REGULAR TOPOLOGIES FOR GIGABIT WIDE-AREA NETWORKS: CONGESTION AVOIDANCE TESTBED EXPERIMENTS

ITAD-8600-FR-94-005

Volume 3

Project 8600

Volume 3

Barbara A. Denny, Computer Scientist
Paul E. McKenney, Sr. Computer Scientist
Danny Lee, Research Engineer
Information and Telecommunications Sciences Center

Prepared for:

National Aeronautics and Space Administration
Ames Research Center
Moffett Field, California 94035

Attn: Dr. Henry Lum, Code RI, M/S: 244-7

and

Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, Virginia 22203-1714

Attn: Major Michael St. Johns

Approved by:

Michael S. Frankel, Vice President and Director
Information, Telecommunications, and Automation Division

SRI International 333 Ravenswood Avenue • Menlo Park, CA 94025-3493 • (415) 326-6200 • FAX: (415) 326-5512 • Telex: 334486

CONTENTS

1 INTRODUCTION 1

2 TRAFFIC GENERATOR SOURCE 3

2.1 TG DECLARATIONS 4

2.2 TG SOURCE 53

2.3 DCAT DECLARATIONS 195

2.4 DCAT SOURCE 205

3 SOURCE FOR TG ANALYSIS TOOLS 219

4 SFQ SOURCE 255

4.1 INSTALLATION NOTES 255

4.2 SOURCE 257

5 SFQ PLUS VIRTUAL CLOCK SOURCE 325

1 INTRODUCTION

This document is Volume 3 of the final technical report on the work performed by SRI International (SRI) on SRI Project 8600. The document includes source listings for all software developed by SRI under this effort. Since some of our work involved the use of ST-II and the Sun Microsystems, Inc. (Sun) High-Speed Serial Interface* (HSI/S) driver, we have included some of the source developed by LBL and BBN† as well. In most cases, our decision to include source developed by other contractors depended on whether it was necessary to modify the original code. If we have modified the software in any way, it is included in this document. In the case of the Traffic Generator (TG), however, we have included all the ST-II software, even though BBN performed the integration, because the ST-II software is part of the standard TG release. It is important to note that all the code developed by other contractors is in the public domain, so that all software developed under this effort can be re-created from the source included here. Therefore, the rest of the document is as follows:

- Section 2—TG and dcat source
- Section 3—Source for the TG analysis tools (i.e., the perl scripts)
- Section 4—SFQ‡ source, using the standard UNIX queueing mechanisms
- Section 5—SFQ plus VirtualClock, using the BBN traffic control abstraction.

*All product names mentioned in this document are the trademarks of their respective holders.

†LBL: Lawrence Berkeley Laboratory; BBN: Bolt Beranek and Newman Inc.

‡SFQ: Stochastic Fairness Queueing.

2 TRAFFIC GENERATOR SOURCE

This section contains the source for the TG and dcat, the program that converts the binary output of TG into an ASCII file. The source for each program is divided into two parts. The first part contains the header files (known as .h files) while the second part is the actual C code. Therefore, Subsections 2.1 and 2.2 are the header files and code for TG, while Subsections 2.3 and 2.4 are the header files and code for dcat. All the source in this section, including a Makefile, is available via anonymous FTP on ftp.erg.sri.com in the pub/tg directory.

2.1 TG DECLARATIONS

```
/*****************************************************************************
 *                                                                           *
 * File: config.h                                                            *
 *                                                                           *
 * Protocol definition structures.                                           *
 *                                                                           *
 * Written 08-Aug-90 by Paul E. McKenney, SRI International.                 *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char config_h_rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/
src/tg/RCS/config.h,v 1.9 90/11/26 13:18:55 dlee Exp $";
#endif lint

/* Maximum packet buffer size.      */

#define MAX_PKT_SIZE  8192 /* 3072 /* Sized for Ethernet. */

/* Maximum value from random-number generator.    */

#define MAX_RANDOM  0x7fffffff

/*
 * FD_SET and associated macros are not defined in SunOS 3.5
 * We need to defined them here, if not previously defined.
 */
#ifndef NFDBITS

#define NFDBITS (30) /* bits per mask */

#define FD_SET(n, p) ((p)->fds_bits[(n)/NFDBITS] |= (1 << ((n) % NFDBITS)))
#define FD_CLR(n, p) ((p)->fds_bits[(n)/NFDBITS] &= ~(1 << ((n) % NFDBITS)))
#define FD_ISSET(n, p) ((p)->fds_bits[(n)/NFDBITS] & (1 << ((n) % NFDBITS)))
#define FD_ZERO(p) bzero((char *)(p), sizeof (*(p)))

#endif
```

```
/*****************************************************************************
 *                                                                           *
 * File: decode.h                                                            *
 *                                                                           *
 * Header file for encode and decode compressed integers.                    *
 *                                                                           *
 * Written 12-Sep-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char decode_h_rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/
src/tg/RCS/decode.h,v 1.5 90/11/26 12:29:24 dlee Exp $";
#endif lint

extern char *decode_ulong();
extern int encode_response();
extern int encode_special_response();
extern char *encode_ulong();
```

```
/*****************************************************************************
 *                                                                           *
 * File: distribution.h                                                      *
 *                                                                           *
 * Structs defining additional parameters for those probability              *
 * distributions that need them.                                             *
 *                                                                           *
 * Written 20-Jun-90 by Paul E. McKenney, SRI International.                 *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char distribution_h_rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/
dartnet/src/tg/RCS/distribution.h,v 1.7 90/11/26 12:29:29 dlee Exp $";
#endif lint

/* Type definitions local to this file.    */

typedef struct
 {
 double  (*generate)(); /* generate a new variable. */
 double  par1;  /* distribution parameter 1. */
 double  par2;  /* distribution parameter 2. */
 double  par3;  /* distribution parameter 3. */
 double  par4;  /* distribution parameter 4. */
 char  *pars;  /* pointer to more parameters */
    /* for those distributions that */
    /* need them.   */
 }  distribution;

typedef struct
 {
 double  mean[2]; /* mean time in each state. */
 unsigned long p[2];  /* probability of remaining in */
    /* same state.   */
 distribution *dist[2]; /* distribution in each state. */
 int  state;  /* current state.  */
 }  dist_markov2; /* 2-state markov distribution. */

extern char  *dist_const_init();
extern char  *dist_exp_init();
extern char  *dist_markov2_init();
extern char  *dist_uniform_init();
```

```
/*****************************************************************************
 *                                                                           *
 * File: log.h                                                               *
 *                                                                           *
 * Header file for handling log files                                        *
 *                                                                           *
 * Written 12-Sep-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char log_h_rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/src/
tg/RCS/log.h,v 1.7 90/11/26 12:29:35 dlee Exp $";
#endif lint

#define LOG_VERSION 1
#define LOG_SUBVERSION 0

/*
 * A log file entry consists of a tuple consisting of the following fields:
 *
 * <Record type> <Record control> <Record value>
 */

/* Record type field enumerations */

#define LOGTYPE_RX  1
#define LOGTYPE_TX  2
#define LOGTYPE_SETUP  3
#define LOGTYPE_TEARDOWN 4
#define LOGTYPE_ACCEPT  5
#define LOGTYPE_ERROR  6

/* Control field modifier bit definitions */

#define LOGCTL_SCHED  (0x1<<0)
#define LOGCTL_ADDR  (0x1<<1)
#define LOGCTL_2ADDR  (0x1<<2)
#define LOGCTL_EXCEPT  (0x1<<3)

/* Error codes when record type is set to LOGTYPE_ERROR */

#define LOGERR_INTFMT  1 /* Script format error */
#define LOGERR_MEM  2 /* Out of memory */
#define LOGERR_2SETUP  3 /* Two connections were established */
#define LOGERR_GETTIME  4 /* gettimeofday() failed. */
#define LOGERR_SELECT  5 /* select() failed. */
#define LOGERR_FCNTL  6 /* fcntl() failed. */
#define LOGERR_GETPEER  7 /* getpeername() failed. */

#define BEGIN_HDR_STRING "<Begin TG Header>\n"
```

```
#define END_HDR_STRING  "<End TG Header>\n"

/* The following routines are exported */

FILE *log_open();
int log_init();
void log_tx();
void log_rx();
void log_accept();
void log_setup();
void log_teardown();
void log_error();
```

```
/*****************************************************************************
 *                                                                           *
 * File: protocol.h                                                          *
 *                                                                           *
 * Protocol definition structures.                                           *
 *                                                                           *
 * Written 20-Jun-90 by Paul E. McKenney, SRI International.                 *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char protocol_h_rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/
src/tg/RCS/protocol.h,v 1.8 90/11/26 12:29:54 dlee Exp Locker: dlee $";
#endif lint

/* Convert the double d to timer tvp.     */

#define dtotimeval(d, tvp) \
 { \
 (tvp)->tv_sec = floor(d); \
 (tvp)->tv_usec = ((d) - (tvp)->tv_sec) * 1000000; \
 }

/* Convert the double d to timer tvp, offsetting from current time. */

#define dtotimevalfromnow(d, tvp) \
 { \
 unsigned long seconds; \
 \
 if (gettimeofday(tvp, (struct timezone *)NULL) == -1) \
  { \
  (void)perror("gettimeofday"); \
  abort(); \
  }\
 seconds = floor(d); \
 (tvp)->tv_sec += seconds; \
 (tvp)->tv_usec += ((d) - seconds) * 1000000; \
 if ((tvp)->tv_usec >= 1000000) \
  { \
  (tvp)->tv_usec -= 1000000; \
  (tvp)->tv_sec++; \
  } \
 }

/* Convert the double d to timer tvp, offsetting from specified time. */

#define dtotimevalfromthen(then, d, tvp) \
 { \
 unsigned long seconds; \
 \
 seconds = floor(d); \
 (tvp)->tv_sec = (then)->tv_sec + seconds; \
```

```
(tvp)->tv_usec = (then)->tv_usec + ((d) - seconds) * 1000000; \
if ((tvp)->tv_usec >= 1000000) \
 { \
 (tvp)->tv_usec -= 1000000; \
 (tvp)->tv_sec++; \
 } \
}

/* Add tvp1 and tvp2, putting the result into result. result may */
/* alias either tvp1 or tvp2 or both, if desired.   */

#define timeradd(tvp1, tvp2, result) \
 { \
 (result)->tv_sec = (tvp1)->tv_sec + (tvp2)->tv_sec; \
 (result)->tv_usec = (tvp1)->tv_usec + (tvp2)->tv_usec; \
 if ((result)->tv_usec >= 1000000) \
  { \
  (result)->tv_usec -= 1000000; \
  (result)->tv_sec++; \
  } \
 }

/* Subtract tvp2 from tvp1, putting the result into result. result */
/* may alias either tvp1 or tvp2, if desired.    */

#define timersub(tvp1, tvp2, result) \
 { \
 (result)->tv_sec = (tvp1)->tv_sec - (tvp2)->tv_sec; \
 if ((tvp1)->tv_usec >= (tvp2)->tv_usec) \
  (result)->tv_usec = (tvp1)->tv_usec - (tvp2)->tv_usec; \
 else \
  { \
  (result)->tv_usec = 1000000 + \
     (tvp1)->tv_usec - \
     (tvp2)->tv_usec; \
  (result)->tv_sec--; \
  } \
 }

/* Type definitions local to this file.    */

typedef struct tg_action_  /* TG action structure.  */
 {
 struct tg_action_ *next;
 int  tg_flags; /* TG flags.   */
 struct timeval start_at; /* time to begin this action. */
 struct timeval stop_before; /* time to be done w/ action. */
 distribution arrival; /* interarrival time.  */
 long  data_limit; /* max amt of data to send. */
 long  packet_limit; /* max amt of data to send. */
 distribution length;  /* packet length.  */
 struct timeval patience; /* patience duration.  */
```

```
 distribution resplen; /* response length distribution.*/
 long  seed;  /* new RNG seed.  */
 struct timeval time_limit; /* max amt of time to be sending*/
 }  tg_action;

/* tg_flags definitions.      */

#define TG_ARRIVAL      0x0001  /* Got arrival distribution. */
#define TG_DATA       0x0002  /* Got data send limit.  */
#define TG_LENGTH       0x0004  /* Got packet length distr. */
#define TG_PATIENCE       0x0008  /* Got patience duration. */
#define TG_RESPLEN      0x0010  /* Got response length.  */
#define TG_SEED         0x0020  /* Got RNG seed.  */
#define TG_SETUP      0x0040  /* Got setup command.  */
#define TG_START      0x0080  /* Got explicit start time. */
#define TG_STOP       0x0100  /* Got explicit stop time. */
#define TG_TIME       0x0200  /* Got time limit.  */
#define TG_WAIT       0x0400  /* Got wait command.  */
#define TG_PACKET       0x0800  /* Got packet limit.  */
#define TG_RESET      0x1000  /* Got reset command.  */

/* Protocol switch table definition.     */

typedef struct    /* Protocol table entry. */
 {
 char  *name;  /* name of protocol.  */
 short  af;  /* Address family.  */
 long  (*setup)(); /* connection setup function. */
    /* protocol *  */
    /* returns -1 if cannot setup. */
 int  (*teardown)(); /* connection teardown function.*/
    /* unsigned long cxn */
    /* returns -1 if cannot teardown*/
 int  (*rcv)(); /* to receive incoming pkts. */
    /* TG-SUPPLIED!!!  */
    /* unsigned long rx */
    /* unsigned long tx */
    /* char *buf  */
    /* int len   */
    /* unsigned long pktid */
    /* The routine should return 1 */
    /* if it has disposed of the */
    /* buffer (e.g., by modifying */
    /* it and passing it to send), */
    /* otherwise it should return 0.*/
 int  (*send)(); /* to send out a packet. */
    /*  unsigned long tx */
    /* char *buf  */
    /* char *len  */
    /* struct timeval *endtout */
    /* unsigned long *pktid */
    /* NULL endtout says to wait */
```

```
    /* forever, if necessary. */
    /* Returns the number of bytes */
    /* actually sent, which will */
    /* normally be equal to len. */
    /* Returns -1 if an error */
    /* occurs (such as a timeout) */
    /* and sets errno appropriately.*/
void  (*sleep_till)(); /* Suspends until the specified*/
    /* time, processing any packets */
    /* that arrive in the interim. */
    /*  struct timeval *waketime*/
char  *(*buffer_get)(); /* request buffer to be used */
    /* to compose packets to be */
    /* output.   */
    /* This allows protocols to */
    /* avoid packet copying. */
    /* unsigned long maxlen */
    /* Return NULL if no more bufs. */
void  (*buffer_free)(); /* return buffer to freelist. */
    /*  char *buf  */
int  (*atoaddr)(); /* parse an ascii string into */
    /* a sockaddr structure, return */
    /* false if unsuccessful. */
    /* char *addr  */
    /* struct sockaddr *s */
int  (*addrtoa)(); /* format a sockaddr structure */
    /* into an ascii string, return */
    /* false if unsuccessful. */
    /* struct sockaddr *s */
    /* char *addr  */
char  *(*btoaddr)(); /* parse a binary log address */
    /* a sockaddr structure, return */
    /* pointer to first byte of */
    /* addr that was not consumed. */
    /* char *addr  */
    /* struct sockaddr *s */
char  *(*addrtob)(); /* format a sockaddr structure */
    /* into an binary log address, */
    /* return pointer to first byte */
    /* of addr that was not  */
    /* overwritten.   */
    /* struct sockaddr *s */
    /* char *addr  */
}  protocol_table;

extern protocol_table *find_protocol();

/* Protocol connection definition struct.    */

typedef struct    /* protocol defn structure. */
 {
 int  qos;  /* Quality of Service flags. */
```

```
 long  rcvwin;
 long  sndwin;
 struct sockaddr src;
 struct sockaddr dst;
 double  avg_bandwidth;
 double  peak_bandwidth;
 double  avg_delay;
 double  peak_delay;
 double  avg_loss;
 double  peak_loss;
 double  interval;
 unsigned long mtu;
 protocol_table *prot;  /* pointer to protocol table */
    /* entry.   */
 }  protocol;

/* Quality of service definitions.     */

#define QOS_AVG_BANDWIDTH 0x0001 /* Got average bandwidth. */
#define QOS_PEAK_BANDWIDTH 0x0002 /* Got peak bandwidth.  */
#define QOS_AVG_DELAY 0x0004 /* Got average delay.  */
#define QOS_PEAK_DELAY 0x0008 /* Got peak delay.  */
#define QOS_AVG_LOSS 0x0010 /* Got average loss rate. */
#define QOS_PEAK_LOSS 0x0020 /* Got peak loss rate.  */
#define QOS_INTERVAL 0x0040 /* Got averaging interval. */
#define QOS_MTU 0x0080 /* Got max transmission unit. */
#define QOS_RCVWIN 0x0100  /* Got receive window size. */
#define QOS_SNDWIN 0x0200  /* Got send window size. */


#define QOS_INTERACTIVE 0x1000 /* Simulate interactive session.*/
#define QOS_SRC 0x2000 /* Got source address.  */
#define QOS_DST 0x4000 /* Got destination address. */
#define QOS_SERVER 0x8000 /* Set up server socket to */
  /* accept incoming connections. */
```

```
#ifndef _ST2_API_H_
#define _ST2_API_H_ 1
#ifndef lint
static char rcsid_st2_api_h[] = "\
@(#) $Header: st2_api.h,v 1.11+ 92/04/03 18:30:20 clynn Exp $ \n";
/*-------------------------------------------------------------------*
 | Copyright (c) 1991-1992 by BBN Systems and Technologies,          |
 | A Division of Bolt Beranek and Newman Inc.                        |
 |                                                                   |
 | Permission to use, copy, modify, distribute, and sell this        |
 | software and its documentation for any purpose is hereby          |
 | granted without fee, provided that the above copyright notice     |
 | and this permission appear in all copies and in supporting        |
 | documentation, and that the name of Bolt Beranek and Newman       |
 | Inc. not be used in advertising or publicity pertaining to        |
 | distribution of the software without specific, written prior      |
 | permission. BBN makes no representations about the suitability    |
 | of this software for any purposes. It is provided "AS IS"         |
 | without express or implied warranties.                           |
 *-------------------------------------------------------------------*/
#endif lint

/*
M* st2_api.h ST-II Application Programming Interface Definitions.
M*
M* This include file contains the Application Programming
M* Interface Definitions for ST-II, an IP-layer protocol for
M* experimentation with guaranteed quality of service, etc. (see
M* RFC 1190 for a description or the ST-II Protocol). It
M* contains all definitions required by an ST-II Application, and
M* then some. These Application Interface Definitions are, as is
M* ST-II, intended to provide flexibility and permit
M* experimentation. These definitions are for use with a
M* BSD-style sockets interface. To maximize transparency and
M* extensibility, information crosses the OS/Application
M* interface in network-byte order. This file also contains some
M* architecture and compiler specific datatype definitions.
M*
m* Status:
m* Features:
m* All the definitions that an ST-II Application should require.
m* Configuration & Management definitions.
m* Data Timestamps.
m* Untested Features:
m* Restrictions/Bugs:
m* No support for Full-Duplex, Changes, full HID Negotiation,
m* ??? Notify, or Status.
m* Things to do:
m* Complete object descriptions.
m* Add more InitXxx definitions.
m* Add support for getsockopt/setsockopt.
m*
```

```
 */

/*
 * Experimenter Identification, to provide a challenge for the
 * monitoring software! If you want to experiment with this software,
 * please get yourself a number (they are cheap, send a message to
 * stii-bugs@BBN.Com). If you have comments or suggestions, send them
 * there, too.
 *
 * Initial DARTNet experimenters ...
 */

#define ExperimenterList \
 aExp(0x00000000,ExpOfficial,stii-bugs@BBN.Com) \
 aExp(0x08000000,ExpBBN,CLynn@BBN.Com) \
 aExp(0x04000000,ExpISI,Casner@ISI.Edu) \
 aExp(0x0C000000,ExpLBL,) \
 aExp(0x02000000,ExpMIT,) \
 aExp(0x0A000000,ExpSRI,) \
 aExp(0x06000000,ExpUDel,) \
 aExp(0x0E000000,ExpUSC,) \
 aExp(0x01000000,ExpXerox,) \
 aExp(0x09000000,ExpINRIA,andry@purcell.inria.fr) \


#define Experimenter ExpOfficial /* Your Experimenter Id goes here */


/* Module Revision History
 *
 * bugid 4,0
 * $Log: st2_api.h,v $
 * Revision 1.11 92/04/03 18:30:20 clynn
 * Release for DARTNet. Virtual Clock enforcement & related changes.
 *
 * Revision 1.10 91/11/26 23:10:03 clynn
 * Made print format of config parameters and stats identical.
 *
 * Revision 1.9 91/11/04 09:20:56 clynn
 * Updated for Public Domain Release. Major changes: addition of Source
 * Routing, IP Encapsulation, HELLO protocol between neighbors, tracking
 * of neighbors, detection of component or agent failures & notification
 * to applications. More consistant naming and format, including making
 * all external names begin with "st2_". Moved some routines and data
 * structures to reduce external references.
 *
 * Added more documentation and added missing pcode parameter to
 * InitaFlowSpec.
 *
 * Revision 1.8 91/05/28 16:17:11 clynn
 * New features: Add new targets from Application layer, UserData support,
 * basic bandwidth reservation for point-to-point links, more complete
```

```
 * state tables, added pcode parameter to InitaFlowSpec3, extended packet
 * buffer abstraction, added network interface abstraction; ststat utility.
 * Bug fixes: ADDR IN USE problem, data send problem, causes of some
 * crashes, cleanup of protocol control blocks.
 * Work around: DARTNET receive memory leak.
 * Eliminated several small modules to reduce globals; adeded Makefile.
 *
 * Revision 1.7 91/04/16 16:13:34 clynn
 * Comma missing in definition of InitaTarget.
 *
 * Revision 1.1 91/03/15 18:33:05 clynn
 * Initial revision
 */


/* Definitions that should be in <sys/socket.h> ... */

#ifndef AF_COIP /* Defer to local site */
#define AF_COIP 23 /* COIP protocol family */
#endif AF_COIP /* see also ConfigParm (coip_family) */

#ifndef PF_COIP
#define PF_COIP AF_COIP /* Same value, different semanitcs! */
#endif PF_COIP


/* PF_COIP Definitions for ST (basically, same as for IP) */

#define STPROTO_ST 0 /* Dummy for ST */
/*#define STPROTO_TCP IPPROTO_TCP /* tcp */
/*#define STPROTO_UDP IPPROTO_UDP /* user datagram protocol */
#define STPROTO_RAW 255 /* raw ST packet */

#define IPPROTO_ST 5 /* Assigned Numbers */

/* From <netinet/in.h> ... */

#ifndef INADDR_ANY
#define INADDR_ANY  (u_long)0x00000000
#endif INADDR_ANY

#ifndef s_addr    /* Defer to local site */
/*
 * Internet address
 */
struct in_addr {
 union {
unsigned long  S_addr; /* An Internet Address */
 }    S_un;
#define s_addr S_un.S_addr
};
```

```
/*
 * Socket address, internet style.
 */
struct sockaddr_in {
 short sin_family; /* I.E., AF_INET */
 unsigned short sin_port; /* Transport-layer multiplexing info */
 struct in_addr sin_addr; /* Identity of a system */
 unsigned char sin_zero[8]; /* Unused by AF_INET */
};
#endif  s_addr



/* Portability Definitions */
/* Machine dependent definitions */

#if defined(mc68020) || defined(mc68030) || defined(sparc)
typedef unsigned char  octet1;
typedef unsigned short  octet2;
typedef unsigned long  octet4;
#define octet8 double  /* yea, but what's better?? */
#define NetOrder 1
#define ntoh1(x) x
#define hton1(x) x
#define ntoh2(x) x
#define hton2(x) x
#define ntoh4(x) x
#define hton4(x) x
#define ntoh8(x) x
#define hton8(x) x
#define ntohIP(x) ((INETADDR)(x))
#define htonIP(x) ((octet4)(x))
#endif


#if defined(mc68020) || defined(mc68030)
#define ALIGNMENT (1)
#endif defined(mc68020) || defined(mc68030)


#if defined(sparc)
#define ALIGNMENT (3)
#endif defined(sparc)

    /* Not to exceed ... */
#define MAX_ST_CTL (1024) /* from <sys/mbuf.h> MCLBYTES */
#define MAX_ST_NAM (112) /* from <sys/mbuf.h> MLEN */



/* Miscellaneous macros */
```

```
/* Split a number up into k bytes */
#define Bytes1(n) (n)
#define Bytes2(n) ((n)>>8),(0xff & (n))
#define Bytes3(n) ((n)>>16),(0xff & ((n)>>8)),(0xff & (n))
#define Bytes4(n) ((n)>>24),(0xff & ((n)>>16)),(0xff & ((n)>>8)),(0xff & (n))
/* others can be defined as required */


/* Build a symbol from three components */
#define Cat2(y,z)Ident(y)z


/* Build a comma-separated list */
#define Cma(x) x,


/* Find the dimension of an array */
#define DimensionOf(array) (sizeof (array) / sizeof (array[0]))


/* Read and write IP Addresses */
#define GetIPAdr(p,subscript) (unsigned long) (p)->IPAdr subscript
#define SetIPAdr(p,subscript,v) (p)->IPAdr subscript = (octet4) v


/* Build a symbol from two components */
#define Ident(x)x


/* Compare unsigned long quaantities */
#define LaterThan(A,B) ((((B)-(A)) & ((~0) - ((unsigned long) (~0) >> 1))) !=
0)


/* Bulid an n-bit mask */
#define NBitMask(n) ((1 << (n)) - 1)


/* Round a number up modulo the size of a data type or structure. */
#define ROUND(n,t) (((n) + sizeof (t) - 1) & ~ (sizeof (t) - 1))



/*
P* ST-II API Parameter Definitions.
P*
P* To provide maximum flexibility and extensibility, we make the
P* underlying ST-II Protocol definitions available to the Application.
P* This should make it easier to add new Parameters and Messages for
P* experimental purposes (plus eliminating reformatting by some layer);
P* the dwonside is that the Application isn't isolated from changes in
```

```
P* the underlying Protocols.
P*
 */


/* Maximum value of a Parameter's plen field, ((8*sizeof (plen))-1) & ~ 3 */
#define MaxPlen (252)


/*
p* Macros to build specific structures to be passed to ST by Application.
p*
p* For each Parameter we define a structure prototype and
p* structure initializer. The prototypes may be customized by
p* each Application when instantiated (at compile time), thus
p* simplifying (really!) generation of parameters whose
p* structure does not vary within an Application. See the
p* example program (example.c) for one way to use them. NB:
p* Since the order of parameters is not constrained by ST-II
p* these prototypes cannot be used to "parse" parameters passed
p* by ST to the Application!
p*
 */

/* ------------------------------------------------------------------------ */

/*
P* InstaApplEntity (name,sapbytes,srcruts)
p*
p* Structure to specify Application Entity Identifier pseudo-parameters
p* consisting of the tuple <IPAdr NextPcol SAPBytes SAP>.
p* PCodes: STLclAppEnt, STRmtAppEnt.
p* "name" is the optional name of the variable created.
p* "sapbytes" is the length of the SAP used by the Application (required).
p* "srcruts" is an optional list of Source Route Parameters (InstaSrcRut).
p*
 */
#define InstaApplEntity(name,sapbytes,srcruts)    \
 struct Ident (aApplEntity)name {     \
 octet1 pcode, plen, NextPcol, SAPBytes;    \
 octet4 IPAdr;      \
 octet1 SAP[(((sapbytes)+3)/4)*4];     \
 srcruts } name

/*
P* InitaApplEntity (type,self,ipadr,nextpcol,sapbytes,sap,srcruts)
p*
p* Initializer for an InstaApplEntity.
p* "type" is either STLclAppEnt or STRmtAppEnt.
p* "self" is the variable representing the structure, e.g., "name".
p* "ipadr" is the desired IP Address (or INADDR_ANY).
p* "nextpcol" identifies the protocol layer above ST.
```

```
p* "sapbytes" is the length of the SAP used by the Application (required).
p* "sap" is a comma-separated list of the bytes of the SAP (see Bytes2).
p* "srcruts" is a list of Source Route Initializers (InitaSrcRut).
p*
 */
#define InitaApplEntity(type,self,ipadr,nextpcol,sapbytes,sap,srcruts) \
 { hton1 (type),hton1 (sizeof (self)),hton1 (nextpcol),   \
 hton1 (sapbytes), htonIP (ipadr),{sap},srcruts }

InstaApplEntity (,4,);  /* Generic definition for 1-4 byte SAPs */

/* ------------------------------------------------------------------------ */

/*
P* InstaDetectorIPAdr (name)
p*
p* Structure to specify a Detector IP Address Pseudo Parameter.
p* Passed to application in the control message structure.
p* PCodes: STDetectorIPAdr.
p* "name" is the optional name of the variable created.
p*
 */

#define InstaDetectorIPAdr(name) struct Ident (aDetectorIPAdr)name { \
 octet1 pcode, plen;      \
 octet2 pad2;      \
 octet4 IPAdr; } name

InstaDetectorIPAdr ();   /* Generic Definition */

/* ------------------------------------------------------------------------ */

/*
P* InstaErrdPDU (name,pdubytes)
p*
p* Structure to specify a Errored PDU Parameter.
p* PCode: STpErrdPDU.
p* "name" is the optional name of the variable created.
p* "pdubytes" is the number of bytes to reserve for the Errored PDU.
p*
 */
#define InstaErrdPDU(name,pdubytes) struct Ident (aErrdPDU)name { \
 octet1 pcode, plen,  pdupbytes, erroroffset;   \
 octet1   pduinerror[(((pdubytes)+3)/4)*4]; } name

InstaErrdPDU (,4);  /* Generic definition, not very useful */


/*
P* InstaFlowSpec (name)
p*
p* Structure to specify Resources required by an Application.
```

```
p* Multiple versions are expected during the course or the research.
p* All begin with a common 4-byte header which idenitfies the actual
p* version. Ideally, only the Resource Management Module (see
p* st2_resource.c) actually needs to know the details of the specific
p* versions; we aren't quite there yet.
p* PCodes: STpFlowSpec, STpRFlowSpec.
p* "name" is the optional name of the variable created.
p*
 */
#define InstaFlowSpec(name) struct Ident (aFlowSpec)name {  \
 octet1 pcode, plen, FlowVer, fill1; } name

InstaFlowSpec ();   /* Generic header */

/* ------------------------------------------------------------------------ */

/*
P* InstaFlowSpec3 (name)
p*
p* Structure to specify a Version 3 FlowSpec Parameter.
p* RFC 1190 defines the fields in FlowSpec Version 3.
p* "name" is the optional name of the variable created.
p*
 */
#define STFSVer3 (3)  /* Version of RFC 1190 */
#define STFSImpAll (0x8000)
#define STFSUnknown (0x4000)
#define STFSRevChrg (0x2000)

#define InstaFlowSpec3(name) struct Ident (aFlowSpec3)name {  \
 octet1 pcode, plen, FlowVer, Hops, DutyFactor, ErrorRate,  \
  Precedence, Reliability;     \
 octet2 Tradeoffs, RecoveryTimeout, LimitOnCosts, LimitOnDelay, \
  LimitOnPDUBytes, LimitOnPDURate;    \
 octet4 MinBytesXRate, AccdMeanDelay, AddDelayVariance;  \
 octet2 DesPDUBytes, DesPDURate; } name

/*
P* InitaFlowSpec3 (pcode,minbytes,minpps,minbw,desbytes,despps)
p*
p* Initializer for a Version 3 Flowspec for use by Applications
p* only interested in specifying packet size and rate information.
p* "minbytes" is the minimum acceptable number of bytes per packet.
p* "minpps" is the minimum acceptable number of Data packets per second.
p* "minbw" is the minimum acceptable bandwidth (bytes * packets/second).
p* "desbytes" is the (largest useful) desireable number of bytes per packet.
p* "despps" is the (largest useful) desireable number of packets/second.
p*
 */
#define APIRecoveryTimeout (0)  /* msec (default) */
#define APILimitOnDelay (2000) /* msec */
```

```
#define InitaFlowSpec3(pcode,minbytes,minpps,minbw,desbytes,despps) \
 { hton1 (pcode),hton1 (sizeof (struct aFlowSpec3)),hton1 (STFSVer3), \
 0,0,0,0,0,0,hton2 (APIRecoveryTimeout),0,hton2 (APILimitOnDelay), \
 hton2 ((minbytes)),hton2 (((minpps)*10)),hton4 (((minbw)*10)),0,0, \
 hton2 ((desbytes)),hton2 (((despps)*10)) }

InstaFlowSpec3 (); /* Generic Version 3 FlowSpec */

/* ------------------------------------------------------------------------ */

/*
P* InstaFreeHIDs (name,nmasks)
p*
p* Structure to specify a Free HIDs Parameter.
p* PCodes: STpFreeHIDs.
p* "name" is the optional name of the variable created.
p* "nmasks" is the number of 32-bit HID mask words allocated (required).
p*
 */
#define InstaFreeHIDs(name,nmasks) struct Ident (aFreeHIDs)name { \
 octet1 pcode, plen;      \
 octet2 basehid;      \
 octet4 freehidbitmask[ nmasks ]; } name

InstaFreeHIDs (,2); /* Generic FreeHIDs w/ 64-bits */

/* ------------------------------------------------------------------------ */

/*
P* InstaGroup (name,nsubgroups)
p*
p* Structure to specify a Group or Reverse Group Parameter.
p* PCodes: STpGroup, STpRGroup.
p* "name" is the optional name of the variable created.
p* "nsubgroups" specifies the number of sub-group slots allocated (required).
p*
 */
#define InstaGroup(name,nsubgroups) struct Ident (aGroup)name {  \
 octet1 pcode, plen;      \
 octet2 UniqId;      \
 octet4 IPAdr, Timstmp;      \
 struct msubgroup      \
 { octet2 subgroupid, relation;     \
 } subgroups[nsubgroups]; } name

InstaGroup (,1);  /* Generic Group/RGroup Parameter */

/* ------------------------------------------------------------------------ */

/*
P* InstaHID (name)
p*
```

```
p* Structure to specify a HID or Reverse HID Parameter.
p* An SCM Notify or StatusResponse PDU may contain multiple HID Parameters.
p* PCodes: STpHID, STpRHID.
p* "name" is the optional name of the variable created.
p*
 */
#define InstaHID(name) struct Ident (aHID)name {   \
 octet1 pcode, plen;      \
 octet2 hid; } name

InstaHID ();   /* Generic HID Parameter */

/* ------------------------------------------------------------------------ */

/*
P* InstaMcastAdr (name,mlen)
p*
p* Structure to specify a Multicast Address Parameter.
p* PCodes: STpMcastAdr.
p* "name" is the optional name of the variable created.
p* "mlen" is the number of bytes in the local net layer multicast address
p*  (required).
p*
 */
#define InstaMcastAdr(name,mlen) struct Ident (aMcastAdr)name {  \
 octet1 pcode, plen, localnetbytes, fill1;    \
 octet4 ipmulticastaddress;     \
 octet1 localnetmulticastaddress[ (((mlen)+3)/4)*4 ]; } name

InstaMcastAdr (,6);  /* McastAdr Parameter for Ethernets */

/* ------------------------------------------------------------------------ */

/*
P* InstaName (name)
p*
p* Structure to specify a Name or Reverse Name Parameter.
p* PCodes: STpName, STpRName.
p* "name" is the optional name of the variable created.
p*
 */
#define InstaName(name) struct Ident (aName)name {   \
 octet1 pcode, plen;      \
 octet2 UniqId;      \
 octet4 IPAdr, Timstmp; } name

InstaName ();   /* Generic Name/RName Parameter */

#define sizeofaName (sizeof (struct aName))

/* ------------------------------------------------------------------------ */
```

```
/*
P* InstaNxtHopIPAdr (name)
p*
p* Structure to specify a Next-Hop IP Address Parameter.
p* Used with the Notify SCMP message to identify a potential routing problem.
p* PCodes: STpNxtHopIPAdr.
p* "name" is the optional name of the variable created.
p*
 */
#define InstaNxtHopIPAdr(name) struct Ident (aNxtHopIPAdr)name { \
 octet1 pcode, plen;      \
 octet4 IPAdr; } name

InstaNxtHopIPAdr ();  /* Generic NxtHopIPAdr Parameter */

/* ------------------------------------------------------------------------ */

/*
P* InstaOrigin (name,sapbytes)
p*
p* Structure to specify a Origin Parameter.
p* PCodes: STpOrigin.
p* "name" is the optional name of the variable created.
p* "sapbytes" is the length of the SAP used by the Application (required).
p*
 */
#define InstaOrigin(name,sapbytes) struct Ident (aOrigin)name {  \
 octet1 pcode, plen, NextPcol, SAPBytes;    \
 octet4 IPAdr;      \
 octet1 SAP[ (((sapbytes)+3)/4)*4 ]; } name

InstaOrigin (,4);  /* Generic Origin for 1-3 byte SAPs */

/* ------------------------------------------------------------------------ */

/*
P* InstaOrgTsmp (name)
p*
p* Structure to specify a Origin Timestamp Parameter.
p* PCodes: STpOrgTsmp.
p* "name" is the optional name of the variable created.
p*
 */
#define InstaOrgTsmp(name) struct Ident (aOrgTsmp) {   \
 octet1 pcode, plen;      \
 octet2 fill2;      \
 octet4 timesec, fractsec; } name

InstaOrgTsmp ();   /* Generic Origin Timestamp Param */

/* ------------------------------------------------------------------------ */
```

```
/*
P* InstaParameter (name)
p*
p* Structure to specify a Generic Parameter.
p* PCodes: <all>.
p* "name" is the optional name of the variable created.
p*
 */
#define InstaParameter(name) struct Ident (aParameter)name {  \
 octet1 pcode, plen;      \
 octet2 pdata2;      \
 octet4 pdata4; } name

InstaParameter ();  /* Generic Parameter */

/* ------------------------------------------------------------------------ */

/*
P* InstaReasonCode (name)
p*
p* Structure to specify a Reason Code Pseudo Parameter.
p* Reason (failure) codes are communicated between the Application and
p* ST using this pseudo parameter.
p* PCodes: STReasonCode.
p* "name" is the optional name of the variable created.
p*
 */

#define InstaReasonCode(name) struct Ident (aReasonCode)name {  \
 octet1 pcode, plen;      \
 octet2 reason; } name

/*
P* InitaReasonCode (reason)
p*
p* Initializer for a Reason Code Pseudo Parameter.
p* "reason" is the appropriate enum ST2Errors symbol.
p*
 */
#define InitaReasonCode(reason)      \
 { hton1 (STReasonCode),hton1 (sizeof (struct aReasonCode)),  \
 hton2 (((octet2) reason)) }

InstaReasonCode ();  /* Generic Reason Code Parameter */

/* ------------------------------------------------------------------------ */

/*
P* InstaRecordRoute (name,naddrs)
p*
p* Structure to specify a Record Route Parameter.
p* PCodes: STpRecordRoute.
```

```
p* "name" is the optional name of the variable created.
p* "naddrs" specifies the number of IP Address slots to be allocated
p*  (required).
p*
 */
#define InstaRecordRoute(name,naddrs) struct Ident (aRecordRoute)name { \
 octet1 pcode, plen, fill1, freeoffset;    \
 octet4 IPAdr[naddrs]; } name

InstaRecordRoute (,1);  /* Generic Record Route Parameter */

/* ------------------------------------------------------------------------ */

/*
P* InstaSrcRut (name,naddrs)
p*
p* Structure to specify a Source Route Pseudo Parameter.
p* These pseudo parameters is used within a Target specification to
p* specify source routing information. They may appear multiply and
p* in any combination per Target.
p* PCodes: STpIPLSrcRut, STpIPSSrcRut, STpSTLSrcRut, STpSTSSrcRut.
p* "name" is the optional name of the variable created.
p* "naddrs" specifies the number of IP Address slots to be allocated.
p*
 */
#define InstaSrcRut(name,naddrs) struct Ident (aSrcRut)name {  \
 octet1 pcode, plen;      \
 octet2 fill2;      \
 octet4 IPAdr[naddrs]; } name

/*
P* InitaSrcRut (type,self,ipaddrs)
p*
p* Initializer for a Source Route Parameter.
p* "type" is one of STpIPLSrcRut, STpIPSSrcRut, STpSTLSrcRut, or STpSTSSrcRut.
p* "self" is the variable representing the structure, e.g., "name".
p* "ipaddrs" is a list of "Cma(" IP Address ")" es forming the route.
p*
 */
#define InitaSrcRut(type,self,ipaddrs)     \
 { htonl (type),htonl (sizeof (self)),0,{ipaddrs} }

InstaSrcRut (,1);  /* Generic Source Route Parameter */
   /* NB: sizeof () is minimum valid size */

/* ------------------------------------------------------------------------ */

/*
P* InstaTarget (name,sapbytes,srcruts)
p*
p* Structure to specify a Target, used within a Target List Parameter.
p* "name" is the optional name of the variable created.
```

```
p* "sapbytes" is the length of the SAP used by the Application (required).
p* "srcruts" is an optional list of Source Route Parameters (InstaSrcRut).
p*
 */
#define InstaTarget(name,sapbytes,srcruts) struct Ident (aTarget)name { \
 octet4 IPAdr;      \
 octet1 TargetBytes, SAPBytes;     \
 octet1 SAP[((((sapbytes)+1)/4)*4)+2];    \
 srcruts } name

/*
P* InitaTarget (self,ipadr,sapbytes,sap,srcruts)
p*
p* Initializer for a Target Specification.
p* "self" is the variable representing the structure, e.g., "name".
p* "ipadr" is the desired IP Address (or INADDR_ANY).
p* "sapbytes"is the length of the SAP used by the Application (required).
p* "sap" is a comma-separated list of the bytes of the SAP (see Bytes2).
p* "srcruts" is a list of Source Route Initializers (InitaSrcRut).
p*
 */
#define InitaTarget(self,ipadr,sapbytes,sap,srcruts)   \
 { htonIP ((ipadr)),hton1 (sizeof (self)),hton1 ((sapbytes)),{sap},srcruts },

InstaTarget (,2,);  /* Generic Target for 1-2 byte SAPs */

#define minsizeofaTarget (8)

/* ------------------------------------------------------------------------ */

/*
P* InstaTargetList (name,targets)
p*
p* Structure to specify a Target List Parameter.
p* PCodes: STpTargetList.
p* "name" is the optional name of the variable created.
p* "targets" is a list of Target Specifications (InstaTarget).
p*
 */
#define InstaTargetList(name,targets) struct Ident (aTargetList)name { \
 octet1 pcode, plen;      \
 octet2 TargetCount;      \
 targets } name

/*
P* InitaTargetList (self,ntargets,targets)
p*
p* Initializer for a Target List Parameter.
p* "self" is the variable representing the structure, e.g., "name".
p* "ntargets" specifies the number of Target Specifications in the List.
p* "targets" is a list of Target Specification Initializers (InitaTarget).
p*
```

```
 */
#define InitaTargetList(self,ntargets,targets)    \
 { hton1 (STpTargetList),hton1 (sizeof (self)),hton2 (ntargets),targets }

InstaTargetList (,struct aTarget targets[1];); /* Generic Single Target */

InstaTargetList (header,)aTargetList; /* Generic header w/o any Targets */
#define sizeofaTargetListHeader (sizeof (struct aTargetListheader))

/* ------------------------------------------------------------------------ */

/*
p* InstaUserData (name,bytes)
p*
p* Structure to specify a User Data Parameter.
p* It may be used by an Application to communicate setup(or teardown)
p* information between the Origin and Targets of a Stream.
p* PCodes: STpUserData.
p* "name" is the optional name of the variable created.
p* "bytes" is the number of bytes of space to be allocated.
p*
 */
#define InstaUserData(name,bytes) struct Ident (aUserData)name { \
 octet1 pcode, plen;      \
 octet2 userlen;      \
 octet1 userdata[((bytes+3)/4)*4]; } name

InstaUserData (,4);  /* Generic User Data Parameter, 1-3 bytes */

/* ------------------------------------------------------------------------ */

/*
p* Instsockaddr_st2 (name,st_parms)
p*
p* Structure to specify an ST-II sockaddr.
p* ST Requests (enum STRequests) are specified implicitly with certain
p* BSD-style socket calls (bind, connect, listen, accept) or explicitly
p* in the ST-II sockaddr (and cmsghdr) structures.
p* "name" is the optional name of the variable created.
p* "st_parms" is a list of ST Parameters (InstXxx).
p*
 */
#define Instsockaddr_st2(name,st_parms) struct Ident (sockaddr_st2)name { \
 unsigned short  sa_family; /* AF_COIP */   \
 unsigned char  st_options; /* enum STOptions */  \
 unsigned char  st_request; /* enum STRequests */  \
 unsigned long  st_handle; /* ffs */   \
 unsigned long  st_sec, st_fract; /* NTP-format Data Timestamp */ \
 st_parms } name

/*
p* Initsockaddr_st2 (req,opt)
```

```
p*
p* Initializer for a Target List Parameter.
p* "req" is an element of enum STRequests.
p* "opt" is an "|"-separated list of ST Options (enum STOptions).
p*
 */
#define Initsockaddr_st2(req,opt) AF_COIP,opt,req,0,0,0

Instsockaddr_st2 (,struct aParameter st_parms[1];); /* Generic ST sockaddr */

Instsockaddr_st2 (header,)sockaddr_st2; /* Generic header */
#define sizeofsockaddr_st2header (sizeof (struct sockaddr_st2header))

/* ------------------------------------------------------------------------ */

/*
O* ST Options.
o*
o* Options may appear in the st_options field of a sockaddr_st2 structure.
o*
 * For each field/bit combination, we define the bit mask, name, and
 * description of the Option.
 *
 * aSTOption (value,name,description)
 */

#define STOptionList \
 aSTOption (0x80,STOpt1socket,Bind (no support)) \
 aSTOption (0x40,STOptPBit,CONNECT/ACCEPT PTP option) \
 aSTOption (0x20,STOptSBit,CONNECT/ACCEPT NoRecovery) \
 aSTOption (0x10,STOptLBit,CONNECT LockPath option) \
 aSTOption (0x03,STTSR,CONNECT/ACCEPT Data Timestamp) \
 aSTOption (0x00,STTSRNotImp,CONNECT/ACCEPT) \
 aSTOption (0x01,STTSRNo,CONNECT/ACCEPT) \
 aSTOption (0x02,STTSRYes,CONNECT/ACCEPT) \
 aSTOption (0x03,STTSROpt,CONNECT/ACCEPT) \
 aSTOption (0xE0,STDataPriority,Data priority send/recv) \
 aSTOption (0x10,STDataTBit,Data Timestamp send/recv) \


enum STOptions {
#define aSTOption(bits,name,desc) name = bits,
 STOptionList
 syntaxfixer
#undef aSTOption
};

/* ------------------------------------------------------------------------ */

/*
O* ST Requests.
o*
```

```
o* Requests may appear in the st_request field of a sockaddr_st2 structure
o* or the cmsg_type field of a cmsghdr structure.
o*
 * For each Request or Message OpCode, we define the value, name, and
 * description of the Request. (Not all SCMP Messages are directly
 * available to the Application Layer.)
 *
 * aReq (value,name,comment)
 *
 * ApplicationSpecificRequests is proviced to allow applications to
 * insert additional (non-ST-II) elements in the enumeration, if desired.
 */

#define STRequestList \
 aReq(0,STReqUnspec,App->ST request as per function or ST->App No notification
or data) \
 aReq(1,STReqAccept,App->ST ACCEPT connection or ST->App CONNECT received) \
 xReq(2,Ack,) \
 aReq(3,STReqChange,CHANGE sent/received) \
 aReq(4,STReqChangeRequest,App->ST request origin to make change or ST->App
requested change) \
 aReq(5,STReqConnect,App->ST send CONNECT) \
 aReq(6,STReqDisconnect,App->ST send DISCONNECT) \
 xReq(7,ErrorInRequest,) \
 xReq(8,ErrorInResponse,) \
 xReq(9,Hello,) \
 xReq(10,HIDApprove,) \
 xReq(11,HIDChange,) \
 xReq(12,HIDChangeRequest,) \
 xReq(13,HIDReject,) \
 aReq(14,STReqNotify,NOTIFY sent/received) \
 aReq(15,STReqRefuse,App->ST send REFUSE) \
 aReq(16,STReqStatus,Request for status) \
 aReq(17,STReqStatusResponse,Status response) \
 xReq(18,x18,) \
 xReq(19,x19,) \
 xReq(20,x20,) \
 xReq(21,x21,) \
 xReq(22,x22,) \
 xReq(23,x23,) \
 xReq(24,x24,) \
 xReq(25,x25,) \
 xReq(26,x26,) \
 xReq(27,x27,) \
 xReq(28,x28,) \
 xReq(29,x29,) \
 xReq(30,x30,) \
 xReq(31,x31,) \
 aReq(32,STCtlPending,Control information is waiting for application) \
 aReq(33,STReqBind,App->ST bind () ) \
 aReq(34,STReqGroupName,Request a unique GroupName) \
 aReq(35,STReqGroupRel,Release a GroupName) \
```

```
 aReq(36,STReqParms,App->ST specify parametrers) \


#define MAX_ST_REQUEST (36+1)

#ifndef ApplicationSpecificRequests
#define ApplicationSpecificRequests
#endif

enum STRequests {
#define aReq(value,name,comment) name = value,
#define xReq(value,name,comment)
 STRequestList
 ApplicationSpecificRequests
#undef xReq
#undef aReq
 NUM_ST_REQUESTS
};



/* SCM Protocol definitions */

/* Define each SCMP Message (OpCode) object. For each object, we specify the
 * value
 * name
 * QB(option bit)
 * QM(mandatory parameter)
 * QO(optional parameter)
 * QV(r/s VLId required)
 * as per the ST-II Protocol Specification (RFC 1190).
 * Due to C deficiencies, must be in numberic order, starting at 0, w/o any
 * gaps.
 */

/* ??? Add properties of object, */
/* e.g., permit 0 rvlid (Connect, Hello, Status, ?Notify) */
/* ??? Option bits (OB ()), required recv VLIds */
/* ??? add Query/Response, ..., merge api STReq* */


#define ST2OpCodes \
 anOpCode(0,BadOpCode,,,,) \
 anOpCode(1,Accept,,QM(STDetectorIPAdr) QM(STpName) QM(STpFlowSpec) \
  QM(STpTargetList), \
 QO(STpRecordRoute) QO(STpRFlowSpec) QO(STpRName) \
  QO(STpUserData), \
 QV(R) QV(S)) \
 /* HELLOs don't have Names, so neither can their ACKs */ \
 anOpCode(2,Ack,,QM(STReasonCode),QO(STpName),QV(R) QV(S)) \
 anOpCode(3,Change,QB(G),QM(STDetectorIPAdr) QM(STpName) QM(STpFlowSpec), \
 QO(STpTargetList) QO(STpUserData), \
```

```
 QV(R) QV(S)) \
anOpCode(4,ChangeRequest,QB(G),QM(STDetectorIPAdr) QM(STpName) \
  QM(STpFlowSpec), \
 QO(STpTargetList) QO(STpUserData), \
 QV(R) QV(S)) \
anOpCode(5,Connect,QB(H) QB(P) QB(S) QB(TSP), \
 QM(STDetectorIPAdr) QM(STpName) QM(STpOrigin) QM(STpFlowSpec) \
 QM(STpTargetList),QO(eHID) QO(STpGroup) QO(STpMcastAdr) \
 QO(STpRecordRoute) QO(STpRFlowSpec) QO(STpRGroup) QO(STpRHID) \
 QO(STpUserData), \
 QV(S)) \
anOpCode(6,Disconnect,QB(G),QM(STReasonCode) QM(STDetectorIPAdr) QM(STpName),
\
 QO(STpTargetList) QO(STpUserData), \
 QV(R) QV(S)) \
anOpCode(7,ErrorInRequest,,QM(STReasonCode) QM(STDetectorIPAdr), \
 QO(STpName) QO(STpErrdPDU) QO(STpTargetList), \
 QV(R)) \
anOpCode(8,ErrorInResponse,,QM(STReasonCode) QM(STDetectorIPAdr), \
 QO(STpName) QO(STpErrdPDU) QO(STpTargetList), \
 QV(R) QV(S)) \
anOpCode(9,Hello,QB(R),QM(eHelloTimer),QO(STpOrgTsmp),QV(S)) \
anOpCode(10,HidApprove,,QM(eHID) QM(STpName),QO(STpFreeHIDs), \
 QV(R) QV(S)) \
anOpCode(11,HidChange,QB(A) QB(D),QM(eHID) QM(STpName),, \
 QV(R) QV(S)) \
anOpCode(12,HidChangeRequest,QB(A) QB(D),QM(eHID) QM(STpName),, \
 QV(R) QV(S)) \
anOpCode(13,HidReject,,QM(eHID) QM(STpName),QO(STpFreeHIDs), \
 QV(R) QV(S)) \
anOpCode(14,Notify,,QM(STReasonCode) QM(STDetectorIPAdr),QO(STpErrdPDU) \
 QO(STpFlowSpec) QO(STpHID) QO(STpName) QO(STpNxtHopIPAdr) \
 QO(STpRecordRoute) QO(STpTargetList),) \
anOpCode(15,Refuse,,QM(STReasonCode) QM(STDetectorIPAdr) QM(STpName) \
  QM(STpTargetList), \
 QO(STpErrdPDU) QO(STpRecordRoute) QO(STpUserData), \
 QV(R) QV(S)) \
anOpCode(16,Status,QB(H) QB(Q),QM(eHID),QO(STpRName),QV(S)) \
anOpCode(17,StatusResponse,QB(H) QB(Q),QM(eHID) QM(STpRName), \
 QO(STpFlowSpec) QO(STpGroup) QO(STpHID) QO(STpTargetList), \
 QV(R)) \


/* ??? add connect-request */

#ifdef DOCUMENTATION

a) q - ErrorInRequest, r - ErrorInReply, n - no error
b) a - DoAcknowledge
c) has rx parameters
d) wants a valid vlinkp
e) f- notify_pending_Forwarder, t - notify_pending_Target,
```

```
 d - disconnect_Forwarder, all disconect_Target
f) st2_PCBUpdate
g) relay message, q - queue_accepts
```

	a	b	c	d	e	f	g
BadOpCode	q?						
Accept	q	a	y	na	t		y/q
Ack	r			y	f	y	n
Change	q		+2	y			
ChangeRequest	q		+1	y			
Connect	q		y				
Disconnect	q	a	y	y	d	y	
ErrorInRequest	n			y	f	y	n
ErrorInResponse	n			y	f	y	n
Hello	q						
HidApprove	r			y	f	y	
HidChange	q		y				
HidChangeRequest	q		-1				
HidReject	r			y	f	y	
Notify	q		y				
Refuse	q	a	y	y	?	?	y
Status	q						
statusResponse	q						

```
#endif DOCUMENTATION

enum OpCodes {
#define anOpCode(value,name,bits,mparm,oparm,vlids) Ident (Op)name = value,
 ST2OpCodes
#undef anOpCode

 NUM_ST_OPCODES
};

/* ------------------------------------------------------------------------ */

/* Define each PCode object.
 * For each object, we specify the value, structure prefix, enum
 * prefixm, name, min and max lengths, and <more later>, as per
 * the ST-II spec.
 *
 * aPCode (value,struct-prefix,pcode-prefix,(base)name,minlen,maxlen,x)
 * sPCode - source routes, tPCode - targetlist, xPCode - internal
 */

#define ST2PCodes \
 xPCode(0,x,p,Bogus,0,0,Catch unspecified parameter code) \
 aPCode(1,a,STp,ErrdPDU,4,252,Copy of Errored SCM PDU) \
 aPCode(2,a,STp,FlowSpec,36,36,FlowSpec describing Resource Requirements) \
 aPCode(3,a,STp,FreeHIDs,8,36 \
  ,Masks of HIDs which are available at the sender) \
```

```
aPCode(4,a,STp,Group,12,28,Information specifying a Group of Streams) \
aPCode(5,a,STp,HID,4,4 \
 ,Identifies HID in use by a stream between two ST Agents) \
aPCode(6,a,STp,McastAdr,8,20,Communicate multicast address information) \
aPCode(7,a,STp,Name,12,12,Identify a Stream) \
aPCode(8,a,STp,NxtHopIPAdr,8,8,Communicate a Next Hop IP Address) \
aPCode(9,a,STp,Origin,8,16,Identify the Origin of a Stream) \
aPCode(10,a,STp,OrgTsmp,12,12,Communicate Time at Origin when Data was sent)
\
aPCode(11,a,STp,RecordRoute,4,252 \
 ,Parameter to collect stream path information) \
aPCode(12,a,STpR,FlowSpec,36,36 \
 ,FlowSpec describing reverse Resource Requirements) \
aPCode(13,a,STpR,Group,12,28 \
 ,Information specifying a reverse Group of Streams) \
aPCode(14,a,STpR,HID,4,4,Specify HID to be used by Reverse Stream) \
aPCode(15,a,STpR,Name,12,12,Identify a Reverse Stream) \
sPCode(16,a,STpIPL,SrcRut,4,252,Loose IP Source Route) \
sPCode(17,a,STpIPS,SrcRut,4,252,Strict IP Source Route) \
sPCode(18,a,STpSTL,SrcRut,4,252,Loose ST Source Route) \
sPCode(19,a,STpSTS,SrcRut,4,252,Strict ST Source Route) \
tPCode(20,a,STp,TargetList,12,252,List of Targets for a Stream) \
aPCode(21,a,STp,UserData,4,252 \
 ,Communicate Application Data at Setup/Teardown) \
xPCode(22,no,x,22,0,0,x) \
xPCode(23,no,x,23,0,0,x) \
xPCode(24,no,x,24,0,0,x) \
xPCode(25,aApplEntity,STLclAppEnt,,8,252 \
 ,Local Application Entity <IPAdr NextPcol SAPBytes SAP>) \
xPCode(26,aApplEntity,STRmtAppEnt,,8,252 \
 ,Remote Application Entity <IPAdr NextPcol SAPBytes SAP>) \
xPCode(27,no,e,HelloTimer,4,4,x) \
xPCode(28,no,e,HID,2,2,x) \
xPCode(29,a,ST,DetectorIPAdr,4,4,x) \
xPCode(30,a,ST,ReasonCode,2,2,x) \


/* ??? add accounting info */
/* ??? add stats info */

enum PCodes {
#define aPCode(v,s,p,n,mn,mx,x) Ident (p)n = v,
#define sPCode(v,s,p,n,mn,mx,x) Ident (p)n = v,
#define tPCode(v,s,p,n,mn,mx,x) Ident (p)n = v,
#define xPCode(v,s,p,n,mn,mx,x) Ident (p)n = v,
 ST2PCodes
#undef xPCode
#undef tPCode
#undef sPCode
#undef aPCode

 NUM_ST_PARMS
```

```
};


/*
 * Level number for [gs]etsockopt () to apply to ST.
 */

#define SOL_STII (1190)  /* ST-II (RFC 1190) option level */

/*
 * [gs]etsockopt () options
 *
 * NB: ST socket option codes are an extension of the user request
 * function (pr_usrreq in struct protosw) codes (PRU_xxx) defined
 * in <sys/protosw.h>. A gap is left after PRU_NREQ for growth.
 * (*protosw[].pr_usrreq)(so, PRU_xxx, m, nam[sockaddr], opt);
 *
 * getsockopt ( int s, level, option, caddr_t val, int *vallen )
 * setsockopt ( int s, level, option, caddr_t val, int vallen )
 * However, they use the pr_ctloutput function.
 * (*pr_ctloutput)( PRCO_[GF]SETOPT, so, level, opt, struct mbuf **m0 )
 * returning a UNIX errno.
 */

#define STOptPriority (65)

/* ------------------------------------------------------------------------ */

/*
 * Control structure used with recvmsg () and sendmsg () and [gs]etsockopt ()
 */

/*
P* Instcmsghdr (name,parms)
p*
p* Structure to specify a Control Message Header.
p*
p* Used with getsockopt and setsockopt as well as forming the contents
p* of the msg_accrights field of the msghdr structure used with
p* recvmsg and sendmsg system calls.
p*
p* "name" is the optional name of the variable created.
p* "parms" is a list of ST Parameter specifications (InstaXxx).
p*
 */
#define Instcmsghdr(name,parms) struct Ident (cmsghdr)name {  \
 unsigned int  cmsg_len;     \
 int    cmsg_level, cmsg_type;    \
 parms } name

/*
```

```
P* Initcmsghdr (self,request)
p*
p* Initializer for a Control Message Header.
p* "self" is the variable representing the structure, e.g., "name".
p* "request" is the appropriate enum STRequest symbol.
p*
 */
#define Initcmsghdr(self,request) sizeof (self),SOL_STII,request

Instcmsghdr (,unsigned char cmsg_data[4];); /* Generic cmsghdr */

Instcmsghdr (header,)cmsghdr;  /* Generic Header only */
#define sizeofCMsgHdr (sizeof (struct cmsghdrheader))

/* ------------------------------------------------------------------------ */

/*
P* InitMsg (p,name,namelen,iov,niov,ctl,ctllen)
p*
p* The recvmsg () and sendmsg () calls use the struct msg structure.
p* "p" is a pointer to the struct msg.
p* "name" is a pointer to a struct sockaddr_st2.
p* "namelen" is the length of the sockaddr_st2.
p* "iovp" is a pointer to a struct iov array.
p* "niov" is the number of entries in the struct iov array.
p* "ctlp" is a pointer to a struct cmsghdr containing a Parameter list.
p* "ctllen" is the length of the cmsghdr and Parameters.
p*
 */

#define InitMsg(p,name,namelen,iovp,niov,ctlp,ctllen)   \
 (p)->msg_name = (caddr_t) (name);     \
 (p)->msg_namelen = (namelen);     \
 (p)->msg_iov = (struct iovec *) (iovp);    \
 (p)->msg_iovlen = (niov);     \
 (p)->msg_accrights = (caddr_t) (ctlp);    \
 (p)->msg_accrightslen = (ctllen);



/*
 * Error Handling
 *
 * Needs more work :-)
 * What do you do if you DETECT an error
 * What do you do if you RECEIVE an error
 * Might depend on whether in relation to a previous-hop or a next-hop
 * How does the user get informed so that something meaningful can be done
 * How to do generic translation between ST errors and native (OS) errors
 *
 */
```

```
#define ErrorFlagList \
 anEFlg(FatalGroup,Error requires group to be removed) \
 anEFlg(FatalStream,Error requires stream to be removed) \
 anEFlg(FatalTarget,Error requires target to be removed) \
 anEFlg(FatalHop,Error requires all targets to nexthop to be removed) \
 anEFlg(FatalNet,Error requires all targets reached via net to be removed) \
 anEFlg(Reroute,Error requires rerouting) \
 anEFlg(Warning,No action required) \


enum errorflags {
 errorflag0 = 0,   /* zero is unspecified */
#define anEFlg(name,desc) name,
 ErrorFlagList
#undef anEFlg
};

/* ------------------------------------------------------------------------ */

/*
 * ReasonCode
 * Define ST-II Error Objects. For each error: (internal) value, external
 * name, internal name, control flags, and description.
 *
 * ApplicationSpecificErrors is proviced to allow applications to
 * insert additional (non-ST-II) elements in the enumeration, if desired.
 *
 */

#define ST2ErrorList \
 anError(0,NoError,NoError,Q(Warning) \
  ,No error has been detected) \
 anError(1,ErrorUnknown,ErrorUnknown,Q(FatalStream) \
  ,An error not contained in this list has been detected) \
 anError(2,AcceptTimeout,AcceptTimeout,Q(FatalTarget) \
  ,An Accept has not been acknowledged) \
 anError(3,AccessDenied,AccessDenied,Q(FatalStream) \
  ,Access denied) \
 anError(4,AckUnexpected,AckUnexpected,Q(Warning) \
  ,An unexpected ACK was received) \
 anError(5,ApplAbort,ApplAbort,Q(FatalStream) \
  ,The application aborted the stream abnormally) \
 anError(6,ApplDisconnect,ApplDisconnect,Q(FatalTarget) \
  ,The application closed the stream normally) \
 anError(7,AuthentFailed,AuthentFailed,Q(FatalStream) \
  ,The authentication function failed) \
 anError(8,CantGetResrc,CantGetResrc,Q(FatalTarget) \
  ,Unable to acquire (additional) resources) \
 anError(9,CantRelResrc,CantRelResrc,Q(Warning) \
  ,Unable to release excess resources) \
 anError(10,CksumBadCtl,CksumBadCtl,Q(Warning) \
  ,A received control PDU has a bad message checksum) \
```

```
 anError(11,CksumBadST,CksumBadST,Q(Warning) \
  ,A received PDU has a bad ST Header checksum) \
 anError(12,DropExcdDly,DropExcdDly,Q(Warning) \
  ,A received PDU was dropped because it could not be processed within the
delay specification) \
 anError(13,DropExcdMTU,DropExcdMTU,Q(Warning) \
  ,A received PDU was dropped because its size exceeds the MTU) \
 anError(14,DropFailAgt,DropFailAgt,Q(Warning) \
  ,A received PDU was dropped because of a failed ST agent) \
 anError(15,DropFailHst,DropFailHst,Q(Warning) \
  ,A received PDU was dropped because of a host failure) \
 anError(16,DropFailIfc,DropFailIfc,Q(Warning) \
  ,A received PDU was dropped because of a broken interface) \
 anError(17,DropFailNet,DropFailNet,Q(Warning) \
  ,A received PDU was dropped because of a network failure) \
 anError(18,DropLimits,DropLimits,Q(Warning) \
  ,A received PDU was dropped because it exceeds the resource limits for its
stream) \
 anError(19,DropNoResrc,DropNoResrc,Q(Warning) \
  ,A received PDU was dropped due to no available resources (including
precedence)) \
 anError(20,DropNoRoute,DropNoRoute,Q(Warning) \
  ,A received PDU was dropped because of no available route) \
 anError(21,DropPriLow,DropPriLow,Q(Warning) \
  ,A received PDU was dropped because it has a priority too low to be
processed) \
 anError(22,DuplicateIgn,DuplicateIgn,Q(Warning) \
  ,A received control PDU is a duplicate and is being acknowledged) \
 anError(23,DuplicateTarget,DuplicateTarget,Q(Warning) \
  ,A received control PDU contains a duplicate target or an attempt to add an
existing target) \
 anError(24,FailureRecovery,FailureRecovery,Q(Warning) \
  ,A notification that recovery is being attempted) \
 anError(25,FlowVerBad,FlowVerBad,Q(FatalStream) \
  ,A received control PDU has a FlowSpec Version Number that is not supported)
\
 anError(26,GroupUnknown,GroupUnknown,Q(Warning) \
  ,A received control PDU contains an unknown Group Name) \
 anError(27,HIDDeferredMcst,HIDDeferredMcst,Q(Warning) \
  ,Deferred HID selection specified with multicast group) \
 anError(28,HIDNegFails,HIDNegFails,Q(FatalHop) \
  ,HID negotiation failed) \
 anError(29,HIDUnknown,HIDUnknown,Q(FatalHop) \
  ,A received control PDU contains an unknown HID) \
 anError(30,InconsistHID,InconsistHID,Q(FatalHop) \
  ,An inconsistency has been detected with a stream Name and corresponding HID)
\
 anError(31,InconsistGroup,InconsistGroup,Q(Warning) \
  ,An inconsistency has been detected with the streams forming a group) \
 anError(32,IntfcFailure,IntfcFailure,Q(FatalNet) \
  ,A network interface failure has been detected) \
 anError(33,InvalidHID,InvalidHID,Q(FatalHop) \
  ,A received ST PDU contains an invalid HID) \
```

```
 anError(34,InvalidSender,InvalidSender,Q(FatalStream) \
  ,A received control PDU has an invalid SenderIPAddress field) \
 anError(35,InvalidTotByt,InvalidTotByt,Q(FatalStream) \
  ,A received control PDU has an invalid TotalBytes field) \
 anError(36,LnkRefUnknown,LnkRefUnknown,Q(FatalHop) \
  ,A received control PDU contains an unknown LnkReference) \
 anError(37,NameUnknown,NameUnknown,Q(FatalHop) \
  ,A received control PDU contains an unknown stream Name (bad format?)) \
 anError(38,NetworkFailure,NetworkFailure,Q(FatalNet) \
  ,A network failure has been detected) \
 anError(39,NoRouteToAgent,NoRouteToAgent,Q(Reroute) \
  ,Cannot find a route to an ST agent) \
 anError(40,NoRouteToDest,NoRouteToDest,Q(Reroute) \
  ,Cannot find a route to the destination) \
 anError(41,NoRouteToHost,NoRouteToHost,Q(Reroute) \
  ,Cannot find a route to a host) \
 anError(42,NoRouteToNet,NoRouteToNet,Q(Reroute) \
  ,Cannot find a route to a network) \
 anError(43,OpCodeUnknown,OpCodeUnknown,Q(FatalHop) \
  ,A received control PDU has an invalid OpCode field) \
 anError(44,PCodeUnknown,PCodeUnknown,Q(FatalHop) \
  ,A received control PDU has a parameter with an invalid PCode) \
 anError(45,ParmValueBad,ParmValueBad,Q(FatalHop) \
  ,A received control PDU contains an invalid parameter value) \
 anError(46,PcolIdUnknown,PcolIdUnknown,Q(FatalTarget) \
  ,A received control PDU contains an unknown next-higher layer protocol
identifier) \
 anError(47,ProtocolError,ProtocolError,Q(FatalHop) \
  ,A protocol error was detected) \
 anError(48,PTPError,PTPError,Q(FatalHop) \
  ,Multiple targets were specified for a stream created with the PTP option) \
 anError(49,RefUnknown,RefUnknown,Q(FatalHop) \
  ,A received control PDU contains an unknown Reference) \
 anError(50,RestartLocal,RestartLocal,Q(Reroute) \
  ,The local ST agent has recently restarted) \
 anError(51,RemoteRestart,RemoteRestart,Q(Reroute) \
  ,The remote ST agent has recently restarted) \
 anError(52,RetransTimeout,RetransTimeout,Q(FatalHop) \
  ,An acknowledgment to a control message has not been received after several
retransmissions) \
 anError(53,RouteBack,RouteBack,Q(Warning) \
  ,The routing function indicates that the route to the next-hop is through the
same interface as the previous-hop and is not the previous-hop) \
 anError(54,RouteInconsist,RouteInconsist,Q(Reroute) \
  ,A routing inconsistency has been detected e.g. a route loop) \
 anError(55,RouteLoop,RouteLoop,Q(Reroute) \
  ,A CONNECT was received that specified an existing target) \
 anError(56,SAPUnknown,SAPUnknown,Q(FatalTarget) \
  ,A received control PDU contains an unknown next-higher layer SAP (port)) \
 anError(57,STAgentFailure,STAgentFailure,Q(FatalHop) \
  ,An ST agent failure has been detected) \
 anError(58,StreamExists,StreamExists,Q(FatalHop) \
```

```
  ,A stream with the given Name or HID already exists) \
 anError(59,StreamPreempted,StreamPreempted,Q(Reroute) \
  ,The stream has been preempted by one with a higher precedence) \
 anError(60,STVerBad,STVerBad,Q(FatalHop) \
  ,A received PDU is not ST Version 2) \
 anError(61,TooManyHIDs,TooManyHIDs,Q(FatalHop) \
  ,Attempt to add more HIDs to a stream than the implementation supports) \
 anError(62,TruncatedCtl,TruncatedCtl,Q(FatalHop) \
  ,A received control PDU is shorter than expected) \
 anError(63,TruncatedPDU,TruncatedPDU,Q(FatalHop) \
  ,A received ST PDU is shorter than the ST Header indicates) \
 anError(64,UserDataSize,UserDataSize,Q(FatalHop) \
  ,The UserData parameter is too large to permit a control message to fit into
a networks MTU) \

#define ImplErrorList \
 anError(80,ParmValueBad,BadParmLen,Q(FatalHop) \
  ,A parameter length not multiple of 4) \
 anError(81,ErrorUnknown,CantDeactivateActive,Q(Warning) \
  ,An attempt to deactivate a connection which is still active) \
 anError(82,ParmValueBad,DupParm,Q(FatalHop) \
  ,A control message contained multiple instances of a parameter) \
 anError(83,ProtocolError,HelloTimerandDetectorIPAdr,Q(FatalHop) \
  ,A pdu cannot have both a HelloTimer and a DetectorIPAdr) \
 anError(84,ProtocolError,HIDandReasonCode,Q(FatalHop) \
  ,A pdu cannot have both a HID and a ReasonCode) \
 anError(85,ProtocolError,ImplRest,Q(FatalHop) \
  ,Attempt to exceed capabilities of the implementation) \
 anError(86,ErrorUnknown,Inconsistency,Q(FatalHop) \
  ,An unexpected condition has occurred) \
 anError(87,InconsistHID,InconsistVLID,Q(FatalHop) \
  ,An inconsistency has been detected with a stream Name and corresponding
VLID) \
 anError(88,ProtocolError,InvalidArgument,Q(Warning) \
  ,A subroutine was called with an invalid argument) \
 anError(89,ErrorUnknown,InvalidPointer,Q(Warning) \
  ,An attempt to use an undefined pointer has occurred) \
 anError(90,ProtocolError,InvalidRVLId,Q(FatalHop) \
  ,A received SCMP PDU contains an invalid RVLId) \
 anError(91,ProtocolError,InvalidSVLId,Q(FatalHop) \
  ,A received SCMP PDU contains an inconsistent SVLId) \
 anError(92,ProtocolError,InvalidTarget,Q(FatalTarget) \
  ,A Target is improperly specified) \
 anError(93,ProtocolError,MTUtosmall,Q(FatalHop).\
  ,A network MTU is too small for SCMP messages) \
 anError(94,CantGetResrc,NoConHidVlid,Q(FatalHop) \
  ,Unable to acquire a connection block or HID or VLId) \
 anError(95,ErrorUnknown,NoSupport,Q(FatalHop) \
  ,Attempt to use non-supported feature) \
 anError(96,ProtocolError,ParmMissing,Q(FatalHop) \
  ,A mandatory parameter is not present) \
 anError(97,ProtocolError,SourceRouteError,Q(FatalTarget) \
```

```
  ,A Target has an invalid source route) \
 anError(98,CantGetResrc,InvalidRsrcId,Q(FatalHop) \
  ,Use of an invalid resource identifier) \
 anError(99,ErrorUnknown,Misconfiguration,Q(FatalStream) \
  ,A local configuration error has been detected) \
 anError(100,NoError,InProgress,Q(Warning) \
  ,operation in progress) \
 anError(101,ErrorUnknown,Alignment,Q(Warning) \
  ,A memory object is not correctly aligned) \
 anError(102,ErrorUnknown,EncapsulationError,Q(FatalHop) \
  ,Error processing encapsulated ST) \


#ifndef ApplicationSpecificErrors
#define ApplicationSpecificErrors
#endif

enum ST2Errors {
#define anError(vi,vx,n,f,def) n = vi,
 ST2ErrorList

 NProtocolErrors,  /* gets value last+1, # formal errors */

/* Following are not in protocol spec */

 ImplErrorList

 ApplicationSpecificErrorBase = 256,

 ApplicationSpecificErrors
#undef anError

 /* MAX_ST_ERRORS  /* has value last+1, # implementation errors */
#define MAX_ST_ERRORS (127) /* C won't use an enum to declare array */
    /* sizes! */
};



/*
C* ST-II Configuration Definitions
c*
c* The ConfigParmList defining site's configuration parameters is
c* instantiated in st2_proto so sites can make local additions.
c* Only add to the end of the list unless you recompile all of
c* ST-II source!
c*
c* Placed here so Applications can get at the information, too.
c*
c* WARNING: Exercise caution when changing the definitions.
c*
 * name,current value,min value,max value,compile symbol,format,desc
```

```
 * aCfgParm(name,cv,nv,xv,cs,format,desc)
 *
 * format specifies how to print the value as "i" quantities of "ni"
 * bytes each (i < 5) as <i><n1><n2>...<ni><printf format specification>
 * Currently, the <printf format specification> prints 13 characters.
 */


/* ST2_ConfigVersion (3)  /* Source 1.9 */
/* ST2_ConfigVersion (4)  /* Source Release 1.10 */
#define ST2_ConfigVersion (5)  /* Source Release 1.11 */


#define ConfigParm(x) st2_config.x

#define ConfigParmList \
 aCfgParm(size,sizeof (struct aConfigBlock),8,, \
  ,"14%17u.",size of this block) \
 aCfgParm(version,Experimenter|ST2_ConfigVersion,,, \
  ,"14%17x ",Experimenter|ST2_ConfigVersion version of this block) \
 \
 aCfgParm(api_maxctl,MCLBYTES,128,xx,MCLBYTES \
  ,"14%17u.",MTU through API to applications) \
 aCfgParm(api_maxnam,MLEN,16,xx,MLEN \
  ,"14%17u.",MTU through API to applications) \
 aCfgParm(api_MTU,MCLBYTES,128,xx,MCLBYTES \
  ,"14%17u.",MTU through API to applications) \
 aCfgParm(coip_family,AF_COIP,AF_COIP,AF_COIP,AF_COIP \
  ,"14%17u.",Address family for COIP) \
 aCfgParm(ctlflg,CTLFLG,0,0xffffffff, \
  ,"14%17x ",ST2Flags) \
 aCfgParm(dbgflg,DBGFLG,0,0xffffffff, \
  ,"14%17x ",Control diagnostic output) \
 aCfgParm(def_genifs,DEF_GENIFS,2,(#nets+MAX_STREAMS),DEF_GENIFS \
  ,"14%17u.",# nets + Origins + Targets) \
 aCfgParm(def_nxthops,DEF_NXTHOPS,1,xx,DEF_NXTHOPS \
  ,"14%17u.",# nxthops per stream w/o ExpAry) \
 aCfgParm(def_routechoices,2,1,xx,DEF_ROUTECHOICES \
  ,"14%17u.",estimated # routes to a destination) \
 aCfgParm(ethertype,0x5354,1501,0xFFFF, \
  ,"14%17x ",private ethertype when x800 cannot be used) \
 aCfgParm(flowspec_versions,0,0,0xFFFFFFFF, \
  ,"14%17x ",bitmap of supported flowspec versions) \
 aCfgParm(fhid_masks,FHID_MASKS,1,62,FHID_MASKS \
  ,"14%17u.",RFC1190 recommends at least 64 bits) \
 aCfgParm(hellobackoff,20000,1,0xFFFFFFFF,nodefine \
  ,"14%17u.",HELLO interval backoff msec) \
 aCfgParm(hellofiltfactor,2,1,xx,nodefine \
  ,"14%17u.",HELLO filtering factor) \
 aCfgParm(hellolossfactor,5,1,xx,nodefine \
  ,"14%17u.",HELLO loss factor) \
 aCfgParm(hellotimerholddown,10000,1000,60000,nodefine \
```

```
 ,"14%17u.",restart quiet time in msec) \
 aCfgParm(hid_bits,HID_BITS,3,16,ID_BITS \
 ,"14%17u.",# of LSB HID bits we use) \
 aCfgParm(hid_mask,NBitMask (HID_BITS),7,0xFFFF, \
 ,"14%17x ",mask for LSB HID bits we use) \
 aCfgParm(ip_tos,0x00380038,0,0x00FE00FE,nodefine \
 ,"222 %7x %7x ",IP TOS neighbor/stream) \
 aCfgParm(ip_ttl,0x00400040,0,0x00FF00FF,nodefine \
 ,"222 %7u. %7u.",IP TTL neighbor/stream) \
 aCfgParm(max_hids_per_stream,MAX_HIDS_PER_STREAM,2,xx,MAX_HIDS_PER_STREAM \
 ,"14%17u.",) \
 aCfgParm(max_lclhdrlen,MAX_LCLHDRLEN,0,999,MAX_LCLHDRLEN \
 ,"14%17u.",maximum local network header length) \
 aCfgParm(max_st_errors,127,xx,xx,MAX_ST_ERRORS \
 ,"14%17u.",) \
 aCfgParm(max_streams,MAX_STREAMS,3,65536,MAX_STREAMS \
 ,"14%17u.",max simultaneous streams + 2) \
 aCfgParm(n32freehids,(((1 << HID_BITS) + 31) / 32),1,2084, \
 ,"14%17u.",# 32-bit words for (1 << HID_BITS) bits) \
 aCfgParm(nbrRTT0,4000,1,0xFFFFFFFF,nodefine \
 ,"14%17u.",initial value of RTT to a neighbor) \
 aCfgParm(noc0,0,0,0xFFFFFFFF,nodefine \
 ,"41111 %u.%u.%u.%u",primary NOC) \
 aCfgParm(noc0rt,0,0,0xFFFFFFFF,nodefine \
 ,"222 %7u. %7u.",primary NOC reporting rate (*.5 secs)) \
 aCfgParm(noc1,0,0,0xFFFFFFFF,nodefine \
 ,"41111 %u.%u.%u.%u",backup NOC) \
 aCfgParm(noc1rt,0,0,0xFFFFFFFF,nodefine \
 ,"222 %7u. %7u.",backup NOC reporting rate (*.5 secs)) \
 aCfgParm(noc2,0,0,0xFFFFFFFF,nodefine \
 ,"41111 %u.%u.%u.%u",second backup NOC) \
 aCfgParm(noc2rt,0,0,0xFFFFFFFF,nodefine \
 ,"222 %7u. %7u.",second backup NOC reporting rate (*.5 secs)) \
 aCfgParm(ntraps,MAX_TRAPS,1,10000,MAX_TRAPS \
 ,"14%17u.",number of entries in traptable) \
 aCfgParm(num_nexthop,100,xx,xx,NUM_NEXTHOP \
 ,"14%17u.",) \

aCfgParm(num_st_opcodes,NUM_ST_OPCODES,NUM_ST_OPCODES,NUM_ST_OPCODES,NUM_ST_OP
CODES \
 ,"14%17u.",# of OpCodes) \
 aCfgParm(num_st_parms,NUM_ST_PARMS,NUM_ST_PARMS,NUM_ST_PARMS,NUM_ST_PARMS \
 ,"14%17u.",# of Parameters) \
 aCfgParm(rcvryhold,10000,2500,60000,nodefine \
 ,"14%17u.",(notyet)msec to hold streams pending reconnection after failure) \
 aCfgParm(rcvrytodef,10000,2500,60000,DEF_RCVRYTO \
 ,"14%17u.",Default msec to detect failed path) \
 aCfgParm(rcvrytomin,5000,2500,60000,MIN_RCVRYTO \
 ,"14%17u.",minimum msec to detect failed path) \
 aCfgParm(rcvrytonbr,100000,15000,60000,NBR_RCVRYTO \
 ,"14%17u.",msec to detect failed neighbor when no active streams) \
 aCfgParm(recvspace,1024*4,1024,64*1024, \
```

```
 ,"14%17u.",default socket recv space) \
/* BEGIN following are each struct stRetryParms */ \
aCfgParm(rx_Accept,(3<<16)+1000,xx,xx,xx \
 ,"222 %7u. %7u.",ACCEPT retransmission count & interval) \
aCfgParm(rx_Ack,(0<<16)+0000,xx,xx,xx \
 ,"222 %7u. %7u.",ACK retransmission count & interval) \
aCfgParm(rx_Change,(3<<16)+1000,xx,xx,xx \
 ,"222 %7u. %7u.",CHANGE retransmission count & interval) \
aCfgParm(rx_ChangeRequest,(3<<16)+1000,xx,xx,xx \
 ,"222 %7u. %7u.",CHANGEREQUEST retransmission count & interval) \
aCfgParm(rx_Connect,(5<<16)+1000,xx,xx,xx \
 ,"222 %7u. %7u.",CONNECT retransmission count & interval) \
aCfgParm(rx_Disconnect,(3<<16)+1000,xx,xx,xx \
 ,"222 %7u. %7u.",DISCONNECT retransmission count & interval) \
aCfgParm(rx_End2End,(0<<16)+5000,xx,xx,xx \
 ,"222 %7u. %7u.",END2END retransmission count & interval) \
aCfgParm(rx_ErrorInRequest,(0<<16)+0000,xx,xx,xx \
 ,"222 %7u. %7u.",ERRORINREQUEST retransmission count & interval) \
aCfgParm(rx_ErrorInResponse,(0<<16)+0000,xx,xx,xx \
 ,"222 %7u. %7u.",ERRORINRESPONSE retransmission count & interval) \
aCfgParm(rx_Hello,(0<<16)+0,xx,xx,xx \
 ,"222 %7u. %7u.",HELLO retransmission count & interval) \
aCfgParm(rx_HIDApprove,(0<<16)+0000,xx,xx,xx \
 ,"222 %7u. %7u.",HIDAPPROVE retransmission count & interval) \
aCfgParm(rx_HIDChange,(3<<16)+1000,xx,xx,xx \
 ,"222 %7u. %7u.",HIDCHANGE retransmission count & interval) \
aCfgParm(rx_HIDChangeRequest,(3<<16)+1000,xx,xx,xx \
 ,"222 %7u. %7u.",HIDCHANGEREQUEST retransmission count & interval) \
aCfgParm(rx_HIDReject,(0<<16)+0000,xx,xx,xx \
 ,"222 %7u. %7u.",HIDREJECT retransmission count & interval) \
aCfgParm(rx_Notify,(3<<16)+1000,xx,xx,xx \
 ,"222 %7u. %7u.",NOTIFY retransmission count & interval) \
aCfgParm(rx_Refuse,(3<<16)+1000,xx,xx,xx \
 ,"222 %7u. %7u.",REFUSE retransmission count & interval) \
aCfgParm(rx_Reroute,(5<<16)+1000,xx,xx,xx \
 ,"222 %7u. %7u.",Reroute retransmission count & interval) \
aCfgParm(rx_Status,(3<<16)+1000,xx,xx,xx \
 ,"222 %7u. %7u.",STATUS retransmission count & interval) \
aCfgParm(rx_StatusResponse,(0<<16)+0,xx,xx,xx \
 ,"222 %7u. %7u.",STATUSRESPONSE retransmission count & interval) \
/* END struct stRetryParms */ \
aCfgParm(sendspace,1024*4,1024,64*1024, \
 ,"14%17u.",default socket send space) \
aCfgParm(sweep_interval,20,1,86400, \
 ,"14%17u.",ticks between checking for hung streams) \
aCfgParm(targ_lists,TARG_LISTS,1,xx,TARG_LISTS \
 ,"14%17u.",# target lists to encode per SCMP) \
aCfgParm(targ_per_list,TARG_PER_LIST,1,xx,TARG_PER_LIST \
 ,"14%17u.",# targets to encode per target list) \
aCfgParm(targ_per_scmp_def,TARG_PER_SCMP_DEF,1,xx,TARG_PER_SCMP_DEF \
 ,"14%17u.",# targets parsed w/o ExpAry) \
aCfgParm(timeoutfactor,1,1,100, \
```

```
  ,"14%17u.",retransmit timeout factor - slowdown for debug) \
 aCfgParm(vlnk_bits,VLNK_BITS,1,VLID_BITS,VLNK_BITS \
  ,"14%17u.",number of VLID_BITS used for table index) \
 aCfgParm(vlnk_hash,VLNK_HASH,1,VLNK_HASH,VLNK_HASH \
  ,"14%17u.",number of entries in (remote) VLink hash table) \
 aCfgParm(vlnk_mask,((1<<VLNK_BITS)-1),1,NBitMask (VLNK_BITS),VLNK_MASK \
  ,"14%17x ",mask for index bits in a VLID) \


struct aConfigBlock {
 unsigned long
#define aCfgParm(name,cv,nv,xv,cs,fo,desc) name,
 ConfigParmList
#undef aCfgParm
    end;
};

extern struct aConfigBlock st2_config;

/* ------------------------------------------------------------------------ */

/* ST-II Implementation Control Flags.
 *
 */

#define ST2FlagList \
 aST2Flg(0x00000001,ST2FlgNoSrcChking \
  ,Do not record and check source of ST-II packets) \
 aST2Flg(0x00000080,ST2FlgNoHello \
  ,Do not send HELLOs) \
 aST2Flg(0x00000100,ST2FlgNoFail \
  ,Do not abort streams on neighbor failure) \
 /*following since do not have source access for recompile */ \
 aST2Flg(0x00100000,ST2FlgOwnLE \
  ,Use st2_leoutput instead of leoutput) \
       \
 aST2Flg(0x10000000,ST2FlgSTCksm \
   Accept ST packets with a zero ckecksum) \
 aST2Flg(0x20000000,ST2FlgSCMPCksm,Accept SCMP packets with a zero ckecksum) \
 aST2Flg(0x40000000,ST2FlgBugPrint,Print/Log BUGCHECK info) \


#define ConfigFlag(x) ((st2_config.ctlflg & (unsigned long) (x)) NE 0)


enum ST2Flag {
#define aST2Flg(bit,name,desc) name = bit,
 ST2FlagList
#undef aST2Flg
 ST2FlgDummy
};
```

```
/*
 * Service costs.
 */

#define CostList \
 aCostVar(cost_ctrl,"14%17u.",) \
 aCostVar(cost_ctrl_drop,"14%17u.",) \
 aCostVar(cost_data,"14%17u.",) \
 aCostVar(cost_data_drop,"14%17u.",) \
 aCostVar(cost_fixed,"14%17u.",) \
 aCostVar(cost_os,"14%17u.",) \


/* ---------------------------------------------------------------------- */

/*
 * Packet throughput counters.
 */

#define ThruList \
 aThruVar(ctrl_in_bytes,"14%17u.",) /* NB: must be first */ \
 aThruVar(ctrl_in_pkts,"14%17u.",) \
 aThruVar(ctrl_in_drop_bytes,"14%17u.",) \
 aThruVar(ctrl_in_drop_pkts,"14%17u.",) \
 aThruVar(ctrl_out_bytes,"14%17u.",) \
 aThruVar(ctrl_out_pkts,"14%17u.",) \
 aThruVar(ctrl_out_drop_bytes,"14%17u.",) \
 aThruVar(ctrl_out_drop_pkts,"14%17u.",) \
 aThruVar(data_in_bytes,"14%17u.",) \
 aThruVar(data_in_pkts,"14%17u.",) \
 aThruVar(data_in_drop_bytes,"14%17u.",) \
 aThruVar(data_in_drop_pkts,"14%17u.",) \
 aThruVar(data_out_bytes,"14%17u.",) \
 aThruVar(data_out_pkts,"14%17u.",) \
 aThruVar(data_out_drop_bytes,"14%17u.",) \
 aThruVar(data_out_drop_pkts,"14%17u.",) \



/*
A* ST-II Statistics Definitions
a*
a* The CounterList is instantiated in st2_proto so sites can make
a* local additions. Only add to the end of the list unless you
a* recompile all of ST-II source!
a*
a* Placed here so Applications can get at the information, too.
a*
 * name,format,description
```

```
 * aCounter(name,format,description)
 *
 * format specifies how to print the value as "i" quantities of "ni"
 * bytes each (i < 5) as <i><n1><n2>...<ni><printf format specification>
 * Currently, the <printf format specification> prints 18 characters.
 *
 */

/* ST2StatsVersion (1)  /* Binary Release 1 */
/* ST2StatsVersion (2)  /* Source Release 1.8 */
/* ST2StatsVersion (3)  /* Source 1.9 */
#define ST2StatsVersion (4)  /* Source Release 1.10 */


#define CounterList \
 /* WARNING, Start: Do NOT change this block */ \
 aCounter(length,"14%17u.",)  /* sizeof (st2_stats), not counter */ \
 aCounter(version,"14%17u.",)  /* version of statistics block */ \
 aCounter(agent,"41111 %u.%u.%u.%u",) /* IPAdr of the agent */ \
 aCounter(rptSeconds,"244%10u.%06u ",) /* Timestamp of data, seconds */ \
 aCounter(rptFract,"0",)  /* ... fractional seconds */ \
 aCounter(butSeconds,"244%10u.%06u ",) /* Timestamp when booted, seconds */ \
 aCounter(butFract,"0",)  /* ... fractional seconds */ \
 aCounter(unique_id,"14%17u.",)  /* Unique Id */ \
 aCounter(other,"14%17u.",)  /* Catch-all */ \
 /* WARNING, End: Do NOT change this block */ \
 aCounter(acksdelayed,"14%17u.",) \
 aCounter(bad_mt_type,"14%17u.",) \
 aCounter(bad_stcksum,"14%17u.",) \
 aCounter(bug_checks,"14%17u.",) \
 aCounter(bug_id,"14%17x ",) \
 aCounter(bug_info,"14%17x ",) \
 CostList \
 ThruList \
 aCounter(encodebadparm,"14%17u.",) \
 aCounter(encodewrongparms,"14%17u.",) \
 aCounter(errfreeingcon,"14%17u.",) \
 aCounter(errfreeinghid,"14%17u.",) \
 aCounter(errfreeingvlid,"14%17u.",) \
 aCounter(fasttics,"14%17u.",) \
 aCounter(frwd_nonxthops,"14%17u.",) \
 aCounter(frwd_notclmbuf,"14%17u.",) \
 aCounter(hellomsec,"14%17u.",our HelloTimer) \
 aCounter(hellorecv,"14%17u.",) \
 aCounter(hellosent,"14%17u.",) \
 /* Begin keep together - struct timeval */ \
 aCounter(hellosec,"14%17u.",local tv_sec of HelloTimer) \
 aCounter(hellousec,"14%17u.",local tv_usec of HelloTimer) \
 /* End keep together */ \
 aCounter(lennot4x_parm,"14%17u.",) \
 aCounter(lennot4x_target,"14%17u.",) \
 aCounter(misconfiguration,"14%17u.",) \
```

```
 aCounter(msg_inhidcoll,"14%17u.",) \
 aCounter(msg_innocon,"14%17u.",) \
 aCounter(msg_intrunc,"14%17u.",) \
 aCounter(nbrsactivated,"14%17u.",# neighbors started sending HELLOs to) \
 aCounter(nbrscreated,"14%17u.",# neighbors created) \
 aCounter(nbrsdeactivated,"14%17u.",# neighbors stopped sending HELLOs to) \
 aCounter(nbrsdeleted,"14%17u.",# neighbors deleted) \
 aCounter(nbrsfailed,"14%17u.",# neighbor instanced failed via HELLOs) \
 aCounter(nbrsfailedRemoteRestart,"14%17u.",# neighbor instanced failed:
RemoteRestart) \
 aCounter(nbrsfailedstream,"14%17u.",# neighbor instanced failed via HELLOs) \
 aCounter(nbrsfailedtarget,"14%17u.",# neighbor instanced failed via HELLOs) \
 aCounter(nbrsinserted,"14%17u.",# vlinks monitored to all neighbors) \
 aCounter(nbrsremoved,"14%17u.",# vlinks no longer monitored to all neighbors)
\
 aCounter(no_dspin,"14%17u.",) \
 aCounter(no_ifnet,"14%17u.",) \
 aCounter(no_mt_data,"14%17u.",) \
 aCounter(nofreecon,"14%17u.",) \
 aCounter(nofreehids,"14%17u.",) \
 aCounter(nofreevlids,"14%17u.",) \
 aCounter(nofwdifc,"14%17u.",) \
 aCounter(not_aligned,"14%17u.",) \
 aCounter(not_st2,"14%17u.",) \
 aCounter(parm_missing,"14%17u.",) \
 \
 aCounter(net_in_bytes,"14%17u.",) \
 aCounter(net_in_pkts,"14%17u.",) \
 aCounter(net_in_drop_bytes,"14%17u.",) \
 aCounter(net_in_drop_pkts,"14%17u.",) \
 aCounter(net_out_bytes,"14%17u.",) \
 aCounter(net_out_pkts,"14%17u.",) \
 aCounter(net_out_drop_bytes,"14%17u.",) \
 aCounter(net_out_drop_pkts,"14%17u.",) \
 \
 aCounter(pcb_fail,"14%17u.",pcbs not allocated -- aka nofreecon) \
 aCounter(pcb_get,"14%17u.",pcbs allocated) \
 aCounter(pcb_rel,"14%17u.",pcbs released) \
 aCounter(pkts_in,"14%17u.",) \
 aCounter(pkts_in_drop,"14%17u.",) \
 aCounter(pkts_in_ipv4p,"14%17u.",) \
 aCounter(pkts_in_ipv4p5,"14%17u.",) \
 aCounter(pkts_in_ipv4p5_rawip,"14%17u.",) \
 aCounter(pkts_in_ipv4px,"14%17u.",) \
 aCounter(pkts_in_ipv5,"14%17u.",) \
 aCounter(pkts_in_max,"14%17u.",input queue high water mark) \
 aCounter(pkts_in_qfull,"14%17u.",input dropped by net because queue full) \
 aCounter(sap_next,"14%17u.",) \
 aCounter(scmp_0rvlid,"14%17u.",) \
 aCounter(scmp_0svlid,"14%17u.",) \
 aCounter(scmp_badvlid,"14%17u.",) \
 aCounter(scmp_cksum,"14%17u.",) \
```

```
 aCounter(scmp_failed_ifc,"14%17u.",) \
 aCounter(scmp_failed_neighbor,"14%17u.",) \
 aCounter(scmp_failed_net,"14%17u.",) \
 aCounter(scmp_failed_resources,"14%17u.",) \
 aCounter(scmp_failed_route,"14%17u.",) \
 aCounter(scmp_failed_route_fixed,"14%17u.",) \
 aCounter(scmp_failed_srcrut,"14%17u.",) \
 aCounter(scmp_failed_to_route,"14%17u.",) \
 aCounter(scmp_fragmented,"14%17u.",) \
 aCounter(scmp_leninconsist,"14%17u.",) \
 aCounter(scmp_nobfxhd,"14%17u.",insufficient packet header space) \
 aCounter(scmp_nobfxo1,"14%17u.",insufficient pre-overhead space) \
 aCounter(scmp_nobfxo2,"14%17u.",insufficient post-overhead space) \
 aCounter(scmp_nobfxpk,"14%17u.",insufficient packet space) \
 aCounter(scmp_nobuf0,"14%17u.",no buf to use) \
 aCounter(scmp_nobuf1,"14%17u.",no small bufs) \
 aCounter(scmp_nobuf2,"14%17u.",no large bufs) \
 aCounter(scmp_nobuf3,"14%17u.",no buf big enough) \
 aCounter(scmp_nobufmod,"14%17u.",invalid GetSpace mode) \
 aCounter(scmp_refnum,"14%17u.",) \
 aCounter(scmp_rx,"14%17u.",number of retransmitted control messages) \
 aCounter(scmp_rx_timeout,"14%17u.",number of unacknowledged control messages)
\
 aCounter(scmp_tooshort,"14%17u.",) \
 aCounter(slowtics,"14%17u.",) \
 aCounter(trapcount,"14%17u.",number of traps) \
 aCounter(trapnext,"14%17x ",next free slot in traptable) \
 aCounter(trappage,"14%17u.",) \
 aCounter(trg_fail,"14%17u.",targets not allocated) \
 aCounter(trg_get,"14%17u.",targets allocated) \
 aCounter(trg_rel,"14%17u.",targets released) \
 aCounter(vlinkfail,"14%17u.",) \
 aCounter(vlinkget,"14%17u.",) \
 aCounter(vlinkrel,"14%17u.",) \

struct aStatsBlock {
#define aCostVar(name,format,desc) unsigned long Ident(cost_)name;
#define aCounter(name,format,desc) unsigned long name;
#define aThruVar(name,format,desc) unsigned long name;
 CounterList
#undef aCostVar
#undef aCounter
#undef aThruVar
};

extern struct aStatsBlock st2_stats;


/* Macro to increment a counter */
#define Count(n,v) st2_stats.n += (v)
```

```
/* ??? Define network management access functions, including accounting. */

/*
T* Testing and Debugging Print Routines
t*
t* The ST2Print Module provides routines to print ST-II Protocol
t* Data Units (PrintXxx) and implementation data structures
t* (print_xxx). The subroutine declarations are availble to
t* Applications when USING_ST2Print is defined before including
t* st2_api h.
t*
 */

#ifdef USING_ST2Print

extern void  flush_history (/* */);
extern FILE *g_fp;   /* initialized to stdout */
extern char *LookupSTError (/* enum ST2Errors error */);
extern char *LookupSTOpcode (/* enum anSTOpCode, char *default */);
extern void  print_aNeighborList (/* prefix */);
extern int  print_aNetIf (/* kagifca, agifcp, prefix */);
extern void  print_aNxtHop (/* long anha, prefix */);
extern void  print_aNxtHopList (/* long anhla, prefix */);
extern void  print_aPktDesc (/* long apda, prefix */);
extern void  print_aRouteChoiceList (/* arcla, arclp, prefix */);
extern int  print_arptable (/* prefix */);
extern void  print_aRsrcBuf (/* long arba, prefix */);
extern long  print_aRsrcClnt (/* long arscla, prefix */);
extern void  print_aRsrcCPU (/* long arscua, prefix */);
extern void  print_aRsrcNet (/* long arna, prefix */);
extern int  print_aRsrcNetList (/* prefix */);
extern int  print_aST2pcb (/* long kcp, prefix */);
extern void  print_aTarget (/* long ata, atl, prefix */);
extern void  print_aVLink (/* long vla, prefix */);
extern void  print_config (/* prefix */);
extern int  print_ConHshTable (/* prefix */);
extern int  print_ConTblList (/* prefix */);
extern void  print_driverqtable (/* prefix */);
extern void  print_flow_spec (/* long fsa, prefix */);
extern int  print_gifcs (/* prefix */);
extern void  print_next_hop (/* long nha, prefix */);
extern void  print_route (/* long rp, prefix */);
extern int  print_rxqueue (/* prefix */);
extern void  print_socket (/* long sop, prefix */);
extern int  print_stats (/* prefix */);
extern void  print_streams ();
extern void  print_traps (/* prefix */);
extern void  print_vlinks (/* prefix */);
extern int  PrintIP4Header (/* ipp, len, prefix */);
extern int  PrintMsg (/* struct msghdr *msgp, datalen, prefix */);
```

```
extern int  PrintParms (/* struct aParameter *parmp, len, prefix */);
extern int  PrintSockaddrST (/* struct sockaddr_st2 *sap, len, prefix */);
extern int  PrintSCMPHeader (/* struct aSCMPHeader *scmhp, len, prefix */);
extern int  PrintST2Header (/* struct aST2Header *st2hp, len, prefix */);
extern int  PrintTarget (/* struct aTarget *tp, len, prefix */);

#endif USING_ST2Print


#endif _ST2_API_H_
```

2.2 TG SOURCE

```
/*****************************************************************************
 *                                                                           *
 * File: buffer_generic.c                                                    *
 *                                                                           *
 * Routines to manage a generic buffer pool.                                 *
 *                                                                           *
 * Written 08-Aug-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/src/tg/RCS/
buffer_generic.c,v 1.8 90/11/26 12:28:44 dlee Exp $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <malloc.h>
#include "config.h"

/* Type definitions local to this file.     */

typedef struct generic_freelist_
 {
 struct generic_freelist_ *next;
 }  generic_freelist;

/* Functions exported from this file        */

/* Functions local to this file.          */

/* Variables exported from this file.     */

/* Variables local to this file.     */

static generic_freelist *generic_flist = NULL; /* generic bfr freelist. */


/* Get a buffer. This does nothing fancy, a more sophisticated version */
/* might be able to avoid some packet copies.    */

/* ARGSUSED */
char *
buffer_generic_get(maxlen)

 unsigned long maxlen;

 {
 char  *buf;
```

```
 if (generic_flist == NULL)
  buf = (char *)malloc(MAX_PKT_SIZE);
 else
  {
  buf = (char *)generic_flist;
  generic_flist = generic_flist->next;
  }
 return (buf);
 }


/* Free up a buffer.      */

void
buffer_generic_free(buf)

 char  *buf;

 {
 generic_freelist *fp = (generic_freelist *)buf;

 fp->next = generic_flist;
 generic_flist = fp;
 }
```

```
/*****************************************************************************
 *                                                                           *
 * File: decode.c                                                            *
 *                                                                           *
 * Encode and decode compressed integers.                                    *
 *                                                                           *
 * Written 11-Sep-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/src/tg/RCS/
decode.c,v 1.5 90/11/26 12:29:17 dlee Exp $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <math.h>
#include "config.h"
#include "distribution.h"
#include "protocol.h"
#include "decode.h"

/* Type definitions local to this file.     */

/* Functions exported from this file.     */

/* Functions local to this file.     */

/* Variables exported from this file.     */

/* Variables local to this file.     */


/* Decode an unsigned long from a buffer. The number is packed seven */
/* bits per byte in little-endian order, with the sign bit indicating */
/* that more is to come.      */

char *
decode_ulong(buf, n, len)

 char   *buf;
 int  *n;
 int  len;

 {
```

```
char  *bufend = &(buf[len]);
unsigned int curbyte;
int  shift = 0;
unsigned long tmp = 0;

/* Each pass though the following loop decodes one byte. */

for (;;)
 {

 /* Pick up the next byte.    */

 curbyte = *(buf++);

 /* Shift and mask it into the accumulated value. */

 tmp |= (curbyte & 0x7f) << shift;

 /* If the top bit is not set, this was the last byte. */

 if (curbyte <= 0x7f)
  break;

 /* Increment the shift count, scream if more than 32 */
 /* bits are to be read.     */

 shift += 7;
 if (shift > 32)
  {

  /*@@@@ Log bad response...   */

  (void)fprintf(stderr,
    "decode_ulong: bad format\n");
  abort();
  }

 /* If the encoded number runs off the end of the */
 /* buffer, return NULL so that the caller can try */
 /* again when he gets more input.   */

 if (buf >= bufend)
  return ((char *)NULL);
 }

/* Return the decoded number.     */

*n = tmp;
return (buf);
}
```

```
decode_ulong2(fp, result)

 FILE  *fp;
 int  *result; /* Integer is returned here */

 {
 unsigned int curbyte;
 int  shift = 0, rvalue;
 unsigned long tmp = 0;

 /* Each pass though the following loop decodes one byte. */

 for (; (rvalue = getc(fp)) != EOF;)
  {

  /* Pick up the next byte.    */

  curbyte = (char) rvalue;

  /* Shift and mask it into the accumulated value. */

  tmp |= (curbyte & 0x7f) << shift;

  /* If the top bit is not set, this was the last byte. */

  if (curbyte <= 0x7f)
   break;

  /* Increment the shift count, scream if more than 32 */
  /* bits are to be read.     */

  shift += 7;
  if (shift > 32)
   {

   /*@@@@ Log bad response...   */

   (void)fprintf(stderr,
     "decode_ulong: bad format\n");
   abort();
   }
 }
if (rvalue == EOF) {
 return (-1);
}
/* Return the decoded number.     */

*result = tmp;
return (0);
}
```

```
/* Encode a response request. This consists of the length of the */
/* desired response in bytes, followed by the number of bytes to skip */
/* in order to find the next response-length request. Lose synch, and */
/* you die! But does not require touching every byte of a long packet. */
/* Returns the length of the buffer consumed. If the buffer pointer */
/* is NULL, returns the maximum length of buffer that can be consumed. */

int
encode_response(buf, len, n)

 char  *buf;
 unsigned int len;
 unsigned int n;

 {
 char  *cp = buf;
 static int maxlen = -1;
 int  remainder;

 /* Set up maximum lengths if first time through.  */

 if (maxlen < 0)
  maxlen = 2 * (int)encode_ulong((char *)NULL, 0);

 /* Just return maximum length if NULL buffer.   */

 if (buf == NULL)
  return (maxlen);

 /* Encode the desired value and the length to skip.  */

 cp = encode_ulong(buf, n);
 remainder = len - (cp - buf);
 if (remainder <= 0)
  cp = encode_ulong(cp, 1);
 else
  cp = encode_ulong(cp, remainder);
 return (cp - buf);
 }


/* Encode a special response request. This consists of the length of */
/* the desired response in bytes, followed by a zero byte, followed by */
/* the number of bytes to skip in order to find the next  */
/* response-length request.     */

int
encode_special_response(buf, len, n)

 char  *buf;
 unsigned int len;
```

```
unsigned int n;

{
char  *cp = buf;
static int maxlen = -1;
int  remainder;

/* Set up maximum lengths if first time through.  */

if (maxlen < 0)
 maxlen = 2 * (int)encode_ulong((char *)NULL, 0) + 1;

/* Just return maximum length if NULL buffer.   */

if (buf == NULL)
 return (maxlen);

/* Encode the desired value and the length to skip.  */

cp = encode_ulong(buf, n);
*cp++ = 0x00;
remainder = len - (cp - buf);
if (remainder <= 0)
 cp = encode_ulong(cp, 1);
else
 cp = encode_ulong(cp, remainder);
return (cp - buf);
}


/* Encode an unsigned long. This consists of seven bits of number per */
/* byte of buffer, in little-endian order, with the sign bit indicating */
/* that more is to come.     */

char *
encode_ulong(buf, n)

 char  *buf;
 unsigned long n;

 {

 /* If no buffer, return maximum length.    */

 if (buf == NULL)
  return ((char *)5);

 /* Each pass through the following loop encodes seven bits, */
 /* low-order bits first.     */

 while (n > 127)
  {
```

```
 *(buf++) = (n & 0x7f) | 0x80;
 n >>= 7;
 }

/* Encode the last bits -- leave sign bit clear.  */

*(buf++) = n;
return (buf);
}
```

```
/*****************************************************************************
 *                                                                           *
 * File: distribution.c                                                      *
 *                                                                           *
 * Contains distribution generation functions.                               *
 *                                                                           *
 * Written 17-Aug-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/src/tg/RCS/
distribution.c,v 1.6 90/11/26 12:29:27 dlee Exp $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <math.h>
#include "config.h"
#include "distribution.h"

/* Type definitions local to this file.     */

/* Functions exported from this file.     */

/* Functions local to this file.     */

/* Variables exported from this file.     */

/* Variables local to this file.     */



/* Return constant ''random'' variate.     */

double
dist_const_gen(dist)

 distribution *dist;

 {

 return (dist->par1);
 }


/* Initialize constant ''random'' variate.    */

char *
dist_const_init(dist, par)
```

```
 distribution *dist;
 double  par;

 {

 dist->generate = dist_const_gen;
 dist->par1 = par;
 return (NULL);
 }


/* Return exponentially-distributed random variate.   */

double
dist_exp_gen(dist)

 distribution *dist;
 {

 double  value;

 do
  {
   value = -dist->par1 * log(((double)(random() + 1))/
    (double)(unsigned)(MAX_RANDOM + 1));
  } while ((value < dist->par2) || (value > dist->par3));

 return(value);
 }


/* Initialize exponentially-distributed random variate.   */

char *
dist_exp_init(dist, mean, min, max)

 distribution *dist;
 double  mean;
 double  min;
 double  max;
 {

 dist->generate = dist_exp_gen;
 dist->par1 = mean;
 dist->par2 = min;
 dist->par3 = max;
 return (NULL);
 }


/* Return uniformly-distributed random variate in interval [0, par1). */
```

```
double
dist_markov2_gen(dist)

 distribution *dist;

 {
 dist_markov2 *dm2;
 int  state;
 distribution *subdist;
 double  x;

 /* Get variate from current distribution.   */

 dm2 = (dist_markov2 *)(dist->pars);
 state = dm2->state;
 subdist = dm2->dist[state];
 x = (*(subdist->generate))(subdist);

 /* Switch state, if necessary.     */

 if (random() > dm2->p[state])
  dm2->state = !state;

 /* Pass back the variate.     */

 return (x);
 }


/* Initialize two-state markov random variate.    */

char *
dist_markov2_init(dist, t1, d1, t2, d2)

 distribution *dist;
 double  t1;
 distribution *d1;
 double  t2;
 distribution *d2;

 {
 dist_markov2 *dm2;

 /* Set up generation function.     */

 dist->generate = dist_markov2_gen;

 /* Get memory for markov2 struct.    */

 dm2 = (dist_markov2 *)malloc(sizeof(dist_markov2));
 if (dm2 == NULL)
```

```
 {
 return ("distribution: Out of memory");
 }
dist->pars = (char *)dm2;

/* Get mean and distribution for state 1.   */

dm2->mean[0] = t1;
if (dm2->mean[0] < 1.)
 return ("Mean state-residence time must be at least 1");
else
 dm2->p[0] = (1. - 1. / dm2->mean[0]) *
   (unsigned)0x80000000;
dm2->dist[0] = (distribution *)malloc(sizeof(distribution));
if (dm2->dist[0] == NULL)
 {
 return ("distribution: Out of memory");
 }
*(dm2->dist[0]) = *d1;

/* Get mean and distribution for state 2.   */

dm2->mean[1] = t2;
if (dm2->mean[1] < 1.)
 return ("Mean state-residence time must be at least 1");
else
 dm2->p[1] = (1. - 1. / dm2->mean[1]) *
   (unsigned)0x80000000;
dm2->dist[1] = (distribution *)malloc(sizeof(distribution));
if (dm2->dist[1] == NULL)
 {
 return ("distribution: Out of memory");
 }
*(dm2->dist[1]) = *d2;

/* Initialize state.     */

dm2->state = 0;

return (NULL);
}


/* Return uniformly-distributed random variate in interval [0, par1). */

double
dist_uniform_gen(dist)

 distribution *dist;
 {

 /* dist->par1 is upper limit, dist->par2 is lower limit */
```

```
 return ((dist->par2 - dist->par1) * (((double)random()) /
  (double)(unsigned)(MAX_RANDOM + 1)) + dist->par1);
 }


/* Initialize uniformly-distributed random variate in interval  */
/* [0, par1).       */

char *
dist_uniform_init(dist, min, max)

 distribution *dist;
 double  min;
 double  max;

 {

 dist->generate = dist_uniform_gen;
 dist->par1 = min;
 dist->par2 = max;
 return (NULL);
 }
```

```
/*****************************************************************************
 *                                                                           *
 * File: log.c                                                               *
 *                                                                           *
 * Routines to write out log entries.                                        *
 *                                                                           *
 * Written 20-Sep-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/src/tg/RCS/
log.c,v 1.8 91/01/15 18:48:36 dlee Exp Locker: dlee $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include "config.h"
#include "distribution.h"
#include "protocol.h"
#include "decode.h"
#include "log.h"

/* Type definitions local to this file.    */

#define HOSTNAMELEN  64
#define FPRINTF  (void) fprintf

/* Functions exported from this file.    */

FILE *log_open();
int log_init();

/* Functions local to this file.    */

static char  *log_current_time();
static char  *log_sched_time();
static char  *log_time();
static void  log_write_error();

/* Variables exported from this file.    */

/* Variables impored to this file    */
extern int FlushOutput;

/* Variables local to this file.    */
```

```
static long  on_time_sec;
static protocol *prot;
static FILE  *log_fp;



FILE *
log_open(filename)
 char  *filename;
{
 FILE  *fp;

 if(filename) {
  if ((fp = fopen(filename, "w")) == NULL) {
   perror("fopen: read");
   return (NULL);
  }
 } else {
  fp = stdout;
 }
 return(fp);
}

/*
 * TG log file header is stored in ASCII form and contains the
 * following information:
 *
 * o Data file version identifier (e.g., 1.1)
 * o Script file name
 * o Start time of TG program
 * o Address format identifier
 * o Name of protocol under test
 * o Header termination line
 */

log_init(fp, on_timeval, prot_name, af, script, prot2)

 FILE *fp;   /* Handle to log file */
 struct timeval on_timeval; /* Starting timeval structure */
 char *prot_name;  /* Textual name of protocol under test*/
 short af;   /* Address format specifier */
 char *script;  /* name of script file */
 protocol *prot2;  /* Common protocol structure */
{
 char hostname[HOSTNAMELEN + 1];
 extern char *version;

 /* Store time (in seconds) in global variable */
 on_time_sec = on_timeval.tv_sec;

 prot = prot2;
```

```
log_fp = fp;

if (gethostname (hostname, HOSTNAMELEN) != 0) {
 FPRINTF(stderr,
  "log_init: fatal error: unable to fetch hostname\n");
 abort();
}

/* Build data file information header */
FPRINTF(fp, BEGIN_HDR_STRING);
FPRINTF(fp, "TG program version %s\n", version);
FPRINTF(fp, "TG log file version %d.%d\n", LOG_VERSION, LOG_SUBVERSION);
FPRINTF(fp, "Log file created on %s\n", hostname);

if (script) {
 FPRINTF(fp, "Script filename is %s\n", script);
} else {
 FPRINTF(fp, "Script read from stdin\n");
}

on_timeval.tv_usec = 0L;  /* Zero usec field */
FPRINTF(fp, "Program start time from UNIX epoch %d\n",
 on_timeval.tv_sec);
FPRINTF(fp, "Program start time: %s",
 ctime((time_t *) &on_timeval.tv_sec));

FPRINTF(fp, "Address family identifier is %d\n", af);
FPRINTF(fp, "Protocol under test is %s\n", prot_name);

/* The following entry must immediately precede data records */
FPRINTF(fp, END_HDR_STRING);

return(0);    /* Normal return */
}


static void
log_write_error( /* type */ )
 /* int type;  /* Not currently used */
{

 FPRINTF(stderr, "log_write_error: fatal error when writing log file\n");
 abort();
}


/* Log connection acceptance. ''address2'' may be NULL to indicate */
/* that it is not present. ''errno'' may be -1 to indicate  */
/* exception-free transmission.     */

void
```

```
log_accept(address1, address2, asn, errno)

 struct sockaddr *address1;
 struct sockaddr *address2;
 int  asn;
 int  errno;

 {
 char  buf[100];
 char  *cp = buf;
 char  ctl;

 /* Encode record type.      */

 *cp++ = LOGTYPE_ACCEPT;

 /* Encode control byte.      */

 ctl = 0;
 ctl |= LOGCTL_ADDR;
 if (address2 != NULL)
  ctl |= LOGCTL_2ADDR;
 if (errno >= 0)
  ctl |= LOGCTL_EXCEPT;
 *cp++ = ctl;

 /* Encode current time (accepts are always asynchronous). */

 cp = log_current_time(cp);

 /* Encode address(es) or asn.     */

 cp = (*(prot->prot->addrtob))(address1, cp);
 if (address2 != NULL)
  cp = (*(prot->prot->addrtob))(address2, cp);
 cp = encode_ulong(cp, (unsigned long) asn);

 /* Encode error number, if one is present.   */

 if (errno >= 0)
  cp = encode_ulong(cp, (unsigned long) errno);

 /* Write out the buffer.     */

 if (fwrite(buf, cp - buf, 1, log_fp) != 1)
  log_write_error();
 else
 {
  if (FlushOutput)
  fflush(log_fp);
 }
}
```

```
/* Log program error. ''address1'' and ''address2'' may be NULL */
/* to indicate that they are not present, ''asn'' may be -1 to indicate */
/* that it is not present. ''errcode'' indicates the type of error. */

void
log_error(address1, address2, asn, errcode)

 struct sockaddr *address1;
 struct sockaddr *address2;
 int  asn;
 int  errcode;

 {
 char  buf[100];
 char  *cp = buf;
 char  ctl;

 /* Encode record type.      */

 *cp++ = LOGTYPE_ERROR;

 /* Encode control byte.      */

 ctl = 0;
 if (address1 != NULL)
  {
  ctl |= LOGCTL_ADDR;
  if (address2 != NULL)
   ctl |= LOGCTL_2ADDR;
  }
 *cp++ = ctl;

 /* Encode current time (errors are always asynchronous). */

 cp = log_current_time(cp);

 /* Encode address(es) or asn.     */

 if (address1 == NULL)
  cp = encode_ulong(cp, (unsigned long) asn);
 else
  {
  cp = (*(prot->prot->addrtob))(address1, cp);
  if (address2 != NULL)
   cp = (*(prot->prot->addrtob))(address2, cp);
  }

 /* Encode error code.      */

 cp = encode_ulong(cp, (unsigned long) errcode);
```

```
 /* Write out the buffer.     */

 if (fwrite(buf, cp - buf, 1, log_fp) != 1)
  log_write_error();
 else
 {
  if (FlushOutput)
  fflush(log_fp);
 }
}


/* Log packet reception. ''address1'' and ''address2'' may be NULL */
/* to indicate that they are not present, ''asn'' may be -1 to indicate */
/* that it is not present. ''errno'' may be -1 to indicate  */
/* exception-free transmission.      */

void
log_rx(address1, address2, asn, pktid, len, errno)

 struct sockaddr *address1;
 struct sockaddr *address2;
 int  asn;
 unsigned long pktid;
 unsigned long len;
 int  errno;

 {
 char  buf[100];
 char  *cp = buf;
 char  ctl;

 /* Encode record type.      */

 *cp++ = LOGTYPE_RX;

 /* Encode control byte.      */

 ctl = 0;
 if (address1 != NULL)
  {
  ctl |= LOGCTL_ADDR;
  if (address2 != NULL)
   ctl |= LOGCTL_2ADDR;
  }
 if (errno >= 0)
  ctl |= LOGCTL_EXCEPT;
 *cp++ = ctl;

 /* Encode current time (receives are always asynchronous). */
```

```
cp = log_current_time(cp);

/* Encode address(es) or asn.     */

if (address1 == NULL)
 cp = encode_ulong(cp, (unsigned long) asn);
else
 {
 cp = (*(prot->prot->addrtob))(address1, cp);
 if (address2 != NULL)
  cp = (*(prot->prot->addrtob))(address2, cp);
 }

/* Encode packet ID and length.     */

cp = encode_ulong(cp, pktid);
cp = encode_ulong(cp, len);

/* Encode error number, if one is present.   */

if (errno >= 0)
 cp = encode_ulong(cp, (unsigned long) errno);

/* Write out the buffer.     */

if (fwrite(buf, cp - buf, 1, log_fp) != 1)
 log_write_error();
else
{
 if (FlushOutput)
 fflush(log_fp);
 }
}


/* Log setup. ''tvp'' may be NULL to indicate that this transmission */
/* was not explictly scheduled. ''errno'' may be -1 to indicate  */
/* exception-free transmission.     */

void
log_setup(tvp, errno)

 struct timeval *tvp;
 int  errno;

 {
 char  buf[100];
 char  *cp = buf;
 char  ctl;

 /* Encode record type.     */
```

```
 *cp++ = LOGTYPE_SETUP;

 /* Encode control byte.      */

 ctl = 0;
 if (tvp != NULL)
  ctl |= LOGCTL_SCHED;
 if (errno >= 0)
  ctl |= LOGCTL_EXCEPT;
 *cp++ = ctl;

 /* Encode time, if present, otherwise encode current time. */

 if (tvp == NULL)
  cp = log_current_time(cp);
 else
  cp = log_sched_time(cp, tvp);

 /* Encode error number, if one is present.   */

 if (errno >= 0)
  cp = encode_ulong(cp, (unsigned long) errno);

 /* Write out the buffer.     */

 if (fwrite(buf, cp - buf, 1, log_fp) != 1)
  log_write_error();
 else
 {
  if (FlushOutput)
  fflush(log_fp);
 }
}


/* Log teardown. ''tvp'' may be NULL to indicate that this  */
/* transmission was not explictly scheduled. ''errno'' may be -1 to */
/* indicate exception-free transmission.    */

void
log_teardown(tvp, errno)

 struct timeval *tvp;
 int  errno;

 {
 char  buf[100];
 char  *cp = buf;
 char  ctl;

 /* Encode record type.      */
```

```
*cp++ = LOGTYPE_TEARDOWN;

/* Encode control byte.     */

ctl = 0;
if (tvp != NULL)
 ctl |= LOGCTL_SCHED;
if (errno >= 0)
 ctl |= LOGCTL_EXCEPT;
*cp++ = ctl;

/* Encode time, if present, otherwise encode current time. */

if (tvp == NULL)
 cp = log_current_time(cp);
else
 cp = log_sched_time(cp, tvp);

/* Encode error number, if one is present.   */

if (errno >= 0)
 cp = encode_ulong(cp, (unsigned long) errno);

/* Write out the buffer.     */

if (fwrite(buf, cp - buf, 1, log_fp) != 1)
 log_write_error();
else
{
 if (FlushOutput)
 fflush(log_fp);
}
}


/* Log packet transmission. ''tvp'' may be NULL to indicate that this */
/* transmission was not explictly scheduled, ''address1'' and  */
/* ''address2'' may be NULL to indicate that they are not present, */
/* ''asn'' may be -1 to indicate that it is not present. ''errno'' may */
/* be -1 to indicate exception-free transmission.   */

void
log_tx(tvp, address1, address2, asn, pktid, len, errno)

 struct timeval *tvp;
 struct sockaddr *address1;
 struct sockaddr *address2;
 int  asn;
 unsigned long pktid;
 unsigned long len;
 int  errno;
```

```
{
char  buf[100];
char  *cp = buf;
char  ctl;

/* Encode record type.      */

*cp++ = LOGTYPE_TX;

/* Encode control byte.      */

ctl = 0;
if (tvp != NULL)
 ctl |= LOGCTL_SCHED;
if (address1 != NULL)
 {
 ctl |= LOGCTL_ADDR;
 if (address2 != NULL)
  ctl |= LOGCTL_2ADDR;
 }
if (errno >= 0)
 ctl |= LOGCTL_EXCEPT;
*cp++ = ctl;

/* Encode time, if present, otherwise encode current time. */

if (tvp == NULL)
 cp = log_current_time(cp);
else
 cp = log_sched_time(cp, tvp);

/* Encode address(es) or asn.     */

if (address1 == NULL)
 cp = encode_ulong(cp, (unsigned long) asn);
else
 {
 cp = (*(prot->prot->addrtob))(address1, cp);
 if (address2 != NULL)
  cp = (*(prot->prot->addrtob))(address2, cp);
 }

/* Encode packet ID and length.     */

cp = encode_ulong(cp, pktid);
cp = encode_ulong(cp, len);

/* Encode error number, if one is present.   */

if (errno >= 0)
 cp = encode_ulong(cp, (unsigned long) errno);
```

```
 /* Write out the buffer.     */

 if (fwrite(buf, cp - buf, 1, log_fp) != 1)
  log_write_error();
 else
 {
  if (FlushOutput)
  fflush(log_fp);
 }
}


/* Log current time, as offset from on-time.    */

static char *
log_current_time(buf)

 char  *buf;

 {
 struct timeval t;

 /* Get the current time, adjust for on-time.   */

 if (gettimeofday(&t, (struct timezone *)NULL) == -1)
  {
  perror("log_current_time: gettimeofday");
  abort();
  }
 t.tv_sec -= on_time_sec;

 return (log_time(buf, &t));
 }


/* Log scheduled time (as offset from on-time) and offset from current */
/* time.       */

static char *
log_sched_time(buf, st)

 char  *buf;
 struct timeval *st;

 {
 char  *cp;
 struct timeval stc;
 struct timeval t;

 /* Get current time.      */
 stc = *st;
```

```
if (gettimeofday(&t, (struct timezone *)NULL) == -1)
 {
 perror("log_sched_time: gettimeofday");
 abort();
 }

/* Get delay between scheduled and actual time, in  */
/* microseconds.     */

timersub(&t, &stc, &t);
t.tv_usec += t.tv_sec * 1000000L;

/* Convert scheduled time to offset from on-time.  */

stc.tv_sec -= on_time_sec;

/* Log the scheduled time.     */

cp = log_time(buf, &stc);

/* Log the delay.      */

cp = encode_ulong(cp, (unsigned long)t.tv_usec);
return (cp);
}


/* Log specified time. The time is assumed to be already converted to */
/* an offset from on-time.      */

static char *
log_time(buf, t)

 char  *buf;
 struct timeval *t;

 {
 char  *cp;

 /* Encode the seconds and microseconds.    */

 cp = encode_ulong(buf, (unsigned long)t->tv_sec);
 cp = encode_ulong(cp, (unsigned long)t->tv_usec);
 return (cp);
 }


void
log_close() {

 extern FILE *log_fp;
```

```
 (void) fclose (log_fp);
 exit(-1);
}
```

```
/*************************************************************************
 *                                                                       *
 * File: prot_dgram.c                                                    *
 *                                                                       *
 * Common routines for interfacing to datagram socket protocols          *
 * suite via the normal user-level interface.                            *
 *                                                                       *
 * Written 17-Aug-90 by Paul E. McKenney, SRI International.             *
 * Copyright (c) 1990 by SRI International.                              *
 *                                                                       *
 *************************************************************************/

/*
 * $Log: prot_dgram.c,v $
 * Revision 1.4 90/10/04 19:42:11 mckenney
 * Paul's final revision, prior to his departure.
 *
 * Revision 1.3 90/09/18 20:23:14 mckenney
 * ckpt
 *
 * Revision 1.2 90/09/04 11:27:46 mckenney
 * Checkpoint
 *
 * Revision 1.1 90/08/26 23:41:06 mckenney
 * .
 *
 */

/* Include files.      */

#include <stdio.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <sys/socket.h>
#include "config.h"
#include "log.h"
#include "distribution.h"
#include "protocol.h"

/* Type definitions local to this file.     */

typedef struct dgram_freelist_
 {
 struct dgram_freelist_ *next;
 }  dgram_freelist;
```

```
/* Functions exported from this file.     */

/* Functions local to this file.     */

/* Variables exported from this file.     */

/* Variables local to this file.     */

static dgram_freelist *dgram_flist = NULL; /* dgram bfr freelist. */
static fd_set  fds;  /* FDS to clients.  */
static int  firsttime = 1; /* flag for get_packets. */
static struct sockaddr_in
   from;  /* source of last datagram. */
static int  fromlen = 0; /* length of last dg's address. */
static int  justrcvd = 0; /* rcv since last send?  */
static protocol  *prot = NULL; /* protocol attr. pointer. */
static protocol_table *prtab = NULL; /* PRotocol Table pointer. */
static int  rcving = 0; /* currently receiving?  */
static int  sfd = -1; /* Socket file descriptor. */


/* Get a buffer. This does nothing fancy, a more sophisticated version */
/* might be able to avoid some packet copies. Leaves space for the */
/* packet ID at the front of the buffer.    */

/* ARGSUSED */
char *
buffer_dgram_get(maxlen)

 unsigned long maxlen;

 {
 char  *buf;

 if (dgram_flist == NULL)
  buf = (char *)malloc(MAX_PKT_SIZE);
 else
  {
  buf = (char *)dgram_flist;
  dgram_flist = dgram_flist->next;
  }
 return (buf + sizeof(unsigned long));
 }


/* Free up a buffer.      */

void
buffer_dgram_free(buf)

 char  *buf;
```

```
{
dgram_freelist *fp;

fp = (dgram_freelist *)(buf - sizeof(unsigned long));
fp->next = dgram_flist;
dgram_flist = fp;
}


/* Accept connections and packets while waiting for the opportunity to */
/* write (if wfd specified) or for specified timeout, whichever comes */
/* first. A wfd of -2 means to send to whereever the previous datagram */
/* came from.       */

int
dgram_get_packets(wfd, endtout)

int  wfd;  /* FD for write, -1 if none. */
struct timeval *endtout; /* end of timeout period. */

{
static char *buf = NULL; /* pointer to receive buffer. */
int  fd;
int  fdmax;
struct sockaddr_in from; /* Socket structure client */
int  nitems;  /* Number of items selected. */
int  pklen;  /* Input packet length.  */
fd_set  rfds;  /* Read FDs for select.  */
struct timeval tv;
struct timeval *tvp;
fd_set  wfds;  /* Write FDs for select. */

/* Initialization on first pass, if necessary.   */

if (firsttime)
 {
 FD_ZERO(&fds);
 if (sfd >= 0)
  {
  FD_SET(sfd, &fds);
  }
 firsttime = 0;
 }

/* Calculate initial timeout. We force a select even if the */
/* timeout has already expired in order to prevent the CPU from */
/* starving the I/O.     */

if (endtout == NULL)
 {

 /* No timeout, so just use NULL pointer.  */
```

```
 tvp = NULL;
 }
else
 {

 /* Calculate timeout.     */

 if (gettimeofday(&tv, (struct timezone *)NULL) == -1)
  {
  log_error(NULL, NULL, -1, LOGERR_GETTIME);
  perror("dgram_get_packets: getttimeofday");
  abort();
  }
 if (timercmp(endtout, &tv, >))
  {
  timersub(endtout, &tv, &tv);

  /* Wait at most one hour at a time. SunOS has */
  /* an annoying limitation on the select timeout,*/
  /* one hour is much less than this limitation. */

  if (tv.tv_sec > 3600)
   tv.tv_sec = 3600;
  }
 else
  {
  tv.tv_sec = 0;
  tv.tv_usec = 0;
  }
 tvp = &tv;
 }

/* Each pass through the following loop does one select call */
/* to check the state of the fds.    */

rcving = 1;
justrcvd = 1;
for (;;)
 {

 /* Set up for select: get fd bitmaps.   */

 rfds = fds;
 FD_ZERO(&wfds);
 if (wfd >= 0)
  {
  FD_SET(wfd, &rfds);
  FD_SET(wfd, &wfds);
  }
 else if (wfd == -2)
  {
```

```
        FD_SET(sfd, &rfds);
        FD_SET(sfd, &wfds);
        }
      if ((nitems = select(fdmax = (sfd > wfd ? sfd + 1 : wfd + 1),
         &rfds,
         &wfds,
         (fd_set *)NULL,
         tvp)) < 0)
       {
       if (errno != EINTR)
        {
        log_error(NULL, NULL, -1, LOGERR_SELECT);
        perror("dgram_get_packets: select");
        abort();
        }
       }

      /* Each pass through the following loop checks for */
      /* messages on one file descriptor.   */

      for (fd = 0; fd <= fdmax; fd++)
       {

       /* If current fd was not selected, ignore it. */

       if (!FD_ISSET(fd, &rfds))
        continue;

       /* Each pass through the following loop  */
       /* attempts to receive one segment.  */

       for (;;)
        {

        /* Get a buffer if we do not already */
        /* have one on hand.   */

        if ( (buf == NULL) &&
         ((buf =
         (*(prtab->buffer_get))(MAX_PKT_SIZE) -
         sizeof(unsigned long)) ==
         NULL))
         {
         log_error(NULL, NULL, -1, LOGERR_MEM);
         (void)fprintf(stderr,
           "%s %s\n",
           "dgram_get_packets: ",
           "out of memory!");
         abort();
         }

        /* Receive the segment, scream and die */
```

```
/* if error.    */

fromlen = sizeof(from);
pklen = recvfrom(fd,
   buf,
   MAX_PKT_SIZE,
   0,
   (struct sockaddr *)&from,
   &fromlen);
if ((pklen == 0) ||
 ((pklen < 0) &&
 (errno != EWOULDBLOCK)))
 {

 /* Shut down and tell rcv if */
 /* EOF or error.  */
 if (close(fd) == -1)
  {
  log_rx(NULL, NULL, fd, 0, 0, errno);
  perror("dgram_get_packets: close");

  log_teardown(NULL, errno);
  abort();
  }
 FD_CLR(fd, &fds);
 log_rx(NULL, NULL, fd, 0, 0, errno);
 log_teardown(NULL, -1);
 /* pklen includes beginning header */
 if (prtab->rcv != NULL)
  (void)(*(prtab->rcv))(fd,
     -1,
     NULL,
     pklen,
     0);
 }
else if (pklen > 0)
 {

 /* Pass packet to receiver. */

 log_rx(&from,
  NULL,
  -1,
  *(unsigned long *)buf,
  pklen - sizeof(unsigned long),
  -1);
 if ((prtab->rcv != NULL) &&
  ((*(prtab->rcv))
   (fd,
   -2,
   buf + sizeof(long),
   pklen - sizeof(long),
```

```
    *(unsigned long *)buf)))
    buf = NULL;
   }
  else
   {

   /* Nothing more to read right */
   /* now.    */

   break;
   }
  }

 if (--nitems <= 0)
  break;
 }

/* Check for ability to write...   */

if ((wfd > 0) &&
 (FD_ISSET(wfd, &wfds)))
 {
 rcving = 0;
 return (1);
 }

/* Calculate next timeout. If the timeout has expired, */
/* tell the caller the sad story.   */

if (endtout != NULL)
 {

 if (gettimeofday(&tv, (struct timezone *)NULL) == -1)
  {
  log_error(NULL, NULL, -1, LOGERR_SELECT);
  perror("dgram_get_packets: getttimeofday");
  abort();
  }
 if (timercmp(endtout, &tv, >))
  {
  timersub(endtout, &tv, &tv);

  /* Wait at most one hour at a time. */

  if (tv.tv_sec > 3600)
   tv.tv_sec = 3600;
  }
 else
  {
  rcving = 0;
  errno = ETIME;
  return (0);
```

```
    }
   }
  }

 /* NOTREACHED */
 }


/* Send a packet, subject to the timeout. The special association */
/* numbered -2 means to return to the sender of the most-recently */
/* received datagram. Note that the datagram ID is just a per-packet */
/* sequence number; no attempt is made to maintain separate consecutive */
/* number sequences for each destination.    */

int
dgram_send(asn, buf, len, endtout, pktid)

 long  asn;
 char  *buf;
 int  len;
 struct timeval *endtout;
 unsigned long *pktid;

 {
 int  cc;
 int  fd = asn;


 /* Make sure that fd wasn't closed out from under the sender. */

 if (fd == -1)
  {
  errno = EINVAL;
  return (-1);
  }

 /* Put the packet ID into the packet.    */

 ((unsigned long *)buf)[-1] = *pktid;

 /* If we just got done allowing receives, try the write without */
 /* bothering to do another receive.    */

 if (justrcvd ||
  rcving)
  {

  /* Try to write out the packet.    */

  if (fd != -2)
   {
   cc = write(fd,
```

```
    buf - sizeof(unsigned long),
    len + sizeof(unsigned long));
  }
 else
  {

  /* fd of -2 says to return a datagram to the */
  /* sender of the previously received datagram. */

  if (fromlen == 0)
   {
   errno = EBADF;
   return (-1);
   }
  cc = sendto(sfd,
    buf - sizeof(unsigned long),
    len + sizeof(unsigned long),
    0,
    (struct sockaddr *)&from,
    fromlen);
  }

/* If we succeeded, or if we failed for some reason */
/* other than blocking, or if we are already receiving */
/* packets (and thus do not want to recurse), clean up */
/* and exit.     */

if ((cc >= 0) ||
 (errno != EWOULDBLOCK) ||
 rcving)
 {
 if (fd != -2)
  log_tx(NULL, /* @@@@ fix definition of
     send to include a pointer
     to the time at which this
     send was scheduled, or
     NULL if it is asynchronous.
     For now, treat all sends
     as if they were async. */
   &(prot->dst),
   NULL,
   -1,
   ((unsigned long *)buf)[-1],
   cc - sizeof(unsigned long),
   cc == -1 ? errno : -1);
 else
  log_tx(NULL,
   &(from),
   NULL,
   -1,
   ((unsigned long *)buf)[-1],
   cc - sizeof(unsigned long),
```

```
    cc == -1 ? errno : -1);
  if (cc >= 0)
   (*pktid)++;
  (*(prtab->buffer_free))(buf);
  justrcvd = 0;
  return (cc);
  }
 }

/* The write failed with an EWOULDBLOCK or we need to allow */
/* some receives to happen. Thus, we must honor the timeout */
/* period, unless this would recursively invoke   */
/* dgram_get_packets.      */

/* Invoke the dgram_get_packets routine to receive packets and */
/* accept new connections while we are waiting to write. */

if (!dgram_get_packets(fd, endtout))
 {
 return (-1);
 }
else
 {

 /* Write out the packet!    */

 if (fd != -2)
  cc = write(fd,
    buf - sizeof(unsigned long),
    len + sizeof(unsigned long));
 else
  {

  /* fd of -2 says to return a datagram to the */
  /* sender of the previously received datagram. */

  if (fromlen == 0)
   {
   errno = EBADF;
   return (-1);
   }
  cc = sendto(sfd,
    buf - sizeof(unsigned long),
    len + sizeof(unsigned long),
    0,
    (struct sockaddr *)&from,
    fromlen);
  }
 if (fd != -2)
  log_tx(NULL,
   &(prot->dst),
   NULL,
```

```
        -1,
        ((unsigned long *)buf)[-1],
        cc - sizeof(unsigned long),
        cc == -1 ? errno : -1);
      else
       log_tx(NULL,
        &(from),
        NULL,
        -1,
        ((unsigned long *)buf)[-1],
        cc - sizeof(unsigned long),
        cc == -1 ? errno : -1);
      if (cc >= 0)
       (*pktid)++;
      (*(prtab->buffer_free))(buf);
      justrcvd = 0;
      return (cc);
      }
     }


    /* Record the fact that a setup has occurred.    */

    void
    dgram_setup(pp, fd)

     protocol *pp;
     int  fd;

     {

     if (sfd >= 0)
      {
      log_error(NULL, NULL, sfd, LOGERR_SELECT);
      (void)fprintf(stderr,
        "dgram_setup: %s %s\n",
        "cannot set up new server without",
        "tearing down old one first");
      abort();
      }

    /* Let dgram_get_packets know of the change.   */

    prot = pp;
    prtab = pp->prot;
    if (fd != -1)
     sfd = fd;
    firsttime = 1;
    log_setup(NULL, -1);
    }
```

```
/* Suspend until the (absolute) time specified by waketime, processing */
/* any incoming packets or new connections that occur in that interval. */

void
dgram_sleep_till(waketime)

 struct timeval *waketime;

 {

 (void)dgram_get_packets(-1, waketime);
 }


/* Tear down connection.      */

int
dgram_teardown(asn)

 long  asn;

 {
  int  fd = asn;
  int  sta;

 if (fd < 0)
  {

  /* Can't tear it down if it is already torn down! */

  errno = EINVAL;
  return (-1);
  }

 fromlen = 0;
 justrcvd = 0;
 sta = close(fd);
 log_teardown(NULL, sta == 0 ? -1 : sta);
 return (sta);
 }
```

```
/*****************************************************************************
 *                                                                           *
 * File: prot_ipport.c                                                       *
 *                                                                           *
 * Convert ASCII IP address of the form a.b.c.d.port to                      *
 * sockaddr_in.                                                              *
 *                                                                           *
 * Written 04-Sep-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header: /tmp_mnt/net/usr.projecte/dartnet/src/tg/sun4/
RCS/prot_ipport.c,v 1.5 90/11/26 12:29:40 dlee Exp Locker: denny $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/time.h>
#include <math.h>
#include "config.h"
#include "distribution.h"
#include "protocol.h"

/* Type definitions local to this file.    */

/* Functions exported from this file.    */

/* Functions local to this file.    */

/* Variables exported from this file.    */

/* Variables local to this file.    */



/* Convert an ascii address to a sockaddr.    */

int
ipport_atoaddr(addr, s)

 char  *addr;
 struct sockaddr *s;

 {
 struct sockaddr_in *sin = (struct sockaddr_in *)s;
```

```
 unsigned int a;
 unsigned int b;
 unsigned int c;
 unsigned int d;

 bzero((char *)s, sizeof(*s));
 sin->sin_family = AF_INET;
 if (sscanf(addr, "%u.%u.%u.%u.%hu",
   &a, &b, &c, &d, &(sin->sin_port)) != 5)
  return (0);
 if ((a > 255) ||
  (b > 255) ||
  (c > 255) ||
  (d > 255))
  return (0);
 sin->sin_addr.s_addr = (a << 24) | (b << 16) | (c << 8) | d;
 return (1);
 }


/* Convert a sockaddr structure to an ascii address.   */

int
ipport_addrtoa(s, addr)

 struct sockaddr *s;
 char  *addr;

 {
 struct sockaddr_in *sin = (struct sockaddr_in *)s;
 unsigned int a;
 unsigned int b;
 unsigned int c;
 unsigned int d;
 unsigned long ipaddr;

 ipaddr = sin->sin_addr.s_addr;
 d = ipaddr & 0xff;
 c = (ipaddr >>= 8) & 0xff;
 b = (ipaddr >>= 8) & 0xff;
 a = (ipaddr >>= 8) & 0xff;
 (void)sprintf(addr, "%u.%u.%u.%u.%u", a, b, c, d, sin->sin_port);
 return (1);
 }


/* Convert a binary log address to a sockaddr.    */

char *
ipport_btoaddr(addr, s)

 char  *addr;
```

```
 struct sockaddr *s;

 {
 struct sockaddr_in *sin = (struct sockaddr_in *)s;

 sin->sin_family = AF_INET;
 (void)bcopy(addr, (char *)&(sin->sin_port), sizeof(sin->sin_port));
 (void)bcopy((char *)&(addr[sizeof(sin->sin_port)]),
   (char *)&(sin->sin_addr.s_addr),
   sizeof(sin->sin_addr.s_addr));
 return (&(addr[sizeof(sin->sin_port) + sizeof(sin->sin_addr.s_addr)]));
 }


/* Convert a sockaddr binary log address.    */

char *
ipport_addrtob(s, addr)

 struct sockaddr *s;
 char  *addr;

 {
 struct sockaddr_in *sin = (struct sockaddr_in *)s;

 (void)bcopy((char *)&(sin->sin_port), addr, sizeof(sin->sin_port));
 (void)bcopy((char *)&(sin->sin_addr.s_addr),
   (char *)&(addr[sizeof(sin->sin_port)]),
   sizeof(sin->sin_addr.s_addr));
 return (&(addr[sizeof(sin->sin_port) + sizeof(sin->sin_addr.s_addr)]));
 }
```

```
/*****************************************************************************
 *                                                                           *
 * File: prot_st2.c                                                          *
 *                                                                           *
 * Common routines for interfacing to ST-II socket protocols                 *
 * suite via the normal user-level interface.                                *
 *                                                                           *
 * Modified 06-12-91 by C.Lynn for use with ST-II.                           *
 * Written 09-Aug-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header$";
#endif lint

/* Include files.      */

#include <stdio.h>

#include <sys/types.h> /* Required by sys/socket.h, fd_set */
#include <netinet/in.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <sys/socket.h>  /* Socket defns, including struct msg */
#include "config.h"
#include "log.h"
#include <sys/uio.h>  /* iov for struct msg */
#include "st2_api.h"
#include "distribution.h"
#include "protocol.h"

/* Type definitions local to this file.     */

/* Functions exported from this file.     */

/* Functions local to this file.     */

/* Variables exported from this file.     */

/* Variables local to this file.     */

static int   fdmax = -1; /* maximum value for client FD. */
static fd_set   fds;  /* FDS to clients.  */
static int   firsttime = 1; /* initialization flag.  */
static unsigned long *idr = NULL; /* per-asn Packet IDs for read. */
static unsigned long *idw = NULL; /* per-asn Packet IDs for write.*/
```

```
static int   justrcvd = 0; /* Must rcv before next send. */
static protocol  *prot = NULL; /* pointer to protocol pars. */
static protocol_table *prtab = NULL; /* pointer to protocol table. */
static int   rcving = 0; /* Receiving packets?  */
static struct sockaddr *sa = NULL; /* per-asn socket addresses. */
static int   sfd = -1; /* Socket file descriptor. */

static Instsockaddr_st2(from,
 unsigned char parm0[(MAX_ST_NAM-sizeofsockaddr_st2header)/4];);
    /* Socket structure client */
static Instsockaddr_st2(to,) = { Initsockaddr_st2(STReqUnspec,STDataTBit) };
static struct msghdr  msg;  /* For recvmsg/sendmsg ... */
static struct iovec  dataiov, /* For send/receive data descriptor */
  *iovecp = &dataiov; /* Ptr to a data descriptor */
static union {    /* First component of msg */
 struct cmsghdr  cm;
 char   buf[ MAX_ST_CTL ]; /* For control info from ST-II */
}    ctl;
static struct cmsghdr *ctlp = &( ctl.cm ); /* Contents of ctlbuf */


/*
|* struct aParameter
|* FindParm ( pcode, parmsp, parmslen )
|* Usage:
|* Scan a list of parameters with given length for first
|* parameter with specified parameter code.
|* Arguments:
|* pcode desired parameter code
|* parmsp pointer to parameter list
|* parmslen length of parameter list
|* Returns:
|* pointer to desired parameter if found, or NULL
|*
 */


 static struct aParameter *
FindParm( pcode, parmsp, parmslen )
 octet1   pcode;  /* Desired parameter code */
 struct aParameter *parmsp; /* Pointer ot parameter list */
 int    parmslen; /* Length of parameter list */
{
 struct aParameter *parmp = parmsp; /* Current parameter */

 /* Move it beyond last parameter */
 parmsp = (struct aParameter *) ((long) parmsp + parmslen);

 /* Scan through list */
 while ( parmp < parmsp ) {
 if ( ntohl( parmp->pcode ) == pcode ) {
  return ( parmp );  /* Found match */
```

```
 }

 /* move to next parameter, assuming well formatted */
 parmp = (struct aParameter *) ((long) parmp + parmp->plen);
 }

 return ( (struct aParameter *) NULL ); /* desired parameter not in list */

} /* end of FindParm */


/* Accept connections and packets while waiting for the opportunity to */
/* write (if wfd specified) or for specified timeout, whichever comes */
/* first.       */

 int
st2_get_packets( wfd, endtoutp )
 int    wfd; /* FD for write, -1 if none.  */
 struct timeval *endtoutp; /* end of timeout period.  */
{
 static char  *bufp = NULL; /* pointer to receive buffer. */
 int    fd,
    i,
    len,
    nitems, /* Number of items selected.  */
    pklen; /* Input packet length.   */
 fd_set   rfds, /* Read FDs for select.   */
    wfds; /* Write FDs for select.  */
 struct timeval  tv,
   *tvp;


 /* Initialization on first pass, if necessary.   */

 if ( firsttime )
 {
 FD_ZERO (&fds);
 if ( sfd >= 0 )
 {
  FD_SET (sfd,&fds);
  fdmax = sfd;
 }
 firsttime = 0;
 }

 /* Calculate initial timeout. We force a select even if the */
 /* timeout has already expired in order to prevent the CPU from */
 /* starving the I/O.     */

 if ( endtoutp == NULL )
 {
 /* No timeout, so just use NULL pointer.   */
```

```
tvp = NULL;
}
else
{
/* Calculate timeout.      */

if ( gettimeofday( &tv, (struct timezone *)NULL ) == -1 )
{
 log_error( /*adr1*/ NULL, /*adr2*/ NULL, /*asn*/ -1,
  /*errcod*/ LOGERR_SELECT );
 perror( "st2_get_packets: gettimeofday" );
 abort();
}

if ( timercmp( endtoutp, &tv, >) )
{
 timersub( endtoutp, &tv, &tv );

 /* Wait at most one hour at a time. SunOS has some strange */
 /* limitations on the timeout in the select call, I guess */
 /* they just don't believe in waiting forever. One hour is */
 /* shorter than forever.     */

 if ( tv.tv_sec > 3600 )
 tv.tv_sec = 3600;
}
else
{
 tv.tv_sec = 0;
 tv.tv_usec = 0;
}

tvp = &tv;
}

/* Each pass through the following loop does one select call to */
/* check the state of the fds.     */

rcving = 1;
justrcvd = 1;

for (;;)
{
/* Set up for select: get fd bitmaps.    */

rfds = fds;
FD_ZERO (&wfds);

if ( wfd >= 0 )
{
 FD_SET (wfd,&rfds);
```

```
 FD_SET (wfd,&wfds);
}

if ( (nitems = select( fdmax > wfd ? fdmax + 1 : wfd + 1,
   &rfds, &wfds, (fd_set *) NULL, tvp )) == -1 )
{
 if ( errno != EINTR )
 {
 log_error( /*adr1*/ NULL, /*adr2*/ NULL, /*asn*/ -1,
   /*errcod*/ LOGERR_SELECT );
 perror( "st2_get_packets: select" );
 abort();
 }
}

/* Accept any outstanding connections.    */

if ( (nitems > 0) && FD_ISSET (sfd,&rfds) )
{
 FD_CLR (sfd,&rfds);
 nitems--;

 /* Each pass through the following loop accepts one  */
 /* connection.      */

 for (;;)
 {
 struct aParameter *parmp;


 len = sizeof (from);

 fd = accept( sfd, (struct sockaddr *) &from, &len );

 if ( fd >= 0 )
 {
  FD_SET (fd,&fds);
  idr[fd] = 0;

  if ( fd > fdmax )
  fdmax = fd;

  parmp = FindParm( STpOrigin/*STRmtAppEnt*/,
    (struct aParameter *) &( from.parm0[0] ),
    len - sizeofsockaddr_st2header );

  if ( parmp == (struct aParameter *) NULL )
  {
  log_error( /*adr1*/ NULL, /*adr2*/ NULL, /*asn*/ fd,
    /*errcod*/ LOGERR_GETPEER );
  perror( "st2_get_packets: accept" );
  abort();
```

```
    }

    sa[fd].sa_family = AF_INET; /*from.sa_family*/
    bcopy( (char *) &( ((struct aApplEntity *) parmp)->SAP[0] ),
     (char *) &( sa[fd].sa_data[0] ), 2 );
    bcopy( (char *) &( ((struct aApplEntity *) parmp)->IPAdr ),
     (char *) &( sa[fd].sa_data[2] ), 4 );

    InitMsg (&(msg),&( from ),sizeof (from),
     (struct iovec *) NULL,0,
     &( ctl.cm ),sizeof (ctl));

    len = recvmsg( fd, &msg, 0/*flags*/ );

    if ( (len == -1) || (from.st_request != STReqConnect) )
    {
    log_error( /*adr1*/ NULL, /*adr2*/ NULL, /*asn*/ fd,
      /*errcod*/ 17 );
    perror( "st2_get_packets: recvmsg" );
    abort();
    }

    from.st_request = STReqAccept;
    len = sendmsg( fd, &msg, 0/*flags*/ );

    if ( len == -1 )
    {
    log_error( /*adr1*/ NULL, /*adr2*/ NULL, /*asn*/ fd,
      /*errcod*/ 18 );
    perror( "st2_get_packets: sendmsg" );
    abort();
    }

    log_accept( &(sa[fd]), NULL, fd, -1 );
   }
   else if ( errno != EWOULDBLOCK )
   {
    log_accept( NULL, NULL, -1, errno );
    perror( "st2_get_packets: accept" );
    abort();
   }
   else
    break;
   }
}

/* Each pass through the following loop checks for messages on */
/* one file descriptor.     */

for ( i = 0; (i <= fdmax) && (nitems > 0); i++ )
{
```

```
/* If current fd was not selected, ignore it.  */

if ( ! FD_ISSET (i,&rfds) )
continue;

/* Each pass through the following loop attempts to  */
/* receive one segment.     */

for (;;)
{
/* Get a buffer if we do not already have one on hand. */

if ( (bufp == NULL)
 && ( (bufp = (*(prtab->buffer_get)) ( MAX_PKT_SIZE ))
 == NULL) )
{
 log_error( /*adr1*/ NULL, /*adr2*/ NULL, /*asn*/ -1,
  /*errcod*/ LOGERR_MEM );
 (void) fprintf( stderr, "%s %s\n", "st2_get_packets: ",
   "out of memory!" );
 abort();
}

/* Receive the segment, scream and die if error. */

dataiov.iov_base = (caddr_t) bufp;
dataiov.iov_len = MAX_PKT_SIZE;

InitMsg (&(msg),&( from ),sizeof (from),
  &dataiov,1,
  &( ctl.cm ),sizeof (ctl));

pklen = recvmsg( i, &msg, 0/*flags*/ );

if ( (pklen < 0) && (errno != EWOULDBLOCK) )
{
 /* Shut down and tell rcv if EOF or error.  */

 log_rx( /*adr1*/ &( sa[i]), /*adr2*/ NULL, /*asn*/ i,
  /*id*/ 0, /*len*/ 0, /*errno*/ errno );

 if ( close( i ) == -1 )
 {
 perror( "close" );
 log_teardown( /*tvp*/ NULL, /*errno*/ errno );
 abort();
 }

 FD_CLR (i,&fds);
 pklen = 0;

 if ( prtab->rcv != NULL )
```

```
 (void) (*(prtab->rcv)) ( i, i, NULL, pklen, 0 );
 break;
}
else if ( pklen == 0 )
{
 log_rx( /*adr1*/ &(sa[i]), /*adr2*/ NULL, /*asn*/ i,
  /*id*/ from.st_request, /*len*/ 0, /*errno*/ -1 );

 if ( (prtab->rcv != NULL)
 && ((*(prtab->rcv)) ( i, i, bufp, 0, idr[i] )) )
 bufp = NULL;

 if ( from.st_request == STReqDisconnect )
 {
 if ( close( i ) == -1 )
 {
  perror( "close" );
  log_teardown( /*tvp*/ NULL, /*errno*/ errno );
  abort();
 }

 FD_CLR (i,&fds);
 break;  /* out of forever loop */
 }
}
else if ( pklen > 0 )
{
 /* Pass packet to receiver.    */

 char  *cp = bufp;
 int    j = 5;
 unsigned long *ulp,
    v = 0;

 if ( *cp != 0 ) /* maybe from example: 5-digit seq # */
 {
 for ( ; j > 0 ; j-- )
 {
  if ( *cp < '0' || '9' < *cp )
  break;

  v = v * 10 + *cp++ - '0';
 }

 if ( (j <= 0) /* ok format */
  && ((idr[i] % 100000) != v) )
  j = v, v++;
 else
  j = -1, v = idr[i] + 1;
 }
 else if ( pklen >= (5 * sizeof (unsigned long)) )
 {
```

```
  ulp = (unsigned long *) bufp;
  v = *ulp;

  if ( idr[i] != v )
   j = v;
  else
   j = -1;
  v++;
  }
  else
  j = -1, v = idr[i] + 1;

  log_rx( /*adr1*/ &(sa[i]), /*adr2*/ NULL, /*asn*/ i,
   /*id*/ idr[i], /*len*/ pklen, /*errno*/ j );

  if ( (prtab->rcv != NULL)
  && ((*(prtab->rcv)) ( i, i, bufp, pklen, idr[i] )) )
  bufp = NULL;

  idr[i] = v;
 }
 else
 {
  /* Nothing more to read right now.   */

  break;
 }
 }

 if ( --nitems < 0 )
 break;
}

/* Check for ability to write...    */

if ( (nitems > 0) && (wfd > 0) && (FD_ISSET (wfd,&wfds)) )
{
 rcving = 0;
 return (1);
}

/* Calculate next timeout. If the timeout has expired, tell */
/* the caller the sad story.     */

if ( endtoutp != NULL )
{
 if ( gettimeofday( &tv, (struct timezone *) NULL ) == -1 )
 {
 log_error( /*adr1*/ NULL, /*adr2*/ NULL, /*asn*/ -1,
   /*errcod*/ LOGERR_SELECT );
 perror( "st2_get_packets: getttimeofday" );
 abort();
```

```
  }

  if ( timercmp (endtoutp, &tv, >) )
  {
  timersub( endtoutp, &tv, &tv );

  /* Wait at most one hour per select.   */

  if ( tv.tv_sec > 3600 )
   tv.tv_sec = 3600;
  }
  else
  {
  errno = ETIME;
  rcving = 0;
  return (0);
  }
 }
 }

 /* NOTREACHED */
}


/* Send a packet, subject to the timeout.    */

 int
st2_send( asn, bufp, len, endtoutp, pktidp ) /* from tg.y:generate */
 long    asn;  /* Association number */
 char   *bufp;  /* Ptr to buffer to be used */
 int     len;  /* Length to be used */
 struct timeval *endtoutp; /* Ptr to Timeout */
 unsigned long  *pktidp; /* Ptr to static "ID" ???per ASN */
{
 int    cc,
    fd = asn;
 static int   firsttime = 1; /* Must wait for select to say */
     /* that connect has succeeded. */
 unsigned long *ulp;


 /* Make sure that fd wasn't closed out from under the sender. */

 if ( fd < 0 )
 {
 errno = EINVAL;
 return (-1);
 }

 /* Setup buffer and iov and msghdr */

 dataiov.iov_base = (caddr_t) bufp;
```

```
if ( (dataiov.iov_len = len) >= (5 * sizeof (unsigned long)) )
{
ulp = (unsigned long *) bufp;
*ulp++ = *pktidp;
gettimeofday( (struct timeval *) ulp, 0 );
ulp += sizeof (struct timeval) / sizeof (unsigned long);
*ulp++ = idw[fd];
*ulp = len;
}

InitMsg (&(msg),&( to ),sizeof (to),
 &dataiov,1, (struct cmsghdr *) NULL,0 );


/* If we just got done allowing receives, try the write without */
/* bothering to do another receive.     */
/* However, don't do this the first time around because we must */
/* check for the connect completing.    */

if ( (justrcvd && ! firsttime) || rcving )
{
/* Try to write out the packet.     */
;
} /* ... else */

/* The write failed with an EWOULDBLOCK or we need to allow some */
/* receives to happen. Thus, we must honor the timeout period, */
/* unless we are being recursively invoked.    */

/* Invoke the get_packets routine to receive packets and accept */
/* new connections while we are waiting to write.   */

else if ( ! st2_get_packets( fd, endtoutp ) )
{
buffer_generic_free( bufp );
return (-1);
}
else
{
/* Write out the packet!     */

firsttime = 0;
}

cc = sendmsg( fd, &msg, 0/*flags*/ );

if ( cc >= 0 )
{
log_tx( /*tvp*/ NULL, /*adr1*/ &(sa[fd]), /*adr2*/ NULL,
 /*asn*/ fd, /*id*/ *pktidp, /*len*/ cc, /*errno*/ -1 );
(*pktidp)++;
idw[fd] += cc;
```

```
 }
 else if ( errno != EWOULDBLOCK )
 {
 log_tx( /*tvp*/ NULL, /*adr1*/ &(sa[fd]), /*adr2*/ NULL,
  /*asn*/ fd, /*id*/ *pktidp, /*len*/ 0, /*errno*/ errno );
 }
 else if ( rcving )
 {
 log_tx( /*tvp*/ NULL, /*adr1*/ &(sa[fd]), /*adr2*/ NULL,
  /*asn*/ fd, /*id*/ *pktidp, /*len*/ cc, /*errno*/ errno );
 }

 buffer_generic_free( bufp );
 justrcvd = 0;

 return (cc);
}


/* Record the fact that a setup has occurred.    */

 void
st2_setup( pp, fd )  /* from prot_straw.c: straw_setup */
 protocol  *pp;
 int    fd;
{
 int    nfds;


 if ( sfd >= 0 )
 {
 log_error( /*adr1*/ NULL, /*adr2*/ NULL, /*asn*/ sfd,
   /*errcod*/ LOGERR_2SETUP );
 (void) fprintf( stderr, "dgram_setup: %s %s\n",
   "cannot set up new server without",
   "tearing down old one first" );
 abort();
 }

 /* Allocate memory for ID arrays.     */

 if ( idr == NULL )
 {
 nfds = getdtablesize();

 if ( (idr = (unsigned long *) malloc( nfds * sizeof (unsigned long) ))
  == NULL )
 {
  log_error( /*adr1*/ NULL, /*adr2*/ NULL, /*asn*/ sfd,
   /*errcod*/ LOGERR_MEM );
  (void) fprintf( stderr, "st2_stream_setup: Out of memory!\n" );
  abort();
```

```
}
}

if ( idw == NULL )
{
nfds = getdtablesize();

if ( (idw = (unsigned long *) malloc( nfds * sizeof (unsigned long) ))
 == NULL )
{
 log_error( /*adr1*/ NULL, /*adr2*/ NULL, /*asn*/ sfd,
  /*errcod*/ LOGERR_MEM );
 (void) fprintf( stderr, "st2_stream_setup: Out of memory!\n" );
 abort();
}
}

if ( sa == NULL )
{
nfds = getdtablesize();
if ( (sa = (struct sockaddr *) malloc( nfds * sizeof (struct sockaddr) ))
 == NULL )
{
 log_error( /*adr1*/ NULL, /*adr2*/ NULL, /*asn*/ sfd,
  /*errcod*/ LOGERR_MEM );
 (void) fprintf( stderr, "st2_stream_setup: Out of memory!\n" );
 abort();
}
}

/* Let st2_get_packets know of the change.    */

prot = pp;
prtab = pp->prot;

if ( fd < 0 )
{
int   cfd = -fd - 1;


FD_SET (cfd,&fds);

if ( cfd > fdmax )
 fdmax = cfd;

sa[cfd] = prot->dst;
}
else
{
sfd = fd;
firsttime = 1;
}
```

```
 log_setup( /*tvp*/ NULL, /*errno*/ -1 );

}


/* Suspend until the (absolute) time specified by waketime, processing */
/* any incoming packets or new connections that occur in that interval. */

 void
st2_sleep_till( waketimep )
 struct timeval *waketimep;
{
 (void) st2_get_packets( -1, waketimep );
 justrcvd = 1;
}


/* Tear down connection.      */

 int
st2_teardown( asn )
 long    asn;
{
 int    i,
    fd = asn,
    result,
    tmperrno;


 if ( fd < 0 )
 {
 /* Can't tear it down if it is already torn down!  */

 errno = EINVAL;
 return (-1);
 }

 result = close( fd );
 log_teardown( /*tvp*/ NULL, /*errno*/ result == 0 ? -1 : errno );

 if ( fd == sfd )
 {
 /* Preserve errno from original close().   */

 tmperrno = errno;

 /* Clear out sfd in order to force initialization if it is */
 /* later re-opened.      */

 sfd = -1;
```

```
 /* This is the server socket, so make sure to also kill  */
 /* current connections to any clients.    */

 for ( i = 0; i <= fdmax; i++ )
 {
  if ( FD_ISSET (i,&fds) )
  (void) close( i );
 }

 /* Restore errno from original close().    */

 errno = tmperrno;
 }

 return (result);
}


/*
 Local Variables:
 c-indent-level: 4
 c-brace-offset: -4
 c-brace-imaginary-offset: 0
 c-continued-statement-offset: 4
 comment-column: 40
 End:
*/
```

```
/*********************************************************************
 *                                                                   *
 * File: prot_straw.c                                                *
 *                                                                   *
 * Routines for interfacing to the raw ST-II protocol suite via      *
 * the normal user-level interface.                                  *
 *                                                                   *
 * Modified 11-Jun-91 by C.Lynn, BBN.                                *
 * Copyright (c) 1991 by BBN Systems and Technologies,               *
 *  A Division of Bolt Beranek and Newman Inc.                       *
 * Written 20-Jun-90 by Paul E. McKenney, SRI International.         *
 * Copyright (c) 1990 by SRI International.                          *
 *                                                                   *
 *********************************************************************/

#ifndef lint
static char rcsid[] = "$Header$";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include "log.h"
#include "distribution.h"
#include "protocol.h"
#include <sys/uio.h>  /* iov for struct msg */
#include "st2_api.h"

/* Type definitions local to this file.     */

#define SAPLEN 2
#define Bytesof(x) Bytes2(x)  /* !@#$ cpp lacks an "eval" */
#define DEFAULT_PROTOCOL 255
#define DEFAULT_PORT 1234

int st2_options = STOptLBit | STTSRYes; /* Ought to fix parser ... */

Instsockaddr_st2(local,InstaApplEntity(here,SAPLEN,/*no srcrut*/);) = {
 Initsockaddr_st2(STReqUnspec,0/*opt_xxx*/),
 InitaApplEntity(STLclAppEnt,local.here,
 INADDR_ANY,DEFAULT_PROTOCOL,SAPLEN,Bytesof(DEFAULT_PORT),/*no srcrut*/)
};
```

```
Instsockaddr_st2(remote,
 InstaFlowSpec3(tos);
 InstaTargetList(targlst,
  InstaTarget(targ1,SAPLEN,);
 );
) = {
 Initsockaddr_st2(STReqUnspec,0/*opt_xxx*/),
 InitaFlowSpec3(STpFlowSpec,
   /*min*/ 128/*bytes*/, 2/*pps*/, 128*2/*bandwidth*/,
   /*des*/ 1024/*bytes*/, 10/*pps*/),
 InitaTargetList(remote.targlst,1/*targets*/,
 InitaTarget(remote.targlst.targ1,
  INADDR_ANY,SAPLEN,0,)
 )
};


/* Functions exported from this file.     */

/* Functions local to this file.     */

/* Variables exported from this file.     */

/* Variables local to this file.     */

static protocol_table *prtab = NULL; /* pointer to protocol table. */

static Instsockaddr_st2(from,
 unsigned char parm0[(MAX_ST_NAM-sizeofsockaddr_st2header)/4];);
    /* Socket structure client */
static struct msghdr  msg;  /* For recvmsg/sendmsg ... */
static struct iovec  ctliov; /* For recvfrom data descriptor */
static char   ctlnodata[8]; /* Broken recvmsg! */
static union {    /* First component of msg */
 struct cmsghdr  cm;
 char   buf[ MAX_ST_CTL ]; /* For control info from ST-II */
}    ctl;
static struct cmsghdr *ctlp = &( ctl.cm ); /* Contents of ctlbuf */


/* ST-II stream setup function.     */

 long
straw_setup( protp )   /* from tg.y: do_actions */
 protocol  *protp;
{
 int    flags,
    sfd,  /* Socket file descriptor. */
    len;
```

```
/* Save pointer to protocol table.     */

prtab = protp->prot;

/* Create ST-II socket.      */

if ( (sfd = socket(AF_COIP, SOCK_RAW, 0)) == -1 )
{
return (-1);
}

if ( ! (protp->qos & QOS_SERVER) )
{
unsigned long  bytes,
   ratex10,
   bandwidthx10;


/* Handle client side. Connect up a socket and return  */
/* it to the caller.      */

if ( (protp->qos & QOS_MTU) != 0 )
{
 bytes = protp->mtu;
}
else
{
 printf( "ST-II: packet size not specified (mtu)\n" );
 return (-1);
}

if ( (protp->qos & QOS_INTERVAL) != 0 )
{
 ratex10 = 10.0 / protp->interval;
 if ( ratex10 == 0 )
 {
 printf( "ST-II: specified interval is less than 0.1 pps, 0.1 pps assumed\n"
);
 ratex10 = 1;
 }
}
else
{
 printf( "ST-II: packet interval not specified (interval)\n" );
 return (-1);
}

if ( (protp->qos & QOS_PEAK_BANDWIDTH) != 0 )
{
 bandwidthx10 = 10 * protp->peak_bandwidth;
}
else if ( (protp->qos & QOS_AVG_BANDWIDTH) != 0 )
```

```
{
 bandwidthx10 = 10 * protp->avg_bandwidth;
}
else
{
 bandwidthx10 = bytes * ratex10;
}

/* Establish a connection.    */

local.here.IPAdr = * (unsigned long *) &( protp->src.sa_data[2] );
local.here.NextPcol = DEFAULT_PROTOCOL;
local.here.SAP[0] = protp->src.sa_data[0];
local.here.SAP[1] = protp->src.sa_data[1];

/* Bind socket.      */

if ( bind( sfd, &( local ), sizeof (local) ) == -1 )
{
 (void) close( sfd );
 return (-1);
}

remote.st_options = st2_options;

remote.targlst.targ1.IPAdr = * (unsigned long *) &( protp->dst.sa_data[2] );
remote.targlst.targ1.SAP[0] = protp->dst.sa_data[0];
remote.targlst.targ1.SAP[1] = protp->dst.sa_data[1];
remote.targlst.targ1.SAPBytes = 2;

remote.tos.LimitOnPDUBytes = bytes;
remote.tos.DesPDUBytes = bytes;
remote.tos.LimitOnPDURate = ratex10;
remote.tos.DesPDURate = ratex10;
remote.tos.MinBytesXRate = bandwidthx10;

if ( (connect( sfd, (struct sockaddr *) &( remote ),
  sizeof (remote) ) == -1 )
 && (errno != EINPROGRESS) )
{
 (void) close( sfd );
 return (-1);
}

/* Prevent slow connection-setup from killing us.  */

(void) signal( SIGPIPE, SIG_IGN );

ctliov.iov_base = (caddr_t) ctlnodata;
ctliov.iov_len = sizeof (ctlnodata);
InitMsg(&(msg),&( from ),sizeof (from), &ctliov,1,
 &( ctl.cm ),sizeof (ctl));
```

```
len = recvmsg( sfd, &msg, 0/*flags*/ );
if ( len == -1 )
{
 log_error(NULL, NULL, sfd, 17);
 perror( "st2_get_packets: recvmsg Accept" );
 abort();
}

if ( from.st_request != STReqAccept )
{
 (void) close( sfd );
 return ( -1 );
}

if ( (flags = fcntl( sfd, F_GETFL, 0 )) == -1 )
{
 log_error( NULL, NULL, sfd, LOGERR_FCNTL );
 perror( "fcntl F_GETFL" );
 abort();
}
if (fcntl( sfd, F_SETFL, flags | FNDELAY ) == -1 )
{
 (void) close( sfd );
 return (-1);
}

/* Tell the get_packets routine about the new client socket. */

st2_setup( protp, -sfd - 1 );
}
else
{

/* Handle server side. This will need to be modified to allow */
/* the server to accept multiple connections.   */

local.here.IPAdr = * (unsigned long *) &( protp->src.sa_data[2] );
local.here.NextPcol = DEFAULT_PROTOCOL;
local.here.SAP[0] = protp->src.sa_data[0];
local.here.SAP[1] = protp->src.sa_data[1];

/* Bind socket.      */

if ( bind( sfd, &( local ), sizeof (local) ) == -1 )
{
 (void) close( sfd );
 return (-1);
}

/* Initialize socket to set no-delay mode.   */

if ( (flags = fcntl( sfd, F_GETFL, 0 )) == -1 )
```

```
{
 log_error( NULL, NULL, sfd, LOGERR_FCNTL );
 perror( "fcntl F_GETFL" );
 abort();
}

if ( fcntl( sfd, F_SETFL, flags | FNDELAY ) == -1 )
{
 log_error( NULL, NULL, sfd, LOGERR_FCNTL );
 perror( "fcntl F_SETFL FNDELAY" );
 abort();
}

/* Declare willingness to accept connections.   */

(void) listen( sfd, 3 );

/* Tell the get_packets routine about the new server socket. */

st2_setup( protp, sfd );
}

return (sfd);
}


/*
 Local Variables:
 c-indent-level: 4
 c-brace-offset: -4
 c-brace-imaginary-offset: 0
 c-continued-statement-offset: 4
 comment-column: 40
 End:
*/
```

```
/*****************************************************************************
 *                                                                           *
 * File: prot_stream.c                                                       *
 *                                                                           *
 * Common routines for interfacing to stream socket protocols                *
 * suite via the normal user-level interface.                                *
 *                                                                           *
 * Written 09-Aug-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/src/tg/RCS/
prot_stream.c,v 1.7 90/11/26 12:29:42 dlee Exp Locker: dlee $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <sys/socket.h>
#include "config.h"
#include "log.h"
#include "distribution.h"
#include "protocol.h"

/* Type definitions local to this file.     */

/* Functions exported from this file.     */

/* Functions local to this file.     */

/* Variables exported from this file.     */

/* Variables local to this file.     */

static int  fdmax = -1; /* maximum value for client FD. */
static fd_set  fds;  /* FDS to clients.  */
static int  firsttime = 1; /* initialization flag.  */
static unsigned long *idr = NULL; /* per-asn Packet IDs for read. */
static unsigned long *idw = NULL; /* per-asn Packet IDs for write.*/
static int  justrcvd = 0; /* Must rcv before next send. */
static protocol  *prot = NULL; /* pointer to protocol pars. */
static protocol_table *prtab = NULL; /* pointer to protocol table. */
```

```
static int  rcving = 0; /* Receiving packets?  */
static struct sockaddr *sa = NULL; /* per-asn socket addresses. */
static int  sfd = -1; /* Socket file descriptor. */
static void (*setup_qos)();
 /* Function to set up FD for */
 /* desired quality of service. */



/* Accept connections and packets while waiting for the opportunity to */
/* write (if wfd specified) or for specified timeout, whichever comes */
/* first.       */

int
stream_get_packets(wfd, endtout)

 int  wfd;  /* FD for write, -1 if none. */
 struct timeval *endtout; /* end of timeout period. */

 {
 static char *buf = NULL; /* pointer to receive buffer. */
 int  fd;
 struct sockaddr_in from; /* Socket structure client */
 int  i;
 int  len;
 int  nitems;  /* Number of items selected. */
 int  pklen;  /* Input packet length.  */
 fd_set  rfds;  /* Read FDs for select.  */
 struct timeval tv;
 struct timeval *tvp;
 fd_set  wfds;  /* Write FDs for select. */

 /* Initialization on first pass, if necessary.   */

 if (firsttime)
  {
  FD_ZERO(&fds);
  if (sfd >= 0)
   {
   FD_SET(sfd, &fds);
   fdmax = sfd;
   }
  firsttime = 0;
  }

 /* Calculate initial timeout. We force a select even if the */
 /* timeout has already expired in order to prevent the CPU from */
 /* starving the I/O.     */

 if (endtout == NULL)
  {
```

```
 /* No timeout, so just use NULL pointer.  */

 tvp = NULL;
 }
else
 {

 /* Calculate timeout.     */

 if (gettimeofday(&tv, (struct timezone *)NULL) == -1)
  {
  log_error(NULL, NULL, -1, LOGERR_SELECT);
  perror("stream_get_packets: gettimeofday");
  abort();
  }
 if (timercmp(endtout, &tv, >))
  {
  timersub(endtout, &tv, &tv);

  /* Wait at most one hour at a time. SunOS has */
  /* some strange limitations on the timeout in */
  /* the select call, I guess they just don't */
  /* believe in waiting forever. One hour is */
  /* shorter than forever.   */

  if (tv.tv_sec > 3600)
   tv.tv_sec = 3600;
  }
 else
  {
  tv.tv_sec = 0;
  tv.tv_usec = 0;
  }
 tvp = &tv;
 }

/* Each pass through the following loop does one select call */
/* to check the state of the fds.    */

rcving = 1;
justrcvd = 1;
for (;;)
 {

 /* Set up for select: get fd bitmaps.   */

 rfds = fds;
 FD_ZERO(&wfds);
 if (wfd >= 0)
  {
  FD_SET(wfd, &rfds);
```

```
 FD_SET(wfd, &wfds);
 }
if ((nitems = select(fdmax > wfd ? fdmax + 1 : wfd + 1,
   &rfds,
   &wfds,
   (fd_set *)NULL,
   tvp)) < 0)
 {
 if (errno != EINTR)
  {
  log_error(NULL, NULL, -1, LOGERR_SELECT);
  perror("stream_get_packets: select");
  abort();
  }
 }

/* Accept any outstanding connections.   */

if ((nitems > 0) &&
 FD_ISSET(sfd, &rfds))
 {
 FD_CLR(sfd, &rfds);
 nitems--;

 /* Each pass through the following loop accepts */
 /* one connection.    */

 for (;;)
  {
  len = sizeof(from);
  fd = accept(sfd,
    (struct sockaddr *)&from,
    &len);

  if (fd >= 0)
   {
   int  namelen;

   FD_SET(fd, &fds);
   idr[fd] = 0;
   if (fd > fdmax)
    fdmax = fd;
   namelen = sizeof(sa[fd]);
   if (getpeername(fd,
     &(sa[fd]),
     &namelen) == -1)
    {
    log_error(NULL,
      NULL,
      fd,
      LOGERR_GETPEER);
    perror("stream_get_packets: getpeername");
```

```
    abort();
    }
   /* Apply desired quality-of-service to nfd. */

   (*setup_qos)(fd);

   log_accept(&(sa[fd]), NULL, fd, -1);
   }
  else if (errno != EWOULDBLOCK)
   {
   log_accept(NULL, NULL, -1, errno);
   perror("stream_get_packets: accept");
   abort();
   }
  else
   break;
  }
 }

/* Each pass through the following loop checks for */
/* messages on one file descriptor.   */

for (i = 0; (i <= fdmax) && (nitems > 0); i++)
 {

 /* If current fd was not selected, ignore it. */

 if (!FD_ISSET(i, &rfds))
  continue;

 /* Each pass through the following loop  */
 /* attempts to receive one segment.  */

 for (;;)
  {

  /* Get a buffer if we do not already */
  /* have one on hand.   */

  if ((buf == NULL) &&
   ((buf =
   (*(prtab->buffer_get))(MAX_PKT_SIZE)) ==
   NULL))
   {
   log_error(NULL, NULL, -1, LOGERR_MEM);
   (void)fprintf(stderr,
     "%s %s\n",
     "stream_get_packets: ",
     "out of memory!");
   abort();
   }
```

```
/* Receive the segment, scream and die */
/* if error.    */

pklen = read(i, buf, MAX_PKT_SIZE);
if ((pklen == 0) ||
 ((pklen < 0) &
 (errno != EWOULDBLOCK)))
 {

 /* Shut down and tell rcv if */
 /* EOF or error.  */

 if (close(i) == -1)
  {
  log_rx(&(sa[i]), NULL, -1, 0, 0,
   pklen == 0 ? -1 : errno);
  perror("close");
  log_teardown (NULL, -1);
  abort();
  }
 FD_CLR(i, &fds);
 len = 0;
 if (prtab->rcv != NULL)
  (void)(*(prtab->rcv))(i,
     i,
     NULL,
     pklen,
     0);
 break;
 }
else if (pklen > 0)
 {

 /* Pass packet to receiver. */

 log_rx(&(sa[i]), NULL, -1, idr[i], pklen,
  -1);
 if ((prtab->rcv != NULL) &&
  ((*(prtab->rcv))(i,
    i,
    buf,
    pklen,
    idr[i])))
  buf = NULL;
 idr[i] += pklen;
 }
else
 {

 /* Nothing more to read right */
 /* now.    */
```

```
    break;
    }
   }

  if (--nitems <= 0)
   break;
  }

 /* Check for ability to write...   */

 if ((nitems > 0) &&
  (wfd > 0) &&
  (FD_ISSET(wfd, &wfds)))
  {
  rcving = 0;
  return (1);
  }

 /* Calculate next timeout. If the timeout has expired, */
 /* tell the caller the sad story.   */

 if (endtout != NULL)
  {

  if (gettimeofday(&tv, (struct timezone *)NULL) == -1)
   {
   log_error(NULL, NULL, -1, LOGERR_SELECT);
   perror("stream_get_packets: getttimeofday");
   abort();
   }
  if (timercmp(endtout, &tv, >))
   {
   timersub(endtout, &tv, &tv);

   /* Wait at most one hour per select. */

   if (tv.tv_sec > 3600)
    tv.tv_sec = 3600;
   }
  else
   {
   errno = ETIME;
   rcving = 0;
   return (0);
   }
  }

 }

/* NOTREACHED */
}
```

```
/* Send a packet, subject to the timeout.    */

int
stream_send(asn, buf, len, endtout, pktid)

 long  asn;
 char  *buf;
 int  len;
 struct timeval *endtout;
 unsigned long *pktid;

{
 int  cc;
 int  fd = asn;
 static int firsttime = 1; /* Must wait for select to say */
    /* that connect has succeeded. */
 char  *tmpbuf = buf;

 /* Make sure that fd wasn't closed out from under the sender. */

 if (fd < 0) {
  errno = EINVAL;
  return (-1);
 }

 /* If we just got done allowing receives, try the write without */
 /* bothering to do another receive.    */
 /* However, don't do this the first time around because we must */
 /* check for the connect completing.    */

 if ((justrcvd && !firsttime) || rcving)
 {

   /* Try to write out the packet.    */

  cc = write(fd, tmpbuf, len);
  if ((cc >= 0) || ((cc <0) && (errno != EWOULDBLOCK)) || rcving)
  {
  log_tx(NULL, &(sa[fd]), NULL, -1, idw[fd],
   cc, cc == -1 ? errno : -1);
  if (cc >= 0) {
   *pktid = idw[fd];
   idw[fd] += cc;
  }
  if ((cc == len) || (cc < 0))
  {
   buffer_generic_free(buf);
   justrcvd = 0;
   return (cc);
  }
  }
```

```
 /* Adjust buffer pointers if partial write occurred. */

 if (cc > 0)
  {
 tmpbuf += cc;
 len -= cc;
 }

}

/* The write failed with an EWOULDBLOCK or we need to allow */
/* some receives to happen. Thus, we must honor the timeout */
/* period, unless we are being recursively invoked.  */


/* Each pass through the following loop attempts to write the */
/* packet, more than one pass will be needed if partial writes */
/* occur.      */

for (;;)
{

 /* Invoke the get_packets routine to receive packets and accept */
 /* new connections while we are waiting to write.  */

if (!stream_get_packets(fd, endtout)) {
 return (-1);
} else {

 /* Write out the packet!    */

 firsttime = 0;
 cc = write(fd, tmpbuf, len);
 if (cc >= 0)
{
   log_tx(NULL, &(sa[fd]), NULL, -1, idw[fd], cc, -1);
  *pktid = idw[fd];
   idw[fd] += cc;
   if (cc != len)
   {
 tmpbuf += cc;
 len -= cc;
 continue;
   }
 } else if (errno != EWOULDBLOCK) {
  log_tx(NULL, &(sa[fd]), NULL, -1, idw[fd], 0, errno);
 }
 buffer_generic_free(buf);
 justrcvd = 0;
 return (cc);
 }
}
```

```
}


/* Record the fact that a setup has occurred.    */

void
stream_setup(pp, fd,my_setup_qos)

 protocol *pp;
 int  fd;
 void (*my_setup_qos)();

 {
 int  nfds;

 if (sfd >= 0)
  {
  log_error(NULL, NULL, sfd, LOGERR_2SETUP);
  (void)fprintf(stderr,
    "dgram_setup: %s %s\n",
    "cannot set up new server without",
    "tearing down old one first");
  abort();
  }

 /* Allocate memory for ID arrays.    */

 if (idr == NULL)
  {
  nfds = getdtablesize();
  if ((idr =
   (unsigned long *)malloc(nfds * sizeof(unsigned long))) ==
   NULL)
   {
   log_error(NULL, NULL, sfd, LOGERR_MEM);
   (void)fprintf(stderr, "stream_setup: Out of memory!\n");
   abort();
   }
  }
 if (idw == NULL)
  {
  nfds = getdtablesize();
  if ((idw =
   (unsigned long *)malloc(nfds * sizeof(unsigned long))) ==
   NULL)
   {
   log_error(NULL, NULL, sfd, LOGERR_MEM);
   (void)fprintf(stderr, "stream_setup: Out of memory!\n");
   abort();
   }
  }
 if (sa == NULL)
```

```
    {
    nfds = getdtablesize();
    if ((sa =
     (struct sockaddr *)malloc(nfds *
       sizeof(struct sockaddr))) ==
     NULL)
     {
     log_error(NULL, NULL, sfd, LOGERR_MEM);
     (void)fprintf(stderr, "stream_setup: Out of memory!\n");
     abort();
     }
    }

   /* Let stream_get_packets know of the change.   */

   prot = pp;
   prtab = pp->prot;
   if (fd < 0)
    {
    int  cfd = -fd - 1;

    FD_SET(cfd, &fds);
    if (cfd > fdmax)
     fdmax = cfd;
    sa[cfd] = prot->dst;
    }
   else
    {
    sfd = fd;
    firsttime = 1;
     }

   setup_qos = my_setup_qos;

   log_setup (NULL, -1);

   }


/* Suspend until the (absolute) time specified by waketime, processing */
/* any incoming packets or new connections that occur in that interval. */

void
stream_sleep_till(waketime)

 struct timeval *waketime;

 {

 (void)stream_get_packets(-1, waketime);
 justrcvd = 1;
 }
```

```
/* Tear down connection.      */

int
stream_teardown(asn)

 long  asn;

 {
 int  i;
 int  fd = asn;
 int  result;
 int  tmperrno;

 if (fd < 0)
  {

  /* Can't tear it down if it is already torn down! */

  errno = EINVAL;
  return (-1);
  }

 result = close(fd);
 log_teardown(NULL, result == 0 ? -1 : errno);

 if (fd == sfd)
  {

  /* Preserve errno from original close().  */

  tmperrno = errno;

  /* Clear out sfd in order to force initialization if */
  /* it is later re-opened.    */

  sfd = -1;

  /* This is the server socket, so make sure to also kill */
  /* current connections to any clients.   */

  for (i = 0; i <= fdmax; i++)
   {
   if (FD_ISSET(i, &fds))
    (void)close(i);
   }

  /* Restore errno from original close().   */

  errno = tmperrno;
  }

 return (result);
 }
```

```
/*****************************************************************************
 *                                                                           *
 * File: prot_tcp.c                                                          *
 *                                                                           *
 * Routines for interfacing to the TCP protocol suite via the                *
 * normal user-level interface.                                              *
 *                                                                           *
 * Written 20-Jun-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header: /tmp_mnt/net/usr.projecte/dartnet/src/tg/sun4/
RCS/prot_tcp.c,v 1.7 90/11/26 12:29:45 dlee Exp Locker: denny $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <sys/time.h>
#include <math.h>
#include "log.h"
#include "config.h"
#include "distribution.h"
#include "protocol.h"

/* Type definitions local to this file.    */

/* Functions exported from this file.    */

/* Functions local to this file.    */

/* Variables exported from this file.    */

/* Variables local to this file.    */

static protocol_table *prtab = NULL; /* pointer to protocol table. */
static protocol *prot = NULL; /* pointer to protocol struct. */


/* Apply QOS parameters to an existing fd.    */
```

```
void tcp_qos(fd)
int fd;
{

 /* Handle QOS parameters, if any are set. */

 if ((prot->qos & QOS_RCVWIN) != 0)
 {
 /* Set up receive buffer size. */
 if (setsockopt(fd,SOL_SOCKET, SO_RCVBUF,&(prot->rcvwin),
 sizeof(prot->rcvwin)) == -1)
 {
 perror("setsockopt SO_RCVBUF failed");
 abort();
 }
 }
 if ((prot->qos & QOS_SNDWIN) != 0)
 {
 /* Set up send buffer size. */
 if (setsockopt(fd,SOL_SOCKET,SO_SNDBUF,&(prot->sndwin),
 sizeof(prot->sndwin)) == -1)
 {
  perror("setsockopt SO_SNDBUF failed");
  abort();
 }

 }
}



/* TCP connection setup function.     */

long
tcp_setup(proto)

 protocol *proto;

 {
 int  flags;
 int  sfd;  /* Socket file descriptor. */

 /* Save pointer to protocol table.    */

 prtab = proto->prot;

 /* Set up global pointer to protocol structure. */
 prot = proto;

 /* Create TCP socket.      */

 if ((sfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1)
```

```
 {
 return (-1);
 }

if (!(prot->qos & QOS_SERVER))
 {

 /* Handle client side. Connect up a socket and return */
 /* it to the caller.    */

 /* Establish a connection.    */

 if ((flags = fcntl(sfd, F_GETFL, 0)) == -1)
  {
  log_error(NULL, NULL, sfd, LOGERR_FCNTL);
  perror("fcntl F_GETFL");
  abort();
  }
 if (fcntl(sfd, F_SETFL, flags | FNDELAY) == -1)
  {
  (void)close(sfd);
  return(-1);
  }
 /* handle any QOS parameters */

 tcp_qos(sfd);

 if ((connect(sfd, &(prot->dst), sizeof(prot->dst)) < 0) &
  (errno != EINPROGRESS))
  {
  (void)close(sfd);
  return (-1);
  }

 /* Prevent slow connection-setup from killing us. */

 (void)signal(SIGPIPE, SIG_IGN);

 /* Tell the get_packets routine about the new client */
 /* socket.     */

 stream_setup(prot, -sfd - 1,tcp_qos);
 }
else
 {

 /* Handle server side. This will need to be modified */
 /* to allow the server to accept multiple connections. */

 /* Bind socket.     */

 if (bind(sfd, &(prot->src), sizeof(prot->src)) < 0)
```

```
 {
 (void)close(sfd);
 return (-1);
 }

/* Initialize socket to set no-delay mode.  */

if ((flags = fcntl(sfd, F_GETFL, 0)) == -1)
 {
 log_error(NULL, NULL, sfd, LOGERR_FCNTL);
 perror("fcntl F_GETFL");
 abort();
 }
if (fcntl(sfd, F_SETFL, flags | FNDELAY) == -1)
 {
 log_error(NULL, NULL, sfd, LOGERR_FCNTL);
 perror("fcntl F_SETFL FNDELAY");
 abort();
 }

/* Declare willingness to accept connections.  */

(void)listen(sfd, 3);

/* Tell the get_packets routine about the new server */
/* socket.     */

stream_setup(prot, sfd,tcp_qos);
}

return (sfd);
}
```

```
/*****************************************************************************
 *                                                                           *
 * File: prot_test.c                                                         *
 *                                                                           *
 * Test ''protocol'' that uses keyboard and screen to check out              *
 * timing and scheduling at a coarse-grained level.                          *
 *                                                                           *
 * Written 24-Aug-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header: /tmp_mnt/net/usr.projecte/dartnet/src/tg/sun4/
RCS/prot_test.c,v 1.6 90/11/26 12:29:47 dlee Exp Locker: denny $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>

#include "log.h"
#include "config.h"
#include "distribution.h"
#include "protocol.h"

/* Type definitions local to this file.    */

/* Functions exported from this file.    */

/* Functions local to this file.    */

/* Variables exported from this file.    */

/* Variables local to this file.    */

static protocol_table *prtab = NULL; /* pointer to protocol table. */
static int  sfd = -1;


/* Accept packets while waiting for the opportunity to write (if wfd */
/* specified) or for specified timeout, whichever comes first.  */
```

```
int
test_get_packets(prtab, wfd, endtout)

protocol_table *prtab;  /* PRotocol Table pointer. */
int  wfd;  /* FD for write, -1 if none. */
struct timeval *endtout; /* end of timeout period. */

{
static char *buf = NULL; /* pointer to receive buffer. */
int  pklen;  /* Input packet length.  */
static unsigned long pktid = 0; /* received packet id.  */
fd_set  rfds;  /* Read FDs for select.  */
struct timeval tv;
struct timeval *tvp;
fd_set  wfds;  /* Write FDs for select. */

/* If we are passed a NULL prtab, the caller is trying to tell */
/* us about a new server socket. But that is too bad, since */
/* we will ignore it in this test protocol.   */

if (prtab == NULL)
 {
 return (0);
 }

/* Calculate initial timeout. We force a select even if the */
/* timeout has already expired in order to prevent the CPU from */
/* starving the I/O.     */

if (endtout == NULL)
 {

 /* No timeout, so just use NULL pointer.  */

 tvp = NULL;
 }
else
 {

 /* Calculate timeout.     */

 if (gettimeofday(&tv, (struct timezone *)NULL) == -1)
  {
  log_error(NULL, NULL, -1, LOGERR_GETTIME);
  perror("test_get_packets: gettimeofday");
  abort();
  }
 if (timercmp(endtout, &tv, >))
  {
  timersub(endtout, &tv, &tv);
```

```
 /* Kludge around strange SunOS restrictions. */

 if (tv.tv_sec > 3600)
  tv.tv_sec = 3600;
 }
else
 {
 tv.tv_sec = 0;
 tv.tv_usec = 0;
 }
tvp = &tv;
}

/* Each pass through the following loop does one select call */
/* to check the state of the fds.    */

for (;;)
 {

 /* Set up for select: get fd bitmaps.   */

 FD_ZERO(&rfds);
 if (sfd >= 0)
  FD_SET(sfd, &rfds);
 FD_ZERO(&wfds);
 if (wfd >= 0)
  {
  FD_SET(wfd, &rfds);
  FD_SET(wfd, &wfds);
  }
 if (select(wfd > sfd ? wfd + 1 : sfd + 1,
   &rfds,
   &wfds,
   (fd_set *)NULL,
   tvp) < 0)
  {
  if (errno != EINTR)
   {
   log_error(NULL, NULL, -1, LOGERR_SELECT);
   perror("test_get_packets: select");
   abort();
   }
  }

 /* If stdin fd was not selected, ignore it.  */

 if ((sfd >= 0) &&
  FD_ISSET(sfd, &rfds))
  {

  /* Each pass through the following loop  */
  /* attempts to receive one segment.  */
```

```
for (;;)
 {

 /* Get a buffer if we do not already */
 /* have one on hand.   */

 if ((buf == NULL) &&
  ((buf =
  (*(prtab->buffer_get))(MAX_PKT_SIZE)) ==
  NULL))
  {
  log_error(NULL, NULL, -1, LOGERR_MEM);
  (void)fprintf(stderr,
    "%s %s\n",
    "test_get_packets: ",
    "out of memory!");
  abort();
  }

 /* Receive the segment, scream and die */
 /* if error.    */

 pklen = read(sfd, buf, MAX_PKT_SIZE);
 if ((pklen == 0) ||
  ((pklen < 0) &
  (errno != EWOULDBLOCK)))
  {

  /* Shut down and tell rcv if */
  /* EOF or error.  */

  /*&&&&*/if (pklen < 0)
   perror("read");
  sfd = -1;
  log_rx(NULL,
   NULL,
   sfd,
   0,
   0,
   pklen == 0 ? -1 : errno);
  if (prtab->rcv != NULL)
   (void)(*(prtab->rcv))(sfd,
      sfd,
      NULL,
      pklen,
      0);
  break;
  }
 else if (pklen > 0)
  {
```

```
    /* Pass packet to receiver. */

    log_rx(NULL,
     NULL,
     sfd,
     pktid,
     pklen,
     -1);
    if ((prtab->rcv != NULL) &&
     ((*(prtab->rcv))(sfd,
       sfd,
       buf,
       pklen,
       pktid++)))
     buf = NULL;
    }
   else
    {

    /* Nothing more to read right */
    /* now.    */

    break;
    }
   }
  }

/* Check for ability to write...   */

if ((wfd > 0) &&
 (FD_ISSET(wfd, &wfds)))
 return (1);

/* Calculate next timeout. If the timeout has expired, */
/* tell the caller the sad story.   */

if (endtout != NULL)
 {

 if (gettimeofday(&tv, (struct timezone *)NULL) == -1)
  {
  log_error(NULL, NULL, -1, LOGERR_GETTIME);
  perror("test_get_packets: gettimeofday");
  abort();
  }
 if (timercmp(endtout, &tv, >))
  {
  timersub(endtout, &tv, &tv);

  /* Kludge around strange SunOS  */
  /* restrictions.   */
```

```
    if (tv.tv_sec > 3600)
     tv.tv_sec = 3600;
    }
   else
    {
    errno = ETIME;
    return (0);
    }
   }

  }

 /* NOTREACHED */
 }


/* Test protocol connection setup function.    */

long
test_setup(prot)

 protocol *prot;

 {
 int  flags;

 /* Save pointer to protocol table.    */

 prtab = prot->prot;

 /* Dump out QOS info. (debug only)    */

#ifdef NOTDEF
 if ((prot->qos & QOS_AVG_BANDWIDTH) != 0)
  (void)printf("Average bandwidth..%g\n", prot->avg_bandwidth);
 if ((prot->qos & QOS_PEAK_BANDWIDTH) != 0)
  (void)printf("Peak bandwidth.....%g\n", prot->peak_bandwidth);
 if ((prot->qos & QOS_AVG_DELAY) != 0)
  (void)printf("Average delay......%g\n", prot->avg_delay);
 if ((prot->qos & QOS_PEAK_DELAY) != 0)
  (void)printf("Peak delay.........%g\n", prot->peak_delay);
 if ((prot->qos & QOS_AVG_LOSS) != 0)
  (void)printf("Average loss.......%g\n", prot->avg_loss);
 if ((prot->qos & QOS_PEAK_LOSS) != 0)
  (void)printf("Peak loss..........%g\n", prot->peak_loss);
 if ((prot->qos & QOS_INTERVAL) != 0)
  (void)printf("Interval...........%g\n", prot->interval);
 if ((prot->qos & QOS_MTU) != 0)
  (void)printf("MTU................%u\n", prot->mtu);
 (void)printf("flags:.............");
 if ((prot->qos & QOS_INTERACTIVE) != 0)
  (void)printf(" INTERACTIVE");
```

```
 if ((prot->qos & QOS_SERVER) != 0)
  (void)printf(" SERVER");
 (void)printf("\n");
#endif NOTDEF

 if (!(prot->qos & QOS_SERVER))
  {

  /* Handle client side.     */

  return (open("/dev/tty", O_RDWR));
  }
 else
  {

  /* Handle server side.     */

  sfd = open("/dev/tty", O_RDWR);

  /* Set no-delay mode.      */

  if ((flags = fcntl(sfd, F_GETFL, 0)) == -1)
   {
   log_error(NULL, NULL, -1, LOGERR_FCNTL);
   perror("fcntl F_GETFL");
   abort();
   }
  if (fcntl(sfd, F_SETFL, flags | FNDELAY) == -1)
   {
   (void)close(sfd);
   return(-1);
   }

  return (sfd);
  }
 }


/* Tear down connection.      */

int
test_teardown(asn)

 long  asn;

 {
 int  fd = asn;
 int  flags;

 if (fd < 0)
  {
```

```
 /* Can't tear it down if it is already torn down! */

 errno = EINVAL;
 return (-1);
 }

/* Set no-delay mode.     */

if ((flags = fcntl(fd, F_GETFL, 0)) == -1)
 {
 log_error(NULL, NULL, -1, LOGERR_FCNTL);
 perror("fcntl F_GETFL");
 abort();
 }
if (fcntl(fd, F_SETFL, flags & ~FNDELAY) == -1)
 {
 (void)close(fd);
 return(-1);
 }
return (0);
}


/* Send a packet, subject to the timeout.    */

int
test_send(asn, buf, len, endtout, pktid)

 long  asn;
 char  *buf;
 int  len;
 struct timeval *endtout;
 unsigned long *pktid;

 {
 int  cc;
 int  fd = asn;
 static unsigned long id = 0;
 char  s[30];

 /* Make sure that fd wasn't closed out from under the sender. */

 if (fd < 0)
  {
  errno = EINVAL;
  return (-1);
  }

 /* Invoke the get_packets routine to receive packets and accept */
 /* new connections while we are waiting to write.  */

 if (!test_get_packets(prtab, fd, endtout))
```

```
    {
    return (-1);
    }
   else
    {

    /* Write out the packet!    */

    (void)sprintf(s, "%d,", len);
    cc = write(fd, s, strlen(s));
    log_tx(NULL,
     NULL,
     NULL,
     fd,
     id,
     len,
     cc >= 0 ? -1 : errno);
    if (cc >= 0)
     *pktid = id++;
    buffer_generic_free(buf);
    return (cc);
    }
   }


/* Suspend until the (absolute) time specified by waketime, processing */
/* any incoming packets or new connections that occur in that interval. */

void
test_sleep_till(waketime)

 struct timeval *waketime;

 {

 (void)test_get_packets(prtab, -1, waketime);
 }
```

```
/*****************************************************************************
 *                                                                           *
 * File: prot_udp.c                                                          *
 *                                                                           *
 * Routines for interfacing to the TCP protocol suite via the                *
 * normal user-level interface.                                              *
 *                                                                           *
 * Written 04-Sep-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/src/tg/RCS/
prot_udp.c,v 1.5 90/11/26 12:29:49 dlee Exp Locker: denny $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/file.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <sys/time.h>
#include <math.h>
#include "log.h"
#include "config.h"
#include "distribution.h"
#include "protocol.h"

/* Type definitions local to this file.    */

/* Functions exported from this file.    */

/* Functions local to this file.    */

/* Variables exported from this file.    */

/* Variables local to this file.    */

static protocol_table *prtab = NULL; /* pointer to protocol table. */


/* Apply QOS parameters to an existing fd. */

static void udp_qos(fd)
```

```
int fd;

{
 /* Handle QOS parameters here if UDP ever gets any. */

}


/* TCP connection setup function.    */

long
udp_setup(prot)

 protocol *prot;

 {
 int  flags;
 int  sfd;  /* Socket file descriptor. */

 /* Save pointer to protocol table.    */

 prtab = prot->prot;

 /* Create TCP socket.     */

 if ((sfd = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
  {
  return (-1);
  }

 if (!(prot->qos & QOS_SERVER))
  {

  /* Handle client side. Connect up a socket and return */
  /* it to the caller.    */

  /* Establish a connection.    */

  if ((flags = fcntl(sfd, F_GETFL, 0)) == -1)
   {
   log_error(NULL, NULL, -1, LOGERR_FCNTL);
   perror("fcntl F_GETFL");
   abort();
   }
  if (fcntl(sfd, F_SETFL, flags | FNDELAY) == -1)
   {
   (void)close(sfd);
   return(-1);
   }
 /* Handle any QOS */
  udp_qos(sfd);
```

```
 if (prot->qos & QOS_SRC)
  {
  if (bind(sfd, &(prot->src), sizeof(prot->src)) < 0)
   {
   (void)close(sfd);
   return (-1);
   }
  }
 if ((connect(sfd, &(prot->dst), sizeof(prot->dst)) < 0) &
  (errno != EINPROGRESS))
  {
  (void)close(sfd);
  return (-1);
  }

 /* Prevent slow connection-setup from killing us. */

 (void)signal(SIGPIPE, SIG_IGN);


 /* Tell the get_packets routine about the new client */
 /* socket.     */

 dgram_setup(prot, -1);
 }
else
 {

 /* Handle server side. This will need to be modified */
 /* to allow the server to accept multiple connections. */

 /* Bind socket.     */

 if (bind(sfd, &(prot->src), sizeof(prot->src)) < 0)
  {
  (void)close(sfd);
  return (-1);
  }

 /* Initialize socket to set no-delay mode.  */

 if ((flags = fcntl(sfd, F_GETFL, 0)) == -1)
  {
  log_error(NULL, NULL, -1, LOGERR_FCNTL);
  perror("fcntl F_GETFL");
  abort();
  }
 if (fcntl(sfd, F_SETFL, flags | FNDELAY) == -1)
  {
  log_error(NULL, NULL, -1, LOGERR_FCNTL);
  perror("fcntl F_SETFL FNDELAY");
```

```
        abort();
        }

    /* Declare willingness to accept connections.  */

    (void)listen(sfd, 3);

    /* Tell the get_packets routine about the new server */
    /* socket.      */

    dgram_setup(prot, sfd);
    }


/* Pass the association ID back to the caller.   */

return (sfd);
}
```

```
/*****************************************************************************
 *                                                                           *
 * File: protocol.c                                                          *
 *                                                                           *
 * Contains protocol table and functions that search this table.             *
 *                                                                           *
 * Written 20-Jun-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/src/tg/RCS/
protocol.c,v 1.7 90/11/26 12:29:52 dlee Exp $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <math.h>
#include "config.h"
#include "distribution.h"
#include "protocol.h"
#include "st2_api.h"

extern int  stream_teardown();
extern int  stream_send();
extern void  stream_sleep_till();

extern long  tcp_setup();

extern int  dgram_teardown();
extern int  dgram_send();
extern void  dgram_sleep_till();

extern long  udp_setup();

extern int  st2_teardown();
extern int  st2_send();
extern void  st2_sleep_till();

extern long  straw_setup();


extern long  test_setup();
extern int  test_teardown();
extern int  test_send();
extern void  test_sleep_till();
```

```
extern char  *buffer_dgram_get();
extern void  buffer_dgram_free();

extern char  *buffer_generic_get();
extern void  buffer_generic_free();

extern int  ipport_atoaddr();
extern int  ipport_addrtoa();
extern char  *ipport_btoaddr();
extern char  *ipport_addrtob();

/* Type definitions local to this file.    */

/* Functions exported from this file.    */

/* Functions local to this file.    */

/* Variables exported from this file.    */

/* Variables local to this file.    */

/* NOTE: The protocol table must be kept in alphabetical order. */

static protocol_table ptab[] =
    {
     {
     "straw", AF_COIP,
      straw_setup, st2_teardown,
      NULL, st2_send,
      st2_sleep_till,
      buffer_generic_get,
      buffer_generic_free,
      ipport_atoaddr,
      ipport_addrtoa,
      ipport_btoaddr,
      ipport_addrtob,
     },
     {
     "tcp", AF_INET,
      tcp_setup, stream_teardown,
      NULL, stream_send,
      stream_sleep_till,
      buffer_generic_get,
      buffer_generic_free,
      ipport_atoaddr,
      ipport_addrtoa,
      ipport_btoaddr,
      ipport_addrtob,
     },
     {
     "test", AF_INET,
      test_setup, test_teardown,
```

```
     NULL, test_send,
     test_sleep_till,
     buffer_generic_get,
     buffer_generic_free,
     ipport_atoaddr,
     ipport_addrtoa,
     ipport_btoaddr,
     ipport_addrtob,
    },
    {
    "udp", AF_INET,
     udp_setup, dgram_teardown,
     NULL, dgram_send,
     dgram_sleep_till,
     buffer_dgram_get,
     buffer_dgram_free,
     ipport_atoaddr,
     ipport_addrtoa,
     ipport_btoaddr,
     ipport_addrtob,
    },
    {
    NULL, AF_UNSPEC,
     NULL, NULL,
     NULL, NULL,
     NULL,
     NULL,
     NULL,
     NULL,
     NULL,
     NULL,
     NULL,
    },
   };


/* search ptab for the specified protocol, return a pointer to the */
/* entry or NULL if no match.     */

protocol_table *
find_protocol(name)

 char  *name;

 {
 int  cmp;
 protocol_table *p;

 for (p = ptab; p->name != NULL; p++)
  {
  cmp = strcmp(name, p->name);
```

```
 if (cmp == 0)
  return (p);
 if (cmp < 0)
  return (NULL);
 }

return (NULL);
}
```

```
%{
/*****************************************************************************
 *                                                                           *
 * File: scan.l                                                              *
 *                                                                           *
 * lexical analyser for CATE traffic generator.                              *
 * Hacked from extractdoc lexer, in turn hacked from an ANSI C               *
 * lexer.                                                                    *
 *                                                                           *
 * Written 18-Jun-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 SRI International.                                     *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char scan_l_rcsid[] = "$Header: /tmp_mnt/net/sfo.a/dlee/workspace/tg/
src/tg/RCS/scan.l,v 1.4 92/07/29 19:26:31 dlee Exp Locker: dlee $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <strings.h>
#include "y.tab.h"

/* Type definitions local to this file.     */

#undef input
#define input() \
 (((yytchar = \
  yysptr > yysbuf ? \
  U(*--yysptr) : \
  (curarg < 0 ? \
   getc(yyin) : \
   ((yytchar = *(curchar++)) != 0 ? \
    yytchar : \
    (gargv[++curarg] != NULL) ? \
     curchar = &(gargv[curarg][0]), \
     '\n' : \
     (strcpy(filename, "<stdin>"), \
      curarg = -1, \
      lineno = 1, \
      '\n' \
     ) \
    ) \
   ) \
  ) == 10 ? \
  (yylineno++, yytchar) : \
  yytchar \
  ) == EOF ? \
  0 : \
  yytchar \
```

```
 )

/* Functions exported from this file.     */

/* Functions local to this file.     */

static void  count(); /* COUNT columns for error use. */

/* Variables exported from this file.     */

/* Variables local to this file.     */

static int  column = 0; /* input COLUMN number.  */
static int  curarg;  /* arglist arg pointer.  */
static char  *curchar; /* arglist char pointer. */
static char  **gargv; /* global version of argv. */
static int  lineno = 1; /* input LINE NOmber.  */
static char  filename[BUFSIZ] = "<arglist>";

%}


 /* Definitions.      */

%a 8000
%e 2000
%n 1000
%o 20000
%p 10000

O  [0-7]
D  [0-9]
L  [a-zA-Z_]
H  [a-fA-F0-9]
E  [Ee][+-]?{D}+
W  [ \t\v\f]
WW  {W}*
ID  {L}({L}|{D})*

%START LEX_NORMAL LEX_ADDRESS

%%

"include"  { count(); BEGIN LEX_NORMAL; return(INCLUDE); }
"="   { count(); BEGIN LEX_NORMAL; return(EQUAL); }

"server"  { count(); BEGIN LEX_NORMAL; return(SERVER); }

"average"  { count(); BEGIN LEX_NORMAL; return(AVERAGE); }
"bandwidth"  { count(); BEGIN LEX_NORMAL; return(BANDWIDTH); }
"delay"   { count(); BEGIN LEX_NORMAL; return(DELAY); }
"interactive"  { count(); BEGIN LEX_NORMAL; return(INTERACTIVE); }
```

```
"interval"  { count(); BEGIN LEX_NORMAL; return(INTERVAL); }
"loss"   { count(); BEGIN LEX_NORMAL; return(LOSS); }
"MTU"   { count(); BEGIN LEX_NORMAL; return(MTU); }
"mtu"   { count(); BEGIN LEX_NORMAL; return(MTU); }
"peak"   { count(); BEGIN LEX_NORMAL; return(PEAK); }
"rcvwin" { count(); BEGIN LEX_NORMAL; return(RCVWIN); }
"sndwin" { count(); BEGIN LEX_NORMAL; return(SNDWIN); }

"constant"  { count(); BEGIN LEX_NORMAL; return(DIST_CONST); }
"exponential"  { count(); BEGIN LEX_NORMAL; return(DIST_EXP); }
"exp"   { count(); BEGIN LEX_NORMAL; return(DIST_EXP); }
"markov"  { count(); BEGIN LEX_NORMAL; return(DIST_MARKOV); }
"markov2"  { count(); BEGIN LEX_NORMAL; return(DIST_MARKOV2); }
"uniform"  { count(); BEGIN LEX_NORMAL; return(DIST_UNIFORM); }

"arrival"  { count(); BEGIN LEX_NORMAL; return(ARRIVAL); }
"at"   { count(); BEGIN LEX_NORMAL; return(AT); }
"data"   { count(); BEGIN LEX_NORMAL; return(DATA); }
"length"  { count(); BEGIN LEX_NORMAL; return(LENGTH); }
"on"   { count(); BEGIN LEX_NORMAL; return(ON); }
"patience"  { count(); BEGIN LEX_NORMAL; return(PATIENCE); }
"responselength" { count(); BEGIN LEX_NORMAL; return(RESPONSELENGTH); }
"resplen"  { count(); BEGIN LEX_NORMAL; return(RESPONSELENGTH); }
"seed"   { count(); BEGIN LEX_NORMAL; return(SEED); }
"setup"   { count(); BEGIN LEX_NORMAL; return(SETUP); }
"time"   { count(); BEGIN LEX_NORMAL; return(TIME); }
"wait"   { count(); BEGIN LEX_NORMAL; return(WAIT); }

"packet"  { count(); BEGIN LEX_NORMAL; return(PACKET); }
"reset"   { count(); BEGIN LEX_NORMAL; return(RESET); }

<LEX_NORMAL>":"  { count(); return(':'); }

<LEX_NORMAL>{ID} {
   count();
   if ((yylval.prot.prot = find_protocol(yytext)) != NULL)
    return(PROTOCOL);
   else
    {
    SYMcpy(yylval.n, yytext);
    return(IDENTIFIER);
    }
   }


<LEX_NORMAL>"/"({ID}("/"{ID})*)?|{ID}("/"{ID})+ {

   /* Note that filenames must contain at least */
   /* one slash.     */

   count();
   SYMcpy(yylval.n, yytext);
```

```
    return(FILENAME);
    }

<LEX_NORMAL>0[xX]{H}+ {
    unsigned long tmp;

    count();
    SYMcpy(yylval.n, &yytext[2]); /* skip past 0[xX] */
    (void)sscanf(yylval.n, "%lx", &tmp);
    yylval.d = tmp;
    return(HEX_INTEGER);
    }
<LEX_NORMAL>0+{O}{O}+ {
    unsigned long tmp;

    count();
    SYMcpy(yylval.n, &yytext[1]); /* skip initial 0*/
    (void)sscanf(yytext, "%lo", &tmp);
    yylval.d = tmp;
    return(OCTAL_INTEGER);
    }
<LEX_NORMAL>0+{O} {
    unsigned long tmp;

    count();
    SYMcpy(yylval.n, yytext);
    (void)sscanf(yytext, "%lu", &tmp);
    yylval.d = tmp;
    return(SMALL_INTEGER);
    }
<LEX_NORMAL>{D}+ {
    unsigned long tmp;

    count();
    SYMcpy(yylval.n, yytext);
    (void)sscanf(yytext, "%lu", &tmp);
    yylval.d = tmp;
    return(INTEGER);
    }
<LEX_NORMAL>'(\\.|[^\\'])*' {
    char  *qp;

    count();

    /* Strip quotes from character constant. */

    qp = &yytext[strlen(yytext) - 1];
    if (*qp == '\'')
     *qp = '\0';
    SYMcpy(yylval.n, yytext + 1);
    return(STRING_LITERAL);
    }
```

```
<LEX_NORMAL>{D}+{E} {
   count();
   SYMcpy(yylval.n, yytext);
   (void)sscanf(yytext, "%lf", &yylval.d);
   return(FLOATING_POINT);
   }
<LEX_NORMAL>{D}*"."{D}+({E})? {
   count();
   SYMcpy(yylval.n, yytext);
   (void)sscanf(yytext, "%lf", &yylval.d);
   return(FLOATING_POINT);
   }
<LEX_NORMAL>{D}+"."{D}*({E})? {
   count();
   SYMcpy(yylval.n, yytext);
   (void)sscanf(yytext, "%lf", &yylval.d);
   return(FLOATING_POINT);
   }

<LEX_NORMAL>\"(\\.|[^\\"])*\" {
   char  *qp;

   count();

   /* Strip quotes from string.   */

   qp = &yytext[strlen(yytext) - 1];
   if (*qp == '"')
    *qp = '\0';
   SYMcpy(yylval.n, yytext + 1);
   return(STRING_LITERAL);
   }
<LEX_NORMAL>\"[^"\n]* {
   count();
   yyerror("Unterminated string");
   exit(-1);
   }

<LEX_NORMAL>[ \t\n\014] { count(); }
<LEX_NORMAL>.  {
   count();
   yyerror("Illegal character");
   exit(-1);
   }

<LEX_NORMAL>^#.*$ {
   /* Comment lines...    */
   count();
   }

<LEX_ADDRESS>[ \t\n\014] { count(); }
```

```
<LEX_ADDRESS>[\41-\176]* {
   count();
   if ((*(lexprot.prot->atoaddr))(yytext,
       &(yylval.tmpaddr)))
    return (ADDR);
   else
    {
    BEGIN LEX_NORMAL;
    REJECT;
    }
   }
%%

/* File record structure used for holding file include state */
struct file_entry {
 FILE  *fd;
 char  name[MAXSYM];
 int  cnt;
};

#define MAX_INCLUDE_LEVEL 10
struct file_entry file_tbl[MAX_INCLUDE_LEVEL];
char   current_config_file[MAXSYM];
int   include_level = 0; /* Current inclusion depth */

yywrap()
{
 /*
 * If we are currently in an included file, don't exit, but pop back
 * to the calling file.
 */
 if (include_level > 0) {
 include_level--;
 yyin = file_tbl[include_level].fd;
 yylineno = file_tbl[include_level].cnt;
 strcpy(current_config_file, file_tbl[include_level].name);
 return(0);
 } else {
 return(1);
 }
}


static void
count()
{
 int i;

 for (i = 0; yytext[i] != '\0'; i++)
  if (yytext[i] == '\n')
   {
```

```
    column = 0;
    lineno++;
    }
   else if (yytext[i] == '\t')
    column += 8 - (column % 8);
   else
    column++;
}


/* Lex initialization.      */

lex_init(argc, argv)

 int  argc;
 char  *argv[];
{
 int  c;
 int  errflg = NULL;

 extern char *optarg, *ifile, *ofile;
 extern int optind;

 gargv = argv;

 /*
  * We should use LEX to parse the command line.
  * For now, we use the standard getopt() call to parse
  * command line options.
 */
 while ((c = getopt(argc, argv, "fi:o:")) != -1) {
  switch (c) {
   case 'i':
   if (ifile) {
    errflg++;
    break;
   }
   ifile = optarg;
   break;
   case 'f':
   FlushOutput = 1;
   break;
 case 'o':
   if (ofile) {
    errflg++;
    break;
   }
   ofile = optarg;
 break;
   case '?':
 errflg++;
 default:
```

```
 errflg++;
 fprintf (stderr, "lex_init: unknown option [%s]\n",
   argv[optind]);
 }
 }
 if (errflg) {
  fprintf (stderr, "lex_init: command line error\n");
  exit (-1);
 }

 if(ifile) {
  if (freopen (ifile, "r", stdin) == (FILE *) NULL) {
   exit (-1);
  }
 }
#ifdef stuff
 if (argc < 2) {
  curarg = -1;
 } else {
  curarg = 1;
  curchar = &(gargv[curarg][0]);
 }
#endif stuff
 curarg = -1;
}
```

```
%{
/*****************************************************************************
 *                                                                           *
 * File: tg.y                                                                *
 *                                                                           *
 * Traffic Generation command grammar.                                       *
 *                                                                           *
 * Written 19-Jun-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1989 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header: /tmp_mnt/net/sfo.a/dlee/workspace/tg/src/tg/
RCS/tg.y,v 1.4 92/07/29 15:07:38 dlee Exp Locker: dlee $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <search.h>
#include <malloc.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <math.h>
#include "config.h"
#include "distribution.h"
#include "protocol.h"
#include "log.h"

/* Type definitions local to this file.    */

#define MAXSYM  512

#define SYMcpy(a, b) { \
   (void)strncpy((a), (b), MAXSYM); \
   (a)[MAXSYM - 1] = '\0'; \
   }

#define SYMcat(a, b) { \
   (void)strncat((a), (b), MAXSYM - strlen(a) - 1); \
   }

typedef enum srvr_state_flag_  /* server packet parse state. */
 {
 srvr_len,   /* Accumulate reply length. */
 srvr_skip,   /* Accumulate skip distance. */
 srvr_skipping   /* Skip to next reply length. */
 } srvr_state_flag;
```

```
typedef struct srvr_state_  /* per-asn state variables. */
 {
 struct srvr_state_ *next;
 long  asn;  /* asn that this state is for. */
 int  bad;  /* asn has corrupted data. */
 srvr_state_flag state;  /* Current state.  */
 unsigned long acc;  /* accumulate compressed number.*/
 int  nbytes;  /* # bytes accumulated so far. */
 int  special; /* acc contains special number. */
 unsigned long skip;  /* how many bytes to skip. */
 } srvr_state;

typedef struct    /* Yacc Symbol table entry TYPE.*/
 {
 int  flags;  /* FLAGS, same as symbol below. */
 int  lineno;  /* Line number for construct. */
 double  d;  /* token numeric value.  */
 char  n[MAXSYM]; /* token name.   */
 distribution tmpdist; /* Distribution when we aren't */
    /* sure which one we have. */
 struct sockaddr tmpaddr; /* Address when we aren't sure */
    /* which one we have.  */
 tg_action action;  /* Action descriptor.  */
 protocol prot;  /* Protocol descriptor.  */
 }  strbuf;

#define YYSTYPE strbuf

/* Each entry of the expected-replies queue represents a packet that */
/* is expected to be received from a server. If a given packet is */
/* not received by its timeout, we scream and die. Packets are  */
/* identified by the sequence number of the first byte following them, */
/* the sequence number of the first byte of the first packet is zero. */

typedef struct xpctd_replies_  /* expected-replies queue. */
 {
 struct xpctd_replies_ *next;
 unsigned long byte_seqno; /* seqno of 1st byte after */
    /* packet represented by this */
    /* struct.   */
 struct timeval timeout; /* time at which patience gives */
    /* out.    */
 } xpctd_replies;

/* Functions exported from this file.     */

extern void  yyerror();

/* Functions local to this file.     */

extern void  fix_times();
```

```
extern void  generate();
extern void  generate_interactive();
extern int  check_deadline();
extern void  do_actions();
extern void  node_init();
extern int  rcv_pkt();
extern int  rcv_pkt_interactive();
extern int  rcv_pkt_interactive_srvr();
extern srvr_state *srvr_state_get();
extern void  tg_append_element();
extern void  wait_start();
extern xpctd_replies *xpctd_replies_get();
extern void  xpctd_replies_free();
extern void  yyerror();

/* Variables exported from this file.     */

double  global_start; /* global start time.  */
struct timeval  global_start_tv; /* global start timeval. */
protocol  lexprot; /* lex's protocol definition. */
protocol  prot;   /* Protocol definition. */
tg_action  *tg_first = NULL; /* first TG action. */
tg_action  **tg_last = &tg_first; /* last TG action. */
char *version = "1.0"; /* TG program version. */
char   *ofile = NULL;
char   *ifile = NULL;
int   FlushOutput = 0; /* whether to flush after each


/* Variables local to this file.     */

static int  got_errors = 0; /* Got errors during parse? */
static int  got_setup = 0; /* Got a setup clause?  */
static int  got_setup_implicit = 0; /* an implicit one? */
static srvr_state *srvr_state_h = NULL; /* per-asn srvr state list*/
static xpctd_replies *xpctd_replies_flist = NULL;
    /* xr freelist.   */
static xpctd_replies *xpctd_replies_qh = NULL; /* xr queue header. */
static xpctd_replies **xpctd_replies_qt = &xpctd_replies_qh;
    /* xr queue tail-pointer. */

/* Symbol table */
struct sym_entry {
 char  *sym_name;
 int  sym_type;
 union {
  int  intval;
  char *strval;
  float fltval;
  double dblval;
 } sym;
```

```
#define sym_intval sym.intval
#define sym_strval sym.strval
#define sym_fltval sym.fltval
#define sym_dblval sym.dblval

};

enum {
 SYM_INT_TYPE = 1,
 SYM_STR_TYPE,
 SYM_FLT_TYPE,
 SYM_DBL_TYPE,
};

#define SYM_TBL_SIZE 100

struct sym_entry sym_tbl[SYM_TBL_SIZE];

int sym_tbl_index = 0;

%}

/* Lex token definitions.      */

%token INCLUDE EQUAL

%token DIST_CONST DIST_EXP DIST_MARKOV DIST_MARKOV2 DIST_UNIFORM

%token ARRIVAL AT AVERAGE BANDWIDTH DATA DELAY INTERACTIVE INTERVAL
%token LENGTH LOSS MTU ON PATIENCE PEAK RCVWIN RESPONSELENGTH SEED SERVER
%token SETUP SNDWIN TIME WAIT

%token ADDR FILENAME FLOATING_POINT HEX_INTEGER IDENTIFIER INTEGER
%token OCTAL_INTEGER PROTOCOL SMALL_INTEGER STRING_LITERAL

%token ADDRESS

%token PACKET RESET

/* Starting state.       */

%start statements



%%

statements
 : statement
 | statements statement
 ;
```

```
statement
 : include
 | macro
 | commands
 ;

include
 : INCLUDE STRING_LITERAL
  {
  FILE *fd;

  fprintf(stderr, "including \"%s\"\n", $2.n);

  if (include_level > MAX_INCLUDE_LEVEL) {
   fprintf(stderr, "too many levels of inclusion\n");
   exit(-1);
  }
  if ((fd = fopen($2.n, "r")) == NULL) {
   fprintf(stderr, "can't open include file \"%s\"\n",
    $2.n);
   exit(-1);
  }
  file_tbl[include_level].fd = yyin;
  file_tbl[include_level].cnt = yylineno;
  strcpy(file_tbl[include_level].name, current_config_file);

  yyin = fd;
  yylineno = 0;
  strcpy(current_config_file, $2.n);
  include_level++;
  }
 ;

macro
 : IDENTIFIER EQUAL integer
  {
  extern char *malloc();
  char  *cp;

  if (!(cp = malloc (strlen($1.n) + 1))) {
   perror("malloc");
   exit (-1);
  }
  strcpy (cp, $1.n);
  sym_tbl[sym_tbl_index].sym_name = cp;
  sym_tbl[sym_tbl_index].sym_type = SYM_INT_TYPE;
  sym_tbl[sym_tbl_index].sym_intval = (int) $3.d;

#ifdef DEBUG
  fprintf(stderr, "sym_tbl_index [%d] macro [%s], value [%d]\n",
   sym_tbl_index, cp, (int) $3.d);
#endif
```

```
    sym_tbl_index++;
    }
  ;

commands
  : start_time association_spec tg_entry_list
    {
    prot = $2.prot;
    }
  ;

start_time
  : ON time_literal
    {
    struct timeval tp;
    unsigned long modulus = $2.d;

    if (modulus != $2.d)
      {
      yyerror("start_time: ON value must be integral");
      }
    if (gettimeofday(&tp, (struct timezone *)NULL) == -1)
      {
      perror("gettimeofday");
      exit(-1);
      }
    global_start = ((tp.tv_sec + modulus - 1) / modulus) * modulus;
    global_start_tv.tv_sec = global_start;
    global_start_tv.tv_usec = 0;
    }
  ;

association_spec
  : protocol_and_addresses quality_of_service
    {
    $$ = $2;
    $$.prot.qos |= $1.prot.qos;
    $$.prot.src = $1.prot.src;
    $$.prot.dst = $1.prot.dst;
    $$.prot.prot = $1.prot.prot;
    }
  ;

protocol_and_addresses
  : protocol ADDR
    {
    $$ = $1;
    $$.prot.dst = $2.tmpaddr;
    $$.prot.qos = QOS_DST;
    }
  | protocol ADDR SERVER
    {
```

```
  $$ = $1;
  $$.prot.src = $2.tmpaddr;
  $$.prot.qos = QOS_SRC | QOS_SERVER;
  }
 | protocol ADDR ADDR
  {
  $$ = $1;
  $$.prot.src = $2.tmpaddr;
  $$.prot.dst = $3.tmpaddr;
  $$.prot.qos = QOS_SRC | QOS_DST;
  }
 ;

protocol
 : PROTOCOL
  {
  node_init(&$$);
  $$.prot.prot = $1.prot.prot;
  lexprot = $$.prot;
  BEGIN LEX_ADDRESS;
  }
 ;

quality_of_service
 :
  {
  node_init(&$$);
  }
 | quality_of_service AVERAGE BANDWIDTH number
  {
  $$ = $1;
  $$.prot.avg_bandwidth = $4.d;
  $$.prot.qos |= QOS_AVG_BANDWIDTH;
  }
 | quality_of_service PEAK BANDWIDTH number
  {
  $$ = $1;
  $$.prot.peak_bandwidth = $4.d;
  $$.prot.qos |= QOS_PEAK_BANDWIDTH;
  }
 | quality_of_service AVERAGE DELAY number
  {
  $$ = $1;
  $$.prot.avg_delay = $4.d;
  $$.prot.qos |= QOS_AVG_DELAY;
  }
 | quality_of_service PEAK DELAY number
  {
  $$ = $1;
  $$.prot.peak_delay = $4.d;
  $$.prot.qos |= QOS_PEAK_DELAY;
  }
```

```
 | quality_of_service AVERAGE LOSS number
  {
  $$ = $1;
  $$.prot.avg_loss = $4.d;
  $$.prot.qos |= QOS_AVG_LOSS;
  }
 | quality_of_service PEAK LOSS number
  {
  $$ = $1;
  $$.prot.peak_loss = $4.d;
  $$.prot.qos |= QOS_PEAK_LOSS;
  }
 | quality_of_service RCVWIN integer
  {
  $$ = $1;
  $$.prot.rcvwin = $3.d;
  $$.prot.qos |= QOS_RCVWIN;
  }
 | quality_of_service SNDWIN integer
  {
  $$ = $1;
  $$.prot.sndwin = $3.d;
  $$.prot.qos |= QOS_SNDWIN;
  }
 | quality_of_service INTERACTIVE
  {
  $$ = $1;
  $$.prot.qos |= QOS_INTERACTIVE;
  }
 | quality_of_service INTERVAL number
  {
  $$ = $1;
  $$.prot.interval = $3.d;
  $$.prot.qos |= QOS_INTERVAL;
  }
 | quality_of_service MTU integer
  {
  $$ = $1;
  $$.prot.mtu = $3.d;
  $$.prot.qos |= QOS_MTU;
  }
 ;

tg_entry_list
 :
 | tg_entry_list tg_entry
  {

  /* Add the new entry to the linked list. Note that */
  /* the chain represented by tg_entry_list has already */
  /* been added to the list, and need not be referred to, */
  /* thus there is no need for node_init in the above */
```

```
 /* empty clause.     */

 *tg_last = (tg_action *)malloc(sizeof(tg_action));
 if (*tg_last == NULL)
  {
  yyerror("tg_entry_list: Out of memory");
  }
 **tg_last = $2.action;
 tg_last = &((*tg_last)->next);
 }
;

tg_entry
 : at_clause tg_action
  {
  $$ = $2;
  $$.action.tg_flags |= $1.action.tg_flags;
  $$.action.start_at = $1.action.start_at;
  }
 | tg_action
  {
  $$ = $1;
  }
 ;

at_clause
 : AT time_literal
  {
  double  tmp;

  node_init(&$$);
  tmp = global_start + $2.d;
  dtotimeval(tmp, &$$.action.start_at);
  $$.action.tg_flags |= TG_START;
  }
 ;

tg_action
 : tg_action_setup
  {
  $$ = $1;
  }
 | tg_action_wait
  {
  $$ = $1;
  }
 | tg_action_arrival tg_action_length tg_action_modifier_list
  {
  $$ = $1;
  tg_append_element(&$$.action, &$2.action);
  tg_append_element(&$$.action, &$3.action);
  }
```

```
 | tg_action_arrival tg_action_length tg_action_resplen
  tg_action_modifier_list
  {
  $$ = $1;
  tg_append_element(&$$.action, &$2.action);
  tg_append_element(&$$.action, &$3.action);
  tg_append_element(&$$.action, &$4.action);
  }
 | tg_action_arrival tg_action_length tg_action_resplen
  tg_action_patience tg_action_modifier_list
  {
  $$ = $1;
  tg_append_element(&$$.action, &$2.action);
  tg_append_element(&$$.action, &$3.action);
  tg_append_element(&$$.action, &$4.action);
  tg_append_element(&$$.action, &$5.action);
  }
 ;

tg_action_setup
 : SETUP
  {
  node_init(&$$);
  if (got_setup)
   {
   yyerror("Multiple setup clauses not supported");
   YYERROR;
   }
  if (got_setup_implicit)
   {
   yyerror("setup clause not legal after implicit setup");
   YYERROR;
   }
  got_setup = 1;
  $$.action.tg_flags |= TG_SETUP;
  }
 ;

tg_action_wait
 : WAIT
  {
  node_init(&$$);
  if (!got_setup)
   got_setup_implicit = 1;
  $$.action.tg_flags |= TG_WAIT;
  }
 | WAIT time_literal
  {
  node_init(&$$);
  if (!got_setup)
   got_setup_implicit = 1;
  $$.action.tg_flags |= TG_WAIT | TG_TIME;
```

```
 dtotimeval($2.d, &$$.action.time_limit);
 }
;

tg_action_arrival
 : ARRIVAL distribution
 {
 node_init(&$$);
 if (!got_setup)
  got_setup_implicit = 1;
 $$.action.tg_flags |= TG_ARRIVAL;
 $$.action.arrival = $2.tmpdist;
 }
;

tg_action_length
 : LENGTH distribution
 {
 node_init(&$$);
 $$.action.tg_flags |= TG_LENGTH;
 $$.action.length = $2.tmpdist;
 }
;

tg_action_resplen
 : RESPONSELENGTH distribution
 {
 node_init(&$$);
 $$.action.tg_flags |= TG_RESPLEN;
 $$.action.resplen = $2.tmpdist;
 }

tg_action_patience
 : PATIENCE time_literal
 {
 node_init(&$$);
 $$.action.tg_flags |= TG_PATIENCE;
 dtotimeval($2.d, &$$.action.patience);
 }

tg_action_modifier_list
 :
 {
 node_init(&$$);
 }
 | tg_action_modifier_list tg_action_modifier
 {
 $$ = $1;
 tg_append_element(&$$.action, &$2.action);
 }
;
```

```
tg_action_modifier
 : DATA number
  {
  node_init(&$$);
  $$.action.tg_flags |= TG_DATA;
  $$.action.data_limit = $2.d;
  }
 | RESET
  {
  node_init(&$$);
  $$.action.tg_flags |= TG_RESET;
  }
 | PACKET number
  {
  node_init(&$$);
  $$.action.tg_flags |= TG_PACKET;
  $$.action.packet_limit = $2.d;
  }
 | SEED integer
  {
  node_init(&$$);
  $$.action.tg_flags |= TG_SEED;
  $$.action.seed = $2.d;
  }
 | TIME time_literal
  {
  node_init(&$$);
  $$.action.tg_flags |= TG_TIME;
  dtotimeval($2.d, &$$.action.time_limit);
  }
 ;

distribution
 : number
  {
  char  *cp;

  if ((cp = dist_const_init(&$$.tmpdist, $1.d)) != NULL)
   yyerror(cp);
  }
 | DIST_CONST number
  {
  char  *cp;

  if ((cp = dist_const_init(&$$.tmpdist, $2.d)) != NULL)
   yyerror(cp);
  }
 | DIST_EXP number
  {
  char  *cp;

  if ((cp = dist_exp_init(&$$.tmpdist, $2.d, (double) 0,
```

```
   (double) MAX_RANDOM)) != NULL)
 yyerror(cp);
}
| DIST_EXP number number number  /* exp mean min max */
{
char  *cp;

if ($3.d < $4.d)
 {
 if ((cp = dist_exp_init(&$$.tmpdist, $2.d, $3.d,
   $4.d)) != NULL)
  yyerror(cp);
 }
else
 {
 if ((cp = dist_exp_init(&$$.tmpdist, $2.d, $4.d,
   $3.d)) != NULL)
  yyerror(cp);
 }
}
| DIST_MARKOV2 number distribution number distribution
{
char  *cp;

if ((cp = dist_markov2_init(&$$.tmpdist,
    $2.d,
    &($3.tmpdist),
    $4.d,
    &($5.tmpdist))) != NULL)
 yyerror(cp);
}
| DIST_UNIFORM number   /* max */
{
char  *cp;

if ((cp = dist_uniform_init(&$$.tmpdist,
    (double) 0, $2.d)) != NULL)
 yyerror(cp);
}
| DIST_UNIFORM number number  /* min max */
{
char  *cp;

if ($2.d < $3.d)
 {
 if ((cp = dist_uniform_init(&$$.tmpdist,
    $2.d, $3.d)) != NULL)
  yyerror(cp);
 }
else
 {
 if ((cp = dist_uniform_init(&$$.tmpdist,
```

```
        $3.d, $2.d)) != NULL)
      yyerror(cp);
     }
    }
   ;


symbol
   : IDENTIFIER
   {
    int i, found;

    /* Perform symbol lookup if we're working with a string */
    for (i = 0, found = 0; i < sym_tbl_index; i++) {
     if ((strcmp(sym_tbl[i].sym_name, $1.n) == 0) &&
      (sym_tbl[i].sym_type == SYM_INT_TYPE)) {
      found++;
      break;
     }
    }

    if (found) {
     $$.d = sym_tbl[i].sym_intval;
    } else {
     fprintf(stderr, "reference to unknown symbol \"%s\"",
      $1.n);
     exit(1);
    }
   }
   | decimal_number
   | number
   | integer
   | decimal_integer
   | integer
    {
    $$ = $1;
    }
   ;

time_literal
   : decimal_number
    {
    $$.d = $1.d;
    }
   | decimal_integer ':' decimal_number
    {
    $$.d = $1.d * 60 + $3.d;
    }
   | decimal_integer ':' decimal_integer ':' decimal_number
    {
    $$.d = ($1.d * 60 + $3.d) * 60 + $5.d;
    }
```

```
;

decimal_number
 : FLOATING_POINT
  {
  $$.d = $1.d;
  }
 | decimal_integer
  {
  $$.d = $1.d;
  }
 ;

number
 : FLOATING_POINT
  {
  $$.d = $1.d;
  }
 | integer
  {
  $$.d = $1.d;
  }
 ;

integer
 : HEX_INTEGER
  {
  $$.d = $1.d;
  }
 | INTEGER
  {
  $$.d = $1.d;
  }
 | OCTAL_INTEGER
  {
  $$.d = $1.d;
  }
 ;

decimal_integer
 : SMALL_INTEGER
  {
  $$.d = $1.d;
  }
 | INTEGER
  {
  $$.d = $1.d;
  }
 ;

%%
#include "lex.yy.c"
```

```
/* MAINprogram for extractdoc.     */

main(argc, argv)

 int  argc;
 char  *argv[];

 {
 extern int yydebug;
 void  sigint();
 FILE  *fp;

 /* Set debugging if it is desired.    */

# if YYDEBUG
 /* yydebug = 1; */
# endif

 /* Initialize lex state.     */

 lex_init(argc, argv);
 BEGIN LEX_NORMAL;

 /* Convert input.      */

 if (yyparse() != 0)
  exit(1);
 if (got_errors)
  exit(1);

 (void) signal(SIGINT, sigint);
 (void) signal(SIGTERM, sigint);

 /*
  * Set up logging. If ofile is set, then we write to the
  * specified file.
  */

 if ((fp = log_open(ofile)) == (FILE *) NULL) {
  exit (-1);
 }

 log_init(fp, global_start_tv, prot.prot->name, prot.prot->af,
  ifile, &prot);

 /* Fix up start and stop times.     */

 fix_times();

 /* Wait for global start time, scream if it has already passed. */
```

```
 wait_start();

 /* Generate traffic.      */

 do_actions();
 return (0);
 }


/* Perform tasks based on list of actions.    */

void
do_actions()

 {
 tg_action *cur_tg;
 struct timeval lasttime;
 long  tx_asn = -1;

 /* Set up receive routine.     */

 if ((prot.qos & QOS_INTERACTIVE) == 0)
  prot.prot->rcv = rcv_pkt;
 else if ((prot.qos & QOS_SERVER) == 0)
  prot.prot->rcv = rcv_pkt_interactive;
 else
  prot.prot->rcv = rcv_pkt_interactive_srvr;

 /* If there is no explicit setup clause, do an immediate setup. */

 if (got_setup_implicit &&
  ((tx_asn = (*(prot.prot->setup))(&prot)) == -1))
  {

  /* log the setup error. */

  perror("do_actions: protocol setup");
  exit(-1);
  }

 /* Each pass through the following loop processes one tg_action */
 /* element from the list.     */

 for (cur_tg = tg_first; cur_tg != NULL; cur_tg = cur_tg->next)
  {

  /* Wait for start time, if one was specified. Remember */
  /* the start time (or the current time, if the start */
  /* time is not specified) in ''lasttime''.  */

  if ((cur_tg->tg_flags & TG_START) != 0)
```

```
 {
 (*(prot.prot->sleep_till))(&(cur_tg->start_at));
 lasttime = cur_tg->start_at;
 }
else
 {
 if (gettimeofday(&lasttime,
    (struct timezone *)NULL) == -1)
  {
  perror("do_actions: gettimeofday");
  abort();
  }
 }

/* Compute stop time on the fly, if needed.  */

if ((cur_tg->tg_flags & TG_STOP) == 0)
 {
 if ((cur_tg->tg_flags & TG_TIME) == 0)
  {

  /* No definite stop time, set it for */
  /* some time in the year 2038... */

  cur_tg->stop_before.tv_sec = 0x7fffffff;
  }
 else
  {

  /* Add the time limit to the current */
  /* time to get the stop time.  */

  timeradd(&lasttime,
    &cur_tg->time_limit,
    &cur_tg->stop_before);
  }
 }

/* Perform the specified action.   */

if ((cur_tg->tg_flags & TG_SETUP) != 0)
 {

 /* Perform setup phase.    */

 if ((tx_asn = (*(prot.prot->setup))(&prot)) == -1)
  {

  /* log the setup error. */

  perror("do_actions: protocol setup");
  exit(-1);
```

```
 }
 }
else if ((cur_tg->tg_flags & TG_WAIT) != 0)
 {

 /* If we would run out of patience before the */
 /* end of the wait, wait for the patience */
 /* interval.    */

 while ((xpctd_replies_qh != NULL) &&
  (timercmp(&(xpctd_replies_qh->timeout),
    &(cur_tg->stop_before),
    <)))
  {

  /* Check to see if timeout has already */
  /* passed..    */

  if (check_deadline
    (&(xpctd_replies_qh->timeout)))
   {

   /* @@@@ log frustration! */

   (void)fprintf(stderr,
     "Patience exceeded!\n");
   exit(-1);
   }

  /* Wait for the timeout if not.  */

  (*(prot.prot->sleep_till))
   (&(xpctd_replies_qh->timeout));
  }

 /* Just wait until the interval is over. */

 (*(prot.prot->sleep_till))(&(cur_tg->stop_before));
 }
else if ((prot.qos & QOS_INTERACTIVE) != 0)
 {

 /* Generate traffic as specified by arrival */
 /* and length.    */

 generate_interactive(tx_asn, cur_tg, lasttime);
 }
else
 {

 /* Generate traffic as specified by arrival */
 /* and length.    */
```

```
    generate(tx_asn, cur_tg, lasttime);
    }
   }

  /* Finished, tear down connection.    */

  if ((*(prot.prot->teardown))(tx_asn) == -1)
   {

   /* log the teardown error. */

   perror("do_actions: protocol teardown");
   exit(-1);
   }

  return;
  }


/* Fix up start and stop times and perform semantic checks.  */

void
fix_times()

 {
 tg_action *cur_tg;

 /* Scan action list, computing all times that can be computed */
 /* beforehand.      */

 for (cur_tg = tg_first; cur_tg != NULL; cur_tg = cur_tg->next)
  {

  /* If we aren't interactive and someone is trying to */
  /* do a responselength, scream.    */

  if (((prot.qos & QOS_INTERACTIVE) == 0) &&
   ((cur_tg->tg_flags & TG_RESPLEN) != 0))
   {
   (void)fprintf(stderr,
     "resplen legal only if %s specified",
     "interactive");
   exit(-1);
   }

  /* If there is no stop time, try to infer one.  */

  if ((cur_tg->tg_flags & TG_STOP) == 0)
   {

   if ((cur_tg->tg_flags & (TG_START | TG_TIME)) ==
```

```
 (TG_START | TG_TIME))
 {

 /* Infer from start time and limit. */

 cur_tg->tg_flags |= TG_STOP;
 timeradd(&(cur_tg->start_at),
   &(cur_tg->time_limit),
   &(cur_tg->stop_before));
 }
else if ((cur_tg->next != NULL) &&
  ((cur_tg->next->tg_flags & TG_START) != 0))
 {

 /* Infer from next actions start time. */

 if (!timercmp(&(cur_tg->next->start_at),
   &(cur_tg->start_at),
   >))
  {
  (void)fprintf(stderr,
    "fix_times: %s\n",
    "time clash");
  }
 cur_tg->tg_flags |= TG_STOP;
 cur_tg->stop_before = cur_tg->next->start_at;
 }
}

/* If there is now a stop time, make sure that it */
/* does not conflict with the next item's start time. */
/* If the next item does not have a start time, then */
/* copy in the stop time from this item.  */

if (((cur_tg->tg_flags & TG_STOP) != 0) &&
 (cur_tg->next != NULL))
 {
 if ((cur_tg->next->tg_flags & TG_START) == 0)
  {
  cur_tg->next->tg_flags |= TG_START;
  }
 else
  {
  if (timercmp(&(cur_tg->stop_before),
    &(cur_tg->next->start_at),
    >))
   {
   (void)fprintf(stderr,
     "fix_times: %s\n",
     "time clash");
   }
  }
```

```
    }
   }
  }

/* Check to see if current time has exceeded deadline.   */

int
check_deadline(deadline)

 struct timeval *deadline;

 {
 struct timeval tv;

 if (gettimeofday(&tv, (struct timezone *)NULL) == -1)
  {
  perror("check_deadline: gettimeofday");
  abort();
  }
 return (timercmp(deadline, &tv, <));
 }


/* Generate traffic as specified by cur_tg.    */

void
generate(tx_asn, cur_tg, lasttime)

 long  tx_asn;
 tg_action *cur_tg;
 struct timeval lasttime;
 {
 unsigned long datasent = 0, pktsent = 0;
 static unsigned long pktid = 0;

 /* Set random-number generator seed, if so specified.  */

 if ((cur_tg->tg_flags & TG_SEED) != 0) {
  (void)srandom(cur_tg->seed);
 }

 /* Reset packet counter */

 if ((cur_tg->tg_flags & TG_RESET) != 0) {
  pktid = 0;
 }

 /* Generate traffic until either packet, data, or time limit */
 /* is exceeded.      */

 for (;;)
```

```
{
char  *buf;
struct timeval nextpkt_tv;
unsigned long pktlen;
double  arrival;

/* Find arrival time for next packet.   */

arrival = (*(cur_tg->arrival.generate))(&cur_tg->arrival);
if (arrival != 0.)
 {

 /* The interarrival time is not exactly zero, */
 /* count the time to start when the last packet */
 /* was scheduled to leave, rather than when it */
 /* actually left.    */

 dtotimevalfromthen(&lasttime, arrival, &nextpkt_tv);
 }
else
 {

 /* The interarrival time is zero, so get the */
 /* current time of day to check for stop times. */

 if (gettimeofday(&nextpkt_tv,
    (struct timezone *)NULL) == -1)
  {
  perror("generate: gettimeofday");
  abort();
  }
 }

/* If we are to stop this action before the next packet */
/* is to arrive, just wait for the stop time.  */

if (!timercmp(&(cur_tg->stop_before), &nextpkt_tv, >))
 {
 (*(prot.prot->sleep_till))(&(cur_tg->stop_before));
 break;
 }

/* Otherwise, wait until nextpkt_tv to transmit the */
/* packet.      */

if (arrival != 0)
 (*(prot.prot->sleep_till))(&nextpkt_tv);
lasttime = nextpkt_tv;

/* Did we exceed the limit on the number of packets to send? */

if (((cur_tg->tg_flags & TG_PACKET) != 0) &&
```

```
    (++pktsent > cur_tg->packet_limit)) {

    break;
   }

   /* Get the packet length and see if the data limit has */
   /* been exceeded.     */

   pktlen = (*(cur_tg->length.generate))(&cur_tg->length);
   if (((cur_tg->tg_flags & TG_DATA) != 0) &&
    ((datasent += pktlen) > cur_tg->data_limit)) {

    /* The current packet would put us over the */
    /* limit, quit!     */

    break;
   }

   /* Get a buffer for the packet and transmit it.  */

   if ((buf = (*(prot.prot->buffer_get))(pktlen)) == NULL)
    {

    log_error(NULL, NULL, tx_asn, LOGERR_MEM);
    (void)fprintf(stderr,
      "main: buffer %s\n",
      "allocation failure");
    }
   else
    {

    (void)(*(prot.prot->send))(tx_asn,
        buf,
        pktlen,
        &(cur_tg->stop_before),
        &pktid);
    }
   }
  }


/* Generate interactive-like traffic as specified by cur_tg.  */

void
generate_interactive(tx_asn, cur_tg, lasttime)

 long  tx_asn;
 tg_action *cur_tg;
 struct timeval lasttime;

 {
 unsigned long datasent = 0, pktsent = 0;
```

```
int  encodelen;
static unsigned long minbuflen = 0;
unsigned long pktid;
static unsigned long replydatawaiting = 0;
unsigned long resplen;

/* Get minimum buffer length if we don't yet know it.  */

if (minbuflen == 0)
 minbuflen = encode_special_response((char *)NULL, 0, 0);

/* Set random-number generator seed, if so specified.  */

if ((cur_tg->tg_flags & TG_SEED) != 0)
 {
 (void)srandom(cur_tg->seed);
 }

/* Generate traffic until either data limit or time limit */
/* exceeded.      */

for (;;)
 {
 char  *buf;
 struct timeval nextpkt_tv;
 unsigned long pktlen;
 double  arrival;

 /* Find arrival time for next packet.   */

 arrival = (*(cur_tg->arrival.generate))(&cur_tg->arrival);
 if (arrival != 0.)
  {

  /* The interarrival time is not exactly zero, */
  /* count the time to start when the last packet */
  /* was scheduled to leave, rather than when it */
  /* actually left.    */

  dtotimevalfromthen(&lasttime, arrival, &nextpkt_tv);
  }
 else
  {

  /* The interarrival time is zero, so get the */
  /* current time of day to check for stop times. */

  if (gettimeofday(&nextpkt_tv,
     (struct timezone *)NULL) == -1)
   {
   perror("generate: gettimeofday");
   abort();
```

```
    }
   }

  /* If we would run out of patience before the stop */
  /* time and before the next packet transmission time, */
  /* wait for the patience interval.   */

  while ((xpctd_replies_qh != NULL) &&
   (timercmp(&(xpctd_replies_qh->timeout),
     &(cur_tg->stop_before),
     <)) &&
   (timercmp(&(xpctd_replies_qh->timeout), &nextpkt_tv, <)))
   {

   /* Check to see if timeout has already passed.. */

   if (check_deadline(&(xpctd_replies_qh->timeout)))
    {

    /* @@@@ log frustration!  */

    (void)fprintf(stderr, "Patience exceeded!\n");
    exit(-1);
    }

   /* Wait for the timeout if not.   */

   (*(prot.prot->sleep_till))
    (&(xpctd_replies_qh->timeout));
   }

  /* If we are to stop this action before the next packet */
  /* is to arrive, just wait for the stop time.  */

  if (!timercmp(&(cur_tg->stop_before), &nextpkt_tv, >))
   {
   (*(prot.prot->sleep_till))(&(cur_tg->stop_before));
   break;
   }

  /* Otherwise, wait until nextpkt_tv to transmit the */
  /* packet.     */

  if (arrival != 0)
   (*(prot.prot->sleep_till))(&nextpkt_tv);
  lasttime = nextpkt_tv;

  /* Did we exceed the limit on the number of packets to send? */

  if (((cur_tg->tg_flags & TG_PACKET) != 0) &&
   (++pktsent > cur_tg->packet_limit)) {
```

```
 break;
}

/* Get the packet length and see if the data limit has */
/* been exceeded.     */

pktlen = (*(cur_tg->length.generate))(&cur_tg->length);
if (((cur_tg->tg_flags & TG_DATA) != 0) &&
 ((datasent += pktlen) > cur_tg->data_limit)) {

 /* The current packet would put us over the */
 /* limit, quit!     */

 break;
}

/* Get a buffer for the packet and transmit it.  */

if ((buf = (*(prot.prot->buffer_get))(pktlen > minbuflen ?
     pktlen :
     minbuflen)) == NULL)
 {

 log_error(NULL, NULL, tx_asn, LOGERR_MEM);
 (void)fprintf(stderr,
   "main: buffer %s\n",
   "allocation failure");
 }
else
 {

 if ((cur_tg->tg_flags & TG_RESPLEN) == 0)
  {

  /* Encode a special zero to say that */
  /* we want no response.   */

  encodelen = encode_special_response(buf,
       pktlen,
       0);
  }
 else
  {

  /* Generate desired response length. */

  resplen = (*(cur_tg->resplen.generate))
    (&(cur_tg->resplen));

  /* Encode the response length into the */
  /* packet and try to send it.  */
```

```
 encodelen = encode_response(buf,
     pktlen,
     resplen);
 }

/* Send the packet.    */

for (;;)
 {
 struct timeval *tvp;

 /* Get a pointer to a timeout, either */
 /* the patience timeout or the end of */
 /* this action, whichever comes first. */

 if (xpctd_replies_qh == NULL)
  tvp = &(cur_tg->stop_before);
 else if (timercmp(&(cur_tg->stop_before),
    &(xpctd_replies_qh->timeout),
    >))
  tvp = &(xpctd_replies_qh->timeout);
 else
  tvp = &(cur_tg->stop_before);
 if ((*(prot.prot->send))(tx_asn,
     buf,
     pktlen > encodelen ?
     pktlen :
     encodelen,
     tvp,
     &pktid) == -1)
  {

  /* If we had a hard error or if */
  /* we hit the end of the action,*/
  /* quit.   */

  if ((errno != ETIME) ||
   (tvp == &(cur_tg->stop_before)))
   break;

  /* Check to see if patience */
  /* timeout has already passed.. */

  if ((xpctd_replies_qh != NULL) &&
   (check_deadline
   (&(xpctd_replies_qh->timeout))))
   {

   /* @@@@ log frustration!*/

   (void)fprintf(stderr,
     "%s exceeded!\n",
```

```
      "Patience");
     exit(-1);
     }
    }
   else
    {

    /* If this action is impatient, */
    /* make an xpctd-replies entry. */

    if ((cur_tg->tg_flags &
     TG_PATIENCE) != 0)
     {
     xpctd_replies *p =
      xpctd_replies_get();

     p->byte_seqno =
      replydatawaiting;
     timeradd(&nextpkt_tv,
       &(cur_tg->patience),
       &(p->timeout));
     *xpctd_replies_qt = p;
     xpctd_replies_qt = &(p->next);

     /* Accumulate the total */
     /* amount of outstanding*/
     /* reply data expected. */

     replydatawaiting += resplen;
     }

    break;
    }
   }
  }
 }

/* NODE INITialize. Forces all fields to zero/NULL.   */

void
node_init(node)

 YYSTYPE  *node;

 {

 (void)bzero((char *)node, sizeof(*node));
 }
```

```
/* Receive and log packet.     */

int
rcv_pkt(rx_asn, tx_asn, buf, len, pktid)

 long  rx_asn;
 long  tx_asn;
 char  *buf;
 int  len;
 unsigned long pktid;

 {

 /* Do nothing, since the protocol logs it. This serves as a */
 /* debugging hook.     */

 return (0);
 }


/* Receive a response from an interactive server (we sent a packet */
/* requesting a response, the server responded, and we just received */
/* that response). Log the packet and modify the xpctd_replies list */
/* to account for that packet.     */

int
rcv_pkt_interactive(rx_asn, tx_asn, buf, len, pktid)

 long  rx_asn;
 long  tx_asn;
 char  *buf;
 int  len;
 unsigned long pktid;

 {
 static unsigned long replydatarcvd = 0;
 int  retval;
 xpctd_replies *xp = xpctd_replies_qh;

 /* Invoke rcv_pkt to log the packet's arrival.   */

 retval = rcv_pkt(rx_asn, tx_asn, buf, len, pktid);

 /* Each pass through the following loop removes one  */
 /* xpctd_replies entry from the list. Note that stream  */
 /* protocols such as TCP can fuse packets; this can result in */
 /* one receive removing several xpctd_replies.   */

 replydatarcvd += len;
 while (xp != NULL)
  {
```

```
 /* Quit if no more xpctd_replies are covered.  */

 if (replydatarcvd <= xp->byte_seqno)
  break;

 /* Remove the current xpctd_replies entry.  */

 xpctd_replies_qh = xp->next;
 xpctd_replies_free(xp);
 if ((xp = xpctd_replies_qh) == NULL)
  xpctd_replies_qt = &xpctd_replies_qh;
 }

/* Tell the caller what rcv_pkt did with the buffer.  */

return (retval);
}


/* Receive a packet from an interactive client that must be responded */
/* to. Since packets may be glued and chopped, the commands from the */
/* client must be pasted back together, in general. Therefore, a */
/* separate state structure is maintained for each association.  */

int
rcv_pkt_interactive_srvr(rx_asn, tx_asn, buf, len, pktid)

 long  rx_asn;
 long  tx_asn;
 char  *buf;
 int  len;
 unsigned long pktid;

 {
 char  *bufend;
 char  *cp = buf;
 srvr_state *p;
 int  rplpktid;
 char  *replbuf;
 int  retval;

 /* Invoke rcv_pkt to log the packet's arrival.   */

 retval = rcv_pkt(rx_asn, tx_asn, buf, len, pktid);

 /* If we really have a packet, get a pointer to the asn's */
 /* state information. If we have an EOF or error, delete */
 /* the asn's state information.     */

 if (buf != NULL)
  p = srvr_state_get(rx_asn, 0);
 else
```

```
 {
 p = srvr_state_get(rx_asn, 1);
 return (retval);
 }

/* If this association has given us bad data in the past, */
/* ignore it.      */

if (p->bad)
 return (retval);

/* Each pass through the following loop consumes one byte of */
/* control information or skips as much filler as possible. */

bufend = &(buf[len]);
while (cp < bufend)
 {

 /* Handle current byte as specified by current state. */

 switch (p->state)
 {
 case srvr_len :

  /* Accumulate the response-length field. */

  p->acc |= (*cp & 0x7f) << (p->nbytes * 7);
  if (++p->nbytes > 5)
   {

   /* Failed plausibility check, so ignore */
   /* this association from here on out. */

   p->bad = 1;
   log_error(NULL, NULL, rx_asn, LOGERR_INTFMT);
   }

  /* If this is the last byte, send the reply. */
  /* The special form of zero represented by */
  /* 0x80 0x00 says ''send no packet''. This */
  /* can be detected by *cp==0x00, since putting */
  /* a zero at the end of a number does not */
  /* change its value.    */

  if ((*cp & 0x80) == 0)
   {
   if (*cp == 0)
    ;
   else if ((replbuf =
    (*(prot.prot->buffer_get))(p->acc)) ==
    NULL)
    {
```

```
    log_error(NULL, NULL, -1, LOGERR_MEM);
    (void)fprintf(stderr,
      "%s%s: Out of memory\n",
      "rcv_pkt_",
      "interactive_srvr");
    abort();
    }
   else
    {
    (void)(*(prot.prot->send))(tx_asn,
       replbuf,
       p->acc,
       NULL,
       &rplpktid);
    }

   /* Set the state to pick up the skip */
   /* count.    */

   p->state = srvr_skip;
   p->acc = 0;
   p->nbytes = 0;
   }

  cp++;
  break;

 case srvr_skip :

  /* Accumulate the skip field.   */

  p->acc |= (*cp & 0x7f) << (p->nbytes * 7);

  /* If we have the full field, skip the  */
  /* specified number of bytes.   */

  if ((*cp & 0x80) != 0)
   {
   if (++p->nbytes > 5)
    {

    /* Failed plausibility check, */
    /* so ignore this association */
    /* from here on out.  */

    p->bad = 1;
    log_error(NULL,
      NULL,
      rx_asn,
      LOGERR_INTFMT);
    }
   cp++;
```

```
        break;
        }
       else
        {
        p->skip = p->acc - p->nbytes;
        if ((p->skip & ~0x7fffffff) != 0)
         {

         /* Failed plausibility check, */
         /* so ignore this association */
         /* from here on out.  */

         p->bad = 1;
         log_error(NULL,
           NULL,
           rx_asn,
           LOGERR_INTFMT);
         }
        p->state = srvr_skipping;
        p->nbytes = 0;

        /* drop into next leg of switch... */

        }

      case srvr_skipping :

       /* Skip the bytes.    */

       cp += p->skip;

       /* If past the end of the packet, adjust the */
       /* count and skip the first part of the next */
       /* packet. Otherwise, just skip the specified */
       /* length.     */

       if ((cp > bufend) ||
        (cp < buf))
        p->skip = cp - bufend;
       else
        {
        p->state = srvr_len;
        p->acc = 0;
        }
       break;


      }
      }

     return (retval);
     }
```

```
/* Get the state entry for the specified association, creating it if */
/* necessary, and returning a pointer to it. If the delete flag is */
/* set, the entry is deleted if it is present and a NULL pointer */
/* returned.       */

srvr_state *
srvr_state_get(asn, delete)

 long  asn;
 int  delete;

 {
 srvr_state *p;
 srvr_state **q;

 /* Search the list for the specified entry, if found, move it */
 /* to the front of the list.     */

 for (p = srvr_state_h, q = &srvr_state_h;
  p != NULL;
  q = &(p->next), p = *q)
  {
  if (p->asn == asn)
   {
   *q = p->next;
   if (delete)
    {
    (void)free((char *)p);
    return (NULL);
    }
   else
    {
    p->next = srvr_state_h;
    srvr_state_h = p;
    break;
    }
   }
  }

 /* If no entry was found, create one (unless it was to be */
 /* deleted anyway...).      */

 if ((p == NULL) &&
  !delete)
  {
  if ((p = (srvr_state *)malloc(sizeof(srvr_state))) == NULL)
   {

   log_error(NULL, NULL, -1, LOGERR_MEM);
   (void)fprintf(stderr,
```

```
    "srvr_state_get: Out of memory\n");
   abort();
   }
  p->next = srvr_state_h;
  srvr_state_h = p;
  p->state = srvr_len;
  p->asn = asn;
  p->acc = 0;
  p->nbytes = 0;
  p->bad = 0;
  }

 /* Return a pointer to the entry.    */

 return (p);
 }


/* Append new element to action structure.    */

void
tg_append_element(ap1, ap2)

 tg_action *ap1;
 tg_action *ap2;

 {

 ap1->tg_flags |= ap2->tg_flags;
 if ((ap2->tg_flags & TG_ARRIVAL) != 0)
  ap1->arrival = ap2->arrival;
 if ((ap2->tg_flags & TG_DATA) != 0)
  ap1->data_limit = ap2->data_limit;
 if ((ap2->tg_flags & TG_PACKET) != 0)
  ap1->packet_limit= ap2->packet_limit;
 if ((ap2->tg_flags & TG_LENGTH) != 0)
  ap1->length = ap2->length;
 if ((ap2->tg_flags & TG_PATIENCE) != 0)
  ap1->patience = ap2->patience;
 if ((ap2->tg_flags & TG_RESPLEN) != 0)
  ap1->resplen = ap2->resplen;
 if ((ap2->tg_flags & TG_SEED) != 0)
  ap1->seed = ap2->seed;
 if ((ap2->tg_flags & TG_TIME) != 0)
  ap1->time_limit = ap2->time_limit;
 }


/* Wait for start time. Currently we assume that initial page-faults */
/* are not a performance problem. If this is not the case, we need */
/* to add code to reference all pages in order to fault them in. */
```

```
void
wait_start()

 {

 if (check_deadline(&global_start_tv))
  {
  (void)fprintf(stderr, "Starting time already passed!\n");
  (void)fprintf(stderr, "Restart program!\n");
  exit(-1);
  }
 (*(prot.prot->sleep_till))(&global_start_tv);
 }


/* Get an expected-replies queue element.    */

xpctd_replies *
xpctd_replies_get()

 {
 xpctd_replies *p;

 /* If the freelist is empty, malloc up another entry.  */

 if (xpctd_replies_flist == NULL)
  {
  p = (xpctd_replies *)malloc(sizeof(xpctd_replies));
  if (p == NULL)
   {
   (void)fprintf(stderr,
     "xpctd_replies_get: out of memory!\n");
   abort();
   }
  p->next = NULL;
  return (p);
  }

 /* Freelist is nonempty, just grab the next element.  */

 p = xpctd_replies_flist;
 xpctd_replies_flist = p->next;
 return (p);
 }


/* Free up an expected-replies queue element.    */

void
xpctd_replies_free(p)

 xpctd_replies *p;
```

```
 {

 p->next = xpctd_replies_flist;
 xpctd_replies_flist = p;
 return;
 }


/* Yacc error routine.      */

void
yyerror(s)

 char  *s;

 {
 (void)fflush(stdout);
 (void)fprintf(stderr, "%s, line %d: %s\n", filename, lineno, s);
 (void)fflush(stderr);
 got_errors = 1;
 }

void
sigint(sig, code, scp, addr)

 int  sig, code;
 struct sigcontext *scp;
 char  *addr;
{

 log_close();
}
```

2.3 DCAT DECLARATIONS

```
/*****************************************************************************
 *                                                                           *
 * File: config.h                                                            *
 *                                                                           *
 * Protocol definition structures.                                           *
 *                                                                           *
 * Written 08-Aug-90 by Paul E. McKenney, SRI International.                 *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char config_h_rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/
src/tg/RCS/config.h,v 1.9 90/11/26 13:18:55 dlee Exp $";
#endif lint

/* Maximum packet buffer size.      */

#define MAX_PKT_SIZE  8192 /* 3072 /* Sized for Ethernet. */

/* Maximum value from random-number generator.    */

#define MAX_RANDOM  0x7fffffff

/*
 * FD_SET and associated macros are not defined in SunOS 3.5
 * We need to defined them here, if not previously defined.
 */
#ifndef NFDBITS

#define NFDBITS (30) /* bits per mask */

#define FD_SET(n, p) ((p)->fds_bits[(n)/NFDBITS] |= (1 << ((n) % NFDBITS)))
#define FD_CLR(n, p) ((p)->fds_bits[(n)/NFDBITS] &= ~(1 << ((n) % NFDBITS)))
#define FD_ISSET(n, p) ((p)->fds_bits[(n)/NFDBITS] & (1 << ((n) % NFDBITS)))
#define FD_ZERO(p) bzero((char *)(p), sizeof (*(p)))

#endif
```

```
/*****************************************************************************
 *                                                                           *
 * File: decode.h                                                            *
 *                                                                           *
 * Header file for encode and decode compressed integers.                    *
 *                                                                           *
 * Written 12-Sep-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char decode_h_rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/
src/tg/RCS/decode.h,v 1.5 90/11/26 12:29:24 dlee Exp $";
#endif lint

extern char *decode_ulong();
extern int encode_response();
extern int encode_special_response();
extern char *encode_ulong();
```

```
/*****************************************************************************
 *                                                                           *
 * File: distribution.h                                                      *
 *                                                                           *
 * Structs defining addtional parameters for those probability               *
 * distributions that need them.                                             *
 *                                                                           *
 * Written 20-Jun-90 by Paul E. McKenney, SRI International.                 *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char distribution_h_rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/
dartnet/src/tg/RCS/distribution.h,v 1.7 90/11/26 12:29:29 dlee Exp $";
#endif lint

/* Type definitions local to this file.    */

typedef struct
 {
 double  (*generate)(); /* generate a new variable. */
 double  par1;  /* distribution parameter 1. */
 double  par2;  /* distribution parameter 2. */
 double  par3;  /* distribution parameter 3. */
 double  par4;  /* distribution parameter 4. */
 char  *pars;  /* pointer to more parameters */
    /* for those distributions that */
    /* need them.   */
 }  distribution;

typedef struct
 {
 double  mean[2]; /* mean time in each state. */
 unsigned long p[2];  /* probability of remaining in */
    /* same state.   */
 distribution *dist[2]; /* distribution in each state. */
 int  state;  /* current state.  */
 }  dist_markov2; /* 2-state markov distribution. */

extern char  *dist_const_init();
extern char  *dist_exp_init();
extern char  *dist_markov2_init();
extern char  *dist_uniform_init();
```

```
/*****************************************************************************
 *                                                                           *
 * File: log.h                                                               *
 *                                                                           *
 * Header file for handling log files                                        *
 *                                                                           *
 * Written 12-Sep-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char log_h_rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/src/
tg/RCS/log.h,v 1.7 90/11/26 12:29:35 dlee Exp $";
#endif lint

#define LOG_VERSION 1
#define LOG_SUBVERSION 0

/*
 * A log file entry consists of a tuple consisting of the following fields:
 *
 * <Record type> <Record control> <Record value>
 */

/* Record type field enumerations */

#define LOGTYPE_RX  1
#define LOGTYPE_TX  2
#define LOGTYPE_SETUP  3
#define LOGTYPE_TEARDOWN 4
#define LOGTYPE_ACCEPT  5
#define LOGTYPE_ERROR  6

/* Control field modifier bit definitions */

#define LOGCTL_SCHED  (0x1<<0)
#define LOGCTL_ADDR  (0x1<<1)
#define LOGCTL_2ADDR  (0x1<<2)
#define LOGCTL_EXCEPT  (0x1<<3)

/* Error codes when record type is set to LOGTYPE_ERROR */

#define LOGERR_INTFMT  1 /* Script format error */
#define LOGERR_MEM  2 /* Out of memory */
#define LOGERR_2SETUP  3 /* Two connections were established */
#define LOGERR_GETTIME  4 /* gettimeofday() failed. */
#define LOGERR_SELECT  5 /* select() failed. */
#define LOGERR_FCNTL  6 /* fcntl() failed. */
#define LOGERR_GETPEER  7 /* getpeername() failed. */

#define BEGIN_HDR_STRING "<Begin TG Header>\n"
```

```
#define END_HDR_STRING  "<End TG Header>\n"

/* The following routines are exported */

FILE *log_open();
int log_init();
void log_tx();
void log_rx();
void log_accept();
void log_setup();
void log_teardown();
void log_error();
```

```
/*****************************************************************************
 *                                                                           *
 * File: protocol.h                                                          *
 *                                                                           *
 * Protocol definition structures.                                           *
 *                                                                           *
 * Written 20-Jun-90 by Paul E. McKenney, SRI International.                 *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char protocol_h_rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/
src/tg/RCS/protocol.h,v 1.8 90/11/26 12:29:54 dlee Exp Locker: dlee $";
#endif lint

/* Convert the double d to timer tvp.     */

#define dtotimeval(d, tvp) \
 { \
 (tvp)->tv_sec = floor(d); \
 (tvp)->tv_usec = ((d) - (tvp)->tv_sec) * 1000000; \
 }

/* Convert the double d to timer tvp, offsetting from current time. */

#define dtotimevalfromnow(d, tvp) \
 { \
 unsigned long seconds; \
 \
 if (gettimeofday(tvp, (struct timezone *)NULL) == -1) \
  { \
  (void)perror("gettimeofday"); \
  abort(); \
  }\
 seconds = floor(d); \
 (tvp)->tv_sec += seconds; \
 (tvp)->tv_usec += ((d) - seconds) * 1000000; \
 if ((tvp)->tv_usec >= 1000000) \
  { \
  (tvp)->tv_usec -= 1000000; \
  (tvp)->tv_sec++; \
  } \
 }

/* Convert the double d to timer tvp, offsetting from specified time. */

#define dtotimevalfromthen(then, d, tvp) \
 { \
 unsigned long seconds; \
 \
 seconds = floor(d); \
 (tvp)->tv_sec = (then)->tv_sec + seconds; \
```

```
 (tvp)->tv_usec = (then)->tv_usec + ((d) - seconds) * 1000000; \
 if ((tvp)->tv_usec >= 1000000) \
  { \
  (tvp)->tv_usec -= 1000000; \
  (tvp)->tv_sec++; \
  } \
 }

/* Add tvp1 and tvp2, putting the result into result. result may */
/* alias either tvp1 or tvp2 or both, if desired.   */

#define timeradd(tvp1, tvp2, result) \
 { \
 (result)->tv_sec = (tvp1)->tv_sec + (tvp2)->tv_sec; \
 (result)->tv_usec = (tvp1)->tv_usec + (tvp2)->tv_usec; \
 if ((result)->tv_usec >= 1000000) \
  { \
  (result)->tv_usec -= 1000000; \
  (result)->tv_sec++; \
  } \
 }

/* Subtract tvp2 from tvp1, putting the result into result. result */
/* may alias either tvp1 or tvp2, if desired.    */

#define timersub(tvp1, tvp2, result) \
 { \
 (result)->tv_sec = (tvp1)->tv_sec - (tvp2)->tv_sec; \
 if ((tvp1)->tv_usec >= (tvp2)->tv_usec) \
  (result)->tv_usec = (tvp1)->tv_usec - (tvp2)->tv_usec; \
 else \
  { \
  (result)->tv_usec = 1000000 + \
     (tvp1)->tv_usec - \
     (tvp2)->tv_usec; \
  (result)->tv_sec--; \
  } \
 }

/* Type definitions local to this file.    */

typedef struct tg_action_  /* TG action structure.  */
 {
 struct tg_action_ *next;
 int  tg_flags; /* TG flags.   */
 struct timeval start_at; /* time to begin this action. */
 struct timeval stop_before; /* time to be done w/ action. */
 distribution arrival; /* interarrival time.  */
 long  data_limit; /* max amt of data to send. */
 long  packet_limit; /* max amt of data to send. */
 distribution length;  /* packet length.  */
 struct timeval patience; /* patience duration.  */
```

```
distribution resplen; /* response length distribution.*/
long  seed;  /* new RNG seed.  */
struct timeval time_limit; /* max amt of time to be sending*/
}  tg_action;

/* tg_flags definitions.      */

#define TG_ARRIVAL 0x0001  /* Got arrival distribution. */
#define TG_DATA  0x0002  /* Got data send limit.  */
#define TG_LENGTH 0x0004  /* Got packet length distr. */
#define TG_PATIENCE 0x0008  /* Got patience duration. */
#define TG_RESPLEN 0x0010  /* Got response length.  */
#define TG_SEED  0x0020  /* Got RNG seed.  */
#define TG_SETUP 0x0040  /* Got setup command.  */
#define TG_START 0x0080  /* Got explicit start time. */
#define TG_STOP  0x0100  /* Got explicit stop time. */
#define TG_TIME  0x0200  /* Got time limit.  */
#define TG_WAIT  0x0400  /* Got wait command.  */
#define TG_PACKET 0x0800  /* Got packet limit.  */
#define TG_RESET 0x1000  /* Got reset command.  */

/* Protocol switch table definition.     */

typedef struct    /* Protocol table entry. */
 {
char  *name;  /* name of protocol.  */
short  af;  /* Address family.  */
long  (*setup)(); /* connection setup function. */
    /* protocol *  */
    /* returns -1 if cannot setup. */
int  (*teardown)(); /* connection teardown function.*/
    /* unsigned long cxn */
    /* returns -1 if cannot teardown*/
int  (*rcv)(); /* to receive incoming pkts. */
    /* TG-SUPPLIED!!!  */
    /* unsigned long rx */
    /* unsigned long tx */
    /* char *buf  */
    /* int len   */
    /* unsigned long pktid */
    /* The routine should return 1 */
    /* if it has disposed of the */
    /* buffer (e.g., by modifying */
    /* it and passing it to send), */
    /* otherwise it should return 0.*/
int  (*send)(); /* to send out a packet. */
    /*  unsigned long tx */
    /* char *buf  */
    /* char *len  */
    /* struct timeval *endtout */
    /* unsigned long *pktid */
    /* NULL endtout says to wait */
```

```
    /* forever, if necessary. */
    /* Returns the number of bytes */
    /* actually sent, which will */
    /* normally be equal to len. */
    /* Returns -1 if an error */
    /* occurs (such as a timeout) */
    /* and sets errno appropriately.*/
  void  (*sleep_till)(); /* Suspends until the specified*/
    /* time, processing any packets */
    /* that arrive in the interim. */
    /*  struct timeval *waketime*/
  char  *(*buffer_get)(); /* request buffer to be used */
    /* to compose packets to be */
    /* output.   */
    /* This allows protocols to */
    /* avoid packet copying. */
    /* unsigned long maxlen */
    /* Return NULL if no more bufs. */
  void  (*buffer_free)(); /* return buffer to freelist. */
    /*  char *buf  */
  int  (*atoaddr)(); /* parse an ascii string into */
    /* a sockaddr structure, return */
    /* false if unsuccessful. */
    /* char *addr  */
    /* struct sockaddr *s */
  int  (*addrtoa)(); /* format a sockaddr structure */
    /* into an ascii string, return */
    /* false if unsuccessful. */
    /* struct sockaddr *s */
    /* char *addr  */
  char  *(*btoaddr)(); /* parse a binary log address */
    /* a sockaddr structure, return */
    /* pointer to first byte of */
    /* addr that was not consumed. */
    /* char *addr  */
    /* struct sockaddr *s */
  char  *(*addrtob)(); /* format a sockaddr structure */
    /* into an binary log address, */
    /* return pointer to first byte */
    /* of addr that was not  */
    /* overwritten.   */
    /* struct sockaddr *s */
    /* char *addr  */
  }  protocol_table;

extern protocol_table *find_protocol();

/* Protocol connection definition struct.    */

typedef struct    /* protocol defn structure. */
 {
 int  qos;  /* Quality of Service flags. */
```

```
 long  rcvwin;
 long  sndwin;
 struct sockaddr src;
 struct sockaddr dst;
 double  avg_bandwidth;
 double  peak_bandwidth;
 double  avg_delay;
 double  peak_delay;
 double  avg_loss;
 double  peak_loss;
 double  interval;
 unsigned long mtu;
 protocol_table *prot;  /* pointer to protocol table */
    /* entry.   */
 }  protocol;

/* Quality of service definitions.     */

#define QOS_AVG_BANDWIDTH 0x0001 /* Got average bandwidth. */
#define QOS_PEAK_BANDWIDTH 0x0002 /* Got peak bandwidth.  */
#define QOS_AVG_DELAY 0x0004  /* Got average delay.  */
#define QOS_PEAK_DELAY 0x0008  /* Got peak delay.  */
#define QOS_AVG_LOSS 0x0010  /* Got average loss rate. */
#define QOS_PEAK_LOSS 0x0020  /* Got peak loss rate.  */
#define QOS_INTERVAL 0x0040  /* Got averaging interval. */
#define QOS_MTU  0x0080  /* Got max transmission unit. */
#define QOS_RCVWIN 0x0100  /* Got receive window size. */
#define QOS_SNDWIN 0x0200 /* Got send window size. */


#define QOS_INTERACTIVE 0x1000  /* Simulate interactive session.*/
#define QOS_SRC  0x2000  /* Got source address.  */
#define QOS_DST  0x4000  /* Got destination address. */
#define QOS_SERVER 0x8000  /* Set up server socket to */
    /* accept incoming connections. */
```

2.4 DCAT SOURCE

```
/*****************************************************************************
 *                                                                           *
 * File: dcat.c                                                              *
 *                                                                           *
 * A filter for reading TG log files                                         *
 *                                                                           *
 * Written 12-Sep-90 by Danny Lee, SRI International.                        *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include "log.h"

#ifndef TRUE
#define TRUE 1
#endif TRUE

#ifndef FALSE
#define FALSE 0
#endif FALSE

main(argc, argv)
 int   argc;
 char  **argv;

{
 register char *buff;
 int   match, index;
 char  *malloc();
 unsigned long  result;
 int   filestat;
 FILE  *fi;

 /* Open specified file */
 if ( argc <= 1 )
 fi = stdin;
 else {
 if ((fi = fopen(*(++argv), "r")) == NULL) {
  perror("dcat cant open input ");
  exit (-1);
 }
 }

 /* Allocate temporary buffer space */
 if ((buff = malloc (BUFSIZ)) == NULL) {
 perror ("dcat: malloc");
 exit (-1);
```

```
 }

 /* Read past the header portion of the file. */
 for (match = FALSE; (match == FALSE) && (fgets (buff, BUFSIZ, fi) != NULL); )
{

 if (!strcmp (buff, END_HDR_STRING)) {
  match = TRUE;
  index = fi->_ptr - fi->_base;
 } else {
  printf ("%s", buff);
 }
 }
 /*
 * End of file was detected without finding the
 * special end of header entry.
 */
 if (match == FALSE) {
 fprintf (stderr, "Error: premature end of file\n");
 exit (-1);
 }

 /* Print a header for the forthcoming table */
 puts ("\n\nEvent Time\tType\t\tAddress\t\tId\tLength");
 printf ("-----------------------------------------------------");
 printf ("---------------\n\n");

 /* Read the record type until we run out */
 for (; (filestat = decode_ulong2(fi, &result)) != EOF; ) {
 log_parse (fi, result);
 }

 free(buff);

 if (fi != stdin) {
 fclose (fi);
 } else {
 clearerr (fi);   /* reset sticky eof */
 }
 if (ferror (stdout)) {
 fprintf (stderr, "cat: output write error\n");
 exit (-1);
 }
 exit (0);
}

/*
 *
 *
 */

log_parse (fp, rec_type)
```

```
FILE  *fp;
unsigned long rec_type;
{
int  ctrl;
unsigned long ac_t_usec, t_sec, t_usec, errno, assoc;

/* Fetch record control/modifier word */
if ((ctrl = getc (fp)) == EOF) {
 fprintf (stderr, "log_parse: premature EOF\n");
 fprintf (stdout, "log_parse: premature EOF\n");
 exit (-1);
}
/* Print event time */
if (ctrl & LOGCTL_SCHED) {
 decode_ulong2 (fp, &t_sec); /* time in seconds */
 decode_ulong2 (fp, &t_usec); /* time in useconds */
 decode_ulong2 (fp, &ac_t_usec); /* time in useconds */
 printf ("%u.%06u + %u\t", t_sec, t_usec, ac_t_usec);
} else {
 decode_ulong2 (fp, &t_sec); /* time in seconds */
 decode_ulong2 (fp, &t_usec); /* time in useconds */
 printf ("%u.%06u\t", t_sec, t_usec);
}

switch(rec_type) {
 case LOGTYPE_RX:
  printf ("Receive ");
  {
   unsigned long pkt_id, pkt_len, assoc, lerrno;

   /* Extract address information */
   log_parse_address (fp, ctrl);

   /* Extract packet length and packet id */
   decode_ulong2(fp, &pkt_id);
   decode_ulong2(fp, &pkt_len);
   printf ("\t%u\t%u", pkt_id, pkt_len);
  }
  break;
 case LOGTYPE_TX:
  printf ("Transmit ");
  {
   unsigned long pkt_id, pkt_len, assoc, lerrno;

   /* Extract address information */
   log_parse_address (fp, ctrl);

   /* Extract packet length and packet id */
   decode_ulong2(fp, &pkt_id);
   decode_ulong2(fp, &pkt_len);
   printf ("\t%u\t%u", pkt_id, pkt_len);
```

```
    }
    break;
   case LOGTYPE_SETUP:
    printf ("Setup ");
    break;
   case LOGTYPE_TEARDOWN:
    printf ("Teardown ");
    break;
   case LOGTYPE_ACCEPT:
    printf ("Accept ");
    {
     /* Extract address information */
     log_parse_address(fp, ctrl);

     /* Association is supplied */
     decode_ulong2(fp, &assoc);
     printf ("\tAssociation %d", assoc);
    }
    break;
   case LOGTYPE_ERROR:
    printf ("Error ");
    {
     /* Extract and print address information */
     log_parse_address (fp, ctrl);

     /* Extract Unix errno. */
     decode_ulong2(fp, &errno);
     printf ("\tError Entry: Unix Error Code %u",
      errno);

     /* log_parse_error (fi, (char) ctrl); */
    }
    break;
   default:
    fprintf (stderr, "log_parse: unknown record type\n");
  }
  /* Check for error */
  if (ctrl & LOGCTL_EXCEPT) {
   decode_ulong2 (fp, &errno);
   printf ("\tUnix Errno %d\n", errno);
  } else {
   putchar('\n');
  }
}

/*
 * Print either the association identification number or the
 * packet source-destination pair
 *
 */
```

```
log_parse_address(fp, ctrl)
 FILE  *fp;
 unsigned char ctrl;
{
 char buffer[30]; /* contains dotted decimal ip address + port */
 struct sockaddr_in sin;
 unsigned long assoc;

 if (ctrl & LOGCTL_ADDR) {
  fread (buffer, sizeof (sin.sin_port) + sizeof (sin.sin_addr),
   1, fp);
  ipport_btoaddr(buffer, &sin);
  ipport_addrtoa(&sin, buffer);
  printf ("\t%s", buffer);  /* Source address */
  if (ctrl & LOGCTL_2ADDR) {
   fread (buffer, sizeof (sin.sin_port) +
    sizeof (sin.sin_addr), 1, fp);
   ipport_btoaddr (buffer, &sin);
   ipport_addrtoa (&sin, buffer);
   printf ("\t%s", buffer); /* Destination address */
  }
 } else {
  decode_ulong2(fp, &assoc);
  printf ("\tAssociation %d", assoc);
 }
}
```

```
/*****************************************************************************
 *                                                                           *
 * File: decode.c                                                            *
 *                                                                           *
 * Encode and decode compressed integers.                                    *
 *                                                                           *
 * Written 11-Sep-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header: /tmp_mnt/net/usr.projectb/dartnet/src/tg/RCS/
decode.c,v 1.5 90/11/26 12:29:17 dlee Exp $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <math.h>
#include "config.h"
#include "distribution.h"
#include "protocol.h"
#include "decode.h"

/* Type definitions local to this file.     */

/* Functions exported from this file.     */

/* Functions local to this file.     */

/* Variables exported from this file.     */

/* Variables local to this file.     */


/* Decode an unsigned long from a buffer. The number is packed seven */
/* bits per byte in little-endian order, with the sign bit indicating */
/* that more is to come.      */

char *
decode_ulong(buf, n, len)

 char   *buf;
 int  *n;
 int  len;

 {
```

```
char  *bufend = &(buf[len]);
unsigned int curbyte;
int  shift = 0;
unsigned long tmp = 0;

/* Each pass though the following loop decodes one byte. */

for (;;)
 {

 /* Pick up the next byte.    */

 curbyte = *(buf++);

 /* Shift and mask it into the accumulated value. */

 tmp |= (curbyte & 0x7f) << shift;

 /* If the top bit is not set, this was the last byte. */

 if (curbyte <= 0x7f)
  break;

 /* Increment the shift count, scream if more than 32 */
 /* bits are to be read.     */

 shift += 7;
 if (shift > 32)
  {

  /*@@@@ Log bad response...   */

  (void)fprintf(stderr,
    "decode_ulong: bad format\n");
  abort();
  }

 /* If the encoded number runs off the end of the */
 /* buffer, return NULL so that the caller can try */
 /* again when he gets more input.   */

 if (buf >= bufend)
  return ((char *)NULL);
 }

/* Return the decoded number.     */

*n = tmp;
return (buf);
}
```

```
decode_ulong2(fp, result)

 FILE  *fp;
 int  *result; /* Integer is returned here */

 {
 unsigned int curbyte;
 int  shift = 0, rvalue;
 unsigned long tmp = 0;

 /* Each pass though the following loop decodes one byte. */

 for (; (rvalue = getc(fp)) != EOF;)
  {

  /* Pick up the next byte.    */

  curbyte = (char) rvalue;

  /* Shift and mask it into the accumulated value. */

  tmp |= (curbyte & 0x7f) << shift;

  /* If the top bit is not set, this was the last byte. */

  if (curbyte <= 0x7f)
   break;

  /* Increment the shift count, scream if more than 32 */
  /* bits are to be read.     */

  shift += 7;
  if (shift > 32)
   {

   /*@@@@ Log bad response...   */

   (void)fprintf(stderr,
     "decode_ulong: bad format\n");
   abort();
   }
  }
 if (rvalue == EOF) {
  return (-1);
 }
 /* Return the decoded number.     */

 *result = tmp;
 return (0);
 }
```

```
/* Encode a response request. This consists of the length of the */
/* desired response in bytes, followed by the number of bytes to skip */
/* in order to find the next response-length request. Lose synch, and */
/* you die! But does not require touching every byte of a long packet. */
/* Returns the length of the buffer consumed. If the buffer pointer */
/* is NULL, returns the maximum length of buffer that can be consumed. */

int
encode_response(buf, len, n)

 char  *buf;
 unsigned int len;
 unsigned int n;

 {
 char  *cp = buf;
 static int maxlen = -1;
 int  remainder;

 /* Set up maximum lengths if first time through.  */

 if (maxlen < 0)
  maxlen = 2 * (int)encode_ulong((char *)NULL, 0);

 /* Just return maximum length if NULL buffer.   */

 if (buf == NULL)
  return (maxlen);

 /* Encode the desired value and the length to skip.  */

 cp = encode_ulong(buf, n);
 remainder = len - (cp - buf);
 if (remainder <= 0)
  cp = encode_ulong(cp, 1);
 else
  cp = encode_ulong(cp, remainder);
 return (cp - buf);
 }


/* Encode a special response request. This consists of the length of */
/* the desired response in bytes, followed by a zero byte, followed by */
/* the number of bytes to skip in order to find the next  */
/* response-length request.     */

int
encode_special_response(buf, len, n)

 char  *buf;
 unsigned int len;
```

```
unsigned int n;

{
char  *cp = buf;
static int maxlen = -1;
int  remainder;

/* Set up maximum lengths if first time through.  */

if (maxlen < 0)
 maxlen = 2 * (int)encode_ulong((char *)NULL, 0) + 1;

/* Just return maximum length if NULL buffer.   */

if (buf == NULL)
 return (maxlen);

/* Encode the desired value and the length to skip.  */

cp = encode_ulong(buf, n);
*cp++ = 0x00;
remainder = len - (cp - buf);
if (remainder <= 0)
 cp = encode_ulong(cp, 1);
else
 cp = encode_ulong(cp, remainder);
return (cp - buf);
}


/* Encode an unsigned long. This consists of seven bits of number per */
/* byte of buffer, in little-endian order, with the sign bit indicating */
/* that more is to come.     */

char *
encode_ulong(buf, n)

 char  *buf;
 unsigned long n;

 {

 /* If no buffer, return maximum length.    */

 if (buf == NULL)
  return ((char *)5);

 /* Each pass through the following loop encodes seven bits, */
 /* low-order bits first.     */

 while (n > 127)
  {
```

```
 *(buf++) = (n & 0x7f) | 0x80;
 n >>= 7;
 }

/* Encode the last bits -- leave sign bit clear.  */

*(buf++) = n;
return (buf);
}
```

```
/*****************************************************************************
 *                                                                           *
 * File: prot_ipport.c                                                       *
 *                                                                           *
 * Convert ASCII IP address of the form a.b.c.d.port to                      *
 * sockaddr_in.                                                              *
 *                                                                           *
 * Written 04-Sep-90 by Paul E. McKenney, SRI International.                 *
 * Copyright (c) 1990 by SRI International.                                  *
 *                                                                           *
 *****************************************************************************/

#ifndef lint
static char rcsid[] = "$Header: /tmp_mnt/net/usr.projecte/dartnet/src/tg/sun4/
RCS/prot_ipport.c,v 1.5 90/11/26 12:29:40 dlee Exp Locker: denny $";
#endif lint

/* Include files.      */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/time.h>
#include <math.h>
#include "config.h"
#include "distribution.h"
#include "protocol.h"

/* Type definitions local to this file.    */

/* Functions exported from this file.    */

/* Functions local to this file.    */

/* Variables exported from this file.    */

/* Variables local to this file.    */



/* Convert an ascii address to a sockaddr.    */

int
ipport_atoaddr(addr, s)

 char  *addr;
 struct sockaddr *s;

 {
 struct sockaddr_in *sin = (struct sockaddr_in *)s;
```

```
 unsigned int a;
 unsigned int b;
 unsigned int c;
 unsigned int d;

 bzero((char *)s, sizeof(*s));
 sin->sin_family = AF_INET;
 if (sscanf(addr, "%u.%u.%u.%u.%hu",
   &a, &b, &c, &d, &(sin->sin_port)) != 5)
  return (0);
 if ((a > 255) ||
  (b > 255) ||
  (c > 255) ||
  (d > 255))
  return (0);
 sin->sin_addr.s_addr = (a << 24) | (b << 16) | (c << 8) | d;
 return (1);
 }


/* Convert a sockaddr structure to an ascii address.   */

int
ipport_addrtoa(s, addr)

 struct sockaddr *s;
 char  *addr;

 {
 struct sockaddr_in *sin = (struct sockaddr_in *)s;
 unsigned int a;
 unsigned int b;
 unsigned int c;
 unsigned int d;
 unsigned long ipaddr;

 ipaddr = sin->sin_addr.s_addr;
 d = ipaddr & 0xff;
 c = (ipaddr >>= 8) & 0xff;
 b = (ipaddr >>= 8) & 0xff;
 a = (ipaddr >>= 8) & 0xff;
 (void)sprintf(addr, "%u.%u.%u.%u.%u", a, b, c, d, sin->sin_port);
 return (1);
 }


/* Convert a binary log address to a sockaddr.    */

char *
ipport_btoaddr(addr, s)

 char  *addr;
```

```
 struct sockaddr *s;

 {
 struct sockaddr_in *sin = (struct sockaddr_in *)s;

 sin->sin_family = AF_INET;
 (void)bcopy(addr, (char *)&(sin->sin_port), sizeof(sin->sin_port));
 (void)bcopy((char *)&(addr[sizeof(sin->sin_port)]),
   (char *)&(sin->sin_addr.s_addr),
   sizeof(sin->sin_addr.s_addr));
 return (&(addr[sizeof(sin->sin_port) + sizeof(sin->sin_addr.s_addr)]));
 }


/* Convert a sockaddr binary log address.    */

char *
ipport_addrtob(s, addr)

 struct sockaddr *s;
 char  *addr;

 {
 struct sockaddr_in *sin = (struct sockaddr_in *)s;

 (void)bcopy((char *)&(sin->sin_port), addr, sizeof(sin->sin_port));
 (void)bcopy((char *)&(sin->sin_addr.s_addr),
   (char *)&(addr[sizeof(sin->sin_port)]),
   sizeof(sin->sin_addr.s_addr));
 return (&(addr[sizeof(sin->sin_port) + sizeof(sin->sin_addr.s_addr)]));
 }
```

3 SOURCE FOR TG ANALYSIS TOOLS

This section contains the four perl scripts used in the analysis of the data presented in Volume 2 of the final report. The scripts involve two protocols, ST-II and UDP, and each protocol has two scripts. One script generates a textual summary of statistics, while the other script creates files suitable for graphing by grtool. The statistics generated by the scripts include average offer rate, average throughput, average delay, and delay variance. The files for graphing contain data that can become a graph of delay, including a scatter diagram of the dropped packets, a graph of the average offer rate on a 10-second interval, and a graph of the average throughput on a 10-second interval. These scripts are tailored for a network whose link speed is 1.344 Mb/s, but this speed can easily be changed. Note that the scripts have been used and tested only for a constant packet size. See the comments at the beginning of each script for more details.

```
#!/usr/local/bin/perl

# st2stats.perl

# read compressed TG client and server log files for ST-II traffic and
# produce summary statistics including average offer rate, average
# throughput, average delay, and delay variance for the experiment. The
# offer rate and throughput are also recorded for each 10 second period.
# To change the period, modify the variable $period in the routine
# thruput. Note: these scripts assume a network bandwidth 1.344 Mgb/s and
# have only been used where the packet size is constant. To change the
# link capacity, modify the variable $byte_rate. Requires client.log.Z,
# server.log.Z, and st2.c.tg (tg script file for the client).

# Copyright (c) 1993 SRI International. All rights reserved.

# Redistribution and use in source and binary forms are permitted
# provided that the above copyright notice and this paragraph are
# duplicated in all such forms and that any documentation,
# advertising materials, and other materials related to such
# distribution and use acknowledge that the software was developed
# by SRI International, Menlo Park, CA. The name SRI International
# may not be used to endorse or promote products derived from this
# software without specific prior written permission.

# THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
# IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
# WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.



$Done = 0;
$byte_rate = 168000.0; # 192000 for full speed T1
$script_file = "st2.c.tg";
$client_file = "client.log.Z";
$server_file = "server.log.Z";
$dcat = "dcat";
$dir = 'basename \'pwd\'';
chop $dir;


if ($#ARGV > 0) { # $#ARGV counts starting with 0 after command name!
 printf("Usage: %s\n",$0);
 exit(0);
}

select((select(STDOUT), $| = 1)[$[]); # unbuffer stdout

$total_exp_rbytes = 0;
$total_exp_xbytes = 0;
$total_exp_delay = 0;
$variance = 0;
```

```
&get_rates();
&open_infiles();
&get_skew();

$pseqno = -1;
while(&next_rcv()) {
 # print $_;
 # find transmitted pkt matching sequence number
 for (;;) {
  $prev_xmt_time = $save_xmt_time;
  $_ = <CLIENT>; # read next client line
  ($xtime,$xdirection,$xhost,$xseqno,$xlength) = split(' ');
   if ($xdirection eq 'Teardown') {
  $last_xmt_time = $prev_xmt_time;
 }

  next if ($xdirection ne 'Transmit');
  $total_exp_xbytes += $xlength + 16;
  $save_xmt_time = $xtime;
  &thruput();
  last if ($xseqno >= $rseqno);
 }

 if ($rseqno != $xseqno) { # out of order!!
  printf("seqno %d OUT OF ORDER!\n",$rseqno);
  next;
 }
 $delay = $rtime - $xtime - $skew;
 &num_packets();
 $total_exp_rbytes += $rlength + 16;
 $total_exp_delay += $delay;
}

printf("Done with RCVS last=%d\n",$rseqno);
while (<CLIENT>) { # drain client xmt data
 $prev_xmt_time = $save_xmt_time;
 ($xtime, $xdirection, $xhost, $xseqno, $xlength) = split(' ');
 $save_xmt_time = $xtime;
 if ($xdirection eq 'Teardown') {
 $last_xmt_time = $prev_xmt_time;
}
 next if ($xdirection ne 'Transmit');
#  &num_packets(); bug??? only used for average so don't want to count
#probably need this here
 $total_exp_xbytes += $xlength + 16;
 &thruput();
}
printf("Done with XMTS last=%d\n",$pxseqno);

# force length change to generate a tp_per_length report
$rlength = 1;
```

```
$Done = 1;
&thruput();
&close_files();

$XXX =  $total_exp_xbytes/($last_xmt_time - $first_xmt_time);
$XXX = $XXX/$byte_rate;
$PPP = $total_exp_delay/$num_packets;
$RRR = $total_exp_rbytes/($last_rcv_time - $first_rcv_time);
$RRR = $RRR/$byte_rate;


printf("\nAverages: TP=%0.4lf OR=%0.4lf DELAY=%0.4lf \n",
$RRR, $XXX, $PPP);

$AV_PKT_RCV = $total_rcv_pkts/($last_rcv_time - $first_rcv_time);
$AV_PKT_XMT = $total_xmt_pkts/($last_xmt_time - $first_xmt_time);

&get_variance();

$variance = $variance/($num_packets - 1);
printf("Variance=%0.4lf \n", $variance);


printf("\nAverage Pkt Offer Rate %0.4lf, Average Pkt Rcv Rate %0.4lf\n",
$AV_PKT_XMT,
$AV_PKT_RCV);


sub get_variance { #computes delay variance

&init_variables();
&open_infiles();
&set_file_ptr();

$pseqno = -1;
printf("skew=%d\n",$skew);
while(&next_rcv()) {
 # print $_;
 # find transmitted pkt matching sequence number
 for (;;) {
  $_ = <CLIENT>; # read next client line
  ($xtime,$xdirection,$xhost,$xseqno,$xlength) = split(' ');
  next if ($xdirection ne 'Transmit');
  &thruput();
  last if ($xseqno >= $rseqno);
 }
 if ($rseqno != $xseqno) { # out of order!!
  printf("rseqno %d xseqno %d OUT OF ORDER!\n",$rseqno,$xseqno);
  next;
 }
 $delay = $rtime - $xtime - $skew;
 &num_packets();
```

```
  $variance = $variance + (($delay - $PPP)**2);

 }
 printf("Done with RCVS last=%d\n",$rseqno);
 while (<CLIENT>) { # drain client xmt data
  ($xtime, $xdirection, $xhost, $xseqno, $xlength) = split(' ');
  next if ($xdirection ne 'Transmit');
  &thruput();
 }
 printf("Done with XMTS last=%d\n",$pxseqno);

 # force length change to generate a tp_per_length report
 $rlength = 1;
 $Done = 1;
 &thruput();
 &close_files();

}

sub init_variables { #initialize variables for second pass of log file
$delay = 0;
$rseqno = 0;
$rtime = 0;
$rlength = 0;
$xseqno = 0;
$xtime = 0;
$num_packets = 0;
$save_seqno = 0;
$slength = 0;
$stime = 0;
$save_rcv_time = 0;
$prev_rcv_time = 0;
$sseqno = 0;
$pseqno = 0;
$temp = 0;
$total_exp_delay = 0;
$endOfSec = 0;
$Done = 0;
$rcv_exhausted = 0;
$next_rcv_init = 0;
$total_rcv_pkts = 0;
$total_xmt_pkts = 0;
}


sub next_rcv { # does local 2 line sort on seqno for server log

 if ($next_rcv_init == 0) { # first time read an extra line
  $next_rcv_init++;
  &next_rcv_line();
 }
```

```
$rtime = $stime;
$rlength = $slength;
$rseqno = $sseqno;


if(! &next_rcv_line()) { # read next line and test for end of file
 if (($save_seqno) && ($sseqno != 6) &&($slength !=0)){
   # end of file, saved one left

  $rtime = $stime;
  $rlength = $slength;
  $rseqno = $save_seqno;
  $save_seqno = 0;
  return 1;
 }
 else { # file and saved line exhausted
  return 0;
 }
}

if ($pseqno >= $sseqno) {
 printf("\nseqno %d precedes %d\n",$pseqno,$sseqno);
}
$pseqno = $sseqno;

# now have two pkts to choose from, use the one with lower seqno
if ($sseqno < $rseqno) {
 $temp = $rtime;
 $rtime = $stime;
 $stime = $temp;

 $temp = $rlength;
 $rlength = $slength;
 $slength = $temp;

 $temp = $rseqno;
 $rseqno = $sseqno;
 $sseqno = $temp;
}
$save_seqno = $sseqno;

return 1;
}

sub next_rcv_line { # gets next Receive line from server log
 return 0 if $rcv_exhausted;
 while (<SERVER>) {
  $prev_rcv_time = $save_rcv_time;
  ($stime, $sdirection, $shost, $sseqno, $slength) = split(' ');
  $save_rcv_time = $stime;
 if (($sdirection eq 'Receive') && ($sseqno == 6) && ($slength == 0)) {
  $last_rcv_time = $prev_rcv_time;
```

```
    return 0;
  }
    if (($sseqno == 0) && ($sdirection eq 'Receive')) {
    $first_rcv_time = $stime;
  }
   return 1 if ($sdirection eq 'Receive');
  }
  $rcv_exhausted = 1;
  return 0;
}

sub num_packets { # increments packet count - packets have to be successfully
received
 $num_packets++;
 if(!($num_packets % 100)) {
  printf(".");
  if(!($num_packets % 10000)) {
   printf("n_xmt_pkts=%d rate=%d\n",$num_packets,$rate);
  }
 }
}

sub close_files {
 close(SERVER);
 close(CLIENT);
}

sub thruput { # compute/print thruput per length, sec and offerrate per sec
 $period = 10.0; # length of time period for each thruput value

 if ($endOfSec == 0) { # initial call
  $endOfSec = $xtime + $period;
  $sec = 0;
  $prseqno = -1;

  $t1oflength = $xtime;
  $plength = $rlength;
  $first_xmt_time = $xtime;
 }
 else {
  $this_period = $period;
  if($Done) {
   printf(" Last xtime = $last_xmt_time, End of Section = $endOfSec \n");
   $ShortTime = ($endOfSec - $last_xmt_time);
   $endOfSec = $last_xmt_time;
   $this_period = $period - $ShortTime;
   printf(" Last Period Short by %2.4f seconds\n",$ShortTime);
   printf("\n");
  }
  if($xtime >= $endOfSec) {
   $sec++;
   $endOfSec += $period;
```

```
  $rcvLastSec = $RcvBytesThisSec/($byte_rate * $this_period);
  $xmtLastSec = $XmtBytesThisSec/($byte_rate * $this_period);
  printf("period=%d tp=%f or=%f\n",
   $sec,$rcvLastSec,$xmtLastSec);

  $RcvBytesThisSec = 0;
  $XmtBytesThisSec = 0;
 }
}

$XmtBytesThisSec += $xlength + 16;
$total_xmt_pkts++;
$pxseqno = $xseqno;


return if ($rseqno == $prseqno);

# rcvd a pkt at server!

$prseqno = $rseqno;
$RcvBytesThisSec += $rlength + 16;
$total_rcv_pkts++;

# if new packet length, summarize previous stats
if ($rlength != $plength) {
 $tptime = $xtime - $t1oflength; #thruput time of length
 if ($tptime == 0.0) {
  $thruput = 0.0;
 }
 else {
  $thruput = $rcvBytesThisLength / ($tptime * $byte_rate);
 }
 $t1oflength = $xtime;
 $rcvBytesThisLength = 0;

 # rates increase only when lengths decrease
 if ($rlength < $plength) {
   $rate = ++$rate % ($#rates + 1);
 }
 $plength = $rlength;
}
$rcvBytesThisLength += $rlength + 16;
}

sub get_rates {
 $#rates = 0;
 unless (open(SCRIPT,$script_file)) {
  warn "Cannot open client script $script_file: $!\n";
  return;
 }
 while(<SCRIPT>) {
  ($f1,$f2,$f3,$f4) = split(' ');
```

```
  next unless ($f1 eq "arrival");
  if ($f2 eq "uniform") {
   printf("%s \n",$f3/2);
   push(rates,$f3/2);
  }
  else {
   printf("%s \n",$f2);
   push(rates,$f2);
  }
 }
 close(SCRIPT);
}

sub open_infiles {
 $command = "zcat $client_file | $dcat |";
 open(CLIENT,$command) || die "Can't open $client_file: $!\n";
 $command = "zcat $server_file | $dcat |";
 open(SERVER,$command) || die "Can't open $server_file: $!\n";
}


sub get_skew {
 # find client/server log clock skew
 for (;;) {
  $_ = <SERVER>;
  ($f1,$f2,$f3,$f4,$f5,$f6,$f7) = split(' ');
  last if ($f6 eq 'epoch');
 }
 $skew = $f7;
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2,$f3,$f4,$f5,$f6,$f7) = split(' ');
  last if ($f6 eq 'epoch');
 }
 $skew = $f7 - $skew;
 printf("skew=%d\n",$skew);

 # now look for "Program start time: Fri Nov 15 10:21:05 1991"
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2,$f3,$f4,$f5,$f6,$f7) = split(' ');
  last if ($f1 eq 'Program'); # must be next line
 }
 s/Program start time://; # chop off front
 $date = $_;
 chop $date;
return;
 # find client/server log clock skew (based on 'Setup')
 for (;;) {
  $_ = <SERVER>;
  ($f1,$f2) = split(' ');
  last if ($f2 eq 'Setup');
```

```
 }
 $skew = $f1;
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2) = split(' ');
  last if ($f2 eq 'Setup');
 }
 $skew = $f1 - $skew;
 printf("skew=%d\n",$skew);

}

sub set_file_ptr{ #sets the file to the setup packet
 for (;;) {
  $_ = <SERVER>;
  ($f1,$f2) = split(' ');
  last if ($f2 eq 'Setup');
 }
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2) = split(' ');
  last if ($f2 eq 'Setup');
 }
}
```

```
#!/usr/local/bin/perl

# st2tables.perl

# read compressed TG client and server log files for ST-II traffic and
# produce tables to be used by grtool. The dltim.tbl file contains the
# per packet delay and a scatter diagram of the dropped packets below
# the X-axis. The or.tbl file contains the average offer rate (fraction
# of 1.344 Mgb/s) for 10 second intervals. The tptim.tbl file contains
# the average throughput for 10 second intervals. It is expected the
# output in or.tbl and tptim.tbl will be displayed in one graph so
# or.tbl does include any headings. The vc_or.tbl file contains the
# average offer rate (packets per second) for 10 second intervals. The
# vc_tptim.tbl file contains the average received rate (packets per
# second) for 10 second intervals. It is expected the output in
# vc_or.tbl and vc_tptim.tbl will be displayed in one graph so vc_or.tbl
# does include any headings. To change this interval, modify the
# variable $period in the routine thruput. Note: these scripts assume a
# network bandwidth 1.344 Mgb/s and have only been used where the packet
# size is constant. To change the link capacity, modify the variable
# $byte_rate. Requires client.log.Z, server.log.Z and st2.c.tg (tg
# script file for the client). If a readme file is present, the first
# line from this file will be used as the graph title.


# Copyright (c) 1993 SRI International. All rights reserved.

# Redistribution and use in source and binary forms are permitted
# provided that the above copyright notice and this paragraph are
# duplicated in all such forms and that any documentation,
# advertising materials, and other materials related to such
# distribution and use acknowledge that the software was developed
# by SRI International, Menlo Park, CA. The name SRI International
# may not be used to endorse or promote products derived from this
# software without specific prior written permission.

# THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
# IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
# WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.


 $Done = 0;
 $byte_rate = 168000.0; # 192000 for full speed T1
 $script_file = "st2.c.tg";
 $client_file = "client.log.Z";
 $server_file = "server.log.Z";
 $dcat = "dcat";
 $dir = 'basename \'pwd\'';
 chop $dir;
```

```
if ($#ARGV > 0) { # $#ARGV counts starting with 0 after command name!
 printf("Usage: %s\n",$0);
 exit(0);
}

select((select(STDOUT), $| = 1)[$[]); # unbuffer stdout

$total_exp_rbytes = 0;
$total_exp_xbytes = 0;
$total_exp_delay = 0;

&get_rates();
&open_infiles();
&get_skew();
&open_outfiles();

$pseqno = -1;
while(&next_rcv()) {
 # print $_;
 # find transmitted pkt matching sequence number
 for (;;) {
  $prev_xmt_time = $save_xmt_time;
  $_ = <CLIENT>; # read next client line
  ($xtime,$xdirection,$xhost,$xseqno,$xlength) = split(' ');
   if ($xdirection eq 'Teardown') {
  $last_xmt_time = $prev_xmt_time;
}
  next if ($xdirection ne 'Transmit');
  $total_exp_xbytes += $xlength + 16;
#  printf("!");
  $save_xmt_time = $xtime;
  &thruput();
  last if ($xseqno >= $rseqno);
  printf (DLYTIM "%0.41f\n",-0.2 + rand(.05));
 }

 if ($rseqno != $xseqno) { # out of order!!
  printf("seqno %d OUT OF ORDER!\n",$rseqno);
  next;
 }
 $delay = $rtime - $xtime - $skew;
 printf (DLYTIM "%0.61f\n",$delay);
 &num_packets();
 $total_exp_rbytes += $rlength + 16;
 $total_exp_delay += $delay;
}

printf("Done with RCVS last=%d\n",$rseqno);
while (<CLIENT>) { # drain client xmt data
 $prev_xmt_time = $save_xmt_time;
 ($xtime, $xdirection, $xhost, $xseqno, $xlength) = split(' ');
 $save_xmt_time = $xtime;
```

```
 if ($xdirection eq 'Teardown') {
 $last_xmt_time = $prev_xmt_time;
}
 next if ($xdirection ne 'Transmit');
#  &num_packets(); bug.....shouldn't count the ones at the end
 &thruput();
#probably need this here
 $total_exp_xbytes += $xlength + 16;
 printf (DLYTIM "%0.41f\n",-0.2 + rand(.05));
}
printf("Done with XMTS last=%d\n",$pxseqno);

# force length change to generate a tp_per_length report
$rlength = 1;
$Done = 1;
&thruput();
&close_files();


sub next_rcv { # does local 2 line sort on seqno

if ($next_rcv_init == 0) { # first time read an extra line
 $next_rcv_init++;
 &next_rcv_line();
}

$rtime = $stime;
$rlength = $slength;
$rseqno = $sseqno;

if(! &next_rcv_line()) { # read next line and test for end of file
 if (($save_seqno) && ($sseqno != 6) && ($slength != 0)){
   # end of file, saved one left

  $rtime = $stime;
  $rlength = $slength;
  $rseqno = $save_seqno;
  $save_seqno = 0;
  return 1;
 }
 else { # file and saved line exhausted
  return 0;
 }
}

if ($pseqno >= $sseqno) {
 printf("\nseqno %d precedes %d\n",$pseqno,$sseqno);
 printf(BADSEQ "seqno %d preceeds %d\n",$pseqno,$sseqno);
}
$pseqno = $sseqno;

# now have two pkts to choose from, use the one with lower seqno
```

```
 if ($sseqno < $rseqno) {
  $temp = $rtime;
  $rtime = $stime;
  $stime = $temp;

  $temp = $rlength;
  $rlength = $slength;
  $slength = $temp;

  $temp = $rseqno;
  $rseqno = $sseqno;
  $sseqno = $temp;
 }
 $save_seqno = $sseqno;

 return 1;
}

sub next_rcv_line {
 return 0 if $rcv_exhausted;
 while (<SERVER>) {
  $prev_rcv_time = $save_rcv_time;
  ($stime, $sdirection, $shost, $sseqno, $slength) = split(' ');
  $save_rcv_time = $stime;
 if (($sdirection eq 'Receive') && ($sseqno == 6) && ($slength == 0)) {
  $last_rcv_time = $prev_rcv_time;
  return 0;
 }
   if (($sseqno == 0) && ($sdirection eq 'Receive')) {
   $first_rcv_time = $stime;
 }
  return 1 if ($sdirection eq 'Receive');
 }
 $rcv_exhausted = 1;
 return 0;
}

sub num_packets {
 $num_packets++;
 if(!($num_packets % 100)) {
  printf(".");
  if(!($num_packets % 10000)) {
   printf("n_xmt_pkts=%d rate=%d\n",$num_packets,$rate);
  }
 }
}

sub close_files {
 close(SERVER);
 close(CLIENT);
 close(DLYTIM);
 close(TPTIM);
```

```
close(ORTBL);
close(VC_ORTBL);
close(VC_TPTIM);
close(BADSEQ);
}

sub thruput { # compute/print thruput per length, sec and offerrate per sec
 $period = 10.0; # length of time period for each thruput value
 if ($endOfSec == 0) { # initial call
  $endOfSec = $xtime + $period;
  $sec = 0;
  $prseqno = -1;

  $t1oflength = $xtime;
  $plength = $rlength;
  $first_xmt_time = $xtime;
 }
 else {
  $this_period = $period;
  if($Done) {
   printf(" Last xtime = $last_xmt_time, End of Section = $endOfSec \n");
   $ShortTime = ($endOfSec - $last_xmt_time);
   $endOfSec = $last_xmt_time;
   $this_period = $period - $ShortTime;
   printf(" Last Period Short by %2.4f seconds\n",$ShortTime);
   printf("\n");
  }
  if($xtime >= $endOfSec) {
   $sec++;
   $endOfSec += $period;
   $rcvLastSec = $RcvBytesThisSec/($byte_rate * $this_period);
   $xmtLastSec = $XmtBytesThisSec/($byte_rate * $this_period);
   $rcvPktsLastSec = $RcvPktsThisSec/$this_period;
   $xmtPktsLastSec = $XmtPktsThisSec/$this_period;
   printf("period=%d tp=%f or=%f\n",
    $sec,$rcvLastSec,$xmtLastSec);
   $posttime = $sec * $period;
   printf(TPTIM "%f %f %f\n",
    $endOfSec,$rcvLastSec,$xmtLastSec);
   printf(ORTBL "%f %f %d\n",
    $endOfSec,$xmtLastSec,$posttime);
   printf(VC_ORTBL "%f %f %d\n",
    $endOfSec,$xmtPktsLastSec,$posttime);
   printf(VC_TPTIM "%f %f %f\n",
    $endOfSec,$rcvPktsLastSec,$posttime);

   $RcvBytesThisSec = 0;
   $XmtBytesThisSec = 0;
   $RcvPktsThisSec = 0;
   $XmtPktsThisSec = 0;
  }
 }
```

```
 $XmtBytesThisSec += $xlength + 16;
 $XmtPktsThisSec++;
 $pxseqno = $xseqno;


 return if ($rseqno == $prseqno);

 # rcvd a pkt at server!
 $prseqno = $rseqno;
 $RcvBytesThisSec += $rlength + 16;
 $RcvPktsThisSec++;

 # if new packet length, summarize previous stats
 if ($rlength != $plength) {
  $tptime = $xtime - $t1oflength; #thruput time of length
  if ($tptime == 0.0) {
   $thruput = 0.0;
  }
  else {
   $thruput = $rcvBytesThisLength / ($tptime * $byte_rate);
  }
#  printf("len=%d rate=%f thruput=%f\n",
#   $plength,@rates[$rate],$thruput);
  $t1oflength = $xtime;
  $rcvBytesThisLength = 0;

  # rates increase only when lengths decrease
  if ($rlength < $plength) {
    $rate = ++$rate % ($#rates + 1);
  }
  $plength = $rlength;
 }
 $rcvBytesThisLength += $rlength + 16;
}

sub get_rates {
 $#rates = 0;
 unless (open(SCRIPT,$script_file)) {
  warn "Cannot open client script $script_file: $!\n";
  return;
 }
 while(<SCRIPT>) {
  ($f1,$f2,$f3,$f4) = split(' ');
  next unless ($f1 eq "arrival");
  if ($f2 eq "uniform") {
   printf("%s \n",$f3/2);
   push(rates,$f3/2);
  }
  else {
   printf("%s \n",$f2);
   push(rates,$f2);
  }
```

```
 }
 close(SCRIPT);
}

sub open_infiles {
 $command = "zcat $client_file | $dcat |";
 open(CLIENT,$command) || die "Can't open $client_file: $!\n";
 $command = "zcat $server_file | $dcat |";
 open(SERVER,$command) || die "Can't open $server_file: $!\n";
}

sub open_outfiles {
 # is there a readme file with a Title?
 if (-e "readme") {
  $title = 'head -1 readme';
  chop($title);
  printf("title=$title\n");
 }
 open(DLYTIM, >dlytim.tbl") || warn "Can't open dlytim.tbl: $!\n";
 printf(DLYTIM "\@title $title\n") if ($title);
 printf(DLYTIM "\@subtitle Delay vs Experiment Time \[$dir:$date\]\n");
 printf(DLYTIM "\@xlabel Packets Sent\n");
 printf(DLYTIM "\@ylabel Delay (seconds) \n");
 printf(DLYTIM "\@setprops 0 0 1 1\n");


 open(TPTIM,">tptim.tbl") || warn "Can't open tp.tbl: $!\n";
 select((select(TPTIM), $| = 1)[$[]); # unbuffer file output
 printf(TPTIM "\@title $title\n") if ($title);
 printf(TPTIM "\@subtitle OfferRate,Thruput \[$dir:$date\]\n");
 printf(TPTIM "\@xlabel Time (seconds)\n");
 printf(TPTIM "\@ylabel Fraction of 1.344 Mbps \n");
 # dual data set, must use "grtool -n"
 printf(TPTIM "\@setprops 0 1 7 1\n");

 open(VC_TPTIM,">vc_tptim.tbl") || warn "Can't open vc_tptim.tbl: $!\n";
 select((select(VC_TPTIM), $| = 1)[$[]); # unbuffer file output
 printf(VC_TPTIM "\@title $title\n") if ($title);
 printf(VC_TPTIM "\@subtitle Offer Rate,Receive Rate \[$dir:$date\]\n");
 printf(VC_TPTIM "\@xlabel Time (seconds)\n");
 printf(VC_TPTIM "\@ylabel Packets Per Second \n");
 # dual data set, must use "grtool -n"
 printf(VC_TPTIM "\@setprops 0 1 7 1\n");

 open(VC_ORTBL,">vc_or.tbl") || warn "Can't open vc_or.tbl: $!\n";
 select((select(VC_ORTBL), $| = 1)[$[]); # unbuffer file output
# printf(VC_ORTBL "\@title $title\n") if ($title);
# printf(VC_ORTBL "\@subtitle OfferRate \[$dir:$date\]\n");
# printf(VC_ORTBL "\@xlabel Time (seconds)\n");
# printf(VC_ORTBL "\@ylabel Packets Per Second \n");
 # dual data set, must use "grtool -n"
# printf(VC_ORTBL "\@setprops 0 1 7 1\n");
```

```
 open(ORTBL,">or.tbl") || warn "Can't open or.tbl: $!\n";
 select((select(ORTBL), $| = 1)[$[]); # unbuffer file output
# printf(ORTBL "\@title $title\n") if ($title);
# printf(ORTBL "\@subtitle OfferRate,Thruput \[$dir:$date\]\n");
# printf(ORTBL "\@xlabel Time (seconds)\n");
# printf(ORTBL "\@ylabel Fraction of 1.344 Mbps \n");
 # dual data set, must use "grtool -n"
# printf(ORTBL "\@setprops 0 1 7 1\n");


 open(BADSEQ,">badseq.tbl") || warn "Can't open badseq.tbl: $!\n";
 printf(BADSEQ "\@title $title\n") if ($title);
 printf(BADSEQ "\@subtitle Out of Sequence Pkts \[$dir:$date\]\n");
}


sub get_skew {
 # find client/server log clock skew
 for (;;) {
  $_ = <SERVER>;
  ($f1,$f2,$f3,$f4,$f5,$f6,$f7) = split(' ');
  last if ($f6 eq 'epoch');
 }
 $skew = $f7;
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2,$f3,$f4,$f5,$f6,$f7) = split(' ');
  last if ($f6 eq 'epoch');
 }
 $skew = $f7 - $skew;
 printf("skew=%d\n",$skew);

 # now look for "Program start time: Fri Nov 15 10:21:05 1991"
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2,$f3,$f4,$f5,$f6,$f7) = split(' ');
  last if ($f1 eq 'Program'); # must be next line
 }
 s/Program start time://; # chop off front
 $date = $_;
 chop $date;
return;
 # find client/server log clock skew (based on 'Setup')
 for (;;) {
  $_ = <SERVER>;
  ($f1,$f2) = split(' ');
  last if ($f2 eq 'Setup');
 }
 $skew = $f1;
```

```
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2) = split(' ');
  last if ($f2 eq 'Setup');
 }
 $skew = $f1 - $skew;
 printf("skew=%d\n",$skew);

}
```

```
#!/usr/local/bin/perl

# udpstats.perl

# read compressed TG client and server log files for UDP traffic and
# produce summary statistics including average offer rate, average
# throughput, average delay, delay variance for the experiment. The
# offer rate and throughput are also recorded for each 10 second period.
# To change the period, modify the variable $period in the routine
# thruput. Note: these scripts assume a network bandwidth 1.344 Mgb/s and
# have only been used where the packet size is constant. To change the
# link capacity, modify the variable $byte_rate. Requires client.log.Z,
# server.log.Z and udp.c.tg (tg script file for the client).

# Copyright (c) 1993 SRI International. All rights reserved.

# Redistribution and use in source and binary forms are permitted
# provided that the above copyright notice and this paragraph are
# duplicated in all such forms and that any documentation,
# advertising materials, and other materials related to such
# distribution and use acknowledge that the software was developed
# by SRI International, Menlo Park, CA. The name SRI International
# may not be used to endorse or promote products derived from this
# software without specific prior written permission.

# THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
# IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
# WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.




$Done = 0;
$byte_rate = 168000.0; # 192000 for full speed T1
$script_file = "udp.c.tg";
$client_file = "client.log.Z";
$server_file = "server.log.Z";
$dcat = "dcat";
$dir = 'basename \'pwd\'';
chop $dir;


if ($#ARGV > 0) { # $#ARGV counts starting with 0 after command name!
 printf("Usage: %s\n",$0);
 exit(0);
}

select((select(STDOUT), $| = 1)[$[]); # unbuffer stdout
```

```
$total_exp_rbytes = 0;
$total_exp_xbytes = 0;
$total_exp_delay = 0;
$variance = 0;

&get_rates();
&open_infiles();
&get_skew();

$pseqno = -1;
while(&next_rcv()) {
 # print $_;
 # find transmitted pkt matching sequence number
 for (;;) {
  $prev_xmt_time = $save_xmt_time;
  $_ = <CLIENT>; # read next client line
  ($xtime,$xdirection,$xhost,$xseqno,$xlength) = split(' ');
   if ($xdirection eq 'Teardown') {
  $last_xmt_time = $prev_xmt_time;
 }
  next if ($xdirection ne 'Transmit');
  $total_exp_xbytes += $xlength + 32;
  $save_xmt_time = $xtime;
  &thruput();
  last if ($xseqno >= $rseqno);
 }

 if ($rseqno != $xseqno) { # out of order!!
  printf("seqno %d OUT OF ORDER!\n",$rseqno);
  next;
 }
 $delay = $rtime - $xtime - $skew;
 &num_packets();
 $total_exp_rbytes += $rlength + 32;
 $total_exp_delay += $delay;
}

printf("Done with RCVS last=%d\n",$rseqno);
while (<CLIENT>) { # drain client xmt data
 $prev_xmt_time = $save_xmt_time;
 ($xtime, $xdirection, $xhost, $xseqno, $xlength) = split(' ');
 $save_xmt_time = $xtime;

 if ($xdirection eq 'Teardown') {
 $last_xmt_time = $prev_xmt_time;
}

 next if ($xdirection ne 'Transmit');
 $total_exp_xbytes += $xlength + 32;
#  &num_packets(); bug??? only used for average so don't want to count
 &thruput();
}
```

```
printf("Done with XMTS last=%d\n",$pxseqno);

# force length change to generate a tp_per_length report
$rlength = 1;
$Done = 1;
&thruput();
&close_files();


$XXX =  $total_exp_xbytes/($last_xmt_time - $first_xmt_time);
$XXX = $XXX/$byte_rate;
$PPP = $total_exp_delay/$num_packets;
$RRR = $total_exp_rbytes/($last_rcv_time - $first_rcv_time);
$RRR = $RRR/$byte_rate;

printf("\nAverages: TP=%0.4lf OR=%0.4lf DELAY=%0.4lf \n",$RRR, $XXX, $PPP);

&get_variance();

$variance = $variance/($num_packets - 1);
printf("Variance=%0.4lf \n", $variance);


sub get_variance { #computes delay variance

&init_variables();
&open_infiles();
&set_file_ptr();

$pseqno = -1;
printf("skew=%d\n",$skew);
while(&next_rcv()) {
 # print $_;
 # find transmitted pkt matching sequence number
 for (;;) {
  $_ = <CLIENT>; # read next client line
  ($xtime,$xdirection,$xhost,$xseqno,$xlength) = split(' ');
  next if ($xdirection ne 'Transmit');
  &thruput();
  last if ($xseqno >= $rseqno);
 }
 if ($rseqno != $xseqno) { # out of order!!
  printf("rseqno %d xseqno %d OUT OF ORDER!\n",$rseqno,$xseqno);
  next;
 }
 $delay = $rtime - $xtime - $skew;
 &num_packets();
 $variance = $variance + (($delay - $PPP)**2);

}
printf("Done with RCVS last=%d\n",$rseqno);
while (<CLIENT>) { # drain client xmt data
```

```
  ($xtime, $xdirection, $xhost, $xseqno, $xlength) = split(' ');
  next if ($xdirection ne 'Transmit');
  &thruput();
 }
 printf("Done with XMTS last=%d\n",$pxseqno);

 # force length change to generate a tp_per_length report
 $rlength = 1;
 $Done = 1;
 &thruput();
 &close_files();

}

sub init_variables { #initialize variables for second pass of log file
$delay = 0;
$rseqno = 0;
$rtime = 0;
$rlength = 0;
$xseqno = 0;
$xtime = 0;
$num_packets = 0;
$save_seqno = 0;
$slength = 0;
$stime = 0;
$sseqno = 0;
$pseqno = 0;
$temp = 0;
$total_exp_delay = 0;
$endOfSec = 0;
$Done = 0;
$rcv_exhausted = 0;
$next_rcv_init = 0;
}



sub next_rcv { # does local 2 line sort on seqno for server log

 if ($next_rcv_init == 0) { # first time read an extra line
  $next_rcv_init++;
  &next_rcv_line();
 }

 $rtime = $stime;
 $rlength = $slength;
 $rseqno = $sseqno;

 if(! &next_rcv_line()) { # read next line and test for end of file
  if ($save_seqno) { # end of file, saved one left
   $rtime = $stime;
   $rlength = $slength;
```

```
   $rseqno = $save_seqno;
   $save_seqno = 0;
   return 1;
  }
  else { # file and saved line exhausted
   return 0;
  }
 }

 if ($pseqno >= $sseqno) {
  printf("\nseqno %d precedes %d\n",$pseqno,$sseqno);
 }
 $pseqno = $sseqno;

 # now have two pkts to choose from, use the one with lower seqno
 if ($sseqno < $rseqno) {
  $temp = $rtime;
  $rtime = $stime;
  $stime = $temp;

  $temp = $rlength;
  $rlength = $slength;
  $slength = $temp;

  $temp = $rseqno;
  $rseqno = $sseqno;
  $sseqno = $temp;
 }
 $save_seqno = $sseqno;

 return 1;
}

sub next_rcv_line { # gets next Receive line from server log
 return 0 if $rcv_exhausted;
 while (<SERVER>) {
  ($stime, $sdirection, $shost, $sseqno, $slength) = split(' ');
   if ( $sdirection eq 'Receive') {
   if ($sseqno == 0) {
  $first_rcv_time = $stime;
 }
   $save_time = $stime;
  }

  return 1 if ($sdirection eq 'Receive');
 }
 $last_rcv_time = $save_time;
 $rcv_exhausted = 1;
 return 0;
}

sub num_packets { #increments packet count - packets have to be successfully
```

```
received
 $num_packets++;
 if(!($num_packets % 100)) {
  printf(".");
  if(!($num_packets % 10000)) {
   printf("n_xmt_pkts=%d rate=%d\n",$num_packets,$rate);
  }
 }
}

sub close_files {
 close(SERVER);
 close(CLIENT);
}

sub thruput { # compute/print thruput per length, sec and offerrate per sec
 $period = 10.0; # length of time period for each thruput value

 if ($endOfSec == 0) { # initial call
  $endOfSec = $xtime + $period;
  $sec = 0;
  $prseqno = -1;

  $tloflength = $xtime;
  $plength = $rlength;
  $first_xmt_time = $xtime;
 }
 else {
  $this_period = $period;
  if($Done) {
   printf(" Last xtime = $last_xmt_time, End of Section = $endOfSec \n");
   $ShortTime = ($endOfSec - $last_xmt_time);
   $endOfSec = $last_xmt_time;
   $this_period = $period - $ShortTime;
   printf(" Last Period Short by %2.4f seconds\n",$ShortTime);
   printf("\n");
  }
  if($xtime >= $endOfSec) {
   $sec++;
   $endOfSec += $period;
   $rcvLastSec = $RcvBytesThisSec/($byte_rate * $this_period);
   $xmtLastSec = $XmtBytesThisSec/($byte_rate * $this_period);
   printf("period=%d tp=%f or=%f\n",
    $sec,$rcvLastSec,$xmtLastSec);

   $RcvBytesThisSec = 0;
   $XmtBytesThisSec = 0;
  }
 }
 $XmtBytesThisSec += $xlength + 32;
 $pxseqno = $xseqno;
```

```
return if ($rseqno == $prseqno);

# rcvd a pkt at server!
$prseqno = $rseqno;
$RcvBytesThisSec += $rlength + 32;

# if new packet length, summarize previous stats
if ($rlength != $plength) {
 $tptime = $xtime - $t1oflength; #thruput time of length
 if ($tptime == 0.0) {
  $thruput = 0.0;
 }
 else {
  $thruput = $rcvBytesThisLength / ($tptime * $byte_rate);
 }
 $t1oflength = $xtime;
 $rcvBytesThisLength = 0;

 # rates increase only when lengths decrease
 if ($rlength < $plength) {
   $rate = ++$rate % ($#rates + 1);
 }
 $plength = $rlength;
}
$rcvBytesThisLength += $rlength + 32;
}

sub get_rates {
 $#rates = 0;
 unless (open(SCRIPT,$script_file)) {
  warn "Cannot open client script $script_file: $!\n";
  return;
 }
 while(<SCRIPT>) {
  ($f1,$f2,$f3,$f4) = split(' ');
  next unless ($f1 eq "arrival");
  if ($f2 eq "uniform") {
   printf("%s \n",$f3/2);
   push(rates,$f3/2);
  }
  else {
   printf("%s \n",$f2);
   push(rates,$f2);
  }
 }
 close(SCRIPT);
}

sub open_infiles {
 $command = "zcat $client_file | $dcat |";
 open(CLIENT,$command) || die "Can't open $client_file: $!\n";
 $command = "zcat $server_file | $dcat |";
```

```
 open(SERVER,$command) || die "Can't open $server_file: $!\n";
}


sub get_skew {
 # find client/server log clock skew
 for (;;) {
  $_ = <SERVER>;
  ($f1,$f2,$f3,$f4,$f5,$f6,$f7) = split(' ');
  last if ($f6 eq 'epoch');
 }
 $skew = $f7;
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2,$f3,$f4,$f5,$f6,$f7) = split(' ');
  last if ($f6 eq 'epoch');
 }
 $skew = $f7 - $skew;
 printf("skew=%d\n",$skew);

 # now look for "Program start time: Fri Nov 15 10:21:05 1991"
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2,$f3,$f4,$f5,$f6,$f7) = split(' ');
  last if ($f1 eq 'Program'); # must be next line
 }
 s/Program start time://; # chop off front
 $date = $_;
 chop $date;
return;
 # find client/server log clock skew (based on 'Setup')
 for (;;) {
  $_ = <SERVER>;
  ($f1,$f2) = split(' ');
  last if ($f2 eq 'Setup');
 }
 $skew = $f1;
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2) = split(' ');
  last if ($f2 eq 'Setup');
 }
 $skew = $f1 - $skew;
 printf("skew=%d\n",$skew);

}

sub set_file_ptr{ #sets the file to the setup packet
 for (;;) {
  $_ = <SERVER>;
  ($f1,$f2) = split(' ');
  last if ($f2 eq 'Setup');
```

```
 }
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2) = split(' ');
  last if ($f2 eq 'Setup');
 }
}
```

```
#!/usr/local/bin/perl

# udptables.perl

# read compressed TG client and server log files for UDP traffic and
# produce tables to be used by grtool. The dltim.tbl file contains the
# per packet delay and a scatter diagram of the dropped packets below
# the X-axis. The or.tbl file contains the average offer rate (fraction
# of 1.344 Mgb/s) for 10 second intervals. The tptim.tbl file contains
# the average throughput for 10 second intervals. It is expected the
# output in or.tbl and tptim.tbl will be displayed in one graph so
# or.tbl does include any headings. To change this interval, modify the
# variable $period in the routine thruput. Note: these scripts assume a
# network bandwidth 1.344 Mgb/s and have only been used where the packet
# size is constant. To change the link capacity, modify the variable
# $byte_rate. Requires client.log.Z, server.log.Z and udp.c.tg
# (tg script file for the client). If a readme file is present,
# the first line from this file will be used as the graph title.


# Copyright (c) 1993 SRI International. All rights reserved.

# Redistribution and use in source and binary forms are permitted
# provided that the above copyright notice and this paragraph are
# duplicated in all such forms and that any documentation,
# advertising materials, and other materials related to such
# distribution and use acknowledge that the software was developed
# by SRI International, Menlo Park, CA. The name SRI International
# may not be used to endorse or promote products derived from this
# software without specific prior written permission.

# THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
# IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
# WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.



$Done = 0;
$byte_rate = 168000.0; # 192000 for full speed T1
$script_file = "udp.c.tg";
$client_file = "client.log.Z";
$server_file = "server.log.Z";
$dcat = "dcat";
$dir = 'basename \'pwd\'';
chop $dir;


if ($#ARGV > 0) { # $#ARGV counts starting with 0 after command name!
 printf("Usage: %s\n",$0);
 exit(0);
}
```

```
select((select(STDOUT), $| = 1)[$[]); # unbuffer stdout

$total_exp_rbytes = 0;
$total_exp_xbytes = 0;
$total_exp_delay = 0;

&get_rates();
&open_infiles();
&get_skew();
&open_outfiles();

$pseqno = -1;
while(&next_rcv()) {
 # print $_;
 # find transmitted pkt matching sequence number
 for (;;) {
  $prev_xmt_time = $save_xmt_time;
  $_ = <CLIENT>; # read next client line
  ($xtime,$xdirection,$xhost,$xseqno,$xlength) = split(' ');
   if ($xdirection eq 'Teardown') {
  $last_xmt_time = $prev_xmt_time;
 }

  next if ($xdirection ne 'Transmit');
  $total_exp_xbytes += $xlength + 32;
  $save_xmt_time = $xtime;
#  printf("!");
  &thruput();
  last if ($xseqno >= $rseqno);
  printf (DLYTIM "%0.41f\n",-0.2 + rand(.05));
 }

 if ($rseqno != $xseqno) { # out of order!!
  printf("seqno %d OUT OF ORDER!\n",$rseqno);
  next;
 }
 $delay = $rtime - $xtime - $skew;
 printf (DLYTIM "%0.61f\n",$delay);
 &num_packets();
 $total_exp_rbytes += $rlength + 32;
 $total_exp_delay += $delay;
}

printf("Done with RCVS last=%d\n",$rseqno);
while (<CLIENT>) { # drain client xmt data
 $prev_xmt_time = $save_xmt_time;
 ($xtime, $xdirection, $xhost, $xseqno, $xlength) = split(' ');
 $save_xmt_time = $xtime;
 if ($xdirection eq 'Teardown') {
 $last_xmt_time = $prev_xmt_time;
}
```

```
  next if ($xdirection ne 'Transmit');
#  &num_packets(); bug.....shouldn't count the ones at the end
  &thruput();
#probably need this here
  $total_exp_xbytes += $xlength + 16;

  printf (DLYTIM "%0.4lf\n",-0.2 + rand(.05));
 }
 printf("Done with XMTS last=%d\n",$pxseqno);

 # force length change to generate a tp_per_length report
 $rlength = 1;
 $Done = 1;
 &thruput();
 &close_files();


sub next_rcv { # does local 2 line sort on seqno

 if ($next_rcv_init == 0) { # first time read an extra line
  $next_rcv_init++;
  &next_rcv_line();
 }

 $rtime = $stime;
 $rlength = $slength;
 $rseqno = $sseqno;

 if(! &next_rcv_line()) { # read next line and test for end of file
  if ($save_seqno) { # end of file, saved one left
   $rtime = $stime;
   $rlength = $slength;
   $rseqno = $save_seqno;
   $save_seqno = 0;
   return 1;
  }
  else { # file and saved line exhausted
   return 0;
  }
 }

 if ($pseqno >= $sseqno) {
  printf("\nseqno %d precedes %d\n",$pseqno,$sseqno);
  printf(BADSEQ "seqno %d preceeds %d\n",$pseqno,$sseqno);
 }
 $pseqno = $sseqno;

 # now have two pkts to choose from, use the one with lower seqno
 if ($sseqno < $rseqno) {
  $temp = $rtime;
  $rtime = $stime;
```

```
 $stime = $temp;

 $temp = $rlength;
 $rlength = $slength;
 $slength = $temp;

 $temp = $rseqno;
 $rseqno = $sseqno;
 $sseqno = $temp;
}
$save_seqno = $sseqno;

return 1;
}

sub next_rcv_line {
return 0 if $rcv_exhausted;
while (<SERVER>) {
 ($stime, $sdirection, $shost, $sseqno, $slength) = split(' ');
 return 1 if ($sdirection eq 'Receive');
}
$rcv_exhausted = 1;
return 0;
}

sub num_packets {
$num_packets++;
if(!($num_packets % 100)) {
 printf(".");
 if(!($num_packets % 10000)) {
  printf("n_xmt_pkts=%d rate=%d\n",$num_packets,$rate);
 }
}
}

sub close_files {
close(SERVER);
close(CLIENT);
close(DLYTIM);
close(TPTIM);
close(ORTBL);
close(BADSEQ);
}

sub thruput { # compute/print thruput per length, sec and offerrate per sec
$period = 10.0; # length of time period for each thruput value

if ($endOfSec == 0) { # initial call
 $endOfSec = $xtime + $period;
 $sec = 0;
 $prseqno = -1;
```

```
    $t1oflength = $xtime;
    $plength = $rlength;
    $first_xmt_time = $xtime;
   }
   else {
    $this_period = $period;
    if($Done) {
     printf(" Last xtime = $last_xmt_time, End of Section = $endOfSec \n");
     $ShortTime = ($endOfSec - $last_xmt_time);
     $endOfSec = $last_xmt_time;
     $this_period = $period - $ShortTime;
     printf(" Last Period Short by %2.4f seconds\n",$ShortTime);
     printf("\n");
    }
    if($xtime >= $endOfSec) {
     $sec++;
     $endOfSec += $period;
     $rcvLastSec = $RcvBytesThisSec/($byte_rate * $this_period);
     $xmtLastSec = $XmtBytesThisSec/($byte_rate * $this_period);
     printf("period=%d tp=%f or=%f\n",
      $sec,$rcvLastSec,$xmtLastSec);
     $posttime = $sec * $period;
     printf(TPTIM "%f %f %f\n",
      $endOfSec,$rcvLastSec,$xmtLastSec);
     printf(ORTBL "%f %f %d\n",
      $endOfSec,$xmtLastSec,$posttime);

     $RcvBytesThisSec = 0;
     $XmtBytesThisSec = 0;
    }
   }
   $XmtBytesThisSec += $xlength + 32;
   $pxseqno = $xseqno;

   return if ($rseqno == $prseqno);

   # rcvd a pkt at server!
   $prseqno = $rseqno;
   $RcvBytesThisSec += $rlength + 32;

   # if new packet length, summarize previous stats
   if ($rlength != $plength) {
    $tptime = $xtime - $t1oflength; #thruput time of length
    if ($tptime == 0.0) {
     $thruput = 0.0;
    }
    else {
     $thruput = $rcvBytesThisLength / ($tptime * $byte_rate);
    }
#   printf("len=%d rate=%f thruput=%f\n",
#    $plength,@rates[$rate],$thruput);
    $t1oflength = $xtime;
```

```
  $rcvBytesThisLength = 0;

  # rates increase only when lengths decrease
  if ($rlength < $plength) {
    $rate = ++$rate % ($#rates + 1);
  }
  $plength = $rlength;
 }
 $rcvBytesThisLength += $rlength + 32;
}

sub get_rates {
 $#rates = 0;
 unless (open(SCRIPT,$script_file)) {
  warn "Cannot open client script $script_file: $!\n";
  return;
 }
 while(<SCRIPT>) {
  ($f1,$f2,$f3,$f4) = split(' ');
  next unless ($f1 eq "arrival");
  if ($f2 eq "uniform") {
   printf("%s \n",$f3/2);
   push(rates,$f3/2);
  }
  else {
   printf("%s \n",$f2);
   push(rates,$f2);
  }
 }
 close(SCRIPT);
}

sub open_infiles {
 $command = "zcat $client_file | $dcat |";
 open(CLIENT,$command) || die "Can't open $client_file: $!\n";
 $command = "zcat $server_file | $dcat |";
 open(SERVER,$command) || die "Can't open $server_file: $!\n";
}

sub open_outfiles {
 # is there a readme file with a Title?
 if (-e "readme") {
  $title = 'head -1 readme';
  chop($title);
  printf("title=$title\n");
 }
 open(DLYTIM,">dlytim.tbl") || warn "Can't open dlytim.tbl: $!\n";
 printf(DLYTIM "\@title $title\n") if ($title);
 printf(DLYTIM "\@subtitle Delay vs Experiment Time \[$dir:$date\]\n");
 printf(DLYTIM "\@xlabel Packets Sent\n");
 printf(DLYTIM "\@ylabel Delay (seconds) \n");
 printf(DLYTIM "\@setprops 0 0 1 1\n");
```

```
 open(TPTIM,">tptim.tbl") || warn "Can't open tp.tbl: $!\n";
 select((select(TPTIM), $| = 1)[$[]); # unbuffer file output
 printf(TPTIM "\@title $title\n") if ($title);
 printf(TPTIM "\@subtitle OfferRate,Thruput \[$dir:$date\]\n");
 printf(TPTIM "\@xlabel Time (seconds)\n");
 printf(TPTIM "\@ylabel Fraction of 1.344 Mbps \n");
 # dual data set, must use "grtool -n"
 printf(TPTIM "\@setprops 0 1 7 1\n");


 open(ORTBL,">or.tbl") || warn "Can't open or.tbl: $!\n";
 select((select(ORTBL), $| = 1)[$[]); # unbuffer file output
# printf(ORTBL "\@title $title\n") if ($title);
# printf(ORTBL "\@subtitle OfferRate,Thruput \[$dir:$date\]\n");
# printf(ORTBL "\@xlabel Time (seconds)\n");
# printf(ORTBL "\@ylabel Fraction of 1.344 Mbps \n");
 # dual data set, must use "grtool -n"
# printf(ORTBL "\@setprops 0 1 7 1\n");


 open(BADSEQ,">badseq.tbl") || warn "Can't open badseq.tbl: $!\n";
 printf(BADSEQ "\@title $title\n") if ($title);
 printf(BADSEQ "\@subtitle Out of Sequence Pkts \[$dir:$date\]\n");
}


sub get_skew {
 # find client/server log clock skew
 for (;;) {
  $_ = <SERVER>;
  ($f1,$f2,$f3,$f4,$f5,$f6,$f7) = split(' ');
  last if ($f6 eq 'epoch');
 }
 $skew = $f7;
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2,$f3,$f4,$f5,$f6,$f7) = split(' ');
  last if ($f6 eq 'epoch');
 }
 $skew = $f7 - $skew;
 printf("skew=%d\n",$skew);

 # now look for "Program start time: Fri Nov 15 10:21:05 1991"
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2,$f3,$f4,$f5,$f6,$f7) = split(' ');
  last if ($f1 eq 'Program'); # must be next line
 }
```

```
 s/Program start time://; # chop off front
 $date = $_;
 chop $date;
return;
 # find client/server log clock skew (based on 'Setup')
 for (;;) {
  $_ = <SERVER>;
  ($f1,$f2) = split(' ');
  last if ($f2 eq 'Setup');
 }
 $skew = $f1;
 for (;;) {
  $_ = <CLIENT>;
  ($f1,$f2) = split(' ');
  last if ($f2 eq 'Setup');
 }
 $skew = $f1 - $skew;
 printf("skew=%d\n",$skew);

}
```

4 SFQ SOURCE

This section contains the implementation of SFQ that uses the standard queueing techniques available in a UNIX-based kernel, unlike the SFQ implementation documented in Section 5. To use SFQ, the enqueue and dequeue macros located in the drivers are replaced with special SFQ macros that perform the necessary functions. Subsection 4.1, therefore, contains notes for installing the software in a UNIX-based kernel. Subsection 4.2 contains the source itself, which includes the following files: a config file for an SFQ kernel; conf/files; and the files sfq.h, sfq.c, and hsis.c. Hsis.c serves as an example of the installation of SFQ in an existing driver. In the presentation of this material, we assume that the reader is familiar with the process of building a kernel and with the kernel directory structure.

4.1 INSTALLATION NOTES

This is a prototype implementation of Stochastic Fairness Queueing. SFQ is a probabilistic variant of strict fair queueing. Instead of requiring that each flow have its own queue, SFQ has a fixed number of queues and uses a hashing function to map the IP source and destination address into one of the queues. A seed to the hash function is occasionally perturbed, to allow a redistribution of the address pair mapping. This is done to ensure that flows are not consistently mapped into the same queue, so that a well-behaved source is not penalized by an ill-behaved source if the flows happen to map to the same queue at some point in time. For a more complete description of SFQ, see the following article.

McKenney, P.E. 1991. "Stochastic Fairness Queuing," in *Internetworking: Research and Experience*, Vol. 2, pp. 113-131.

SRI provides the code as is and without any express or implied warranties, including, without limitation, the implied warranties of merchantability and fitness for a particular purpose. The current implementation only supports IP packets. It also aliases the send head queue pointer, if_qhead, to a new data type. It is therefore necessary to modify the if_down routine in if.c, so that SFQ queues are not flushed with the routine if_qflush when the interface is marked down via ifconfig. Because this involves Sun source code, the file if.c is not included with this distribution. However, a SFQ flush routine, if_sfqflush, is provided in case you are able to make this change.

Besides the source to SFQ, this distribution includes the source to the HSIS driver used on DARTnet, a sample kernel config file, and the file sun4c/conf/files. Hsis.c provides a model of the way to integrate SFQ into an existing driver, and sun4c/conf/SFQ and sun4c/conf/files show how to integrate SFQ into the kernel. To install SFQ into any system, use the following procedure:

1. Start with a GENERIC kernel config file and copy it to a new name, like SFQ. To the generic config file, add the SFQ option by specifying

   ```
   options SFQ
   ```

2. Add the following to conf/files:

   ```
   sfq/sfq.c optional SFQ
   ```

3. Create a directory called sfq containing sfq.c and sfq.h. This directory should be at the top level of the kernel distribution.

4. If source is available, replace the call to if_qflush with if_sfqflush and add the following external definition in if.c:

```
extern void if_sfqflush();
```

   Otherwise, DO NOT mark an interface as down, using SFQ with ifconfig. If you do, the system will crash.

5. For every driver:
   A. Add #include "../sfq/sfq.h" to the end of the #include section.
   B. Add a call to IF_QINIT before the call to if_attach: for example,

```
IF_QINIT(&ifp->if_snd,hash1)
```

   if you want to use hash1 as the hash function.
   C. Replace calls to IF_QFULL with IF_SFQFULL. Note: a second parameter has been added that is the pointer to the mbuf. If the IP packet is not available yet, use NULL (i.e., 0) as the second parameter.
   D. Replace calls to IF_ENQUEUE with IF_SFQENQUEUE.
   E. For optimization, if the driver is structured with a call to IF_QFULL followed by IF_ENQUEUE, replace the two calls with a call to IF_SFQ_ENQUEUE FAIL. This call will cause the hashing function to be performed only once, and the routine will return TRUE if a packet cannot be added to the queue.
6. For compatibility with all other kernel configurations, you should use conditional compilation around any modifications to the existing kernel source (i.e., #ifdef SFQ followed by #endif SFQ).

4.2 SOURCE

```
# SFQ config file
# This config file describes the "released" Sun-4c kernel for use in DARTnet,
# including the HSIS driver and IP multicast support.
#
# The following lines include support for all Sun-4c CPU types.
# There is little to be gained by removing support for particular
# CPUs, so you might as well leave them all in.
#
machine  "sun4c"
cpu  "SUN4C_60" # Sun-4/60 (any Sun-4c)

ident  "SFQ_1"

#
# This kernel supports about 8 users. Count one user for each
# timesharing user, one for each window that you typically use, and one
# for each diskless client you serve. This is only an approximation used
# to control the size of various kernel data structures, not a hard limit.
#
maxusers 16

options GENERIC
options INET  # basic networking support - mandatory
options UFS  # filesystem code for local disks
options NFSCLIENT # NFS client side code
options NFSSERVER # NFS server side code
options MULTICAST # IP multicast support
options MROUTING # IP multicast routing support
options HSFS  # High Sierra (ISO 9660) CD-ROM file system

options TCPDEBUG # TCP debugging, see trpt(8)

options SFQ # Stochastic Fairness Queueing (SFQ)

#
# The following option adds support for SunView 1 journaling.
#
options WINSVJ  # SunView 1 journaling support

#
# Build one kernel based on this basic configuration.
# It will use the generic swap code so that you can have
# your root filesystem and swap space on any supported device.
# Put the kernel configured this way in a file named "vmunix".
#
config vmunix swap generic

#
# Include support for all possible pseudo-devices.
#
# The first few are mostly concerned with networking.
```

```
# You should probably always leave these in.
#
pseudo-device pty  # pseudo-tty's, also needed for SunView
pseudo-device ether  # basic Ethernet support
pseudo-device loop  # loopback network - mandatory

pseudo-device dbx

#
# The next few are for SunWindows support, needed to run SunView 1.
#
pseudo-device win128  # window devices, allow 128 windows
pseudo-device dtop1  # desktops (screens), allow 4
pseudo-device ms  # mouse support

#
# The following is needed to support the Sun keyboard, with or
# without the window system.
#
pseudo-device kb  # keyboard support

#
# The "open EEPROM" pseudo-device is required to support the
# eeprom command.
#
pseudo-device openeepr # onboard configuration NVRAM

pseudo-device bpfilter 16 # Berkeley packet filter

#
# The following is for the "clone" device, used with streams devices.
# This is required if you include streams NIT support, RFS, or an audio
# device.
#
pseudo-device clone  # clone device

#
# The following section describes which standard device drivers this
# kernel supports.
#
device-driver sbus  # 'driver' for sbus interface
device-driver bwtwo  # monochrome frame buffer
device-driver cgthree  # 8-bit color frame buffer
device-driver cgsix  # 8-bit accelerated color frame buffer
device-driver dma  # 'driver' for dma engine on sbus interface
device-driver esp  # Emulex SCSI interface
device-driver fd  # Floppy disk
device-driver audioamd # AMD79C30A sound chip
device-driver le  # LANCE ethernet
device-driver zs  # UARTs
device-driver hsis
options  HSIS_TRACE
```

```
#options  HSIS_EXTERNAL_RCVDONE

#
# The following section describes SCSI device unit assignments.
#
scsibus0 at esp    # declare first scsi bus
disk sd0 at scsibus0 target 3 lun 0 # first hard SCSI disk
disk sd1 at scsibus0 target 1 lun 0 # second hard SCSI disk
disk sd2 at scsibus0 target 2 lun 0 # third hard SCSI disk
disk sd3 at scsibus0 target 0 lun 0 # fourth hard SCSI disk
tape st0 at scsibus0 target 4 lun 0 # first SCSI tape
tape st1 at scsibus0 target 5 lun 0 # second SCSI tape
disk sr0 at scsibus0 target 6 lun 0 # CD-ROM device
```

```
# @(#)files 1.48 90/08/22 SMI
# Updated by 4.1.1-GFX Rev.2 on 91/03/11 to 1.49
#
include ../../conf.common/files.cmn
#
# Only one of i386dev,sbusdev, and sundev/fd.c should be defined
#
#i386dev/fd.c  optional fd device-driver not-supported
i386dev/hd.c  optional hd device-driver not-supported
i386dev/pp.c  optional pp device-driver not-supported
i386dev/wds.c  optional wds device-driver not-supported
pixrect/../cg12/cg12_colormap.c optional win cgtwelve device-driver
pixrect/../cg12/cg12_ioctl.c optional win cgtwelve device-driver
pixrect/../gt/gt_colormap.c optional win gt device-driver
pixrect/../gt/gt_ioctl.c optional win gt device-driver
pixrect/../gt/gt_rop.c optional win gt device-driver
pixrect/../mem/mem_colormap.c optional win device-driver
pixrect/../mem/mem_kern.c optional win device-driver
pixrect/../mem/mem_rop.c optional win device-driver
pixrect/../pr/pr_clip.c  optional win device-driver
pixrect/../pr/pr_dblbuf.c optional win device-driver
pixrect/../pr/pr_plngrp.c optional win device-driver
sbusdev/audio.c  optional audioamd
sbusdev/audio_79C30.c optional audioamd device-driver
sbusdev/amd7930.c optional audio device-driver
sbusdev/bwtwo.c  optional bwtwo device-driver
sbusdev/cgthree.c optional cgthree device-driver
sbusdev/cgsix.c  optional cgsix device-driver
sbusdev/cgeight.c optional cgeight not-supported
sbusdev/cgtwelve.c optional cgtwelve device-driver
sbusdev/dbx_sbusdev.c optional dbx symbolic-info
sbusdev/gt.c  optional gt device-driver
sbusdev/dmaga.c  optional dma device-driver
sbusdev/fbutil.c optional bwtwo device-driver
sbusdev/fbutil.c optional cgthree device-driver
sbusdev/fbutil.c optional cgsix device-driver
sbusdev/fbutil.c optional cgtwelve device-driver
sbusdev/fd.c  optional fd device-driver
#
# SCSI for Campus
#
scsi/adapters/esp.c  optional esp scsibus device-driver
scsi/conf/scsi_confdata.c optional scsibus
scsi/conf/scsi_confsubr.c optional scsibus
scsi/impl/scsi_capabilities.c optional scsibus
scsi/impl/scsi_control.c optional scsibus
scsi/impl/scsi_resource.c optional scsibus
scsi/impl/scsi_data.c  optional scsibus
scsi/impl/scsi_subr.c  optional scsibus
scsi/impl/scsi_transport.c optional scsibus
scsi/targets/sd_conf.c  optional sd scsibus
scsi/targets/sd.c  optional sd scsibus
```

```
scsi/targets/sf.c  optional sf scsibus not-supported
scsi/targets/sg.c  optional sg scsibus not-supported
scsi/targets/sr.c  optional sr scsibus
scsi/targets/st_conf.c  optional st scsibus
scsi/targets/st.c  optional st scsibus
#
sparc/addupc.s  standard
sparc/copy.s  standard
sparc/crt.s  standard
sparc/float.s  standard
sparc/fpu/addsub.c standard
sparc/fpu/compare.c standard
sparc/fpu/div.c  standard
sparc/fpu/fpu.c  standard
sparc/fpu/fpu_simulator.c standard
sparc/fpu/iu_simulator.c standard
sparc/fpu/mul.c  standard
sparc/fpu/pack.c standard
sparc/fpu/unpack.c standard
sparc/fpu/utility.c standard
sparc/fpu/uword.c standard
sparc/kgdb_stub.c optional KGDB
sparc/mcount.s  optional profiling-routine
sparc/ocsum.s  standard
sparc/overflow.s standard
sparc/sparc_subr.s standard
sparc/swtch.s  standard
sparc/underflow.s standard
sparc/zs_asm.s  optional zs device-driver
sun/conf.c  standard
sun/cons.c  standard
sun/consfb.c  standard device-driver
sun/conskbd.c  standard
sun/consms.c  optional ms
#sun/dkbad.c  standard
sun/in_cksum.c  optional INET
sun/mb_machdep.c standard device-driver
sun/openprom_util.c standard
sun/probe.c  standard
sun/seg_kmem.c  standard
sun/str_conf.c  standard
sun/subr_crash.c standard
sun/swapgeneric.c standard
sun/ufs_machdep.c standard
sun/vdconf.c  optional VDDRV
sun/vddrv.c  optional VDDRV
sun/vdmodsw.c  optional VDDRV
sun/wscons.c  standard
sun4/vm_hat.c  standard
sun4/vm_hatasm.s standard
sun4c/audio_79C30_intr.s optional audioamd device-driver
sun4c/autoconf.c standard device-driver
```

```
sun4c/openprom_xxx.c standard device-driver
sun4c/clock.c  standard device-driver
sun4c/dbx_machdep.c optional dbx symbolic-info
sun4c/fd_asm.s  standard device-driver
sun4c/kgdb_glue.s optional KGDB
sun4c/kprof.s  optional profiling-routine
sun4c/locore.s  standard special
sun4c/lwp/low.s  optional LWP
sun4c/lwp/lwpmachdep.c optional LWP
sun4c/lwp/lwputil.c optional LWP
sun4c/lwp/stack.c optional LWP
sun4c/machdep.c  standard
sun4c/map.s  standard
sun4c/mem.c  standard
sun4c/memerr.c  standard
sun4c/mmu.c  standard
sun4c/subr.s  standard
sun4c/trap.c  standard
sun4c/vm_machdep.c standard
sunchat/chat.c  optional chat device-driver
sunchat/chatunit.c optional chat device-driver
sundev/ar.c  optional ar not-supported
sundev/cgtwo.c  optional cgtwo not-supported
sundev/cgfour.c  optional cgfour not-supported
sundev/cgnine.c  optional cgnine not-supported
sundev/db.c  optional db
sundev/dbx_sundev.c optional dbx symbolic-info
sundev/des.c  optional des not-supported
sundev/fpa.c  optional fpa not-supported
#sundev/fd.c  optional fd device-driver
sundev/ft.c  optional ft device-driver
sundev/gpone.c  optional gpone not-supported
sundev/hrc.c  optional hrc device-driver
sundev/hrc_common.c optional hrc device-driver
sundev/id.c  optional id device-driver not-supported
sundev/ipi.c  optional ipi device-driver not-supported
sundev/ipi_trace.c optional ipi device-driver not-supported
sundev/is.c  optional is device-driver not-supported
sundev/is_conf.c optional is device-driver not-supported
sundev/kbd.c  optional kb
sundev/keytables.c optional kb
sundev/kg.c  optional kg device-driver
sundev/lightpen.c optional gt device-driver
# no mb.c for sun4c
#sundev/mb.c  standard device-driver
sundev/mcp.c optional mcp not-supported
sundev/mcp_async.c optional mcpa not-supported
sundev/mcp_bsc.c optional mcpb not-supported
sundev/mcp_bsctables.c optional mcpb not-supported
sundev/mcp_conf.c optional mcp not-supported
sundev/mcp_isdlc.c optional mcps not-supported
sundev/mcp_proto.c optional mcp not-supported
```

```
sundev/mcp_sdlc.c optional mcph not-supported
sundev/ms.c  optional ms
sundev/mti.c  optional mti not-supported
sundev/mti_conf.c optional mti not-supported
sundev/ns.c  optional ns device-driver
sundev/openprom.c optional openeepr device-driver
sundev/pc.c  optional pc device-driver
sundev/pc_conf.c optional pc device-driver
sundev/pi.c  optional pi not-supported
sundev/rd.c  optional rd device-driver
sundev/sc.c  optional sc OLDSCSI device-driver
sundev/sc_conf.c optional OLDSCSI device-driver
sundev/sd.c  optional sd OLDSCSI device-driver
sundev/se.c  optional se OLDSCSI device-driver
sundev/sf.c  optional sf OLDSCSI device-driver not-supported
sundev/si.c  optional si OLDSCSI device-driver
sundev/st.c  optional st OLDSCSI device-driver
sundev/sw.c  optional sw OLDSCSI device-driver
sundev/taac.c  optional taac not-supported
sundev/tm.c  optional mt not-supported
sundev/tod.c  optional tod not-supported
sundev/tvone.c  optional tvone not-supported
sundev/vp.c  optional vp not-supported
sundev/vpa.c optional vpa not-supported
sundev/vpc.c  optional vpc not-supported
sundev/vuid_queue.c optional kb
sundev/vuid_store.c optional win device-driver
sundev/xd.c  optional xd not-supported
sundev/xd_conf.c optional xd not-supported
sundev/xt.c  optional xt not-supported
sundev/xy.c  optional xy not-supported
sundev/xy_conf.c optional xy not-supported
sundev/zs_async.c optional zs device-driver
sundev/zs_bsc.c  optional zsb device-driver
sundev/zs_bsctables.c optional zsb device-driver
sundev/zs_common.c optional zs device-driver
sundev/zs_conf.c optional zs device-driver
sundev/zs_isdlc.c optional zsi device-driver
sundev/zs_midi.c optional zs MIDI device-driver
sundev/zs_proto.c optional zs device-driver
sundev/zs_sdlc.c optional zss device-driver
sunif/dbx_sunif.c optional dbx INET symbolic-info
sunif/dcp.c  optional dcp device-driver
sunif/ie_conf.c  optional ie device-driver
sunif/if_dcp.c  optional dcp device-driver
sunif/if_ec.c  optional ec INET device-driver
sunif/if_en.c  optional en INET device-driver
sunif/if_hy.c  optional hy not-supported
sunif/if_ie.c  optional ie INET device-driver
sunif/if_le.c  optional le INET device-driver
sunif/if_me.c  optional pc INET device-driver
sunif/if_subr.c  optional ether INET
```

```
sunif/le_conf.c  optional le device-driver
sunif/tbi.c  optional tbi device-driver
sunwindow/rect/rect.c  optional win device-driver
sunwindow/rect/rect_data.c optional win device-driver
sunwindow/rect/rectlist.c optional win device-driver
sunwindowdev/dtopnub.c  optional dtop device-driver
sunwindowdev/win.c  optional win device-driver
sunwindowdev/win_syscall.c optional win device-driver
sunwindowdev/wincms.c  optional win device-driver
sunwindowdev/windevconf.c optional win device-driver
sunwindowdev/windt.c  optional win device-driver
sunwindowdev/winioctl.c  optional win device-driver
sunwindowdev/winlock.c  optional win device-driver
sunwindowdev/winshared.c optional win device-driver
sunwindowdev/wintree.c  optional win device-driver
sunwindowdev/ws.c  optional dtop device-driver
sunwindowdev/ws_dispense.c optional win device-driver
sunwindowdev/ws_interrupt.c optional win device-driver
hsisdev/hsis.c  optional hsis device-driver
hsisdev/dbx_hsis.c optional hsis symbolic-info
sfq/sfq.c  optional SFQ
```

```
#ifdef notdef
/* #ifndef lint */
static char rcsid[] = "@(#)$Id: sfq.h,v 1.6 92/10/23 17:35:49 root Exp Locker:
root $";
static char copyright[] = "Copyright (c) 1992 SRI International,
denny@erg.sri.com";
/* #endif lint */
#endif notdef

/*
 * Copyright (c) 1992 SRI International. All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by SRI International, Menlo Park, CA. The name SRI International
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */

#ifdef SFQ

#include <sys/param.h>
#include <sys/mbuf.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/errno.h>
#include <sys/time.h>

#include <net/if.h>
#include <netinet/in.h>
#include <netinet/in_var.h>
#include <netinet/in_systm.h>
#include <netinet/ip.h>

#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/mbuf.h>
/*
 * Debug register. Set to 0xffffffff to enable debug statements.
 */
extern long   sfqdebug;

#define DPRT(c, x)  if(sfqdebug&c)printf x;

/* debugging flags */
```

```
#define TR_ENQ   1<<0 /* Enqueue flag */
#define TR_DEQ   1<<1 /* Dequeue flag */
#define TR_STA   1<<2 /* Statistics and
 /* error checking */
 /* flag */

#ifdef KERNEL
#define malloc(x)  new_kmem_zalloc((u_int)(x), KMEM_SLEEP)
#define free    kmem_free
#endif KERNEL

/*
 * Overview of SFQ implementation
 */

/* Replace generic BSD queuing macros with our own */

#undef IF_PREPEND   /* (ifq, m) */
#undef IF_DEQUEUEIF   /* (ifq, m, ifp) */
#undef IF_DEQUEUE   /* (ifq, m) */


#define MOD_INPUT_VALUE 8 /* used by call to modit */
#define FQ_HASHTBLSIZ   257 /* The size of the hash table */
     /* is 2 * MOD_INPUT_VALUE + 1.*/
 /* This also indicates the */
     /* number of queues allocated.*/
#define FQ_HASHBUFLEN  8 /* 2*sizeof(struct ip_addr) */
#define FQ_MAXFCFSQLEN  100 /* Maximum FCFS Queue length per */
 /* queue */

/* Hashing functions that can be used */

unsigned long hash1 ();
unsigned long hash2 ();
unsigned long hash3 ();
unsigned long hash4 ();
unsigned long hash5 ();

/* Individual queue description */
struct ifqueue2 {
 struct mbuf  *ifq_head; /* pointers to pkts in this */
 struct mbuf  *ifq_tail; /* queue */
 int   ifq_len; /* length of queue */
 int   ifq_maxlen; /* maximum number of entries */
 int   ifq_drops; /* number dropped */
 int   ifq_sent; /* for debugging-number sent */
 struct ifqueue2 *ifq_forw; /* active list pointer */
 struct ifqueue2 *ifq_back; /* i.e. where this queue is in */
    /* the transmission queue */
 int   ifq_label; /* For debugging only */
};
```

```
/* Main top level data structure, replaces datatype for the ifq_head */
/* pointer in the ifnet structure */
struct sfq {
 int   fq_len;  /* Total number of packets in FQ chain */
 unsigned long  (*fq_hash)(); /* Hash function */
 long  fq_hashlen; /* Length of hash data */
 struct ifqueue2 *fq_index; /* Points to the head of the active list */
 long  fq_seed; /* Hash function seed */
 struct ifqueue2 fq_hashtbl[FQ_HASHTBLSIZ]; /* Hash table area */
 u_char  fq_hashbuf[FQ_HASHBUFLEN];/* Temporary work area */

};

/*
 * Miscellaneous comments:
 *
 * Note that all internal variables used within a macro are prefaced
 * with an underscore so as to differentiate them from variables names
 * used the calling program. Be careful when using variable names
 * beginning with an underscore.
 */

/* IF_QINIT: Macro which allocates and initializes SFQ structure */

/*
 * ifq ::= Pointer to ifnet ifqueue structure
 * hashfnc ::= Pointer to hash function
 */
#define IF_QINIT(ifq, hashfnc) {     \
  struct sfq  *_sfq; \
  int   _i;     \
        \
  /* Allocate FQ structure and attach to interface ifq */ \
  _sfq = (struct sfq *) malloc (sizeof (struct sfq));  \
  (ifq)->ifq_head = (struct mbuf *) _sfq;   \
        \
  /* Initialize variables */     \
  _sfq->fq_index = NULL;     \
  _sfq->fq_seed = 0;      \
  _sfq->fq_len = 0;      \
  _sfq->fq_hash = hashfnc;     \
  _sfq->fq_hashlen = FQ_HASHBUFLEN;    \
  for (_i = 0; _i < FQ_HASHTBLSIZ; _i++) {   \
  _sfq->fq_hashtbl[_i].ifq_head = NULL;   \
  _sfq->fq_hashtbl[_i].ifq_tail = NULL;   \
  _sfq->fq_hashtbl[_i].ifq_len = 0;   \
  _sfq->fq_hashtbl[_i].ifq_drops = 0;   \
  _sfq->fq_hashtbl[_i].ifq_sent = 0;   \
  _sfq->fq_hashtbl[_i].ifq_maxlen = FQ_MAXFCFSQLEN; \
  _sfq->fq_hashtbl[_i].ifq_forw = NULL;   \
  _sfq->fq_hashtbl[_i].ifq_back = NULL;   \
```

```
  _sfq->fq_hashtbl[_i].ifq_label = _i;   \
  }       \
 }

#ifndef PPP_HDRSPACE
#define PPP_HDRSPACE 4
#endif

/* IF_SFQENEQUEUE: Macro which enqueues an IP packet, contained in the */
/* mbuf, to the appropriate queue and links that queue into the active */
/* list pointer if necessary. The active list pointer is a linked list */
/* which orders the packets for transmission. */

/*
 * ifq ::= Pointer to ifnet ifqueue structure
 * m ::= Pointer to beginning of mbuf chain
 */
#define IF_SFQENQUEUE(ifq,m) {      \
  unsigned long  _s;    \
  struct sfq  *_sfq;    \
  struct ip  *_ip;    \
  struct ifqueue2  *_q, *_q2;   \
   struct mbuf *_m0;    \
  u_char   *_ucp;    \
  int _index; \
  /* Extract fields to be hashed and put into buffer */ \
  \
  _sfq = (struct sfq *) ifq->ifq_head;    \
    \
  _m0 = m;       \
   \
  /* Checking the size and finding the beginning of the IP pkt */ \
  if (_m0->m_len == PPP_HDRSPACE) {    \
   _m0 = _m0->m_next;      \
  if (_m0 == NULL)      \
   panic("IF_SFQENQUEUE: ppp header only\n");  \
   _ucp = mtod ((_m0), u_char *);    \
  } else if (_m0->m_len > PPP_HDRSPACE) {   \
   _ucp = mtod ((_m0), u_char *);    \
   _ucp = (u_char *) (((int) _ucp) + PPP_HDRSPACE);  \
  } else {       \
   panic("IF_SFQENQUEUE:invalid _m0 elem: no valid data \
  or header\n"); \
  }       \
  _ip = (struct ip *) _ucp;     \
  bcopy (&(_ip->ip_src), _sfq->fq_hashbuf, _sfq->fq_hashlen); \
        \
  /* Compute hash entry */     \
  _s = (*_sfq->fq_hash) (_sfq->fq_seed, _sfq->fq_hashbuf); \
        \
  /* Mod hash result to fit into table */    \
  _index = modit(_s,MOD_INPUT_VALUE);    \
```

```
 if (_index >= FQ_HASHTBLSIZ) \
  panic("IF_SFQENQUEUE: invalid queue index\n"); \
        \
 DPRT (TR_ENQ,("IF_SFQENQUEUE: Adding packet to [%d]\n", _index)); \
        \
 _q = &_sfq->fq_hashtbl[_index];     \
        \
 /* Make active list entry */     \
 if (_q->ifq_head == NULL) {     \
  if (_sfq->fq_index == NULL) {    \
  _sfq->fq_index = _q;     \
  _q->ifq_forw = _q;     \
  _q->ifq_back = _q;     \
  } else {      \
  _q2 = _sfq->fq_index;     \
  _q->ifq_back = _q2->ifq_back;    \
  _q->ifq_forw = _q2;     \
  _q2->ifq_back->ifq_forw = _q;    \
  _q2->ifq_back = _q;     \
 }      \
 }       \
 /* Insert into the queue */ \
 (m)->m_act = 0;       \
 if (_q->ifq_tail == 0) {     \
  _q->ifq_head = m;      \
 } else {      \
  _q->ifq_tail->m_act = m;     \
 }       \
 _q->ifq_tail = m;      \
  \
 _q->ifq_len++;  /* FCFS queue counter */  \
 _sfq->fq_len++; /* SFQ packet counter */ \
 (ifq)->ifq_len++; /* ifnet counter */   \
}

/* IF_DEQUEUE: Macro which dequeues the next packet for tranmission */
/* from the active list. All appropriate fields are updated including */
/* the queue length fields and the number sent from this queue. */
/*
 * NB: m = NULL signals empty queue
 * fq_index == NULL signals empty queue
 */
#define IF_DEQUEUE(ifq,m) {      \
  struct sfq  *_sfq;     \
  struct ifqueue2 *_q;     \
        \
  (m) = NULL;       \
  _sfq = (struct sfq *) ifq->ifq_head;   \
  _q = _sfq->fq_index;     \
        \
  /*       \
  * Conditional will be true, if index field points  \
```

```
 * to a non-empty queue.     \
 */       \
 if (_q) {       \
       \
 (m) = _q->ifq_head;      \
  if ((_q->ifq_head = (m)->m_act) == 0) {  \
  _q->ifq_tail = 0;     \
  }       \
   (m)->m_act = 0;      \
\
   _q->ifq_len--;      \
  _sfq->fq_len--;    \
   (ifq)->ifq_len--;      \
   _q->ifq_sent++; /* number sent on this q */ \
  DPRT (TR_DEQ, ("IF_DEQUEUE: [%d]\n", _q->ifq_label)); \
       \
  /* Remove entry from active list if no more pkts */ \
  if (_q->ifq_head == 0) {     \
   if ((_q->ifq_forw == _q) && (_q->ifq_back== _q)) { \
   _q->ifq_forw = NULL;     \
  /* Perturb the hash seed only when queue is empty */ \
   _sfq->fq_seed++;     \
 DPRT (TR_DEQ, ("IF_DEQUEUE: FQ %d now Empty\n",_q->ifq_label)); \
   } else {      \
   _q->ifq_back->ifq_forw = _q->ifq_forw;  \
   _q->ifq_forw->ifq_back = _q->ifq_back;  \
 DPRT (TR_DEQ, ("IF_DEQUEUE: removing %d from active list\n", \
    _q->ifq_label));     \
   }       \
  }       \
        \
  /* Index to the next non-empty queue */   \
  _sfq->fq_index = _q->ifq_forw;     \
  }       \
 }


/* IF_QPRINT: Macro which prints out information about the SFQ structure */
/* and verifies the integrity of the structure. */

#define IF_QPRINT(ifq) {      \
  int  i, total = 0, nelem;   \
  struct sfq *q;      \
  struct ifqueue2 *fcq;      \
  struct mbuf *m;      \
        \
  q = (struct sfq *) ifq->ifq_head;   \
  DPRT(TR_STA,("IF_QPRINT: Seed = %d SFQ len = %d\n", \
   q->fq_seed, q->fq_len ));   \
        \
  for (i = 0; i < FQ_HASHTBLSIZ; i++) {   \
   fcq = &q->fq_hashtbl[i];     \
```

```
  if (fcq->ifq_sent != 0) \
  DPRT(TR_STA,("IF_QPRINT: FQ = %d, len = %d, \
drops = %d, sent = %d\n", \
fcq->ifq_label, fcq->ifq_len, fcq->ifq_drops, \
fcq->ifq_sent)); \
  if (fcq->ifq_head != 0) {    \
  total += fcq->ifq_len;    \
  for (nelem = 0, m = fcq->ifq_head; m;  \
   m = m->m_act) {    \
   nelem++;     \
  }      \
  if (fcq->ifq_len != nelem) {   \
   DPRT(TR_STA, ("IF_QPRINT: Inconsistency in q \
structure for %d\n",i);  \
  DPRT(TR_STA, ("IF_QPRINT: ifq_len = %d count \
= %d\n", fcq->ifq_len, nelem)); \
  }      \
  }       \
 }       \
 if (total != q->fq_len)     \
  DPRT(TR_STA,("IF_QPRINT: inconsistency in \
total pkt count\n"));   \
  DPRT (TR_STA, ("IF_QPRINT: Total number of \
packets = %d fq_len = %d\n",  \
  total, q->fq_len));     \
}

#endif SFQ
```

```
#ifdef notdef
/* #ifndef lint */
static char rcsid[] = "@(#)$Id: sfq.c,v 1.6 92/10/23 17:35:21 root Exp Locker:
root $";
static char copyright[] = "Copyright (c) 1992 SRI International,
denny@erg.sri.com";
/* #endif lint */
#endif notdef

/*
 * Copyright (c) 1992 SRI International. All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by SRI International, Menlo Park, CA. The name SRI International
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */

#ifdef SFQ

#include <sys/param.h>
#include <sys/mbuf.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/errno.h>
#include <sys/time.h>

#include <net/if.h>
#include <netinet/in.h>
#include <netinet/in_var.h>
#include <netinet/in_systm.h>
#include <netinet/ip.h>


#ifndef FALSE
#define FALSE (0)
#endif

#ifndef TRUE
#define TRUE (1)
#endif

#include "sfq.h"
```

```
long   sfqdebug = 0x0; /* printing debug flag */


/* modit returns a mod (2**m + 1): a must be positive */
/* The algorithm is taken from Volume II of Knuth */

modit(a, m)

 unsigned long  a;
 unsigned long  m;

 {
 int mask, bp1, c, n, x;

 mask = (1 << m) - 1;
 bp1 = mask + 2;

 for (;;)
  {
  c = a & mask;
  n = a >> m;
  x = c - n;
  if (x >= 0)
   return (x);
  if (x >= -bp1)
   return (x + bp1);
  a = -x;  /* masking only works on positive values */

  c = a & mask;
  n = a >> m;
  x = c - n;
  if (x > 0)
   return (bp1 - x);
  if (x > -bp1)
   return (-x);
  a = -x;
  }
 }


/* if_sfqflush frees all queues when an interface is marked down by ifconfig*/
/* should replace call to if_qflush in if_down if running SFQ */
/* These routines are contained in if.c */

if_sfqflush(ifq)
register struct ifqueue *ifq;
{
 int  i;
 struct sfq *q;
 struct ifqueue2 *fcq;
 struct mbuf *m, *n;
```

```
 q = (struct sfq *) ifq->ifq_head;
 for (i = 0; i < FQ_HASHTBLSIZ; i++) /* loop through all the queues */
 {
 fcq = &q->fq_hashtbl[i];
 if (fcq->ifq_head != NULL)   /* Is this queue not empty? */
 {
  n = fcq->ifq_head; /* remove packets in this queue */
  while (m = n)
  {
  n = m->m_act;
  m_freem(m);
 }
 }
 fcq->ifq_head = NULL; /* reinitialize all variables */
 fcq->ifq_tail = NULL;
 fcq->ifq_len = 0;
 fcq->ifq_drops = 0;
 fcq->ifq_sent = 0;
 fcq->ifq_forw = NULL;
 fcq->ifq_back = NULL;
 }
 q->fq_len = 0;
 q->fq_index = NULL;
 q->fq_seed = 0;
}


/* IF_SFQFULL checks to see if a particular queue is full for a given */
/* mbuf which contains an IP packet. Returns TRUE if the queue is */
/* full; FALSE otherwise. Note: Due to the structure in the HSIS driver, */
/* this routine gets called for a raw packet before the driver actually */
/* gets the packet. Since SFQ needs the IP source and destination address */
/* from the packet to figure out what queue to put it in, all raw packets */
/* automatically get put in the same queue regardless of their source or */
/* destination. Since it is assumed that raw packets are not a great */
/* percentage of network traffic, the effects should be negligible. */
/* In addition, a side effect of this routine is that the drop counter */
/* for a queue gets incremented if there is no room for the given IP */
/* packet. */



IF_SFQFULL(ifq, m)
 struct ifqueue *ifq;
 struct mbuf  *m;
{
 struct sfq  *sfq;
 struct mbuf *m0;
 struct ip  *ip;
 struct ifqueue2 *q;
 long  s; /* value from hash function */
 u_char *ucp; /* pointer to IP pkt */
```

```
int index;

if (m != NULL) /* do we have a packet? */
{
sfq = (struct sfq *) ifq->ifq_head;
m0 = m;
if (m0->m_len == PPP_HDRSPACE) { /* pkt size checks */
 m0 = m0->m_next;
 if (m0 == NULL)
 panic("IF_SFQFULL: ppp header only\n");
 ucp = mtod ((m0), u_char *);
} else if (m0->m_len > PPP_HDRSPACE) {
 ucp = mtod ((m0), u_char *);
 ucp = (u_char *) (((int) ucp) + PPP_HDRSPACE);
} else {
 panic("IF_SFQFULL: invalid m0 elem: no valid data or header\n");
}
ip = (struct ip *) ucp;  /* Assumes packet is an IP pkt */


/* Extract fields to be hashed and put into buffer */
/* i.e. IP source and destination address */

bcopy (&ip->ip_src, sfq->fq_hashbuf, sfq->fq_hashlen);

/* Compute hash entry */
s = (*sfq->fq_hash) (sfq->fq_seed, sfq->fq_hashbuf);
/* Mod hash result to fit into table */
index = modit(s,MOD_INPUT_VALUE);
if (index >= FQ_HASHTBLSIZ) /* error check */
 printf("IF_SFQFULL: invalid queue index\n");
}
else
index = FQ_HASHTBLSIZ -1; /* all raw packets get this index */

q = &sfq->fq_hashtbl[index]; /* get the queue for this IP address */
   /* pair */
if (q->ifq_len >= q->ifq_maxlen) /* check the length */
{
if (m != NULL) /* will need to drop this packet */
q->ifq_drops++; /* increment drop counter */
return(TRUE);
}
else
return(FALSE);
}

/* IF_SFQ_ENQUEUE_FAIL returns TRUE if a packet failed to be put on the */
/* associated queue; FALSE otherwise. Note: Due to the structure in the */
/* HSIS driver, this routine can called for a raw packet before the */
/* driver actually gets the packet. Since SFQ needs the IP source and */
/* destination address from the packet to figure out what queue to put */
```

```
/* it in, all raw packets automatically get put in the same queue */
/* regardless of their source or destination. Since it is assumed that */
/* raw packets are not a great percentage of network traffic, the */
/* effects should be negligible. In addition, a side effect of this */
/* routine is that the drop counter for a queue gets incremented if */
/* there is no room for the given IP packet. */


IF_SFQ_ENQUEUE_FAIL(ifq, m)
 struct ifqueue *ifq;
 struct mbuf  *m;
{
 struct sfq  *sfq;
 struct mbuf *m0;
 struct ip  *ip;
 struct ifqueue2 *q, *q2;
 long  s; /* value returned from hash function */
 u_char *ucp;
 int index;

 if (m != NULL) { /* do we have a packet? */

 sfq = (struct sfq *) ifq->ifq_head;
 m0 = m;
 if (m0->m_len == PPP_HDRSPACE) { /* pkt size legality checks */
  m0 = m0->m_next;
  if (m0 == NULL)
  panic("IF_SFQ_ENQUEUE_FAIL: ppp header only\n");
  ucp = mtod ((m0), u_char *);
 } else if (m0->m_len > PPP_HDRSPACE) {
  ucp = mtod ((m0), u_char *);
  ucp = (u_char *) (((int) ucp) + PPP_HDRSPACE);
 } else {
  panic("IF_SFQ_ENQUEUE_FAIL: invalid m0 elem: no valid data or header\n");
 }
 ip = (struct ip *) ucp;  /* Assumes packet is an IP pkt */


 /* Extract fields to be hashed and put into buffer */
 /* i.e., IP source and destination address */

 bcopy (&ip->ip_src, sfq->fq_hashbuf, sfq->fq_hashlen);

 /* Compute hash entry */
 s = (*sfq->fq_hash) (sfq->fq_seed, sfq->fq_hashbuf);
 /* Mod hash result to fit into table */
 index = modit(s,MOD_INPUT_VALUE);
 if (index >= FQ_HASHTBLSIZ)
  printf("IF_SFQ_ENQUEUE_FAIL: invalid queue index\n");
 }
 else
 index = FQ_HASHTBLSIZ -1; /* raw packets go here */
```

```
q = &sfq->fq_hashtbl[index]; /* find the queue */
if (q->ifq_len >= q->ifq_maxlen) /* is length not ok? */
{
if (m != NULL)
q->ifq_drops++; /* incremented drop counter */
return(TRUE);
}
else
{
if (m == NULL) /* hack for raw send and because we */
 return FALSE; /* are combining the enqueue and qfull */
   /* macros for efficiency */
/* There is no packet yet */

DPRT (TR_ENQ, ("IF_SFQ_ENQUEUE_FAIL: Adding packet to [%d]\n", index));

/* Make active list entry */

if (q->ifq_head == NULL) {
 if (sfq->fq_index == NULL) {
 sfq->fq_index = q;
 q->ifq_forw = q;
 q->ifq_back = q;
 } else {
 q2 = sfq->fq_index;
 q->ifq_back = q2->ifq_back;
 q->ifq_forw = q2;
 q2->ifq_back->ifq_forw = q;
 q2->ifq_back = q;
}
}

/* Store packet onto queue */
(m)->m_act = 0;
if (q->ifq_tail == 0) {
 q->ifq_head = m;
} else {
 q->ifq_tail->m_act = m;
}
q->ifq_tail = m;

q->ifq_len++;  /* FCFS queue counter */
sfq->fq_len++; /* SFQ packet counter */
(ifq)->ifq_len++; /* ifnet counter */

return(FALSE);
}
}


/* Possible hash functions to choose from */
```

```
/*
 *
 */
unsigned long
hash1(seq, cp)    /* addxorhash.c */
 char seq;
 unsigned char *cp;
{
 unsigned long l1;
 unsigned long l2;

 l1 = *(unsigned long *)cp;
 l2 = *(((unsigned long *)cp) + 1);
 return ((l1 + seq) ^ l2);
}


/*
 *
 */
unsigned long
hash2(seq, cp)    /* rot11hash.c */
 char seq;
 unsigned char *cp;
{
 static unsigned long maskstay[] = {
 0xffffffff, 0xfffffffe, 0xfffffffc, 0xfffffff8,
 0xfffffff0, 0xffffffe0, 0xffffffc0, 0xffffff80,
 0xffffff00, 0xfffffe00, 0xfffffc00, 0xfffff800,
 0xfffff000, 0xffffe000, 0xffffc000, 0xffff8000,
 0xffff0000, 0xfffe0000, 0xfffc0000, 0xfff80000,
 0xfff00000, 0xffe00000, 0xffc00000, 0xff800000,
 0xff000000, 0xfe000000, 0xfc000000, 0xf8000000,
 0xf0000000, 0xe0000000, 0xc0000000, 0x80000000
 };
 static unsigned maskwrap[] = {
 0x00000000, 0x00000001, 0x00000003, 0x00000007,
 0x0000000f, 0x0000001f, 0x0000003f, 0x0000007f,
 0x000000ff, 0x000001ff, 0x000003ff, 0x000007ff,
 0x00000fff, 0x00001fff, 0x00003fff, 0x00007fff,
 0x0000ffff, 0x0001ffff, 0x0003ffff, 0x0007ffff,
 0x000fffff, 0x001fffff, 0x003fffff, 0x007fffff,
 0x00ffffff, 0x01ffffff, 0x03ffffff, 0x07ffffff,
 0x0fffffff, 0x1fffffff, 0x3fffffff, 0x7fffffff
 };
 unsigned long l1, l2;
 int coseq;

 l1 = *(unsigned long *)cp;
 l2 = *(((unsigned long *)cp) + 1);
 seq &= 0x1f;
```

```
 coseq = 32 - seq;
 l1 = ((l1 << seq) & maskstay[seq]) ^ ((l1 >> (coseq)) & maskwrap[seq]);

 l2 = ((l2 << coseq) & maskstay[coseq]) ^ ((l2 >> (seq)) & maskwrap[coseq]);

 return (l1 + l2);
}


/*
 *
 */
unsigned long
hash3(seq, cp)   /* rot1hash.c */
 char seq;
 unsigned char *cp;
{
 static unsigned long maskstay[] = {
 0xffffffff, 0xfffffffe, 0xfffffffc, 0xfffffff8,
 0xfffffff0, 0xffffffe0, 0xffffffc0, 0xffffff80,
 0xffffff00, 0xfffffe00, 0xfffffc00, 0xfffff800,
 0xfffff000, 0xffffe000, 0xffffc000, 0xffff8000,
 0xffff0000, 0xfffe0000, 0xfffc0000, 0xfff80000,
 0xfff00000, 0xffe00000, 0xffc00000, 0xff800000,
 0xff000000, 0xfe000000, 0xfc000000, 0xf8000000,
 0xf0000000, 0xe0000000, 0xc0000000, 0x80000000
 };
 static unsigned maskwrap[] = {
 0x00000000, 0x00000001, 0x00000002, 0x00000004,
 0x00000008, 0x0000001f, 0x0000002f, 0x0000004f,
 0x0000008f, 0x000001ff, 0x000002ff, 0x000004ff,
 0x000008ff, 0x00001fff, 0x00002fff, 0x00004fff,
 0x00008fff, 0x0001ffff, 0x0002ffff, 0x0004ffff,
 0x0008ffff, 0x001fffff, 0x002fffff, 0x004fffff,
 0x008fffff, 0x01ffffff, 0x02ffffff, 0x04ffffff,
 0x08ffffff, 0x1fffffff, 0x2fffffff, 0x4fffffff,
 0x8fffffff
 };
 unsigned long l1;
 unsigned long l2;
 int coseq;

 l1 = *(unsigned long *)cp;
 l2 = *(((unsigned long *)cp) + 1);
 seq &= 0x1f;
 coseq = 32 - seq;
 l1 = ((l1 << seq) & maskstay[seq]) ^ ((l1 >> (coseq)) & maskwrap[seq]);

 l2 = ((l2 << coseq) & maskstay[coseq]) ^ ((l2 >> (seq)) & maskwrap[coseq]);

 return (l1 + l2);
}
```

```
/*
 *
 */
unsigned long
hash4(seq, cp)   /* rothash.c */
 char seq;
 unsigned char *cp;
{
 static unsigned long maskstay[] = {
 0xffffffff, 0xfffffffe, 0xfffffffc, 0xfffffff8,
 0xfffffff0, 0xffffffe0, 0xffffffc0, 0xffffff80,
 0xffffff00, 0xfffffe00, 0xfffffc00, 0xfffff800,
 0xfffff000, 0xffffe000, 0xfffffc000, 0xffff8000,
 0xffff0000, 0xfffe0000, 0xfffc0000, 0xfff80000,
 0xfff00000, 0xffe00000, 0xffc00000, 0xff800000,
 0xff000000, 0xfe000000, 0xfc000000, 0xf8000000,
 0xf0000000, 0xe0000000, 0xc0000000, 0x80000000
 };
 static unsigned maskwrap[] = {
 0x00000000, 0x00000001, 0x00000002, 0x00000004,
 0x00000008, 0x0000001f, 0x0000002f, 0x0000004f,
 0x0000008f, 0x000001ff, 0x000002ff, 0x000004ff,
 0x000008ff, 0x00001fff, 0x00002fff, 0x00004fff,
 0x00008fff, 0x0001ffff, 0x0002ffff, 0x0004ffff,
 0x0008ffff, 0x001fffff, 0x002fffff, 0x004fffff,
 0x008fffff, 0x01ffffff, 0x02ffffff, 0x04ffffff,
 0x08ffffff, 0x1fffffff, 0x2fffffff, 0x4fffffff,
 0x8fffffff
 };
 unsigned long l1, l2;
 int coseq;

 l1 = *(unsigned long *)cp;
 l2 = *(((unsigned long *)cp) + 1);
 seq &= 0x1f;
 coseq = 32 - seq;
 l1 = ((l1 << seq) & maskstay[seq]) ^ ((l1 >> (coseq)) & maskwrap[seq]);

 l2 = ((l2 << coseq) & maskstay[coseq]) ^ ((l2 >> (seq)) & maskwrap[coseq]);

 return (l1 ^ l2);
}


/*
 *
 */
unsigned long
hash5(seq, cp)   /* xoraddhash.c */
 char seq;
```

```
 unsigned char *cp;
{
 unsigned long l1;
 unsigned long l2;


 l1 = *(unsigned long *)cp;
 l2 = *(((unsigned long *)cp) + 1);
 return ((l1 ^ seq) + l2);
}

#endif SFQ
```

```
#ifndef lint
static char rcsid[] =
 "@(#)$Header: /home/sys_isi/hsisdev/RCS/hsis.c,v 1.4 92/10/23 17:46:43 root
Exp Locker: root $";
static char copyright[] =
 "Copyright (c) 1990 Regents of the University of California";
#endif

/*
 * Copyright (c) 1990 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by the University of California, Lawrence Berkeley Laboratory,
 * Berkeley, CA. The name of the University may not be used to
 * endorse or promote products derived from this software without
 * specific prior written permission.
 * THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */

#include "hsis.h"
#if NHSIS > 0
#include <sys/param.h>
#include <sys/mbuf.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/errno.h>
#include <sys/systm.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/in_var.h>
#include <net/netisr.h>
#include <sys/stream.h>
#include <sys/ttycom.h>
#include <sys/tty.h>
#include <sys/time.h>

#include <sys/sockio.h>
#include <sundev/zsreg.h>
#include <sundev/mbvar.h>
#include <sun4c/intreg.h>

#include <sun4c/auxio.h>

#include "bpfilter.h"
#if NBPFILTER > 0
```

```
#include <net/bpf.h>
#endif

#ifdef SFQ
#include "../sfq/sfq.h"
#endif SFQ


#include "z16c35.h"
#include "hsisreg.h"
#include "syncmode.h"
#include "syncstat.h"
#include "hsiscom.h"
#include "ppp.h"

#ifndef AF_RAWSYNC
#define AF_RAWSYNC AF_IMPLINK
#endif

int hsidebug = 0;
#ifdef HSIS_TRACE
int hsis_trace_lock = 0x7FFFFFFF;
#endif
#define dprintf(x) if(hsidebug)printf x;

/*
 * Like most things that have to deal with Zilog chips, this driver
 * requires two interrupt levels: a hardware level that should be as
 * high as possible (SBus level 5-7) and a software level that should
 * be at or below splimp (i.e., at level 4 or 6 on a sparc). Very little
 * time is spent at hardware interrupt level. Packet copies to and
 * from the board (<300 us, worst case) and almost all the device and
 * system stuff (20-100 us typically, 300 us if we have to call Sun's
 * incredibly slow Streams NIT) is done at the software interrupt level.
 */
#define splboard spl4
#define HARDINT_LEVEL 5

/*
 * Software interrupt level, state and status bits.
 */
#define SOFTINT_LEVEL 4
#define splsoft spl2

u_int hsis_isum;
#define RECV_DONE 1
#define XMIT_DONE 2
#define SOFTINT(softc, event) (hsis_isum |= (event) << \
     ((sc)->sc_if.if_unit << 1))

/*
 * default set-up for scc (possibly modified by user-specified 'syncmode').
```

```
 * see hsis_init for how this is interpreted. Each entry is a particular
 * scc reg. The scc requires register be set in a particular order and
 * not all bits (in particular, the rx & tx enables) can be loaded from
 * here (see hsis_init). Be careful when changing this array.
 */
u_char hsis_scc_setup[] = {
 /* 0 */ 0,
#ifdef HSIS_EXTERNAL_RCVDONE
 /* 1 */ ZSWR1_SIE,
#else
 /* 1 */ ZSWR1_SIE | ZSWR1_RIE_SPECIAL_ONLY,
#endif
 /* 2 */ 0,
 /* 3 */ ZSWR3_RXCRC_ENABLE | ZSWR3_RX_8,
 /* 4 */ ZSWR4_SDLC,
 /* 5 */ ZSWR5_TXCRC_ENABLE | ZSWR5_RTS | ZSWR5_TX_8,
 /* 6 */ 0,
 /* 7 */ ZSWR7_SDLCFLAG,
 /* 8 */ 0,
 /* 9 */ ZSWR9_MASTER_IE | ZSWR9_NO_VECTOR | ZSWR9_VECTOR_INCL_STAT,
 /*10 */ ZSWR10_PRESET_ONES,
 /*11 */ 0,
 /*12 */ 0,
 /*13 */ 0,
 /*14 */ 0,
 /*15 */ ZSR15_SDLC_FIFO_ENA | ZSR15_TX_UNDER,
};

/*
 * default 'syncmode' (clock and loopback options) for scc.
 */
struct syncmode hsis_default_sm = {
 TXC_IS_BAUD, RXC_IS_BAUD, 1, 0, 9600, 0, 0
};


int hsisidentify(), hsisattach();
struct dev_ops hsis_ops = {
 1,
 hsisidentify,
 hsisattach,
};

int nhsisboard;   /* total # of hsis boards found */

struct hsiscom *hsiscom;
struct hsiscom *hsiscom_end;
struct hsis_softc *hsis_softc;
struct hsis_softc *hsis_softc_end;

int hsis_init(), hsis_output(), hsis_ioctl(), hsis_reset(), hsis_watchdog(),
 hsisintr(), hsissoftint();
```

```
void hsis_start();
void rawsyncinit();

void
hsis_scc_attach(unit, hs, addr)
 register u_char unit;
 register struct hsiscom *hs;
 register u_char *addr;
{
 static int nhsischan;
 register struct zscc *zs = &hs->hs_zs[unit];
 register struct zdma *zd = &hs->hs_zd[unit >> 1];
 register u_char *zsdev = addr;
 register struct hsis_softc *sc = &hsis_softc[nhsischan];
 register struct ifnet *ifp = &sc->sc_if;

 zs->zs_unit = unit;
 zs->zs_addr = zsdev;

 /* disable rx, tx and interrupts. */
 SCC_WRITE(1, 0);
 SCC_WRITE(3, 0);
 SCC_WRITE(5, 0);

 /* disable external rcv done interrupt */
 hs->hs_board[unit + HSIS_EINT_ENA_A] = 0;

 /* set up the softc and make us known to the network code. */
 sc->sc_zs = zs;
 sc->sc_zd = zd;
 sc->sc_hs = hs;
 sc->sc_sm = hsis_default_sm;
 sc->sc_fifop = sc->sc_fifo;
 ifp->if_name = "hsis";
 ifp->if_unit = nhsischan++;
 ifp->if_mtu = HSIS_MTU;
#ifdef MULTICAST
 ifp->if_flags = IFF_POINTOPOINT | IFF_MULTICAST;
#else
 ifp->if_flags = IFF_POINTOPOINT;
#endif MULTICAST
 ifp->if_init = hsis_init;
 ifp->if_output = hsis_output;
 ifp->if_ioctl = hsis_ioctl;
 ifp->if_reset = hsis_reset;
 ifp->if_watchdog = hsis_watchdog;
#ifdef SFQ
 ifp->if_snd.ifq_maxlen = FQ_MAXFCFSQLEN; /* may be misleading */
      /* size of one of SFQ's queue */
#else
 ifp->if_snd.ifq_maxlen = HSIS_MAX_SENDQ;
#endif SFQ
```

```
#if NBPFILTER > 0
 bpfattach(&sc->sc_bpf, ifp, DLT_PPP, PPP_HDRSPACE);
#endif

#ifdef SFQ
  IF_QINIT(&ifp->if_snd,hash1);
#endif SFQ

 if_attach(ifp);
}

void
hsis_dma_attach(unit, zd, addr)
 register u_char unit;
 register struct zdma *zd;
 register u_char *addr;
{
 register u_char *zddev = addr;

 zd->zd_addr = zddev;

 /*
  * Reset the dma section. Clear out the 'interrupt vector' address
  * so we get a simple value to switch on when interpreting intr.
  * Set the DCR to increment addresses rather than decrementing.
  * Set the ICR to disable intr
  * on all channels (any event that would generate a dma intr also
  * generates an scc intr -- we waste enough time dealing with this
  * stupid chip and don't need to double all the interrupts),
  * don't put a vector on the bus when req. intr but do include
  * 'status' in the IVR on intr.
  * Zero out the dma address registers so we don't have to write
  * the high bytes when switching buffers.
  */
 DMA_WRITE0(ZSDMA_CCAR_DMA_RESET);
 DMA_WRITE0(ZSDMA_CCAR_RESET_IUS);
 DMA_WRITE(ZSDMA_IVR, 0);
 DMA_WRITE(ZSDMA_ICSR, 0);
 DMA_WRITE(ZSDMA_DCR, 0xf);
 DMA_WRITE(ZSDMA_ICR, ZSDMA_ICR_NV | ZSDMA_ICR_VIS);

 DMA_WRITE(ZSDMA_RDARA, 0);
 DMA_WRITE(ZSDMA_RDARA+1, 0);
 DMA_WRITE(ZSDMA_RDARA+2, 0);
 DMA_WRITE(ZSDMA_RDARA+3, 0);

 DMA_WRITE(ZSDMA_TDARA, 0);
 DMA_WRITE(ZSDMA_TDARA+1, 0);
 DMA_WRITE(ZSDMA_TDARA+2, 0);
 DMA_WRITE(ZSDMA_TDARA+3, 0);
```

```
 DMA_WRITE(ZSDMA_RDARB, 0);
 DMA_WRITE(ZSDMA_RDARB+1, 0);
 DMA_WRITE(ZSDMA_RDARB+2, 0);
 DMA_WRITE(ZSDMA_RDARB+3, 0);

 DMA_WRITE(ZSDMA_TDARB, 0);
 DMA_WRITE(ZSDMA_TDARB+1, 0);
 DMA_WRITE(ZSDMA_TDARB+2, 0);
 DMA_WRITE(ZSDMA_TDARB+3, 0);
}

int
hsisidentify(name)
 char *name;
{
 if (strcmp(name, "HSI") == 0) {
  ++nhsisboard;
  return (1);
 } else
  return (0);
}

int
hsisattach(dev)
 register struct dev_info *dev;
{
 static int curhsis = 0;
 register struct hsiscom *hs;
 register u_char *hsboard;
 register int i;

 dev->devi_unit = curhsis;
 if (hsiscom == NULL) {
  hsiscom = (struct hsiscom *)new_kmem_zalloc(
    (u_int)(nhsisboard * sizeof (struct hsiscom)),
    KMEM_SLEEP);
  if (hsiscom == NULL) {
   printf("hsis: no space for data structures.\n");
   return (-1);
  }
  hsiscom_end = &hsiscom[nhsisboard];

  hsis_softc = (struct hsis_softc *)new_kmem_zalloc(
    (u_int)(nhsisboard * 4 * sizeof (struct hsis_softc)),
    KMEM_SLEEP);
  if (hsis_softc == NULL) {
   printf("hsis: no space for data structures\n");
   return (-1);
  }
  hsis_softc_end = &hsis_softc[nhsisboard * 4];

  rawsyncinit(); /*XXX*/
```

```
}
hs = &hsiscom[dev->devi_unit];
hs->hs_dev = dev;

/*
 * register our interrupt handler, map the board into kernel memory,
 * then reset it. Note that the interrupt level must be <= splimp
 * (and, given the constraints imposed by the braindead Zilog dma,
 * the level should be as high as possible, e.g., splimp).
 */
addintr(dev->devi_intr->int_pri, hsisintr, dev->devi_name, curhsis);
addintr(SOFTINT_LEVEL, hsissoftint, "HSI-soft", curhsis);
hsboard = (u_char *)map_regs(dev->devi_reg->reg_addr,
    dev->devi_reg->reg_size,
    dev->devi_reg->reg_bustype);
hs->hs_board = hsboard;

/*
 * set up the free buffer list.
 */
for (i = HSIS_SRAM_SIZE; (i -= HSIS_BUFSIZE) >= 0; ) {
 register struct hsisbuf *bp;

 bp = (struct hsisbuf *)(hsboard + i + HSIS_SRAM);
 bp->next = hs->hs_free;
 hs->hs_free = bp;
}

/*
 * Initialize the two dma channels and 4 scc channels.
 * First reset each chip and set its bus configuration reg.
 * Then set up the software data structures.
 */
hsboard[HSIS_RST_ISSC0] = 0; /* reset chips */
hsboard[HSIS_RST_ISSC1] = 0;
DELAY(10);
hsboard[HSIS_SCC_A] = 0; /* clear BCR */
hsboard[HSIS_SCC_C] = 0;

hsis_dma_attach(0, &hs->hs_zd[0], &hsboard[HSIS_DMA_0]);
hsis_dma_attach(1, &hs->hs_zd[1], &hsboard[HSIS_DMA_1]);

hsis_scc_attach(0, hs, &hsboard[HSIS_SCC_A]);
hsis_scc_attach(1, hs, &hsboard[HSIS_SCC_B]);
hsis_scc_attach(2, hs, &hsboard[HSIS_SCC_C]);
hsis_scc_attach(3, hs, &hsboard[HSIS_SCC_D]);

report_dev(dev);
++curhsis;
return (0);
}
```

```
int
copy_m_to_b(sc, m, cp)
 register struct hsis_softc *sc;
 register struct mbuf *m;
 register u_char *cp;
{
 register long len, totlen;
 register struct mbuf *m0 = m;

 totlen = 0;
 do {
  len = m->m_len;
  bcopy(mtod(m, caddr_t), (caddr_t)cp, (u_int)len);
  cp += len;
  totlen += len;
 } while (m = m->m_next);

 m_freem(m0);
 if (sc->sc_raw && (sc->sc_raw->so_snd.sb_flags & SB_WAIT ||
    sc->sc_raw->so_snd.sb_sel))
  sbwakeup(sc->sc_raw, &sc->sc_raw->so_snd);
 return (totlen);
}

hsis_output(ifp, m, dst)
 register struct ifnet *ifp;
 register struct mbuf *m;
 struct sockaddr *dst;
{
 register int s;
 register struct hsis_softc *sc;

 sc = (struct hsis_softc *)ifp;
 TRACEL(T_OUTPUT, sc, 0)
#ifdef HSIS_TRACE
 /*
  * keep trace unlocked; ignore miniscule interrupt race.
  */
 if (hsis_trace_lock > 100) hsis_trace_lock = 0x7FFFFFFF;
#endif
 if ((ifp->if_flags & IFF_UP) == 0) {
  m_freem(m);
  return (ENETDOWN);
 }
 if (dst->sa_family != AF_UNSPEC) {
  /* Add a PPP header */
  register u_int off = m->m_off;

  if (off < MMINOFF + PPP_HDRSPACE ||
   (off >= MSIZE && m->m_cltype != MCL_STATIC_HDR)) {
   /* need new mbuf for hdr (should really panic here
    * then fix whatever isn't leaving space for header) */
```

```
    register struct mbuf *m0;

    MGET(m0, M_DONTWAIT, MT_DATA);
    if (m0 == (struct mbuf *)0) {
     m_freem(m);
     return (ENOBUFS);
    }
    m0->m_next = m;
    m0->m_len = PPP_HDRSPACE;
    m = m0;
   } else {
    m->m_off -= PPP_HDRSPACE;
    m->m_len += PPP_HDRSPACE;
   }
   *mtod(m, u_int *) = PPP_INET; /* XXX */
  }
  if (sc->sc_ostate == 0) {
   register struct hsisbuf *bp = sc->sc_curout;
   register int len;

   len = copy_m_to_b(sc, m, BUFtoCP(bp));
   s = splboard();
   TRACE(T_OUT_COPY, sc, (int)bp | len)
   bp->cnt = len;
   sc->sc_ostate = 2;
   hsis_start(sc);
#ifdef HSIS_EXTERNAL_RCVDONE
  } else if (sc->sc_if.if_snd.ifq_len == 0 && sc->sc_nextout->cnt == 0) {
   register struct hsisbuf *bp = sc->sc_nextout;
   register int len;

   len = copy_m_to_b(sc, m, BUFtoCP(bp));
   s = splboard();
   TRACE(T_OUT_COPY, sc, (int)bp | len)
   bp->cnt = len;
   if (sc->sc_ostate == 0) {
    /*
     * last packet completed while we were doing copy --
     * flip buffers & restart output.
     */
    (sc->sc_nextout = sc->sc_curout)->cnt = 0;
    sc->sc_curout = bp;
    sc->sc_ostate = 2;
    hsis_start(sc);
   }
#endif
  } else {
   register struct ifqueue *ifq = &ifp->if_snd;

   s = splboard();
#ifdef SFQ
   if (IF_SFQ_ENQUEUE_FAIL(ifq,m)) {
```

```
    IF_DROP(ifq);
/*   TRACE(T_QFULL, sc, 0) see if can make this call */
    splx(s);
    m_freem(m);
    return (ENOBUFS);
   }
#else
   if (IF_QFULL(ifq)) {
    IF_DROP(ifq);
    TRACE(T_QFULL, sc, 0)
    splx(s);
    m_freem(m);
    return (ENOBUFS);
   }
   IF_ENQUEUE(ifq, m);
#endif SFQ
   if (sc->sc_ostate == 0)
    hsis_start(sc);
 }
 splx(s);
 return (0);
}

/*
 * start new output operation. This routine *must* be called at splboard.
 */
void
hsis_start(sc)
 register struct hsis_softc *sc;
{
 register struct zscc *zs = sc->sc_zs;
 register u_char *zsdev = zs->zs_addr;
 register u_char *zddev = sc->sc_zd->zd_addr;
 register u_char unit = sc->sc_if.if_unit;
 register u_char *cp;
 register int len;
 register u_int baddr;
 register int s;
 register u_int resid;
 register int i;

 cp = BUFtoCP(sc->sc_curout);
 if (sc->sc_ostate == 3)  /* take handoff from hardware intr */
  sc->sc_ostate = 0;
 if (sc->sc_ostate == 0) {
  register struct hsisbuf *bp;

  if (len = (bp = sc->sc_nextout)->cnt) {
   /*
    * 'next' output buffer is full - swap current and
    * next. (The weird assignment below fools Sun-4 cc
    * into generating reasonable code -- maybe one day
```

```
     * Sun will discover ANSI C 'volatile' and this
     * crap can go away.)
     */
    (sc->sc_nextout = sc->sc_curout)->cnt = 0;
    sc->sc_curout = bp;
    cp = BUFtoCP(bp);
   } else {
    /*
     * on-board xmit buffer is empty. try to copy a new
     * packet to it. (we want to overlap the copy with
     * the scc sending the final crc and flag bytes to
     * avoid taking an extra xmit interrupt).
     */
    register struct mbuf *m;

    IF_DEQUEUE(&sc->sc_if.if_snd, m);
    if (m == NULL) {
     sc->sc_if.if_timer = 0;
     TRACE(T_EMPTY_OQ, sc, 0)
     return;
    }
    len = copy_m_to_b(sc, m, cp);
    TRACE(T_OUT_COPY, sc, (int)bp | len)
   }
  } else
   len = sc->sc_curout->cnt;

  /*
   * if the transmit buffer is full it means that crc/flag
   * sending is still in progress. the stupid scc will jam
   * two packets together if we enable dma xmit so we have
   * to turn on the xmit interrupt (which should come when
   * the crc and flag have been sent) and exit waiting for
   * that interrupt. If the transmit buffer is empty, we
   * can just start the next packet (this should be the
   * usual case at T1 speeds).
   */
  sc->sc_if.if_timer = HSIS_WATCHDOG_TIME;
  SCC_READ0(resid);
  if ((resid & ZSRR0_TX_READY) == 0) {
   /*
    * we lose - wait for xmit intr.
    */
   SCC_BIS(1, ZSWR1_TIE);
   sc->sc_ostate = 2;
   sc->sc_curout->cnt = len;
   return;
  }
  /*
   * Transmit buffer empty -- start up dma.
   */
  baddr = cp - sc->sc_hs->hs_board;
```

```
 SCC_WRITE0(ZSWR0_RESET_TXCRC);
 if (unit & 1) {
  /* B channel */
  register u_int c;

  DMA_READ(ZSDMA_TDCRB, resid);
  DMA_READ(ZSDMA_TDCRB + 1, c);
  resid |= c << 8;

  DMA_WRITE(ZSDMA_TDCRB, len);
  DMA_WRITE(ZSDMA_TDCRB+1, len >> 8);

  DMA_WRITE(ZSDMA_TDARB, baddr);
  DMA_WRITE(ZSDMA_TDARB+1, baddr >> 8);

  s = splhigh();
  DMA_WRITE(ZSDMA_CCAR, ZSDMA_CCAR_ENA_TX_B);
 } else {
  /* A channel */
  register u_int c;

  DMA_READ(ZSDMA_TDCRA, resid);
  DMA_READ(ZSDMA_TDCRA + 1, c);
  resid |= c << 8;

  DMA_WRITE(ZSDMA_TDCRA, len);
  DMA_WRITE(ZSDMA_TDCRA+1, len >> 8);

  DMA_WRITE(ZSDMA_TDARA, baddr);
  DMA_WRITE(ZSDMA_TDARA+1, baddr >> 8);

  s = splhigh();
  DMA_WRITE(ZSDMA_CCAR, ZSDMA_CCAR_ENA_TX_A);
 }
 /*
  * Cretinous chip requires that EOM not be reset until dma has
  * loaded first data character into buffer but must be reset
  * before last character loaded into buffer. We've locked out
  * to prevent the obvious race so just wait until character gets
  * there.
  */
 for (i = 10; --i >= 0; ) {
  DELAY(2);
  SCC_READ0(baddr);
  if ((baddr & ZSRR0_TX_READY) == 0)
   goto rdy;
 }
 TRACE(T_NO_XMIT_RDY, sc, baddr)
 dprintf(("hsis%d: hsis_start: xmit didn't load (rr0=0x%x)\n",
  unit, baddr))
rdy:
 SCC_WRITE0(ZSWR0_RESET_EOM);
```

```
 TRACE(T_START_OUT, sc, (resid << 16) | len)
 sc->sc_ostate = 1;
 splx(s);
 if (resid) {
  ++sc->sc_if.if_oerrors;
  ++sc->sc_estats.sse_underrun;
 }
 ++sc->sc_if.if_opackets;
 ++sc->sc_dstats.ssd_opack;
 sc->sc_dstats.ssd_ochar += len;
#if NBPFILTER > 0
 if (sc->sc_bpf)
  bpf_tap(sc->sc_bpf, cp, len);
#endif
#ifdef HSIS_EXTERNAL_RCVDONE
 /*
  * if there's more in the snd q, copy another packet to the
  * on-board 'next' buffer (we do it now to overlap the copy
  * with the send of the last packet to approximate back-to-back
  * output packets).
  */
 if (sc->sc_if.if_snd.ifq_len) {
  register struct mbuf *m;

  IF_DEQUEUE(&sc->sc_if.if_snd, m);
  if (m) {
   register struct hsisbuf *bp = sc->sc_nextout;

   bp->cnt = len = copy_m_to_b(sc, m, BUFtoCP(bp));
   TRACE(T_OUT_COPY, sc, (int)bp | len)
  }
 }
#endif
}

int
hsis_start_dma_read(sc)
 register struct hsis_softc *sc;
{
 register u_char *zddev = sc->sc_zd->zd_addr;
 register int baddr = BUFtoCP(sc->sc_inbuf) - sc->sc_hs->hs_board;
 register u_char bl = baddr;
 register u_char bh = baddr >> 8;
 register int s = splhigh();
 register u_int rl;
 register u_char rh;

 /*
  * disable the dma channel, set it to xfer into sc_inbuf,
  * then re-enable. If there are back-to-back packets inbound,
  * we have to re-enable the dma channel before the 3 byte scc
  * rcv fifo overflows. E.g., at T1 (5.2us/byte) the time
```

```
 * from disable to enable should be no more than 10us. So,
 * we make sure nothing interrupts us during this window and
 * the code below is as fast as I can make it.
 * Data from the next packet probably got stuffed into the
 * buffer containing the current packet so we save and return
 * the current dma count reg. before overwriting it (so the
 * higher level routine can undo the damage done by this
 * braindead dma model).
 */
if (sc->sc_zs->zs_unit & 1) {
 /* B channel */

 /* if the TX DMA is just about finished, wait for it */
 /* to avoid re-enabling it when disabling RX DMA */
 DMA_READ(ZSDMA_TDCRB, rl);
 DMA_READ(ZSDMA_TDCRB + 1, rh);
 rl |= rh << 8;
 if (rl > 0 && rl < 4) {
  do {
   DELAY(2);
   DMA_READ(ZSDMA_DER, rh);
  } while (rh & ZSDMA_DER_TX_B_ENABLE);
 }

 /* spin until the rcv fifo is empty */
 for (rl = 10; --rl != 0; ) {
  register u_char *zsdev = sc->sc_zs->zs_addr;
  SCC_READ0(rh)
  if ((rh & ZSRR0_RX_READY) == 0)
   break;
  DELAY(2);
 }
 if (rl == 0)
  TRACE(T_FAIL, sc, rh);

 /* repeat the disable until it works (harware bug) */
 DMA_READ(ZSDMA_DER, rh);
 do {
  DMA_WRITE(ZSDMA_DER, rh &~ ZSDMA_DER_RX_B_ENABLE);
  DMA_READ(ZSDMA_DER, rh);
 } while (rh & ZSDMA_DER_RX_B_ENABLE);

 DMA_READ(ZSDMA_RDARB, rl);
 DMA_WRITE(ZSDMA_RDARB, bl);
 DMA_READ(ZSDMA_RDARB+1, rh);
 DMA_WRITE(ZSDMA_RDARB+1, bh);

 DMA_WRITE0(ZSDMA_CCAR_ENA_RX_B);

 /* set count after enabling to minimize time */
 /* disabled; lsb first to prevent borrow from msb */
 DMA_WRITE(ZSDMA_RDCRB, HSIS_MAXPACKET & 0xff);
```

```
 DMA_WRITE(ZSDMA_RDCRB+1, HSIS_MAXPACKET >> 8);
 } else {
  /* A channel */
  DMA_READ(ZSDMA_TDCRA, rl);
  DMA_READ(ZSDMA_TDCRA + 1, rh);
  rl |= rh << 8;
  if (rl > 0 && rl < 4) {
   do {
    DELAY(2);
    DMA_READ(ZSDMA_DER, rh);
   } while (rh & ZSDMA_DER_TX_A_ENABLE);
  }

  for (rl = 10; --rl != 0; ) {
   register u_char *zsdev = sc->sc_zs->zs_addr;
   SCC_READ0(rh)
   if ((rh & ZSRR0_RX_READY) == 0)
    break;
   DELAY(2);
  }
  if (rl == 0)
   TRACE(T_FAIL, sc, rh);

  DMA_READ(ZSDMA_DER, rh);
  do {
   DMA_WRITE(ZSDMA_DER, rh &~ ZSDMA_DER_RX_A_ENABLE);
   DMA_READ(ZSDMA_DER, rh);
  } while (rh & ZSDMA_DER_RX_A_ENABLE);

  DMA_READ(ZSDMA_RDARA, rl);
  DMA_WRITE(ZSDMA_RDARA, bl);
  DMA_READ(ZSDMA_RDARA+1, rh);
  DMA_WRITE(ZSDMA_RDARA+1, bh);

  DMA_WRITE0(ZSDMA_CCAR_ENA_RX_A);

  DMA_WRITE(ZSDMA_RDCRA, HSIS_MAXPACKET & 0xff);
  DMA_WRITE(ZSDMA_RDCRA+1, HSIS_MAXPACKET >> 8);
 }
 rl |= rh << 8;
 TRACE(T_START_READ, sc, (((bh << 8) | bl) << 16) | rl)
 splx(s);
 return ((int)(rl - sizeof(struct hsisbuf)) & (HSIS_BUFSIZE - 1));
}

/*
 * This routine is called if an output operation takes longer than
 * HSIS_WATCHDOG_TIME (usually 2 minutes) to complete. Reset and
 * restart the channel (current output packet will be lost).
 */
hsis_watchdog(unit)
 register int unit;
```

```
{
 register struct hsis_softc *sc = &hsis_softc[unit];
 register u_int errcnt = sc->sc_oerrcnt;
 register int s;

 if (sc->sc_inbuf == NULL || (sc->sc_if.if_flags & IFF_UP))
  /* we have been manually turned offline or online */
  return;

 if (errcnt == 0)
  printf("hsis%d: watchdog timeout.\n", unit);
 s = splboard();
 sc->sc_oerrcnt = ++errcnt;
 hsis_reset(unit);
 hsis_init(unit);
 hsis_start(sc);
 splx(s);
}

void
hsis_ierr_timer(sc)
 register struct hsis_softc *sc;
{
 register int unit = sc->sc_if.if_unit;
 register int s;

 if (sc->sc_inbuf == NULL || (sc->sc_if.if_flags & IFF_UP))
  /* we have been manually turned offline or online */
  return;
 printf("hsis%d: reset and restarted.\n", unit);
 s = splboard();
 hsis_init(unit);
 hsis_start(sc);
 splx(s);
}

void
hsis_ierror(sc, zs, zsdev, msg)
 register struct hsis_softc *sc;
 register struct zscc *zs;
 register u_char *zsdev;
 register char *msg;
{
 register int unit = sc->sc_if.if_unit;
 register u_int errcnt = sc->sc_ierrcnt;

 sc->sc_ierrcnt = ++errcnt;
 if (errcnt >= HSIS_RESET_THRESH) {
#ifdef HSIS_TRACE
  if (hsis_trace_lock > 100) hsis_trace_lock = 100;
#endif
  hsis_reset(unit);
```

```
 if (errcnt >= HSIS_OFF_THRESH)
  timeout(hsis_ierr_timer, sc, HSIS_OFF_TIME);
 else
  hsis_ierr_timer(sc);
 return;
}
if (errcnt == 1)
 printf("hsis%d: %s (p#%d).\n", unit, msg,
  sc->sc_dstats.ssd_ipack);

/*
 * book says we have to disable then re-enable
 * fifo to clear it. Then toss input queue in (vain) hope that
 * we'll end up with input stream and fifo in sync.
 */
*(u_char *)AUXIO_REG = AUX_MBO|AUX_EJECT;
SCC_BIC(3, ZSWR3_RX_ENABLE)
SCC_BIC(15, ZSR15_SDLC_FIFO_ENA)
++sc->sc_if.if_ierrors;
SCC_BIS(15, ZSR15_SDLC_FIFO_ENA)
SCC_WRITE0(ZSWR0_RESET_ERRORS);
(void) hsis_start_dma_read(sc);
SCC_BIS(3, ZSWR3_RX_ENABLE)
if (sc->sc_intail) {
 sc->sc_intail->next = sc->sc_hs->hs_free;
 sc->sc_hs->hs_free = sc->sc_inq;
 sc->sc_inq = NULL;
 sc->sc_intail = NULL;
}
bzero(sc->sc_fifo, sizeof(sc->sc_fifo));
sc->sc_fifop = sc->sc_fifo;
sc->sc_inoff = NULL;
*(u_char *)AUXIO_REG = AUX_MBO|AUX_EJECT|AUX_LED;
}

void
hsis_sccrecv_intr(sc, zs, zsdev)
 register struct hsis_softc *sc;
 register struct zscc *zs;
 register u_char *zsdev;
{
 register struct hsiscom *hs;
 register struct hsisbuf *bp;
 register struct hsisbuf *qp;
 register int *fp;

 /*
  * Stash the current contents of the sdlc fifo. This should
  * be done before we call 'start_dma_read' or we can end up
  * with the fifo & dma counts different: For god-only-knows
  * what reason, Zilog made the sdlc fifo count bytes that have
  * entered the 3 byte receive data fifo while the dma counts
```

```
         * bytes that have left the receive data fifo. Thus the fifo
         * can record the end of a packet that the dma hasn't finished.
         */
        fp = sc->sc_fifop;
        while (1) {
         register u_char rr1, rr6, rr7;
         register int i;

         SCC_READ(7, rr7)
         SCC_READ(6, rr6)
         SCC_READ(1, rr1)
         i = (((rr1 << 8) | rr7) << 8) | rr6;
         TRACE(T_RCVINT_FIFO, sc, i)
         if (rr1 & ZSRR1_DO) {
          /*
           * data overrun - the way the Zilog fifo works
           * we don't have a prayer of recovering so toss
           * everything in the fifo and input queue, put
           * the scc in hunt mode to skip to the start of
           * the next packet, then hope the fifo eventually
           * gets back in sync with the input stream.
           */
          hsis_ierror(sc, zs, zsdev, "receive data overrun");
          ++sc->sc_estats.sse_overrun;
          return;
         }
         switch (rr7 & 0xc0) {

         case 0x80:
         case 0xc0:
          /* fifo overflow */
          hsis_ierror(sc, zs, zsdev, "status fifo overflow");
          return;

         case 0x00:
          /* fifo empty */
          goto fifo_empty;
         }
         if (*fp) {
          hsis_ierror(sc, zs, zsdev, "status fifo array overflow");
          return;
         }
         *fp++ = i;
         if (fp >= &sc->sc_fifo[HSIS_NFIFO])
          fp = sc->sc_fifo;
        }
       fifo_empty:
        if (fp == sc->sc_fifop)
         /* did nothing (spurious read interrupt) */
         return;
        sc->sc_fifop = fp;
```

```
 hs = sc->sc_hs;
 if ((bp = sc->sc_inbuf) == 0) {
  /* channel is offline */
  (void)hsis_start_dma_read(sc);
  return;
 }
 if ((qp = hs->hs_free) == NULL) {
  /* no buffer - have to toss input or we lose sync with fifo */
  TRACE(T_RCVINT_BUF, sc, 0)
  hsis_ierror(sc, zs, zsdev, "no free bufs");
  return;
 }
 hs->hs_free = qp->next;
 sc->sc_inbuf = qp;
 if ((bp->cnt = hsis_start_dma_read(sc)) == 0) {
  bp->next = hs->hs_free;
  hs->hs_free = bp;
 } else {
  bp->next = NULL;
  if (qp = sc->sc_intail)
   qp->next = bp;
  else
   sc->sc_inq = bp;
  sc->sc_intail = bp;
 }
 SOFTINT(sc, RECV_DONE);
}

void
hsis_stat_intr(sc, zs, zsdev)
 register struct hsis_softc *sc;
 register struct zscc *zs;
 register u_char *zsdev;
{
 register u_int rr0;

 SCC_READ0(rr0);
 SCC_WRITE0(ZSWR0_RESET_STATUS);
 SCC_WRITE0(ZSWR0_CLR_INTR);
 TRACE(T_STAT_INT, sc, rr0)
 if ((rr0 & ZSRR0_TXUNDER) && sc->sc_ostate) {
  /* packet completed - start a new one if possible */
  sc->sc_oerrcnt = 0;
  sc->sc_ostate = 3; /* don't let hsis_output sneak in */
  SOFTINT(sc, XMIT_DONE);
 } else if (rr0 & ZSRR0_BREAK) {
  /*
   * 'abort' received -- reset input (because scc fifo
   * state is messed up and we have no way to figure
   * out packet boundaries).
   */
  hsis_ierror(sc, zs, zsdev, "received 'abort'");
```

```
        ++sc->sc_estats.sse_abort;
       } else {
        /* XXX - should do something. */
        dprintf(("hsis%d: scc stat interrupt, rr0=0x%x.\n",
         sc->sc_if.if_unit, rr0))
       }
      }

      void
      hsis_sccxmit_intr(sc, zs, zsdev)
       register struct hsis_softc *sc;
       register struct zscc *zs;
       register u_char *zsdev;
      {
       register u_int rr0;

       SCC_READ0(rr0);
       SCC_BIC(1, ZSWR1_TIE);
       SCC_WRITE0(ZSWR0_RESET_TXINT);
       SCC_WRITE0(ZSWR0_CLR_INTR);
       TRACE(T_XMIT_INT, sc, (sc->sc_ostate << 16) | rr0)
       if (sc->sc_ostate == 2) {
        /*
         * we were waiting for CRC/flag send to finish and it
         * has -- start next packet.
         */
        SOFTINT(sc, XMIT_DONE);
       } else {
        /*
         * since we never enabled xmit interrupt, something's weird.
         * it would be nice to print a warning message but there's
         * a small race in hsis_start where we TIE then find xmit is
         * done & turn it off. Zilog says the disable should clear
         * the interrupt but, of course, they lie & we can end
         * up here.
         */
       }
      }

      #define ZS_A_INTR (ZSRR3_IP_A_STAT|ZSRR3_IP_A_RX|ZSRR3_IP_A_TX)
      #define ZS_B_INTR (ZSRR3_IP_B_STAT|ZSRR3_IP_B_RX|ZSRR3_IP_B_TX)

      #define ZINTSCAN(rmask, smask, tmask) { \
       if (rr3 & rmask) { \
        SCC_WRITE0(ZSWR0_RESET_ERRORS); \
        SCC_WRITE0(ZSWR0_CLR_INTR); \
        hsis_sccrecv_intr(sc, zs, zsdev); \
       } \
       if (rr3 & smask) \
        hsis_stat_intr(sc, zs, zsdev); \
       if (rr3 & tmask) \
        hsis_sccxmit_intr(sc, zs, zsdev); \
```

```
}

#define ZCHECK_INT(chan) ZINTSCAN(ZSRR3_IP_/**/chan/**/_RX, \
    ZSRR3_IP_/**/chan/**/_STAT, \
    ZSRR3_IP_/**/chan/**/_TX)
#ifndef HSIS_EXTERNAL_RCVDONE
#define ZCHECK_BOTH { \
 zs = sc->sc_zs; \
 zsdev = zs->zs_addr; \
 SCC_READ(3, rr3); \
 if (rr3 & ZS_A_INTR) { \
  ++did_something; \
  ZCHECK_INT(A) \
 } \
 ++sc; \
 if (rr3 & ZS_B_INTR) { \
  ++did_something; \
  zs = sc->sc_zs; \
  zsdev = zs->zs_addr; \
  ZCHECK_INT(B) \
 } \
 ++sc; \
}
#else
#define ZCHECK_BOTH { \
 zs = sc->sc_zs; \
 zsdev = zs->zs_addr; \
 if ((hsis[HSIS_EINT_STS] & rcvintmask) == 0) {\
  ++did_something; \
  hsis[zs->zs_unit + HSIS_EINT_CLR_A] = 0; \
  hsis_sccrecv_intr(sc, zs, zsdev); \
 } \
 rcvintmask <<= 1; \
 SCC_READ(3, rr3); \
 if (rr3 & ZS_A_INTR) { \
  ++did_something; \
  ZCHECK_INT(A) \
 } \
 ++sc; \
 if ((hsis[HSIS_EINT_STS] & rcvintmask) == 0) {\
  ++did_something; \
  zs = sc->sc_zs; \
  zsdev = zs->zs_addr; \
  hsis[zs->zs_unit + HSIS_EINT_CLR_A] = 0; \
  hsis_sccrecv_intr(sc, zs, zsdev); \
 } \
 rcvintmask <<= 1; \
 if (rr3 & ZS_B_INTR) { \
  ++did_something; \
  zs = sc->sc_zs; \
  zsdev = zs->zs_addr; \
  ZCHECK_INT(B) \
```

```
 } \
 ++sc; \
}
#endif

/*
 * Handle HSIS board(s) interrupt(s).
 */
int
hsisintr()
{
 register struct hsis_softc *sc;
 register struct hsis_softc *sc_end = hsis_softc_end;
 register struct zscc *zs;
 register u_char *zsdev, *hsis;
 register u_char rr3;
 register int did_something = 0, last_round;

 /*
  * It's costly to take an interrupt and likely that some other
  * channel finished while we were servicing the current channel.
  * We loop here until we make one full pass through the status
  * registers and don't find new work to do.
  */
 TRACE(T_INTR_ENTRY, hsis_softc, 0)
 do {
  last_round = did_something;
  /*
   * we can't reliably read the status registers on the
   * zilog chip so we look at the hsis board status register
   * to find interrupt requests. This means we loop over
   * four zilog channels at a time.
   */
  for (sc = hsis_softc; sc < sc_end; ) {
#ifdef HSIS_EXTERNAL_RCVDONE
   register u_char rcvintmask = 1;
#endif
   hsis = sc->sc_hs->hs_board;
   ZCHECK_BOTH
   ZCHECK_BOTH
  }
 } while (did_something != last_round);

 if (hsis_isum)
  set_intreg(IR_SOFT_INT4, 1);

 TRACE(T_INTR_EXIT, hsis_softc, did_something)
 return (did_something);
}

void
hsis_drop_input(sc, len, bufoff, buflen)
```

```
register struct hsis_softc *sc;
register int len, buflen;
register u_char *bufoff;
{
/*
 * Note: To avoid races between the hardware & software
 * interrupt levels, it's important that the load of 'bp'
 * below be done *after* we are at splboard. In particular,
 * this means that the load of bp *cannot* be put in the
 * delay slot of the spl call. The statement order below
 * works with Sun's current compiler technology but may give
 * problems in the future (one day they'll discover "volatile"...)
 */
register int s = splboard();
register struct hsiscom *hs = sc->sc_hs;
register struct hsisbuf *bp = sc->sc_inq;

if (bp == NULL) {
 /*
  * timeout or hardware intr probably did a reset --
  * exit, making sure that input state stays clean.
  */
 sc->sc_inoff = 0;
 splx(s);
 return;
}
while (len) {
 if (buflen <= 0) {
  buflen = bp->cnt;
  bufoff = BUFtoCP(bp);
 }
 if (buflen > len) {
  buflen -= len;
  bufoff += len;
  break;
 } else {
  register struct hsisbuf *np;

  len -= buflen;
  buflen = 0;
  bufoff = NULL;

  np = bp->next;
  bp->next = hs->hs_free;
  hs->hs_free = bp;
  if ((sc->sc_inq = bp = np) == NULL) {
   sc->sc_intail = NULL;
   break;
  }
 }
}
if (buflen)
```

```
  bp->cnt = buflen;
 sc->sc_inoff = bufoff;
 splx(s);
}

/*
 * Following is mbuf offset to get nice alignment of packets.
 * Choice determined by:
 * 1. IP header *must* be aligned on a word (4 byte) boundary.
 * 2. bcopy will go much faster if data is cache aligned (i.e., at
 * 16-byte boundary).
 * 3. We need space to prepend a 14 byte ethernet header (if
 * forwarding packet).
 *
 * MMINOFF is 12 (thanks to Bill Joy for this horrible kludge) and the
 * next mult-of-16 above 12+14 is 32 so we offset 20 (= 32 - MMINOFF)
 * in an mbuf and 16 in a cluster.
 */
#define HDRSPACE (16)
#define MBUF_HDRSPACE (32 - MMINOFF)

/*
 * following macro drops 'len' input bytes (used on input errors).
 */
#define DROP_INPUT(len) { \
 hsis_drop_input(sc, len, bufoff, buflen); \
 if ((bp = sc->sc_inq) == NULL) \
  /* input was reset by timeout or error */ \
  return; \
 if (bufoff = sc->sc_inoff) \
  buflen = bp->cnt; \
 else \
  buflen = 0; \
}

/*
 * This routine goes through the software copy of the scc
 * status fifo (which records the lengths of incoming packets)
 * and sc_inq (the queue of unprocessed, incoming data) and
 * breaks the data up into packets then queues them for higher
 * level network processing. The loop structure is complicated
 * by the fact that packet boundaries are mapped randomly onto
 * the input queue (there may be more than one packet per input
 * buffer and packets may cross buffer boundaries).
 */
void
hsis_recv_done(sc)
 register struct hsis_softc *sc;
{
 register int *fp;
 register u_char *bufoff;
 register int buflen;
```

```
register struct hsiscom *hs = sc->sc_hs;
register struct hsisbuf *bp;

/* find the first unprocessed packet in the fifo array */
for (fp = sc->sc_fifop - 1; ; --fp) {
 if (fp < sc->sc_fifo)
  fp = &sc->sc_fifo[HSIS_NFIFO - 1];
 if (*fp == 0)
  break;
}

if (bufoff = sc->sc_inoff) {
 if ((bp = sc->sc_inq) == NULL)
  return;
 buflen = bp->cnt;
} else
 buflen = 0;

while (1) {
 register struct mbuf *m;
 register u_char *op;
 register int sfifo;

 if (++fp >= &sc->sc_fifo[HSIS_NFIFO])
  fp = sc->sc_fifo;

 if ((sfifo = *fp) == 0)
  break;
 *fp = 0;
 TRACEL(T_RCVINT, sc, sfifo)

 if (sfifo & (ZSRR1_FE << 16)) {
  ++sc->sc_estats.sse_crc;
  ++sc->sc_if.if_ierrors;
  sfifo &= 0x3fff;
  DROP_INPUT(sfifo)
  continue;
 }
 sfifo &= 0x3fff;
 if (sfifo > MCLBYTES - HDRSPACE) {
  /* packet too big */
  ++sc->sc_if.if_ierrors;
  DROP_INPUT(sfifo)
  continue;
 }
 MGET(m, M_DONTWAIT, MT_DATA);
 if (m == (struct mbuf *)0) {
  DROP_INPUT(sfifo)
  continue;
 }
 if (sfifo <= MLEN - MBUF_HDRSPACE)
  m->m_off += MBUF_HDRSPACE;
```

```
else {
 /* too big for mbuf - use cluster */
 MCLGET(m);
 if (m->m_len != MCLBYTES) {
  /* no clusters - drop this packet */
  m_freem(m);
  DROP_INPUT(sfifo)
  continue;
 }
 m->m_off += HDRSPACE;
}
if (! sc->sc_raw)
 m->m_off -= PPP_HDRSPACE;
m->m_len = sfifo - 2;
sc->sc_dstats.ssd_ichar += sfifo;
++sc->sc_dstats.ssd_ipack;
++sc->sc_if.if_ipackets;
op = mtod(m, u_char *);
while (sfifo) {
 if (buflen <= 0) {
  bp = sc->sc_inq;
  if (bp == NULL) {
   /*
    * nothing on input queue - probably
    * had an overrun at hardware intr
    * level. just bail.
    */
   m_freem(m);
   return;
  }
  buflen = bp->cnt;
  bufoff = BUFtoCP(bp);
 }
 if (buflen > sfifo) {
  bcopy(bufoff, op, sfifo);
  buflen -= sfifo;
  bufoff += sfifo;
  break;
 } else {
  register int s;
  register struct hsisbuf *bp;

  bcopy(bufoff, op, buflen);
  sfifo -= buflen;
  op += buflen;
  buflen = 0;
  bufoff = NULL;

  s = splboard();
  if ((bp = sc->sc_inq) == NULL) {
   splx(s);
   m_freem(m);
```

```
      return;
     }
     if ((sc->sc_inq = bp->next) == NULL)
      sc->sc_intail = NULL;
     bp->next = hs->hs_free;
     hs->hs_free = bp;
     splx(s);
    }
   }
   /*
    * queue the packet to the appropriate network protocol.
    */
#if NBPFILTER > 0
   if (sc->sc_bpf) {
    register u_char c;

    op = mtod(m, u_char *);
    c = *op;
    *op = 0; /* 'in' direction */
    bpf_tap(sc->sc_bpf, op, m->m_len);
    *op = c;
   }
#endif
   if (sc->sc_raw) {
    register struct socket *so = sc->sc_raw;

    if (m->m_len > sbspace(&so->so_rcv)) {
     ++sc->sc_idrops;
     m_freem(m);
    } else {
     register struct mbuf *n = so->so_rcv.sb_mb;

     if (n) {
      while (n->m_next)
       n = n->m_next;
      n->m_next = m;
     } else
      so->so_rcv.sb_mb = m;

     so->so_rcv.sb_cc += m->m_len;
     sorwakeup(so);
    }
   } else {
    register struct ifqueue *inq;
    register int s;

    /*
     * Note: the protocol input routines all require
     * an ifp at the front of the buffer. We make
     * use of the fact that a PPP header is the same
     * size as an ifp & just overwrite the 4 bytes
     * of header without allocating new space.
```

```
     */
    s = splimp();
    op = mtod(m, u_char *);
    switch (*(u_int *)op) {

    case PPP_INET:
     inq = &ipintrq;
     schednetisr(NETISR_IP);
     break;
    default:
     splx(s);
     ++sc->sc_ibadtype;
     m_freem(m);
     continue;
    }
    if (IF_QFULL(inq)) {
     ++sc->sc_idrops;
     m_freem(m);
    } else {
     *(struct ifnet **)op = &sc->sc_if;
     IF_ENQUEUE(inq, m);
    }
    splx(s);
   }
   sc->sc_ierrcnt = 0;
  }
  /*
   * done with fifo -- if there's data left in the current buffer,
   * remember where we are.
   */
  if (bufoff && (bp = sc->sc_inq))
   bp->cnt = buflen;
  sc->sc_inoff = bufoff;
}

int
hsissoftint()
{
 register u_int isum;
 register int did_something = 0;

 TRACE(T_SOFTINT, hsis_softc, hsis_isum)
 while (1) {
  register struct hsis_softc *sc;
  register struct hsis_softc *sc_end = hsis_softc_end;
  register int s;

  s = splboard();
  isum = hsis_isum;
  hsis_isum = 0;
  splx(s);
  if (isum == 0)
```

```
    break;

   ++did_something;
   sc = hsis_softc;
   for ( ; isum && sc < sc_end; ++sc) {
    if (isum & XMIT_DONE) {
     s = splboard();
     hsis_start(sc);
     splx(s);
    }
    if (isum & RECV_DONE)
     hsis_recv_done(sc);
    isum >>= 2;
   }
  }
  TRACE(T_SOFTINT_EXIT, hsis_softc, did_something)
  return (did_something);
}

int
hsis_ioctl(ifp, cmd, data)
 register struct ifnet *ifp;
 register int cmd;
 register caddr_t data;
{
 register int unit = ifp->if_unit;
 register struct hsis_softc *sc = &hsis_softc[unit];
 register struct ifreq *ifr = (struct ifreq *)data;
 register int s = splboard(), error = 0;

 TRACE(T_IOCTL, sc, cmd)
 switch (cmd) {

 case SIOCSIFADDR:
  bzero((caddr_t)&sc->sc_dstats, sizeof(sc->sc_dstats));
  bzero((caddr_t)&sc->sc_estats, sizeof(sc->sc_estats));
  error = hsis_init(unit);
  break;

 case SIOCSIFDSTADDR:
  break;

 case SIOCSIFFLAGS:
  switch (ifp->if_flags & (IFF_UP|IFF_RUNNING)) {

  case IFF_UP:
   /* down interface just marked up */
   bzero((caddr_t)&sc->sc_dstats, sizeof(sc->sc_dstats));
   bzero((caddr_t)&sc->sc_estats, sizeof(sc->sc_estats));
   error = hsis_init(unit);
   hsis_start(sc);
   break;
```

```
        case IFF_RUNNING:
         /* up interface just marked down */
         hsis_offline(unit);
         break;
        }
        break;

       case SIOCSSDSTATS:
        *(struct ss_dstats *)ifr->ifr_data = sc->sc_dstats;
        break;

       case SIOCSSESTATS:
        *(struct ss_estats *)ifr->ifr_data = sc->sc_estats;
        break;

       case SIOCGETSYNC:
        *(struct syncmode *)ifr->ifr_data = sc->sc_sm;
        break;


       case SIOCSETSYNC:
        if (ifp->if_flags & IFF_RUNNING)
         hsis_offline(unit);
        sc->sc_sm = *(struct syncmode *)ifr->ifr_data;
        if (sc->sc_sm.sm_baudrate != 0) {
         bzero((caddr_t)&sc->sc_dstats, sizeof(sc->sc_dstats));
         bzero((caddr_t)&sc->sc_estats, sizeof(sc->sc_estats));
         error = hsis_init(unit);
         hsis_start(sc);
        }
        break;
      #ifdef MULTICAST
       case SIOCADDMULTI:
       case SIOCDELMULTI:
        switch (ifr->ifr_addr.sa_family) {
      #ifdef INET
        case AF_INET:
         break;
      #endif INET
        default:
         error = EAFNOSUPPORT;
         break;
        }
        break;
      #endif MULTICAST

       default:
        error = EINVAL;
       }
       splx(s);
       return (error);
```

```
}

int
hsis_init(unit)
 int unit;
{
 register struct hsis_softc *sc = &hsis_softc[unit];
 register struct zscc *zs = sc->sc_zs;
 register u_char *zsdev = zs->zs_addr;
 u_char wreg[sizeof(zs->zs_wreg)];
 register int s;
 register int clk = 0, txin = 0;

 TRACEL(T_INIT, sc, sc->sc_if.if_flags)
 if (sc->sc_if.if_flags & IFF_RUNNING)
  return (0);

 /* set appropriate defaults for scc */
 bcopy((caddr_t)hsis_scc_setup, (caddr_t)wreg, sizeof(wreg));

 /*
  * modify defaults as per Sun's 'syncmode' (hsi compatibility).
  */
 switch (sc->sc_sm.sm_txclock) {

 case TXC_IS_TXC:
  wreg[11] |= ZSWR11_TXCLK_TRXC;
  txin = 1;
  break;

 case TXC_IS_RXC:
  wreg[11] |= ZSWR11_TXCLK_RTXC;
  break;

 case TXC_IS_BAUD:
  wreg[11] |= ZSWR11_TXCLK_BAUD;
  clk = 1;
  break;

 case TXC_IS_PLL:
  wreg[11] |= ZSWR11_TXCLK_DPLL;
  clk = 2;
  break;

 default:
  return (EINVAL);
 }
 switch (sc->sc_sm.sm_rxclock) {

 case RXC_IS_RXC:
  wreg[11] |= ZSWR11_RXCLK_RTXC;
  break;
```

```
case RXC_IS_TXC:
 wreg[11] |= ZSWR11_RXCLK_TRXC;
 txin = 1;
 break;

case RXC_IS_BAUD:
 wreg[11] |= ZSWR11_RXCLK_BAUD;
 clk |= 1;
 break;

case RXC_IS_PLL:
 wreg[11] |= ZSWR11_RXCLK_DPLL;
 clk |= 2;
 break;

default:
 return (EINVAL);
}
if (clk) {
 register long tconst;

 if (clk > 2)
  return (EINVAL);
 tconst = sc->sc_sm.sm_baudrate;
 if (clk == 2) {
  if (! sc->sc_sm.sm_nrzi)
   return (EINVAL);
  tconst <<= 5;
 }
 tconst = HSIS_PCLK / (tconst * 2) - 2;
 if (tconst == 0)
  return (EINVAL);

 sc->sc_sm.sm_baudrate = (HSIS_PCLK / 2) / (tconst + 2);

 wreg[12] = tconst;
 wreg[13] = tconst >> 8;
 wreg[14] |= ZSWR14_BAUD_FROM_PCLK | ZSWR14_BAUD_ENA;
}
if (! txin)
 wreg[11] |= ZSWR11_TRXC_OUT_ENA | ZSWR11_TRXC_XMIT;
if (sc->sc_sm.sm_loopback)
 wreg[14] |= ZSWR14_LOCAL_LOOPBACK;
if (sc->sc_sm.sm_nrzi)
 wreg[10] |= ZSWR10_NRZI;

/*
 * if we don't have read and write buffers yet, get them.
 */
s = splboard();
if (sc->sc_inbuf == NULL) {
```

```
 if ((sc->sc_inbuf = sc->sc_hs->hs_free) == NULL) {
  printf("hsis%d: hsis_init: no free bufs.\n",
   sc->sc_if.if_unit);
  splx(s);
  return (ENOBUFS);
 }
 sc->sc_hs->hs_free = sc->sc_inbuf->next;
}
if (sc->sc_curout == NULL) {
 if ((sc->sc_curout = sc->sc_hs->hs_free) == NULL) {
  printf("hsis%d: hsis_init: no free bufs\n",
   sc->sc_if.if_unit);
  splx(s);
  return (ENOBUFS);
 }
 sc->sc_hs->hs_free = sc->sc_curout->next;
 sc->sc_curout->cnt = 0;
}
if (sc->sc_nextout == NULL) {
 if ((sc->sc_nextout = sc->sc_hs->hs_free) == NULL) {
  printf("hsis%d: hsis_init: no free bufs\n",
   sc->sc_if.if_unit);
  splx(s);
  return (ENOBUFS);
 }
 sc->sc_hs->hs_free = sc->sc_nextout->next;
 sc->sc_nextout->cnt = 0;
}

 /*
  * If we got here, sm contents must be reasonable and wreg contains
  * the new scc configuration. Disable rcvr & xmitter, load in new
  * modes then re-enable.
  */

 sc->sc_hs->hs_board[zs->zs_unit + HSIS_INT_CLK_A] = 0;
#ifdef HSIS_EXTERNAL_RCVDONE
 /*
  * if there's an external receive clock, use the external
  * done interrupt. Otherwise, enable the chip's receive intr.
  */
 if (sc->sc_sm.sm_rxclock == RXC_IS_RXC)
  sc->sc_hs->hs_board[zs->zs_unit + HSIS_EINT_ENA_A] = 0xff;
 else {
  sc->sc_hs->hs_board[zs->zs_unit + HSIS_EINT_ENA_A] = 0;
  wreg[1] |= ZSWR1_RIE_SPECIAL_ONLY;
 }
#endif
 SCC_BIC(3, ZSWR3_RX_ENABLE);
 SCC_BIC(5, ZSWR5_TX_ENABLE);

 SCC_WRITE(4, wreg[4]);
```

```
SCC_WRITE(10, wreg[10]);
SCC_WRITE(6, wreg[6]);
SCC_WRITE(7, wreg[7]);
SCC_WRITE(3, wreg[3]);
SCC_WRITE(5, wreg[5]);
SCC_WRITE(1, wreg[1]);
SCC_WRITE(9, wreg[9]);
SCC_WRITE(11, wreg[11]);
SCC_WRITE(12, wreg[12]);
SCC_WRITE(13, wreg[13]);
if (clk == 2) {
 SCC_WRITE(14, ZSWR14_DPLL_SRC_BAUD);
 SCC_WRITE(14, ZSWR14_DPLL_NRZI);
} else
 SCC_WRITE(14, ZSWR14_DPLL_DISABLE);
SCC_WRITE(14, wreg[14]);
SCC_WRITE(15, wreg[15]);

if (txin) {
 /*
  * TRxC pin is input (i.e., we're using an external clock).
  * Set latch that allows clock to get to the pin.
  */
 register u_char *board = sc->sc_hs->hs_board + zs->zs_unit;

 board[HSIS_INT_CLK_A] = 1;
 /* XXX - ext. clocks are inverted */
 board[HSIS_RX_CLK_A] = 1;
 board[HSIS_TX_CLK_A] = 1;
}
SCC_WRITE0(ZSWR0_RESET_ERRORS);
SCC_WRITE0(ZSWR0_RESET_STATUS);
SCC_WRITE0(ZSWR0_RESET_TXCRC);

/*
 * Everything should be configured. Start a dma read then
 * enable recv dma and the receiver.
 */
(void) hsis_start_dma_read(sc);
SCC_BIS(1, ZSWR1_REQ_ENABLE)
SCC_BIS(3, ZSWR3_RX_ENABLE);
/*
 * Enable the transmitter and turn on DTR.
 */
SCC_BIS(5, ZSWR5_TX_ENABLE|ZSWR5_DTR);

SCC_WRITE0(ZSWR0_RESET_ERRORS);
SCC_WRITE0(ZSWR0_RESET_STATUS);
SCC_WRITE0(ZSWR0_RESET_TXCRC);

sc->sc_ostate = 0;
sc->sc_if.if_flags |= IFF_UP|IFF_RUNNING;
```

```
 TRACE(T_INIT_DONE, sc, *zsdev)
 splx(s);
 return (0);
}

hsis_reset(unit)
 register int unit;
{
 register int s = splboard();
 register struct hsis_softc *sc = &hsis_softc[unit];
 register struct zscc *zs = sc->sc_zs;
 register u_char *zsdev = zs->zs_addr;
 register u_char *zddev = sc->sc_zd->zd_addr;
 register struct hsisbuf *bp;

 TRACE(T_RESET, sc, *zsdev)
 SCC_WRITE(3, 0);
 SCC_WRITE(5, 0);
 SCC_WRITE(15, 0);
#ifdef HSIS_EXTERNAL_RCVDONE
 sc->sc_hs->hs_board[zs->zs_unit + HSIS_EINT_ENA_A] = 0;
#endif
 (void) hsis_start_dma_read(sc);
 if (zs->zs_unit & 1) {
  DMA_BIC(ZSDMA_DER, ZSDMA_DER_RX_B_ENABLE);
 } else {
  DMA_BIC(ZSDMA_DER, ZSDMA_DER_RX_A_ENABLE);
 }
 (void) hsis_start_dma_read(sc);
 if (sc->sc_intail) {
  sc->sc_intail->next = sc->sc_hs->hs_free;
  sc->sc_hs->hs_free = sc->sc_inq;
  sc->sc_inq = NULL;
  sc->sc_intail = NULL;
 }
 bzero(sc->sc_fifo, sizeof(sc->sc_fifo));
 sc->sc_fifop = sc->sc_fifo;
 sc->sc_inoff = NULL;

 sc->sc_ostate = 0;
 if (bp = sc->sc_curout)
  bp->cnt = 0;
 if (bp = sc->sc_nextout)
  bp->cnt = 0;
 sc->sc_if.if_flags &=~ (IFF_UP|IFF_RUNNING);
 splx(s);
}

hsis_offline(unit)
 register int unit;
{
 register int s = splboard();
```

```
        register struct hsis_softc *sc = &hsis_softc[unit];
        register struct hsisbuf *bp;

        /* reset channel, then free any resources it holds */

        hsis_reset(unit);
        if (bp = sc->sc_inbuf) {
         sc->sc_inbuf = NULL;
         bp->next = sc->sc_hs->hs_free;
         sc->sc_hs->hs_free = bp;
        }
        if (bp = sc->sc_curout) {
         sc->sc_curout = NULL;
         bp->next = sc->sc_hs->hs_free;
         sc->sc_hs->hs_free = bp;
        }
        if (bp = sc->sc_nextout) {
         sc->sc_nextout = NULL;
         bp->next = sc->sc_hs->hs_free;
         sc->sc_hs->hs_free = bp;
        }
        splx(s);
       }

       /*
        * Remainder of this code is support for 'raw sync' protocol domain.
        * It should probably be in a separate file since it (should not be)
        * hsis specific but I'm too lazy to set that up just now.
        */

       #include <sys/protosw.h>
       #include <sys/domain.h>
       #include <sys/user.h>
       #include <sys/uio.h>

       int rawsync_usrsend(), rawsync_usrrecv(), rawsync_usrreq();

       extern struct domain rawsyncdomain;

       struct protosw rawsyncsw[] = {
       { SOCK_RAW, &rawsyncdomain, 0,  PR_ATOMIC,
        0,  0,  0,  0,
        rawsync_usrreq,
        0,  0,  0,  0,
        rawsync_usrsend, rawsync_usrrecv},
       };

       struct domain rawsyncdomain =
        { AF_RAWSYNC, "rawsync", 0, 0, 0,
        rawsyncsw, &rawsyncsw[sizeof(rawsyncsw)/sizeof(rawsyncsw[0])] };

       void
```

```
rawsyncinit()
{
 register struct domain *dp;

 rawsyncdomain.dom_next = domains;
 domains = &rawsyncdomain;
}

int
rawsync_usrsend(so, nam, uio, flags, rights)
 register struct socket *so;
 struct mbuf *nam;
 register struct uio *uio;
 int flags;
 struct mbuf *rights;
{
 register int len;
 register int error = 0;
 register int s;
 register struct hsis_softc *sc;
 register struct mbuf *m;
 static struct sockaddr dst = { AF_UNSPEC };

#ifdef lint
 nam = nam; rights = rights;
#endif lint
 if ((sc = (struct hsis_softc *)so->so_pcb) == NULL ||
  so != sc->sc_raw)
  return (EINVAL);

 len = uio->uio_resid;
 if (len <= 0 || len > sc->sc_if.if_mtu)
  return (EMSGSIZE);

 if (so->so_state & SS_CANTSENDMORE) {
  psignal(u.u_procp, SIGPIPE);
  return (EPIPE);
 }
 while (1) {
  s = splboard();
#ifdef SFQ
  if (! IF_SFQFULL(&sc->sc_if.if_snd,0)) /* not correct */
#else
  if (! IF_QFULL(&sc->sc_if.if_snd))
   break;
#endif SFQ
  sbwait(&so->so_snd);
  splx(s);
 }
 (void) splnet();
 MGET(m, M_DONTWAIT, MT_DATA);
 if (m == (struct mbuf *)0) {
```

```
 error = ENOBUFS;
 goto out;
 }
 if (len <= MLEN - MBUF_HDRSPACE)
 m->m_off += MBUF_HDRSPACE;
 else {
 /* too big for mbuf - use cluster */
 MCLGET(m);
 if (m->m_len != MCLBYTES) {
 /* no clusters - drop this packet */
 m_freem(m);
 error = ENOBUFS;
 goto out;
 }
 m->m_off += HDRSPACE;
 }
 m->m_len = len;
 error = uiomove(mtod(m, caddr_t), len, UIO_WRITE, uio);
 if (! error)
 error = hsis_output(&sc->sc_if, m, &dst);
out:
 splx(s);
 return (error);
}

int
rawsync_usrrecv(so, anam, uio, flags, arights)
 register struct socket *so;
 struct mbuf **anam;
 register struct uio *uio;
 int flags;
 struct mbuf **arights;
{
 register int len, mlen;
 register int error = 0;
 register int s;
 register struct hsis_softc *sc;
 register struct mbuf *m, *n;
 register struct sockbuf *sb;

 if (anam)
 *anam = NULL;
 if (arights)
 *arights = NULL;

 if ((sc = (struct hsis_softc *)so->so_pcb) == NULL ||
 so != sc->sc_raw)
 return (EINVAL);

 len = uio->uio_resid;
 if (len <= 0) {
 error = EMSGSIZE;
```

```
 goto out;
}
if (so->so_state & SS_CANTRCVMORE)
 goto out;

sb = &so->so_rcv;
sblock(so, sb);
s = splboard();
if (sb->sb_cc == 0) {
 if (so->so_error) {
  error = so->so_error;
  so->so_error = 0;
  goto release;
 }
 if (so->so_state & SS_NBIO) {
  error = EWOULDBLOCK;
  goto release;
 }
 while (sb->sb_cc == 0) {
  sbunlock(so, sb);
  sbwait(sb);
  if (so->so_error) {
   error = so->so_error;
   so->so_error = 0;
   goto release;
  }
 }
}
m = sb->sb_mb;
sb->sb_mb = m->m_next;
mlen = m->m_len;
sb->sb_cc -= mlen;
splx(s);
sbunlock(so, sb);
m->m_next = NULL;
if (mlen > len) {
 error = EMSGSIZE;
 mlen = len;
}
error = uiomove(mtod(m, caddr_t), mlen, UIO_READ, uio);
MFREE(m, n);
out:
 return (error);
release:
 splx(s);
 sbunlock(so, sb);
 return (error);
}

/*ARGSUSED*/
rawsync_usrreq(so, req, m, nam, rights)
 struct socket *so;
```

```
 int req;
 struct mbuf *m, *nam, *rights;
{
 register struct hsis_softc *sc;
 register int error = 0;

 if (rights && rights->m_len) {
  error = EOPNOTSUPP;
  goto release;
 }
 sc = (struct hsis_softc *)so->so_pcb;
 if (sc != NULL && so != sc->sc_raw) {
  error = EINVAL;
  goto release;
 }
 switch (req) {

 case PRU_ATTACH:
  if ((so->so_state & SS_PRIV) == 0)
   error = EACCES;
  else if (sc)
   error = EINVAL;
  else if (sbreserve(&so->so_snd, 4096) == 0)
   error = ENOBUFS;
  else if (sbreserve(&so->so_rcv, 4096) == 0) {
   sbrelease(&so->so_snd);
   error = ENOBUFS;
  }
  break;

 /*
  * Destroy state just before socket deallocation.
  * Flush data or not depending on the options.
  */
 case PRU_DETACH:
  if (sc == 0)
   error = ENOTCONN;
  else {
   register int s = splboard();

   sc->sc_raw = NULL;
   so->so_pcb = NULL;
   sofree(so);
   splx(s);
  }
  break;

 case PRU_BIND:
  if (sc)
   error = EISCONN;
  else {
   register struct sockaddr *addr =
```

```
    mtod(nam, struct sockaddr *);
  if (addr->sa_family != AF_RAWSYNC) {
   error = EINVAL;
   break;
  }
  sc = (struct hsis_softc *)ifunit(addr->sa_data, MLEN);
  if (sc == NULL)
   error = EADDRNOTAVAIL;
  else if (sc->sc_raw)
   error = EADDRINUSE;
  else {
   register int s = splboard();

   sc->sc_raw = so;
   so->so_pcb = (caddr_t)sc;
   splx(s);
  }
 }
 break;

/*
 * Mark the connection as being incapable of further input.
 */
case PRU_SHUTDOWN:
 socantsendmore(so);
 break;

case PRU_ABORT:
 if (sc != NULL && (so->so_state & SS_NOFDREF)) {
  register int s = splboard();

  sc->sc_raw = NULL;
  so->so_pcb = NULL;
  splx(s);
 }
 sofree(so);
 soisdisconnected(so);
 break;

case PRU_SENSE:
 /*
  * stat: don't bother with a blocksize.
  */
 return (0);

/*
 * Not supported.
 */
case PRU_CONTROL:

case PRU_CONNECT:
case PRU_CONNECT2:
```

```
        case PRU_DISCONNECT:

        case PRU_RCVOOB:
        case PRU_RCVD:

        case PRU_LISTEN:
        case PRU_ACCEPT:
        case PRU_SENDOOB:

        case PRU_SOCKADDR:
        case PRU_PEERADDR:
         error = EOPNOTSUPP;
         break;

        default:
         panic("rawsync_usrreq");
        }
release:
        if (m != NULL)
         m_freem(m);
        return (error);
}

#endif
```

5 SFQ PLUS VIRTUAL CLOCK SOURCE

This section contains the implementation of the hybrid algorithm, SFQ plus VirtualClock. In addition to the code developed by SRI, the implementation uses BBN's traffic control abstraction, BBN's release 1.12 of ST-II, and the HSI/S driver from LBL. All code not developed by SRI is available from BBN. It is important to use the HSI/S driver from BBN, since it has been altered to accommodate the traffic control abstraction.

This section contains all source developed by SRI and any source from BBN that SRI had to modify. In particular, the section includes the following files: a config file for an SFQ plus VirtualClock kernel, files.cmn, sfq.h, sfq.c, hybrid_int.c, if.h (diffs), if_aux.c, st2_resource.h, hsis.c, and st2_proto.c. Except for sfq.c, sfq.h and hybrid_int.c, the files are printed with our modifications in bold type. The code in sfq.c and sfq.h is similar to the code presented in the previous section, except that the macro calls have been made into routines, and all references have been changed from the ifnet structure to the aNetIF structure, which is necessary for BBN's traffic control. The file named hybrid_int.c contains the appropriate calls to SFQ plus VirtualClock encapsulated in the interface definition for a new traffic control abstraction.

Once a kernel is built, it is necessary to invoke the ifconfig program provided in BBN's software release, in order to attach our traffic control algorithm to a particular interface. The name of the traffic-control algorithm is SFQ_VC; so the command, given as root, would be

ifconfig.sun4c hsis0 tc SFQ_VC

to attach this control algorithm to interface 0 of an HSI/S board, for example.

The reader who uses this material for building a kernel containing SFQ plus VirtualClock is assumed to be familiar with the process of building a kernel, the kernel directory structure, and the installation notes from BBN entitled "Installing Generic Traffic Control & Resource Management, and ST-II in a SunOS 4.1.x Kernel." These notes are provided in BBN's software release.

```
# @(#) $Id: DARTNET,v 1.5 92/12/23 14:39:41 casner Exp $
#
# This config file describes the "released" Sun-4c kernel for use in DARTnet,
# including the HSIS driver and IP multicast support.
#
# The following lines include support for all Sun-4c CPU types.
# There is little to be gained by removing support for particular
# CPUs, so you might as well leave them all in.
#
machine  "sun4c"
cpu  "SUN4C_60" # Sun-4/60 (any Sun-4c)

ident  "HYBRID"

#
# This kernel supports about 8 users. Count one user for each
# timesharing user, one for each window that you typically use, and one
# for each diskless client you serve. This is only an approximation used
# to control the size of various kernel data structures, not a hard limit.
#
maxusers 16

options GENERIC
options INET  # basic networking support - mandatory
options TRAFFIC_CONTROL # Generic traffic control support
options DARTNET # DARTNET specific stuff
#options FAIR_SHARE # Fair Share traffic control support
options VIRTUAL_CLOCK # Virtual Clock traffic control support
options STII # Protocol processing software for ST-II
options STIIAPI # Socket interface for ST-II
options STIIDEBUG # ST-II Debugging, if desired
options SFQ # SFQ
options SFQ_VC # SFQ with VC
options MY_FIFO # FIFO queue model
options UFS  # filesystem code for local disks
options NFSCLIENT # NFS client side code
options NFSSERVER # NFS server side code
options MULTICAST # IP multicast support
options MROUTING # IP multicast routing support
options HSFS  # High Sierra (ISO 9660) CD-ROM file system

options TCPDEBUG # TCP debugging, see trpt(8)

#
# The following option includes the COIP-Kernel code for ST-II (RFC 1190)
#
#options STII  # Protocol processing software for ST-II
#options STIIAPI  # Socket interface for ST-II
#options STIIDEBUG # ST-II Debugging, if desired

#
# The following option adds support for loadable kernel modules.
```

```
#
options VDDRV  # loadable modules

#
# The following option adds support for SunView 1 journaling.
#
options WINSVJ  # SunView 1 journaling support

#
# Build one kernel based on this basic configuration.
# It will use the generic swap code so that you can have
# your root filesystem and swap space on any supported device.
# Put the kernel configured this way in a file named "vmunix".
#
config vmunix swap generic

#
# Include support for all possible pseudo-devices.
#
# The first few are mostly concerned with networking.
# You should probably always leave these in.
#
pseudo-device pty  # pseudo-tty's, also needed for SunView
pseudo-device ether  # basic Ethernet support
pseudo-device loop  # loopback network - mandatory
pseudo-device encap4 init encapattach # allow 4 IP tunnels
pseudo-device coip init coipdomaininit # COIP/ST-II

pseudo-device dbx

#
# The next few are for SunWindows support, needed to run SunView 1.
#
pseudo-device win128  # window devices, allow 128 windows
pseudo-device dtop1  # desktops (screens), allow 4
pseudo-device ms  # mouse support

#
# The following is needed to support the Sun keyboard, with or
# without the window system.
#
pseudo-device kb  # keyboard support

#
# The "open EEPROM" pseudo-device is required to support the
# eeprom command.
#
pseudo-device openeepr # onboard configuration NVRAM

pseudo-device bpfilter 32 # Berkeley packet filter

#
```

```
# The following is for the "clone" device, used with streams devices.
# This is required if you include streams NIT support, RFS, or an audio
# device.
#
pseudo-device clone  # clone device

#
# The following section describes which standard device drivers this
# kernel supports.
#
device-driver sbus  # 'driver' for sbus interface
device-driver bwtwo  # monochrome frame buffer
device-driver cgthree  # 8-bit color frame buffer
device-driver cgsix  # 8-bit accelerated color frame buffer
device-driver dma  # 'driver' for dma engine on sbus interface
device-driver esp  # Emulex SCSI interface
device-driver fd  # Floppy disk
#device-driver audioamd # AMD79C30A sound chip
device-driver bsdaudio # BSD audio driver to replace Sun's
device-driver le  # LANCE ethernet
device-driver zs  # UARTs
device-driver hsis
options  HSIS_TRACE
options  HSIS_EXTERNAL_RCVDONE

#
# The following section describes SCSI device unit assignments.
#
scsibus0 at esp    # declare first scsi bus
disk sd0 at scsibus0 target 3 lun 0 # first hard SCSI disk
disk sd1 at scsibus0 target 1 lun 0 # second hard SCSI disk
disk sd2 at scsibus0 target 2 lun 0 # third hard SCSI disk
disk sd3 at scsibus0 target 0 lun 0 # fourth hard SCSI disk
tape st0 at scsibus0 target 4 lun 0 # first SCSI tape
tape st1 at scsibus0 target 5 lun 0 # second SCSI tape
disk sr0 at scsibus0 target 6 lun 0 # CD-ROM device
```

```
#
# @(#)files.cmn 2.83 90/08/02 SMI
# Copyright (c) 1990 by Sun Microsystems, Inc.
#
des/des_crypt.c  optional NFSCLIENT
des/des_crypt.c  optional NFSSERVER
des/des_soft.c  optional NFSCLIENT CRYPT
des/des_soft.c  optional NFSSERVER CRYPT
hsfs/hsfs_node.c optional HSFS
hsfs/hsfs_subr.c optional HSFS
hsfs/hsfs_vfsops.c optional HSFS
hsfs/hsfs_vnodeops.c optional HSFS
krpc/klm_kprot.c standard
krpc/klm_lockmgr.c standard
lofs/lo_subr.c  optional LOFS
lofs/lo_vfsops.c optional LOFS
lofs/lo_vnodeops.c optional LOFS
lwp/alloc.c  optional LWP
lwp/cntxt.c  optional LWP
lwp/condvar.c  optional LWP
lwp/lwperror.c  optional LWP
lwp/monitor.c  optional LWP
lwp/process.c  optional LWP
lwp/schedule.c  optional LWP
net/af.c  standard
net/bpf.c  optional bpfilter
net/bpf_filter.c optional bpfilter
net/if.c  standard
net/ifdev.c  optional ifd
net/nit.c  optional NIT
net/nit_buf.c  optional nbuf
net/nit_if.c  optional snit
net/if_aux.c optional TRAFFIC_CONTROL
#net/fs.c optional TRAFFIC_CONTROL FAIR_SHARE
net/nit_pf.c  optional pf
net/packetfilt.c optional pf
net/raw_cb.c  standard
net/raw_usrreq.c standard
net/route.c  standard
net/if_sl.c  optional sl INET
net/slcompress.c optional sl INET
netat/aarp.c  optional atalk
netat/at_proto.c optional atalk
netat/ddp_usrreq.c optional atalk
netdna/dna.c  optional dnalink device-driver
netinet/if_ether.c optional ether INET
netinet/if_encap.c optional encap INET
netinet/if_loop.c optional loop INET
netinet/igmp.c  optional MULTICAST INET
netinet/in.c  optional INET
netinet/in_pcb.c optional INET
netinet/in_proto.c optional INET
```

```
netinet/ip_icmp.c optional INET
netinet/ip_input.c optional INET
netinet/ip_mroute.c optional MROUTING MULTICAST INET
netinet/ip_output.c optional INET
netinet/raw_ip.c optional INET
netinet/st2.c  optional STII INET
netinet/st2_api.c optional STIIAPI INET
netinet/st2_asm.s optional STII INET
netinet/st2_cmp.c optional STII INET
netinet/st2_proto.c optional STII INET
netinet/st2_resource.c optional STII INET
netinet/st2_routing.c optional STII INET
netinet/dbx_st2.c optional STII dbx INET symbolic-info
netinet/tcp_debug.c optional INET
netinet/tcp_input.c optional INET
netinet/tcp_output.c optional INET
netinet/tcp_subr.c optional INET
netinet/tcp_timer.c optional INET
netinet/tcp_usrreq.c optional INET
netinet/udp_usrreq.c optional INET
netinet/dbx_inet.c optional dbx INET symbolic-info
netns/idp_usrreq.c optional xns
netns/ns_error.c optional xns
netns/ns_ether.c optional xns
netns/ns_input.c optional xns
netns/ns_ip.c  optional xns
netns/ns_output.c optional xns
netns/ns_pcb.c  optional xns
netns/ns_proto.c optional xns
netns/spp_debug.c optional xns
netns/spp_usrreq.c optional xns
netns/xns.c  optional xns
nettli/tcp_tli.c optional tcptli
nettli/tcp_tliaux.c optional tcptli
nettli/tcp_tlisubr.c optional tcptli
nettli/ti_mod.c  optional tim
nettli/ti_rdwr.c optional tirw
nfs/dbx_nfs.c  optional dbx NFSCLIENT symbolic-info
nfs/dbx_nfs.c  optional dbx NFSSERVER symbolic-info
nfs/nfs_client.c optional NFSCLIENT
nfs/nfs_common.c optional NFSCLIENT
nfs/nfs_common.c optional NFSSERVER
nfs/nfs_dump.c  optional NFSCLIENT
nfs/nfs_export.c optional NFSSERVER
nfs/nfs_server.c optional NFSSERVER
nfs/nfs_subr.c  optional NFSCLIENT
nfs/nfs_vfsops.c optional NFSCLIENT
nfs/nfs_vnodeops.c optional NFSCLIENT
nfs/nfs_xdr.c  optional NFSCLIENT
nfs/nfs_xdr.c  optional NFSSERVER
os/au_msg_wrappers.c optional SYSAUDIT IPCMESSAGE
os/au_quot_wrappers.c optional SYSAUDIT QUOTA UFS
```

```
os/au_sem_wrappers.c optional SYSAUDIT IPCSEMAPHORE
os/au_shm_wrappers.c optional SYSAUDIT IPCSHMEM
os/au_wrappers.c optional SYSAUDIT
os/dbx_sys.c  optional dbx symbolic-info
os/heap_kmem.c  standard
os/init_dbx.c  standard symbolic-info
os/init_main.c  standard
os/init_sysent.c standard
os/ipc_msg.c  optional IPCMESSAGE
os/ipc_sem.c  optional IPCSEMAPHORE
os/ipc_shm.c  optional IPCSHMEM
os/ipc_subr.c  optional IPCMESSAGE
os/ipc_subr.c  optional IPCSEMAPHORE
os/ipc_subr.c  optional IPCSHMEM
os/kern_acct.c  optional SYSACCT
os/kern_audit.c  optional SYSAUDIT
os/kern_clock.c  standard
os/kern_descrip.c standard
os/kern_exec.c  standard
os/kern_exit.c  standard
os/kern_fork.c  standard
os/kern_mman.c  standard
os/kern_proc.c  standard
os/kern_prot.c  standard
os/kern_resource.c standard
os/kern_sig.c  standard
os/kern_softint.c standard
os/kern_subr.c  standard
os/kern_synch.c  standard
os/kern_time.c  standard
os/kern_trace.c  optional TRACE
os/kern_xxx.c  standard
os/str_buf.c  standard
os/str_io.c  standard
os/str_sp.c  optional sp
os/str_syscalls.c standard
os/subr_dump.c  standard
os/subr_log.c  standard
os/subr_mcount.c optional profiling-routine
os/subr_prf.c  standard
os/subr_rmap.c  standard
os/subr_xxx.c  standard
os/sys_generic.c standard
os/sys_process.c standard
os/sys_socket.c  standard
os/syscalls.c  optional SYSCALLTRACE
os/tty.c  standard
os/tty_ldterm.c  standard
os/tty_pty.c  optional pty
os/tty_ptyconf.c optional pty
os/tty_subr.c  standard
os/tty_tb.c  optional tb
```

```
os/tty_ttcompat.c standard
os/tty_tty.c  standard
os/uipc_domain.c standard
os/uipc_mbuf.c  standard
os/uipc_proto.c  standard
os/uipc_socket.c standard
os/uipc_socket2.c standard
os/uipc_syscalls.c standard
os/uipc_usrreq.c standard
os/vfs.c  standard
os/vfs_bio.c  standard
os/vfs_conf.c  standard
os/vfs_dnlc.c  standard
os/vfs_io.c  standard
os/vfs_lookup.c  standard
os/vfs_pathname.c standard
os/vfs_syscalls.c standard
os/vfs_vnode.c  standard
os/vfs_xxx.c  standard
os/vm_meter.c  standard
os/vm_pageout.c  standard
os/vm_sched.c  standard
os/vm_subr.c  standard
pcfs/dbx_pcfs.c  optional dbx PCFS symbolic-info
pcfs/pc_alloc.c  optional PCFS
pcfs/pc_dir.c  optional PCFS
pcfs/pc_node.c  optional PCFS
pcfs/pc_subr.c  optional PCFS
pcfs/pc_vfsops.c optional PCFS
pcfs/pc_vnodeops.c optional PCFS
rfs/adv.c  optional RFS
rfs/auth.c  optional RFS
rfs/canon.c  optional RFS
rfs/cirmgr.c  optional RFS
rfs/comm.c  optional RFS
rfs/fumount.c  optional RFS
rfs/netboot.c  optional RFS
rfs/que.c  optional RFS
rfs/queue.c  optional RFS
rfs/recover.c  optional RFS
rfs/rfadmin.c  optional RFS
rfs/rfcanon.c  optional RFS
rfs/rfs_misc.c  optional RFS
rfs/rfs_param.c  optional RFS
rfs/rfs_serve.c  optional RFS
rfs/rfs_syscalls.c optional RFS
rfs/rfs_vfsops.c optional RFS
rfs/rfs_vnodeops.c optional RFS
rfs/rfs_xdr.c  optional RFS
rfs/rfsys.c  optional RFS
rfs/rsc.c  optional RFS
rpc/auth_des.c  optional NFSCLIENT
```

```
rpc/auth_kern.c  standard
rpc/authdes_prot.c optional NFSCLIENT
rpc/authdes_subr.c optional NFSCLIENT
rpc/authunix_prot.c standard
rpc/clnt_kudp.c  standard
rpc/clnt_perror.c standard
rpc/dbx_rpc.c  optional dbx symbolic-info
rpc/key_call.c  optional NFSCLIENT
rpc/key_call.c  optional NFSSERVER
rpc/key_prot.c  optional NFSCLIENT
rpc/key_prot.c  optional NFSSERVER
rpc/kudp_fastsend.c standard
rpc/pmap_kgetport.c standard
rpc/pmap_prot.c  standard
rpc/pmap_rmt.c  standard
rpc/rpc_callmsg.c optional NFSCLIENT
rpc/rpc_callmsg.c optional NFSSERVER
rpc/rpc_prot.c  standard
rpc/subr_kudp.c  standard
rpc/svc.c  optional NFSSERVER
rpc/svc_auth.c  optional NFSSERVER
rpc/svc_auth_unix.c optional NFSSERVER
rpc/svc_kudp.c  optional NFSSERVER
rpc/svcauth_des.c optional NFSSERVER
rpc/xdr.c  standard
rpc/xdr_array.c  standard
rpc/xdr_mbuf.c  standard
rpc/xdr_mem.c  standard
rpc/xdr_reference.c standard
rpcsvc/bootparam_xdr.c optional NFSCLIENT
rpcsvc/mountxdr.c optional NFSCLIENT
sfq/sfq.c optional SFQ
sfq_vc/hybrid_int.c optional SFQ_VC
specfs/bdev_dsort.c standard
specfs/fifo_vnodeops.c standard
specfs/spec_clone.c optional clone
specfs/spec_subr.c standard
specfs/spec_vfsops.c standard
specfs/spec_vnodeops.c standard
sunwindowdev/winsvj.c optional WINSVJ
tfs/tfs_subr.c  optional TFS
tfs/tfs_vfsops.c optional TFS
tfs/tfs_vnodeops.c optional TFS
tfs/tfs_xdr.c  optional TFS
tmpfs/tmp_dir.c  optional TMPFS
tmpfs/tmp_subr.c optional TMPFS
tmpfs/tmp_tnode.c optional TMPFS
tmpfs/tmp_vfsops.c optional TMPFS
tmpfs/tmp_vnodeops.c optional TMPFS
ufs/dbx_ufs.c  optional dbx UFS symbolic-info
ufs/quota.c  optional QUOTA UFS
ufs/quota_syscalls.c optional QUOTA UFS
```

```
ufs/quota_ufs.c  optional QUOTA UFS
ufs/ufs_alloc.c  optional UFS
ufs/ufs_bmap.c  optional UFS
ufs/ufs_dir.c  optional UFS
ufs/ufs_efs.c  optional UFS efs device-driver
ufs/ufs_inode.c  optional UFS
ufs/ufs_lockf.c  optional UFS
ufs/ufs_subr.c  optional UFS
ufs/ufs_tables.c optional UFS
ufs/ufs_vfsops.c optional UFS
ufs/ufs_vnodeops.c optional UFS
vm/dbx_vm.c  optional dbx symbolic-info
vm/seg_dev.c  standard
vm/seg_map.c  standard
vm/seg_u.c  standard
vm/seg_vn.c  standard
vm/vm_anon.c  standard
vm/vm_as.c  standard
vm/vm_mp.c  standard
vm/vm_page.c  standard
vm/vm_pvn.c  standard
vm/vm_rm.c  standard
vm/vm_seg.c  standard
vm/vm_swap.c  standard
```

```
#ifdef notdef
/* #ifndef lint */
static char rcsid[] = "@(#)$Id: sfq.h,v 1.1 93/04/19 20:44:46 denny Exp Locker:
denny $";
static char copyright[] = "Copyright (c) 1992 SRI International,
denny@erg.sri.com";
/* #endif lint */
#endif notdef

/*
 * Copyright (c) 1992 SRI International. All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by SRI International, Menlo Park, CA. The name SRI International
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */

#ifdef SFQ

#include <sys/param.h>
#include <sys/mbuf.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/errno.h>
#include <sys/time.h>

#include <net/if.h>
#include <netinet/in.h>
#include <netinet/in_var.h>
#include <netinet/in_systm.h>
#include <netinet/ip.h>

#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/mbuf.h>
/*
 * Debug register. Set to 0xffffffff to enable debug statements.
 */
extern long   sfqdebug;

#define DPRT(c, x)  if(sfqdebug&c)printf x;

/* debugging flags */
```

```
#define TR_ENQ   1<<0 /* Enqueue flag */
#define TR_DEQ   1<<1 /* Dequeue flag */
#define TR_STA   1<<2 /* Statistics and
 /* error checking */
 /* flag */

#ifdef KERNEL
#define malloc(x)  new_kmem_zalloc((u_int)(x), KMEM_SLEEP)
#define free    kmem_free
#endif KERNEL

/*
 * Overview of SFQ implementation
 */



#define MOD_INPUT_VALUE 8 /* used by call to modit */
#define FQ_HASHTBLSIZ   257 /* The size of the hash table */
     /* is 2 * MOD_INPUT_VALUE + 1.*/
 /* This also indicates the */
     /* number of queues allocated.*/
#define FQ_HASHBUFLEN  8 /* 2*sizeof(struct ip_addr) */
#define FQ_MAXFCFSQLEN  100 /* Maximum FCFS Queue length per */
 /* queue */


/* Hashing functions that can be used */

unsigned long hash1 ();
unsigned long hash2 ();
unsigned long hash3 ();
unsigned long hash4 ();
unsigned long hash5 ();

/* Individual queue description */
struct ifqueue2 {
 struct mbuf  *ifq_head; /* pointers to pkts in this */
 struct mbuf  *ifq_tail; /* queue */
 int   ifq_len; /* length of queue */
 int   ifq_maxlen; /* maximum number of entries */
 int   ifq_drops; /* number dropped */
 int   ifq_sent; /* for debugging-number sent */
 struct ifqueue2 *ifq_forw; /* active list pointer */
 struct ifqueue2 *ifq_back; /* i.e. where this queue is in */
    /* the transmission queue */
 int   ifq_label; /* For debugging only */
};

/* Main top level data structure, replaces datatype for the ifq_head */
/* pointer in the ifnet structure */
struct sfq {
```

```
    int   fq_len;  /* Total number of packets in FQ chain */
    unsigned long  (*fq_hash)(); /* Hash function */
    long  fq_hashlen; /* Length of hash data */
    struct ifqueue2 *fq_index; /* Points to the head of the active list */
    long  fq_seed; /* Hash function seed */
    struct ifqueue2 fq_hashtbl[FQ_HASHTBLSIZ]; /* Hash table area */
    u_char  fq_hashbuf[FQ_HASHBUFLEN];/* Temporary work area */

};

#ifndef PPP_HDRSPACE
#define PPP_HDRSPACE 4
#endif
#ifndef ETHER_HDRSPACE
#define ETHER_HDRSPACE 14
#endif

/* returns true if all SFQ queues are empty */

#define SFQ_EMPTY(extifp) (((struct sfq *) (extifp)->iftc_state2p)->fq_index \
    == (struct ifqueue2 *)NULL)


#endif SFQ
```

```
#ifdef notdef
/* #ifndef lint */
static char rcsid[] = "@(#)$Id: sfq.c,v 1.1 93/04/19 20:42:39 denny Exp Locker:
denny $";
static char copyright[] = "Copyright (c) 1992 SRI International,
denny@erg.sri.com";
/* #endif lint */
#endif notdef

/*
 * Copyright (c) 1992 SRI International. All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by SRI International, Menlo Park, CA. The name SRI International
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */

/* This file contains all the routines for manipulating an SFQ. */
/* It includes the initalization, enqueueing, dequeuing, */
/* queue full, queue okay functionality. The hash functions and */
/* mod function are also located in this file.   */


#ifdef SFQ

#include <sys/param.h>
#include <sys/mbuf.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/errno.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/kmem_alloc.h>

#include <net/if.h>
#include <netinet/in.h>
#include <netinet/in_var.h>
#include <netinet/in_systm.h>
#include <netinet/ip.h>


#ifndef FALSE
#define FALSE (0)
```

```
#endif

#ifndef TRUE
#define TRUE (1)
#endif

#include "sfq.h"

long   sfqdebug = 0x0; /* printing debug flag */


/* modit returns a mod (2**m + 1): a must be positive */
/* The algorithm is taken from Volume II of Knuth */

modit(a, m)

 unsigned long  a;
 unsigned long  m;

 {
 int mask, bp1, c, n, x;

 mask = (1 << m) - 1;
 bp1 = mask + 2;

 for (;;)
  {
  c = a & mask;
  n = a >> m;
  x = c - n;
  if (x >= 0)
   return (x);
  if (x >= -bp1)
   return (x + bp1);
  a = -x;  /* masking only works on positive values */

  c = a & mask;
  n = a >> m;
  x = c - n;
  if (x > 0)
   return (bp1 - x);
  if (x > -bp1)
   return (-x);
  a = -x;
  }
 }


/* sfq_drain_func removes and frees all packets on the queues associated */
/* with SFQ. npktsp contains the count of how many packets were removed. */
/* nbytesp contains the count of how many bytes were freed. */
```

```
void sfq_drain_func(gifcp, npktsp, nbytesp)
register struct aNetIf *gifcp;
unsigned long   *npktsp, *nbytesp;
{
 int  i;
 struct sfq *q;
 struct ifqueue2 *fcq;
 struct mbuf *m, *n, *tmp;
 unsigned long npkts = 0,
 nbytes = 0;

 q = (struct sfq *) gifcp->iftc_state2p;
 for (i = 0; i < FQ_HASHTBLSIZ; i++) /* loop through all the queues */
 {
 fcq = &q->fq_hashtbl[i];
 if (fcq->ifq_head != NULL)   /* Is this queue not empty? */
 {
  n = fcq->ifq_head; /* remove packets in this queue */
  while (m = n)
  {
  n = m->m_act;
  for (tmp = m; tmp != (struct mbuf *) NULL; tmp = tmp->m_next)
 {
   nbytes += tmp->m_len;
  }
  npkts++;
  m_freem(m);
 }
 }

 fcq->ifq_head = NULL; /* reinitialize all variables */
 fcq->ifq_tail = NULL;
 fcq->ifq_len = 0;
 fcq->ifq_drops = 0;
 fcq->ifq_sent = 0;
 fcq->ifq_forw = NULL;
 fcq->ifq_back = NULL;
 }
 q->fq_len = 0;
 q->fq_index = NULL;
 q->fq_seed = 0;
 if ( npktsp != (unsigned long *) NULL )
 *npktsp = npkts;

 if ( nbytesp != (unsigned long *) NULL )
  *nbytesp = nbytes;
}


/* IF_SFQFULL checks to see if a particular queue is full. The mbuf */
/* is used to determine which queue you wish to check. Returns TRUE if */
```

```
/* the queue is full; FALSE otherwise.     */


int IF_SFQFULL(gifcp, m)
 struct aNetIf *gifcp;
 struct mbuf  *m;
{
 struct sfq  *sfq;
 struct mbuf *m0;
 struct ip  *ip;
 struct ifqueue2 *q;
 long  s; /* value from hash function */
 u_char *ucp; /* pointer to IP pkt */
 int   local_hdr_length; /* number of bytes for local hdr */
 int index;

 if ( gifcp->ifc_name[0] == 'h' )
 local_hdr_length = PPP_HDRSPACE;
 else if ( (gifcp->ifc_name[0] == 'l') || (gifcp->ifc_name[0] == 'i') )
 local_hdr_length = ETHER_HDRSPACE;
 else
 panic("IF_SFQFULL: unknown interface name\n");
 sfq = (struct sfq *)gifcp->iftc_state2p;

 if (m != NULL) { /* do we have a packet? */


m0 = m;
if (m0->m_len == local_hdr_length) { /* pkt size legality checks */
 m0 = m0->m_next;
 if (m0 == NULL)
 panic("sfq_nq_func: local header only\n");
 ucp = mtod ((m0), u_char *);
} else if (m0->m_len >= (local_hdr_length + sizeof(struct ip))) {
 ucp = mtod ((m0), u_char *);
 ucp = (u_char *) (((int) ucp) + local_hdr_length);
} else {
 panic("IF_SFQFULL: invalid m0 elem: no valid data or header\n");
}

ip = (struct ip *) ucp;  /* Assumes packet is an IP pkt */
if (ip->ip_v == IPVERSION)
{
 /* Extract fields to be hashed and put into buffer */
/* i.e., IP source and destination address */

 bcopy (&ip->ip_src, sfq->fq_hashbuf, sfq->fq_hashlen);

 /* Compute hash entry */
  s = (*sfq->fq_hash) (sfq->fq_seed, sfq->fq_hashbuf);
/* Mod hash result to fit into table */
 index = modit(s,MOD_INPUT_VALUE);
```

```
 if (index >= FQ_HASHTBLSIZ)
  printf("IF_SFQFULL: invalid queue index\n");
 }
 else
  index = FQ_HASHTBLSIZ -1; /* non-IP, i.e. ST-II control packets
     /* go here */
 }
 else
 index = FQ_HASHTBLSIZ -1; /* raw packets go here */

 q = &sfq->fq_hashtbl[index]; /* find the queue */
 if (q->ifq_len >= q->ifq_maxlen) /* is length not ok? */
 return(TRUE);
 else
 return(FALSE);

}

/* sfq_chk checks the integrity of the SFQ, used for debugging */


sfq_chk(q)
struct sfq *q;
{
 int  i, total = 0, nelem;
 struct ifqueue2 *fcq;
 struct mbuf *m;

 DPRT(TR_STA,("sfq_chk: Seed = %d SFQ len = %d\n",q->fq_seed, q->fq_len ));

 for (i = 0; i < FQ_HASHTBLSIZ; i++)
 {
 fcq = &q->fq_hashtbl[i];
 if (fcq->ifq_len != 0)
 DPRT(TR_STA,("sfq_chk: FQ = %d, len = %d, drops = %d, sent = %d\n",
 fcq->ifq_label, fcq->ifq_len, fcq->ifq_drops, fcq->ifq_sent));
 if (fcq->ifq_head != 0)
 {
   total += fcq->ifq_len;
  for (nelem = 0, m = fcq->ifq_head; m; m = m->m_act)
   nelem++;
 if (fcq->ifq_len != nelem)
 {
   DPRT(TR_STA, ("sfq_chk: Inconsistency in q structure for %d\n",i));
  DPRT(TR_STA, ("ifq_len = %d count = %d\n", fcq->ifq_len, nelem));
  }
 }
 }
 if (total != q->fq_len)
 {
 DPRT(TR_STA,("sfq_chk: inconsistency in total pkt count\n"));
 DPRT (TR_STA, ("sfq_chk: Total number of packets = %d fq_len = %d\n",
```

```
    total, q->fq_len));
  }
}


/* sfq_nq_func returns FALSE if a packet failed to be put on the */
/* associated queue; TRUE otherwise. Note: Due to the structure in the */
/* HSIS driver, this routine can called for a raw packet before the */
/* driver actually gets the packet. Since SFQ needs the IP source and */
/* destination address from the packet to figure out what queue to put */
/* it in, all raw packets automatically get put in the same queue */
/* regardless of their source or destination. Since it is assumed that */
/* raw packets are not a great percentage of network traffic, the */
/* effects should be negligible. If a packet is not an IP packet, it */
/* is also placed in the same queue as raw packets. In addition, */
/* a side effect of this routine is that the drop counter for a queue */
/* gets incremented if there is no room for the given IP packet. */


int sfq_nq_func(gifcp, m)
struct aNetIf  *gifcp;
struct mbuf *m;
{
 struct ifqueue *ifq;
 struct sfq  *sfq;
 struct mbuf *m0;
 struct ip  *ip;
 struct ifqueue2 *q, *q2;
 long  s; /* value returned from hash function */
 u_char *ucp;
 int index;
 int local_hdr_length;


 if ( gifcp->ifc_name[0] == 'h' )
 local_hdr_length = PPP_HDRSPACE;
 else if ( (gifcp->ifc_name[0] == 'l') || (gifcp->ifc_name[0] == 'i') )
 local_hdr_length = ETHER_HDRSPACE;
 else
 panic("sfq_nq_func: unknown interface name\n");
 sfq = (struct sfq *)gifcp->iftc_state2p;

 if (m != NULL) { /* do we have a packet? */


 m0 = m;
 if (m0->m_len == local_hdr_length) { /* pkt size legality checks */
  m0 = m0->m_next;
  if (m0 == NULL)
  panic("sfq_nq_func: local header only\n");
  ucp = mtod ((m0), u_char *);
 } else if (m0->m_len >= (local_hdr_length + sizeof(struct ip))) {
```

```
 ucp = mtod ((m0), u_char *);
 ucp = (u_char *) (((int) ucp) + local_hdr_length);
} else {
 panic("sfq_nq_func: invalid m0 elem: no valid data or header\n");
}

ip = (struct ip *) ucp;  /* Assumes packet is an IP pkt */
if (ip->ip_v == IPVERSION)
{
 /* Extract fields to be hashed and put into buffer */
/* i.e., IP source and destination address */

 bcopy (&ip->ip_src, sfq->fq_hashbuf, sfq->fq_hashlen);

 /* Compute hash entry */
  s = (*sfq->fq_hash) (sfq->fq_seed, sfq->fq_hashbuf);
/* Mod hash result to fit into table */
 index = modit(s,MOD_INPUT_VALUE);
if (index >= FQ_HASHTBLSIZ)
 printf("sfq_nq_func: invalid queue index\n");
}
else
 index = FQ_HASHTBLSIZ -1; /* non-IP, i.e. ST-II control packets
    /* go here */
}
else
index = FQ_HASHTBLSIZ -1; /* raw packets go here */

q = &sfq->fq_hashtbl[index]; /* find the queue */
if (q->ifq_len >= q->ifq_maxlen) /* is length not ok? */
{
if (m != NULL)
q->ifq_drops++; /* incremented drop counter */
return(FALSE);
}
else
{
if (m == NULL) /* hack for raw send and because we */
 return TRUE; /* are combining the enqueue and qfull */
   /* macros for efficiency */
/* There is no packet yet */

if (ip->ip_v == IPVERSION)
{
 DPRT (TR_ENQ,("sfq_nq_func: Adding IP pkt to [%d]\n",index));
/* printf("src %s ", inet_ntoa(ip->ip_src));
printf("dest %s\n", inet_ntoa(ip->ip_dst)); */
 }
 else
   DPRT (TR_ENQ, ("sfq_nq_func: Adding NON-IP packet to [%d]\n", index));
```

```
    /* Make active list entry */

    if (q->ifq_head == NULL) {
     if (sfq->fq_index == NULL) {
     sfq->fq_index = q;
     q->ifq_forw = q;
     q->ifq_back = q;
     } else {
     q2 = sfq->fq_index;
     q->ifq_back = q2->ifq_back;
     q->ifq_forw = q2;
     q2->ifq_back->ifq_forw = q;
     q2->ifq_back = q;
    }
    }

    /* Store packet onto queue */
    (m)->m_act = 0;
    if (q->ifq_tail == 0) {
     q->ifq_head = m;
    } else {
     q->ifq_tail->m_act = m;
    }
    q->ifq_tail = m;

    q->ifq_len++;  /* FCFS queue counter */
    sfq->fq_len++; /* SFQ packet counter */
   /* (ifq)->ifq_len++;*/ /* ifnet counter */
     /* don't increment ifnet counter */
     /* because it is used for the stii */
     /* queue */

    return(TRUE);
    }
   }


   /* sfq_init_func: function which allocates and initializes SFQ structure */

   /*
    * gifcp ::= Pointer to extended ifnet structure
    * hashfnc ::= Pointer to hash function
    */
   int sfq_init_func(gifcp, hashfnc)
   struct aNetIf *gifcp;
   unsigned long  (*hashfnc)();
   {
     struct sfq  *_sfq;
     int   _i;

     /* Allocate FQ structure and attach to interface ifq */
     _sfq = (struct sfq *) malloc (sizeof (struct sfq));
```

```
  if (_sfq == (struct sfq *) NULL)
  return(ENOMEM);
 gifcp->iftc_state2p = (caddr_t) _sfq;

  /* Initialize variables */
  _sfq->fq_index = NULL;
  _sfq->fq_seed = 0;
  _sfq->fq_len = 0;
  _sfq->fq_hash = hashfnc;
  _sfq->fq_hashlen = FQ_HASHBUFLEN;
  for (_i = 0; _i < FQ_HASHTBLSIZ; _i++) {
  _sfq->fq_hashtbl[_i].ifq_head = NULL;
  _sfq->fq_hashtbl[_i].ifq_tail = NULL;
  _sfq->fq_hashtbl[_i].ifq_len = 0;
  _sfq->fq_hashtbl[_i].ifq_drops = 0;
  _sfq->fq_hashtbl[_i].ifq_sent = 0;
  _sfq->fq_hashtbl[_i].ifq_maxlen = FQ_MAXFCFSQLEN;
  _sfq->fq_hashtbl[_i].ifq_forw = NULL;
  _sfq->fq_hashtbl[_i].ifq_back = NULL;
  _sfq->fq_hashtbl[_i].ifq_label = _i;
  }
  return(0);
}


/* sfq_dq_func: function which dequeues the next packet for tranmission */
/* from the active list. All appropriate fields are updated including   */
/* the queue length fields and the number sent from this queue.   */
/*
 * NB: m = NULL signals empty queue
 * fq_index == NULL signals empty queue
 */
struct mbuf *sfq_dq_func(gifcp)
struct aNetIf *gifcp;
{
 struct ifqueue  *ifq;
  struct sfq  *_sfq;
  struct ifqueue2 *_q;
  struct mbuf *m;

  m = (struct mbuf *)NULL;
/* ifq = (struct ifqueue *)&(gifcp->osifcp->if_snd); CHECK !!!!! */
  _sfq = (struct sfq *)gifcp->iftc_state2p;
  _q = _sfq->fq_index;

  /*
  * Conditional will be true, if index field points
  * to a non-empty queue.
  */
  if (_q) {

  (m) = _q->ifq_head;
```

```
    if ((_q->ifq_head = (m)->m_act) == 0) {
    _q->ifq_tail = 0;
    }
    (m)->m_act = 0;

    _q->ifq_len--;
   _sfq->fq_len--;
/*    (ifq)->ifq_len--; */ /* this is the vc queue */
 /* statistics for now */
    _q->ifq_sent++; /* number sent on this q */
  DPRT (TR_DEQ, ("sfq_dq_func: [%d]\n", _q->ifq_label));

  /* Remove entry from active list if no more pkts */
  if (_q->ifq_head == 0) {
   if ((_q->ifq_forw == _q) && (_q->ifq_back== _q)) {
   _q->ifq_forw = NULL;
  /* Perturb the hash seed only when queue is empty */
    _sfq->fq_seed++;
 DPRT (TR_DEQ, ("sfq_dq_func: FQ %d now Empty\n",_q->ifq_label));
   } else {
   _q->ifq_back->ifq_forw = _q->ifq_forw;
   _q->ifq_forw->ifq_back = _q->ifq_back;
 DPRT (TR_DEQ, ("sfq_dq_func: removing %d from active list\n",
    _q->ifq_label));
   }
  }

  /* Index to the next non-empty queue */
  _sfq->fq_index = _q->ifq_forw;
  }
  return(m);
 }

#endif SFQ



/* Possible hash functions to choose from */

/*
 *
 */

unsigned long
hash1(seq, cp)    /* addxorhash.c */
 char seq;
 unsigned char *cp;
{
 unsigned long l1;
 unsigned long l2;
```

```
 l1 = *(unsigned long *)cp;
 l2 = *(((unsigned long *)cp) + 1);
 return ((l1 + seq) ^ l2);
}


/*
 *
 */
unsigned long
hash2(seq, cp)    /* rot11hash.c */
 char seq;
 unsigned char *cp;
{
 static unsigned long maskstay[] = {
 0xffffffff, 0xfffffffe, 0xfffffffc, 0xfffffff8,
 0xfffffff0, 0xffffffe0, 0xffffffc0, 0xffffff80,
 0xffffff00, 0xfffffe00, 0xfffffc00, 0xfffff800,
 0xfffff000, 0xffffe000, 0xffffc000, 0xffff8000,
 0xffff0000, 0xfffe0000, 0xfffc0000, 0xfff80000,
 0xfff00000, 0xffe00000, 0xffc00000, 0xff800000,
 0xff000000, 0xfe000000, 0xfc000000, 0xf8000000,
 0xf0000000, 0xe0000000, 0xc0000000, 0x80000000
 };
 static unsigned maskwrap[] = {
 0x00000000, 0x00000001, 0x00000003, 0x00000007,
 0x0000000f, 0x0000001f, 0x0000003f, 0x0000007f,
 0x000000ff, 0x000001ff, 0x000003ff, 0x000007ff,
 0x00000fff, 0x00001fff, 0x00003fff, 0x00007fff,
 0x0000ffff, 0x0001ffff, 0x0003ffff, 0x0007ffff,
 0x000fffff, 0x001fffff, 0x003fffff, 0x007fffff,
 0x00ffffff, 0x01ffffff, 0x03ffffff, 0x07ffffff,
 0x0fffffff, 0x1fffffff, 0x3fffffff, 0x7fffffff
 };
 unsigned long l1, l2;
 int coseq;

 l1 = *(unsigned long *)cp;
 l2 = *(((unsigned long *)cp) + 1);
 seq &= 0x1f;
 coseq = 32 - seq;
 l1 = ((l1 << seq) & maskstay[seq]) ^ ((l1 >> (coseq)) & maskwrap[seq]);

 l2 = ((l2 << coseq) & maskstay[coseq]) ^ ((l2 >> (seq)) & maskwrap[coseq]);

 return (l1 + l2);
}


/*
 *
 */
```

```
unsigned long
hash3(seq, cp)   /* rot1hash.c */
 char seq;
 unsigned char *cp;
{
 static unsigned long maskstay[] = {
 0xffffffff, 0xfffffffe, 0xfffffffc, 0xfffffff8,
 0xfffffff0, 0xffffffe0, 0xffffffc0, 0xffffff80,
 0xffffff00, 0xfffffe00, 0xfffffc00, 0xfffff800,
 0xfffff000, 0xffffe000, 0xffffc000, 0xffff8000,
 0xffff0000, 0xfffe0000, 0xfffc0000, 0xfff80000,
 0xfff00000, 0xffe00000, 0xffc00000, 0xff800000,
 0xff000000, 0xfe000000, 0xfc000000, 0xf8000000,
 0xf0000000, 0xe0000000, 0xc0000000, 0x80000000
 };
 static unsigned maskwrap[] = {
 0x00000000, 0x00000001, 0x00000002, 0x00000004,
 0x00000008, 0x0000001f, 0x0000002f, 0x0000004f,
 0x0000008f, 0x000001ff, 0x000002ff, 0x000004ff,
 0x000008ff, 0x00001fff, 0x00002fff, 0x00004fff,
 0x00008fff, 0x0001ffff, 0x0002ffff, 0x0004ffff,
 0x0008ffff, 0x001fffff, 0x002fffff, 0x004fffff,
 0x008fffff, 0x01ffffff, 0x02ffffff, 0x04ffffff,
 0x08ffffff, 0x1fffffff, 0x2fffffff, 0x4fffffff,
 0x8fffffff
 };
 unsigned long l1;
 unsigned long l2;
 int coseq;

 l1 = *(unsigned long *)cp;
 l2 = *(((unsigned long *)cp) + 1);
 seq &= 0x1f;
 coseq = 32 - seq;
 l1 = ((l1 << seq) & maskstay[seq]) ^ ((l1 >> (coseq)) & maskwrap[seq]);

 l2 = ((l2 << coseq) & maskstay[coseq]) ^ ((l2 >> (seq)) & maskwrap[coseq]);

 return (l1 + l2);
}


/*
 *
 */
unsigned long
hash4(seq, cp)   /* rothash.c */
 char seq;
 unsigned char *cp;
{
 static unsigned long maskstay[] = {
 0xffffffff, 0xfffffffe, 0xfffffffc, 0xfffffff8,
```

```
 0xfffffff0, 0xffffffe0, 0xffffffc0, 0xffffff80,
 0xffffff00, 0xfffffe00, 0xfffffc00, 0xfffff800,
 0xfffff000, 0xffffe000, 0xffffc000, 0xffff8000,
 0xffff0000, 0xfffe0000, 0xfffc0000, 0xfff80000,
 0xfff00000, 0xffe00000, 0xffc00000, 0xff800000,
 0xff000000, 0xfe000000, 0xfc000000, 0xf8000000,
 0xf0000000, 0xe0000000, 0xc0000000, 0x80000000
 };
 static unsigned maskwrap[] = {
 0x00000000, 0x00000001, 0x00000002, 0x00000004,
 0x00000008, 0x0000001f, 0x0000002f, 0x0000004f,
 0x0000008f, 0x000001ff, 0x000002ff, 0x000004ff,
 0x000008ff, 0x00001fff, 0x00002fff, 0x00004fff,
 0x00008fff, 0x0001ffff, 0x0002ffff, 0x0004ffff,
 0x0008ffff, 0x001fffff, 0x002fffff, 0x004fffff,
 0x008fffff, 0x01ffffff, 0x02ffffff, 0x04ffffff,
 0x08ffffff, 0x1fffffff, 0x2fffffff, 0x4fffffff,
 0x8fffffff
 };
 unsigned long l1, l2;
 int coseq;

 l1 = *(unsigned long *)cp;
 l2 = *(((unsigned long *)cp) + 1);
 seq &= 0x1f;
 coseq = 32 - seq;
 l1 = ((l1 << seq) & maskstay[seq]) ^ ((l1 >> (coseq)) & maskwrap[seq]);

 l2 = ((l2 << coseq) & maskstay[coseq]) ^ ((l2 >> (seq)) & maskwrap[coseq]);

 return (l1 ^ l2);
}


/*
 *
 */
unsigned long
hash5(seq, cp)   /* xoraddhash.c */
 char seq;
 unsigned char *cp;
{
 unsigned long l1;
 unsigned long l2;


 l1 = *(unsigned long *)cp;
 l2 = *(((unsigned long *)cp) + 1);
 return ((l1 * seq) + l2);
}
```

```
/* hybrid_int.c       */
/*
 * Copyright (c) 1993 SRI International. All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by SRI International, Menlo Park, CA. The name SRI International
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */



/* This file contains the routines that interface to the traffic */
/* control algorithms. These routines specify a hybrid algorithm of  */
/* SFQ for IP traffic,best-effort service, and virtual clock   */
/* as embedded in ST-II for resource guarantee. ST-II provides the  */
/* resource protocol setup needed for passing the flow specification  */
/* to the routers. Virtual Clock provides a clock-based scheme for */
/* ensuring proper ordering of arriving packets according to their  */
/* advertised rate. */

#ifdef SFQ_VC
#include <sys/types.h>
#include <sys/mbuf.h>
#include <sys/socket.h>
#include <net/if.h>
#include <sfq/sfq.h>

/* Compare unsigned long quantities */

#define LaterThan(A,B) ((((B)-(A)) & ((~0) - ((unsigned long) (~0) >> 1))) !=
0)


#ifndef FALSE
#define FALSE (0)
#endif

#ifndef TRUE
#define TRUE (1)
#endif

#ifndef NULL
```

```
# define NULL 0
#endif

extern int sfq_init_func();
extern int vc_init_func();
extern int sfq_nq_func();
extern void vc_nq_func();
extern struct mbuf *gen_dq_func();
extern struct mbuf *sfq_dq_func();
extern void gen_drain_func();
extern void sfq_drain_func();
extern int vc_enf_func();
extern unsigned long hash1();
extern unsigned long hash2();
extern unsigned long hash3();
extern unsigned long hash4();
extern unsigned long hash5();

long sfq_vc_debug = 0x0;

#define H_DPRT(c, x) if(sfq_vc_debug&c)printf x;

#ifndef TR_ENQ
#define TR_ENQ 1<<0 /* Enqueue flag */
#endif

#ifndef TR_DEQ
#define TR_DEQ 1<<1 /* Dequeue flag */
#endif

#ifndef TR_STA
#define TR_STA 1<<2 /* Errors */
#endif

/* sfq_vc_init initializes the sfq and vc queueing structures */

int sfq_vc_init(gifcp)
struct aNetIf *gifcp;
{
 int sfq_error, vc_error;

 sfq_error = 0;
 /* initialize SFQ structures */

 sfq_error = sfq_init_func(gifcp,hash1);

 /* initialize VC structures */
 vc_error = vc_init_func(gifcp);

 if (!sfq_error)
 return(vc_error);
 else
```

```
 return(sfq_error);

}

/* sfq_vc_nq enqueues a "packet" onto the appropriate queue.   */
/* If the packet belongs to an ST-II flow, indicated by the    */
/* mbuf type set to MT_TCDATA, the packet is put on the virtual  */
/* clock queue. If the packet is an IP packet, or an ST-II packet */
/* which does not have a flow setup, indicated by the mbuf type set */
/* to MT_DATA, it is handled by SFQ.      */

void sfq_vc_nq(gifcp, mp)
struct aNetIF *gifcp;
struct mbuf *mp;
{
 switch(mp->m_type)
 {
 case MT_DATA:
  H_DPRT(TR_ENQ, ("sfq_vc_nq: mt_data pkt\n"));
  if (sfq_nq_func(gifcp,mp) != TRUE)
   {
   /* qfull routine is not called with correct */
 /* queue structure */
   /* need to free packet, because q might be full */
   /* make sure surrounding calls are at correct */
   /* interrupt level */

   if (mp != NULL) /* not a raw packet */
   {
    H_DPRT(TR_ENQ, ("sfq_vc_nq: FREEING an sfq packet\n"));
    m_freem(mp);
    mp = NULL; /* NOOP for now, doing for cleanliness */
 }
   }
  break;
 case MT_TCDATA:
  H_DPRT(TR_ENQ, ("sfq_vc_nq: mt_tcdata pkt\n"));
  vc_nq_func(gifcp, mp);
  break;
 default:
  printf("sfq_vc_nq: Illegal Classifcation. FATAL ERROR\n");
  break;
 }
}

/* PEAK_VC_QUEUE is a macro which does a one packet lookahead into */
/* the virtual clock queue     */

#define PEAK_VC_QUEUE(timestamp, queue, nextpkt){ \
 \
 nextpkt = (queue)->ifq_head; \
 timestamp = (nextpkt)->m_tckey; \
```

```
}

/* VC_EMPTY is a macro which checks to see if the virtual clock */
/* queue is empty       */

#define VC_EMPTY(queue) ((queue)->ifq_head == (struct mbuf *) NULL)

/* sfq_vc_dq dequeues a packet from either the SFQ or Virtual Clock */
/* queue. See SRI report 8600 ...for details    */

struct mbuf *sfq_vc_dq(gifcp)
struct aNetIf  *gifcp;
{
struct mbuf  *mp;
unsigned long pkttimestamp;
struct timeval time_right_now;
unsigned long nowusec;
struct ifqueue *vc_q;



/* Preliminary version */
/* VC queue has priority */
/*
 if ((mp = gen_dq_func(gifcp)) == NULL)
 mp = sfq_dq_func(gifcp);
 return(mp);
*/


 /* interleaving with times */

/* If it is time to send a VC packet, dequeue a packet from this queue. */
/* If not time to send from VC, take a packet from SFQ. If no packet */
/* available, go back and try VC queue.     */

 vc_q = (struct ifqueue *) &(gifcp->osifcp->if_snd);
 if (!VC_EMPTY(vc_q)) /* something in virtual clock queue */
 {
 PEAK_VC_QUEUE(pkttimestamp, vc_q, mp);
 uniqtime(&time_right_now);
 nowusec = ( (unsigned long) time_right_now.tv_sec) * 1000000 +
 time_right_now.tv_usec;
H_DPRT(TR_DEQ, ("sfq_vc_dq: pkttime = %X, now = %X\n", pkttimestamp,
 nowusec));
 if ( LaterThan(pkttimestamp,nowusec))
 {
 if (SFQ_EMPTY(gifcp))
 {
  H_DPRT(TR_DEQ, ("trying vc q:sfq empty \n"));
  return(gen_dq_func(gifcp));
```

```
 }
  else
 {
  H_DPRT(TR_DEQ, ("dq sfq pkt\n"));
  return(sfq_dq_func(gifcp));
 }
 }
 else
 {
  H_DPRT(TR_DEQ, ("dq vc pkt-pkt time expired\n"));
  return(gen_dq_func(gifcp));
 }
 }
 else
 {
 H_DPRT(TR_DEQ, ("dq sfq pkt: no vc pkts\n"));
 return(sfq_dq_func(gifcp));
 }

}

/* sfq_vc_drain empties all packets from the Virtual Clock and SFQ queue. */


void sfq_vc_drain(gifcp, state1p, state2p, state1ul, state2ul, npktsp,
nbytesp)
struct aNetIf *gifcp;
caddr_t  state1p,
 state2p;
unsigned long state1ul,
 state2ul,
 *npktsp,
 *nbytesp;
{
 unsigned long npkts1,
  npkts2,
 nbytes1,
 nbytes2;


 npkts1 = npkts2 = nbytes1 = nbytes2 = (unsigned long ) 0;
 /* empty all queues associated with each algorithm */
 /* VC queue */
 /* leave order as is due to re-initializing ifq fields */
 gen_drain_func(gifcp, state1p, state2p, state1ul, state2ul, &npkts1,
 &nbytes1);
 /* SFQ queues */
 sfq_drain_func(gifcp, &npkts2, &nbytes2);


 if ( npktsp != (unsigned long *)NULL )
 *npktsp = npkts1 + npkts2;
```

```
 if ( nbytesp != (unsigned long *) NULL)
 *nbytesp = nbytes1 + nbytes2;

 return;
}


/* sfq_vc_classify should classify whether the packet belongs in the */
/* Virtual Clock queue or SFQ      */
/* problem here.....the result of this function gets overwritten */
/* with resource information from stII....something should be done */
/* differently */
/* for initial cut rely on the fact that if an enforcement function */
/* is provided, the mbuf type is MT_TCDATA for resource traffic and */
/* MT_DATA for best-effort so this function is not really used */



caddr_t sfq_vc_classify(gifcp, mp, pf, nethdrp, validlen)
struct aNetIf *gifcp;
struct mbuf  *mp;
int  pf;
caddr_t  nethdrp;
int  validlen;
{
/* IF packet is IP, use SFQ */
/* if packet is STII, use VC */
 switch(pf)
 {
 case PF_INET:
/*  return(PF_INET); */
  return(NULL);
  break;
 case PF_COIP:
/*  return(PF_COIP); */
  return(NULL);
 break;
 default:
  printf("sfq_vc_classify: unknown packet type\n");
  return(NULL);
/*  return(PF_UNSPEC); */
  break;
 }
}

/* sfq_vc_enforce provides the enforcement. Because IP traffic doesn't */
/* require any enforcement, since it represents best-effort service, */
/* only the Virtual Clock enforcement routine is called. */


int sfq_vc_enforce(gifcp, mp, totlen, timevalp)
struct aNetIf  *gifcp;
```

```
struct mbuf   *mp;
unsigned int   totlen;
struct timeval  *timevalp;
{
 return (vc_enf_func(gifcp, mp, totlen, timevalp));
}

/* sfq_vc_qfull checks to see if the queue associated with a given packet */
/* is full. Skeleton only provided for completeness. It is currently not */
/* used.         */


int sfq_vc_qfull(gifcp, mp)
struct aNetIf  *gifcp;
struct mbuf  *mp;
{

}

#endif SFQ_VC
```

```
/* This is diffs for if.h. */
12,13d11
< #include <sys/time.h>
<
112,118d109
< #ifdef TRAFFIC_CONTROL
< /* If we had source, the following struct would be part of struct ifnet
<  * and many things would be easier ...
<  *
<  * NB: without source, we cannot get to things like if_qflush() ...
<  */
<
120,699d110
< D* aNetIf  Yet another network interface abstraction ...
< d*
< d* We want fields:
< d*
< d* AdrOfIfc (ifc) Maps an OS net handle to something to access its info.
< d* ifc_bw_alloc Currently allocated bandwidth (Bytes/sec) for network.
< |*OBS ifc_bw_byte Bandwidth (Bytes) required to send a byte (8 bits).
< d* ifc_bw_cnfg Bandwidth (Bytes/sec) provided by the network.
< d* ifc_bw_load Target load (Bytes/sec) for network from this interface.
< d* ifc_bw_resv Amount of resources (Bytes/sec) that may be reserved.
< d* ifc_bw_pkt Overhead bandwidth (Bytes/pkt) required to send a packet.
< |*OBS ifc_clientsp Ptr to access info about clients reserving resources.
< d* ifc_cost Cost to reserve network resources.
< d* ifc_cost_byte Cost to send a byte.
< d* ifc_cost_msec Cost to use link for a millisecond
< d* ifc_cost_pkt Cost to send a packet.
< d* ifc_cp  [Private: Ptr to Origin/Target's aST2pcb].
< d* ifc_gwcp [Private: Ptr to Target's "Gateway" aST2pcb].
< d* ifc_lclhdrlen Bytes to leave for network layer headers.
< d* ifc_mtu  Network MTU.
< d* ifc_name Name of network interface.
< d* ifc_output Routine to send a packet.
<    from ReplyMsgSend st2_CMPOutput st2_Forward
< d* ifc_selfp [Private: Ptr to memory object holding interface info].
< d* ifc_snd  Output queue
< d* ifc_unit Device unit #
< d*
< d* iftc_alg Current Traffic Control Algorithm
< d* iftc_alg_next Next Traffic Control Algorithm
< d* iftc_classify ( IF_Ext(ifp), mp, PF_xxx, nethdrp, validlen ) -> flowp
< d* iftc_clockfast ( IF_Ext(ifp) )
< d* iftc_control ( IF_Ext(ifp), op, xxxp, xxxl ) -> error
< d* iftc_dq  ( IF_Ext(ifp) ) -> mp
< d* iftc_drain ( IF_Ext(ifp), state1p, state2p, state1ul, state2ul,
< d*     npktsp, nbytesp )
< d* iftc_enforce ( IF_Ext(ifp), mp, totlen, timevalp ) -> result
< d*    -2: no traffic control; -1: drop it; 0: enqueue it;
< d*    N: enqueue it but some other N byte packet was dropped
< d* iftc_init ( IF_Ext(ifp) )
```

```
< d* iftc_nq  ( IF_Ext(ifp), mp ) -> error
< d* iftc_per_alg [] struct
< d* iftc_quit ( IF_Ext(ifp) )
< d* ifrm_rsrcalloc ( IF_Ext(ifp), ridp, arsclpp, ratep, bytesp, fdp, destpp )
< d* ifrm_rsrcgetid ( IF_Ext(ifp), rsrcidp )
< d* ifrm_rsrcprobe ( IF_Ext(ifp) ??? )
< d* ifrm_rsrcrelid ( IF_Ext(ifp), rsrcidp )
< d* ifrm_rsrcrelse ( IF_Ext(ifp), ridp, arsclpp, ratep, bytesp, fdp, destpp )
< d* iftc_state1p pointers for use by algorithm, except iftc_drain
< d* iftc_state2p
< d* iftc_state1ul longs for use by algorithm, except iftc_drain
< d* iftc_state2ul
< d*
<  */
<
< struct aNetIf {
<     char   namebuf[16]; /* Interface name */
<     struct aNetIf *ifc_nextp; /* Ptr to next generic interface */
<     int    lclhdrlen; /* Local network header length */
<     unsigned long  bw_conf, /* Bytes per second on the wire */
<    bw_load, /* Target loading, bytes/sec */
<    bw_resv; /* Reservable bytes per second */
<
<     struct aResourceManagement {
<     /* allocate resources */
<  int      (*rsrcalloc) (/* IF_Ext(ifp), ridp, arsclpp, ratep,
<           bytesp, fdp, destpp */),
<        (*rsrccntrl) (/* IF_Ext(ifp), op, xxxp, xxxl */),
<     /* get a resource id */
<        (*rsrcgetid) (/* IF_Ext(ifp), rsrcidp */),
<     /* probe available resources */
<        (*rsrcprobe) (/* IF_Ext(ifp) ??? */),
<     /* release a resource id */
<        (*rsrcrelid) (/* IF_Ext(ifp), rsrcidp */),
<     /* release resources */
<        (*rsrcrelse) (/* IF_Ext(ifp), ridp, arsclpp, ratep,
<          bytesp, fdp, destpp */);
<     }    rmf;
<
<     struct aTrafficControl {
<  caddr_t      (*classify)  (/* IF_Ext(ifp), mp, PF_xxx,
<           nethdrp, validlen */);
<  void       (*clockfast) (/* IF_Ext(ifp) */);
<  int       (*control)   (/* IF_Ext(ifp), op, xxxp, xxxl */);
<  struct mbuf *(*dq)    (/* IF_Ext(ifp) */);
<  void       (*drain)   (/* IF_Ext(ifp), state1p, state2p, state1ul,
<           state2ul, npktsp, nbytesp */);
<  int       (*enforce)   (/* IF_Ext(ifp), mp, totlen, timevalp */);
<     /* -1: drop it; 0: enqueue it; */
<     /* N: enqueue it but some other N byte */
<     /* packet was dropped */
<  int       (*init)    (/* IF_Ext(ifp) */);
```

```
<  void      (*nq)   (/* IF_Ext(ifp), mp */);
<  void      (*quit)   (/* IF_Ext(ifp) */);
<  struct aResourceManagement rmf; /* per-algorithm hooks */
<  struct aTcRmState {
<       caddr_t  s1p,  /* Traffic Control private */
<      s2p;  /* Traffic Control private */
<       unsigned long s1ul,  /* Traffic Control private */
<      s2ul;  /* Traffic Control private */
<  }   state;
<       }    tcf;
<
<       struct aRsrcNet *rsrcifp; /* Ptr to resource management info */
<
<       short   alg,  /* Current Traffic Control algorithm */
<      alg_next; /* Next Traffic Control algorithm */
<       caddr_t   per_alg; /* private */
<
<       struct ifnet *osifcp; /* Ptr to OS structure */
< #ifdef STII
<       struct aST2pcb *cp,  /* Ptr to api pcb (IFF_PRIVATE) */
<     *gwcp;  /* Ptr to upstream-"gateway" pcb */
<       /* (IFF_PRIVATE) */
<       struct aPktDesc *selfp;  /* Ptr to buffer holding interface */
< #endif STII
<
<   /* Pseudo interface may have OS block here, if required */
< };
<
< /* Generic Network Interface (aNetIf) Mappings (ifnet, aRsrsNet) */
<
< #define AdrOfIfc(ifc) ((ifc)->osifcp)
< #define ifc_bw_alloc rsrcifp->bw_alloc
< #define ifc_bw_cnfg rsrcifp->bw_cnfg
< #define ifc_bw_load rsrcifp->bw_load
< #define ifc_bw_pkt rsrcifp->bw_pkt
< #define ifc_bw_resv rsrcifp->bw_resv
< /*#define ifc_clientsp rsrcifp->clientsp*/
< #define ifc_cost rsrcifp->cost
< #define ifc_cost_byte rsrcifp->cost_byte
< #define ifc_cost_msec rsrcifp->cost_msec
< #define ifc_cost_pkt rsrcifp->cost_pkt
< #define ifc_cp  cp
< #define ifc_gwcp gwcp
< #define ifc_lclhdrlen lclhdrlen
< #define ifc_mtu  osifcp->if_mtu
< #define ifc_name namebuf
< #define ifc_output osifcp->if_output
< #define ifc_selfp selfp
< #define ifc_snd  osifcp->if_snd
< #define ifc_unit osifcp->if_unit
<
< #define ifrm_rsrcalloc rmf.rsrcalloc
```

```
< #define ifrm_rsrccntrl rmf.rsrccntrl
< #define ifrm_rsrcgetid rmf.rsrcgetid
< #define ifrm_rsrcprobe rmf.rsrcprobe
< #define ifrm_rsrcrelid rmf.rsrcrelid
< #define ifrm_rsrcrelse rmf.rsrcrelse
<
< #define iftc_alg alg
< #define iftc_alg_next alg_next
< #define iftc_classify tcf.classify
< #define iftc_clockfast tcf.clockfast
< #define iftc_control tcf.control
< #define iftc_dq  tcf.dq
< #define iftc_drain tcf.drain
< #define iftc_enforce tcf.enforce
< #define iftc_init tcf.init
< #define iftc_nq  tcf.nq
< #define iftc_quit tcf.quit
< #define iftc_rsrcalloc tcf.rmf.rsrcalloc
< #define iftc_rsrccntrl tcf.rmf.rsrccntrl
< #define iftc_rsrcgetid tcf.rmf.rsrcgetid
< #define iftc_rsrcprobe tcf.rmf.rsrcprobe
< #define iftc_rsrcrelid tcf.rmf.rsrcrelid
< #define iftc_rsrcrelse tcf.rmf.rsrcrelse
< #define iftc_state1p tcf.state.s1p
< #define iftc_state2p tcf.state.s2p
< #define iftc_state1ul tcf.state.s1ul
< #define iftc_state2ul tcf.state.s2ul
<
< #define ifo_per_alg per_alg
<
< #define NO_NETIFP ((struct aNetIf *) NULL)
< #define IF_Ext(ifp) (tcif_ifp2gifcp( (caddr_t) ifp ))
< extern struct aNetIf *tcif_ifp2gifcp (/* osifcp */);
<
<
< /* Structure for managing buffer resources. */
<
< struct aRsrcBuf {   /* # of buffers ... */
<     unsigned long  cnfg,  /* ... configured as available */
<     alloc;  /* ... currently allocated */
< };
<
<
< /*
< D* aRsrcId  Structure holding a Resource Identifier.
<  *
<  * (HAP wants 64 bits)
<  */
<
< struct aRsrcId {
<     unsigned long  a,
<     b;
```

```
< };
<
< #define NO_RSRCIDP ((struct aRsrcId *) NULL)
<
<
< /* Structure to hold information about resource requests for a requestor of
<  * services from a network.
<  */
<
< struct aRsrcClnt {
< #define RSRCCLNTID 0xc8b9c551  /* A unique # */
<     int    rsrcclntid; /* MUST BE FIRST w/value RSRCCLNTID */
<     struct aRsrcClnt *nextp;  /* Ptr to next client on this interface */
< #ifdef NOTYET
<     struct aRsrcClnt *parentp,
<        *childp;
<    /* ??? pending/multiple requests for single client */
<     int    usecnt; /* NOTYET */
< #endif NOTYET
<
<     struct aRsrcId  handle; /* Resource id, given to client */
<
<     struct aFlowDesc {   /* Args for xx_aloc_func */
<  unsigned long  src,
<     src_mask,
<     dst,
<     dst_mask,
<     type,
<     type_mask,
< #define FL_TYPE_PRIVATE  0x00000001 /* Externally managed flow */
< #define FL_TYPE_IP  0x00000002 /*  IP flow */
< #define FL_TYPE_IPPROT_SHIFT 8 /* 0000FF00 Proto field */
< #define FL_TYPE_ST  0x00000004 /*  ST-II flow (=>FL_TYPE_RT) */
< #define FL_TYPE_STHID_SHIFT 16 /* FFFF0000 ST-II HID field */
< #define FL_TYPE_STHID_MASK 0xffff0000
< #define FL_TYPE_RT  0x00000008 /*  Real-Time flow */
<     ports,
<     ports_mask;
<     }    flowdesc;
<
<     unsigned long  buf,  /* Resources allocated to client */
<     bw_clnt,
<     bw_local,
<     cpu_clnt,
<     cpu_local;
<
<  /* Outstanding request (?s) for client */
< /*  current request
<     callback function
<     callback argument
<  struct NextHop *
<     pending list
```

```
<      lower layer, handle
<      lower layer, ...
<   */
<      int    enf_id; /* id of enforcement algorithm used */
<      unsigned long  alg_vector[1]; /* Algorithms using this */
<
< #ifdef VIRTUAL_CLOCK
<  struct vc_info {
<       unsigned long
<     drop,  /* # times said to drop pkt */
<   /*  lar,  */
<     last_check_usec,
<     priority, /* microseconds */
<   /*  threshold, /* microseconds */
<     vc,  /* microseconds */
<     auxvc,  /* microseconds */
<     ai,
<   /*  air,  */
<   /*  ar,  */
<   /*  bw_byte, /* fractional seconds per byte */
<     bw_pkt, /* overhead bytes per packet */
<     bytes_in_ai, /* bytes (/sec) used this interval */
<     bytes_per_ai, /* bytes (/sec) allocated */
< #define VTICKSCALE 10  /* Extra precision bits 10^6 << 10 < 2^32 */
<     vtick;  /* realtime expansion factor */
<       struct timeval
<     last_check_tv;
<
<  }   rc_vc;
<
< #define vc_ai   rc_vc.ai
< #define vc_auxvc  rc_vc.auxvc
< #define vc_bw_pkt  rc_vc.bw_pkt
< #define vc_drop   rc_vc.drop
< #define vc_bytes_in_ai  rc_vc.bytes_in_ai
< #define vc_bytes_per_ai  rc_vc.bytes_per_ai
< #define vc_last_check_usec rc_vc.last_check_usec
< #define vc_priority  rc_vc.priority
< #define vc_vc   rc_vc.vc
< #define vc_vtick  rc_vc.vtick
< #define vc_last_check_tv rc_vc.last_check_tv
< #endif VIRTUAL_CLOCK
<
< #ifdef FAIR_SHARE
<  struct flow *flowp;
< #define fs_flowp  flowp
< #endif FAIR_SHARE
<
<  /* add others here ... */
<
<
< };
```

```
<
< #define NO_RSRCCLNTP ((struct aRsrcClnt *) NULL)
<
<
< /* Structure for managing CPU resource. */
<
< struct aRsrcCPU {   /* Fractional seconds ... */
<     unsigned long  cnfg,  /* ... configured as available */
<     alloc;  /* ... currently allocated */
< };
<
<
< /* Logical extension to OS native network interface structure
<  * to support resource management by the network.
<  */
<
< struct aRsrcNet {
< #define RNNameUnit parentp  /* Init: "name|unit" */
<    struct aRsrcNet *parentp, /* NOTYET */
<    *nextp,  /* ptr to next extension block */
<    *childp; /* NOTYET */
<     struct aNetIf *gifcp;  /* ptr to generic interface this extends */
<     struct aRsrcClnt *clientsp; /* ptr to clients of network */
<
<     unsigned long  flags;  /* Services provided by network */
<
< # define NetCanMcast 0x01
< # define NetMayMcast 0x02
< # define NetCanBcast 0x04
< # define NetMayBcast 0x08
< # define NetIsPt2Pt 0x10
<
<     struct aRsrcBuf *bufp;  /* Ptr to buffer resources */
<     struct aRsrcCPU *cpup;  /* Ptr to CPU resources */
<     unsigned long  ber,  /* Bit Error Rate, negative of
<        exponent of ten ( > 0 ) */
<
<     /* On-the-wire Bytes per second ... */
<    bw_cnfg, /* ... available, i.e., bandwidth */
<    bw_load, /* ... target load */
<    bw_resv, /* ... reservable bandwidth */
<  /* OBS bw_byte normalized to be 1 */
<   /*  bw_byte, /* ... required to send a byte */
<    bw_pkt, /* ... required to send a packet, incl.
<            gap, leader, header, trailer */
<    bw_alloc, /* ... allocated bandwidth */
<
<   /* ??? Need units - milli/micro/nano cents/dollars/pounds/DM */
<    cost,  /* Fixed cost per client */
<    cost_byte, /* Per byte cost */
<    cost_msec, /* per millisecond cost */
<    cost_pkt, /* Per packet cost */
```

```
<
<      /* Fractional seconds of CPU ... */
<     cpu_in_byte, /* ... to receive a byte */
<     cpu_in_pkt, /* ... to receive a packet */
<
<     cpu_out_byte, /* ... to send a byte */
<     cpu_out_pkt, /* ... to send a packet */
<
<     droprate, /* Drop rate, fraction of packets that
<         will be dropped */
<      /* ??? need per pkt & per byte? */
<
<      /* Fractional seconds ... */
<     dly_in, /* ... to receive a packet */
<     dly_in_var,
<
<     dly_prop, /* ... of "fixed" propagation delay,
<             (else see aNeighbor delay) */
<      /*     ???need >= 1 second */
<
<    /* ??? array by class, etc., ??? need >= 1 second */
<     dly_que, /* ... of queueing delay, estimate */
<     dly_que_var,
<      /* ??? need per pkt & per byte? */
<     dly_out, /* ... to transmit a packet */
<     dly_out_var,
<
<     lclhdrlen; /* Max length of network-layer header,
<         so clients can leave room for it */
< };
<
< #define NO_RSRCNET ((struct aRsrcNet *) NULL)
<
<
< #ifndef Ident
< /* Build a symbol from two components */
< #define Ident(x)x
< #endif  Ident
<
<
< /* def/name/aloc getid probe relid rlse */
<
< #define aRmVectorList      \
<  aRV (P2P,point-to-point,tcif_PtpAlloc,NoRmCtrl,tcif_RsIdGet,  \
<   NoRmProbe,tcif_RsIdRel,tcif_PtpRlse)   \
<  aRV (BCST,broadcast,tcif_PtpAlloc,NoRmCtrl,tcif_RsIdGet,  \
<   NoRmProbe,tcif_RsIdRel,tcif_PtpRlse)   \
<  aRV (NUN,,NoRmAlloc,NoRmCtrl,NoRmIdGet,NoRmProbe,NoRmIdRel,NoRmRlse) \
<
<
< /* def/name/classify clockfast control dq drain enforce init nq quit
<             aloc ctrl getid probe relid rlse */
```

```
<
< #ifdef FAIR_SHARE
< #define MAYBE_FAIR_SHARE      \
<  aTV (FS1,fair-share,fs_classify_func,fs_clockfast_func,fs_control_func,\
<   gen_dq_func,NoDrain,fs_enforce_func,fs_init_func, \
<   fs_nq1q_func,NoQuit,     \
<   fs_aloc_func,NoRsrcCtrl,NoRsrcGetid,NoRsrcProbe, \
<   NoRsrcRelid,fs_rlse_func)
< #else
< #define MAYBE_FAIR_SHARE
< #endif FAIR_SHARE
<
<
< #ifdef MY_FIFO
< #define MAYBE_MY_FIFO       \
<  aTV (MY_FIFO,my_fifo,gen_classify_func,NoClockfast,NoControl,gen_dq_func,\
<                 gen_drain_func, gen_enforce_func,NoInit,gen_nq_func,      \
<                 NoQuit,NoRsrcAlloc,NoRsrcCtrl,NoRsrcGetid,NoRsrcProbe,    \
<   NoRsrcRelid,NoRsrcRlse)
< #else
< #define MAYBE_MY_FIFO
< #endif MY_FIFO
<
< /*
< #ifdef SFQ
< #define MAYBE_SFQ      \
<  aTV (SFQ,sfq,NoClassify,NoClockfast,NoControl,NoDq,NoDrain,  \
<   NoEnforce,NoInit,NoNq,NoQuit,    \
<   NoRsrcAlloc,NoRsrcCtrl,NoRsrcGetid,NoRsrcProbe,  \
<   NoRsrcRelid,NoRsrcRlse)
< #else
< #define MAYBE_SFQ
< #endif SFQ
< */
<
< #ifdef SFQ_VC
< #define MAYBE_SFQ_VC                \
<  aTV (SFQ_VC,sfq_vc,sfq_vc_classify,NoClockfast,NoControl,sfq_vc_dq,     \
<                 sfq_vc_drain,sfq_vc_enforce,sfq_vc_init,sfq_vc_nq,       \
<                 NoQuit, vc_aloc_func,NoRsrcCtrl,NoRsrcGetid,             \
<                 NoRsrcProbe,NoRsrcRelid,NoRsrcRlse)
< #else
< #define MAYBE_SFQ_VC
< #endif SFQ_VC
<
<
< #ifdef VIRTUAL_CLOCK
< #define MAYBE_VIRTUAL_CLOCK     \
<  aTV (VC,vc,NoClassify,NoClockfast,NoControl,gen_dq_func,NoDrain, \
<   vc_enf_func,vc_init_func,vc_nq_func,NoQuit,  \
<   vc_aloc_func,NoRsrcCtrl,NoRsrcGetid,NoRsrcProbe, \
<   NoRsrcRelid,NoRsrcRlse)
```

```
< #else
< #define MAYBE_VIRTUAL_CLOCK
< #endif VIRTUAL_CLOCK
<
<     /* %^*% cpp! no #ifdefs in a #define */
< #define aTcVectorList       \
<  aTV (FIFO,fifo,NoClassify,NoClockfast,NoControl,gen_dq_func,NoDrain, \
<   gen_enforce_func,NoInit,gen_nq_func,NoQuit,  \
<   NoRsrcAlloc,NoRsrcCtrl,NoRsrcGetid,NoRsrcProbe,  \
<   NoRsrcRelid,NoRsrcRlse)     \
<  MAYBE_FAIR_SHARE       \
<  aTV (RD,random-drop,NoClassify,NoClockfast,NoControl,gen_dq_func, \
<   NoDrain,gen_enforce_func,NoInit,gen_nq_func,NoQuit, \
<   NoRsrcAlloc,NoRsrcCtrl,NoRsrcGetid,NoRsrcProbe,  \
<   NoRsrcRelid,NoRsrcRlse)     \
< /* MAYBE_SFQ  */      \
<  MAYBE_SFQ_VC                        \
<  MAYBE_MY_FIFO                                                          \
<  MAYBE_VIRTUAL_CLOCK       \
<  /* This must be last -- it has the largest enum value */  \
<  aTV (NUN,,gen_classify_func,gen_clockfast_func,gen_control_func, \
<   gen_dq_func,gen_drain_func,gen_enforce_func,  \
<   gen_init_func,gen_nq_func,gen_quit_func,  \
<   NoRsrcAlloc,NoRsrcCtrl,NoRsrcGetid,NoRsrcProbe,  \
<   NoRsrcRelid,NoRsrcRlse)
<
<
< enum RM_Strategy {
< #define aRV(id,name,aloc,ctrl,idget,probe,idrel,rlse) Ident(RM_)id,
<   aRmVectorList
< #undef aRV
< };
<
<
< enum TC_Algorithm {
<  /* @#$% cpp is too primative to allow \ in the formal parameter list */
< #define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
<  Ident(TC_)id,
<   aTcVectorList
< #undef aTV
< };
<
<
< #ifdef lint
<
< #define NoRmAlloc LintRmAlloc
< #define NoRmCtrl LintRmCtrl
< #define NoRmIdGet LintRmIdGet
< #define NoRmProbe LintRmProbe
< #define NoRmIdRel LintRmIdRel
```

```
< #define NoRmRlse LintRmRlse
<
< #define NoClassify LintClassify
< #define NoClockfast LintClockfast
< #define NoControl LintControl
< #define NoDq  LintDq
< #define NoDrain  LintDrain
< #define NoEnforce LintEnforce
< #define NoInit  LintInit
< #define NoNq  LintNq
< #define NoQuit  LintQuit
< #define NoRsrcAlloc LintRsrcAlloc
< #define NoRsrcCtrl LintCtrl
< #define NoRsrcGetid LintRsrcGetid
< #define NoRsrcProbe LintRsrcProbe
< #define NoRsrcRelid LintRsrcRelid
< #define NoRsrcRlse LintRsrcRlse
<
< #else !lint
<
< #define NoRmAlloc ((int (*)()) 0)
< #define NoRmCtrl ((int (*)()) 0)
< #define NoRmIdGet ((int (*)()) 0)
< #define NoRmProbe ((int (*)()) 0)
< #define NoRmIdRel ((int (*)()) 0)
< #define NoRmRlse ((int (*)()) 0)
<
< #define NoClassify ((caddr_t (*)()) 0)
< #define NoClockfast ((void (*)()) 0)
< #define NoControl ((int (*)()) 0)
< #define NoDq  ((struct mbuf *(*)()) 0)
< #define NoDrain  ((void (*)()) 0)
< #define NoEnforce ((int (*)()) 0)
< #define NoInit  ((int (*)()) 0)
< #define NoNq  ((void (*)()) 0)
< #define NoQuit  ((void (*)()) 0)
< #define NoRsrcAlloc ((int (*)()) 0)
< #define NoRsrcCtrl ((int (*)()) 0)
< #define NoRsrcGetid ((int (*)()) 0)
< #define NoRsrcProbe ((int (*)()) 0)
< #define NoRsrcRelid ((int (*)()) 0)
< #define NoRsrcRlse ((int (*)()) 0)
<
< #endif lint
<
< #if 0
< #define Omit(x)
< #define NoFunc(x)x
<
< #define aRV(id,name,aloc,ctrl,idget,probe,idrel,rlse)  \
< extern int aloc (), ctrl (), idget (), \
<   probe (), idrel (), rlse ();
```

```
<   aRmVectorList
< #undef aRV
<
<
< #define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
< extern caddr_t clsfy (); extern void fsclk (); \
< extern int cntrl (); extern struct mbuf *dq (); \
< extern void drain (); extern int enfrc (); \
< extern int init (); extern void nq ();  \
< extern void quit ();     \
< extern int aloc (), ctrl (), idget (), \
<   probe (), idrel (), rlse ();
<   aTcVectorList
< #undef aTV
<
< #undef Omit
< #define Omit(x)x
< #undef NoFunc
< #define NoFunc(x)
<
< #endif 0
<
< /*
706a118,121
> #define IFO_ENQUEUE(ifq, m)  \
>   (st2_ifonq  [st2_ifoid]) (ifq,m,st2_ifoid,0)
> #define IFO_DEQUEUE(ifq, m)  \
>   (st2_ifodq  [st2_ifoid]) (ifq,&(m),st2_ifoid,0)
708,711d122
< #define IFTC_ENQUEUE(ifqp, mp)  tcif_nq_func( (ifqp), (mp) )
< #define IFTC_DEQUEUE(ifqp, mp)  tcif_dq_func( (ifqp), &(mp) )
< #endif TRAFFIC_CONTROL
<
832,841d242
<   int ifru_tc_alg;
<
<   /* Struct for resource management info */
<   struct ifr_rsrcnet_parms {
<    u_long parm1;
<    u_long parm2;
<    u_long parm3;
<    u_long parm4;
<   } ifru_rsrcnet_req;
<
858,862d258
< #define ifr_tc_alg ifr_ifru.ifru_tc_alg /* traffic control algorithm */
<
< /* resource management specific */
< #define ifr_rsrcnet_req ifr_ifru.ifru_rsrcnet_req
<
```

```
881a278,293
>
> /* Driver Output Queue Management Strategies: */
>
> #define IFO_GEN 0 /* Generic: fifo using if_snd queue */
>
> #define IFO_VC 1 /* Virtual Clock: uses single if_snd queue, but    *
>     * m_type MT_DATASORTED mbufs are inserted, before *
>     * other MT_xxxs, by m_key (units of microsecs)    */
> /* ... others ... */
> #define NIFOIDS 8 /* allocated size of st2_ifonq/st2_ifodq arrays    *
>     * ought to be per interface, but no room in ifnet */
>
> extern int      st2_ifoid; /* Strategy being used */
> extern void   (*(st2_ifonq  [])) (/*ifqp,mp,id,aux*/);
> extern struct mbuf *(*(st2_ifodq  [])) (/*ifqp,mpp,id,aux*/);
>
```

```
/*
#ifndef lint
static char rcsid_if_aux_c[] = "\
@(#) $Header: if_aux.c,v 1.98 1.98+ 93/04/08 18:00:00 clynn Exp $ \n"; */
/* @(#)*/ /*-----------------------------------------------------------------*
@(#) | Copyright (c) 1991-1993 by BBN Systems and Technologies,          |
@(#) | A Division of Bolt Beranek and Newman Inc.                        |
@(#) |                                                                   |
@(#) | Permission to use, copy, modify, distribute, and sell this        |
@(#) | software and its documentation for any purpose is hereby          |
@(#) | granted without fee, provided that the above copyright notice     |
@(#) | and this permission appear in all copies and in supporting        |
@(#) | documentation, and that the name of Bolt Beranek and Newman       |
@(#) | Inc. not be used in advertising or publicity pertaining to        |
@(#) | distribution of the software without specific, written prior      |
@(#) | permission. BBN makes no representations about the suitability    |
@(#) | of this software for any purposes. It is provided ''AS IS''       |
@(#) | without express or implied warranties.                            |
@(#) *-----------------------------------------------------------------*/
/* #endif lint */


/*
M* if_aux.c Generic Traffic Control and Resource Management Routines.
M*
 */

/*
m* Status:
m* Features:
m* Untested Features:
m* Restrictions/Bugs:
m* Things to do:
m*
 */


/* Module Revision History
 *
 * $Log: if_aux.c,v $
 */



#include <sys/errno.h>  /* E* */
#include <sys/types.h>  /* for <sys/mbuf.h> */
#include <sys/mbuf.h>  /* needs <sys/types.h> & <sys/param.h> */
#include <sys/socket.h>  /* for <net/if.h> */
#include <sys/socketvar.h> /* for so_proto */

#include <sys/ioccom.h>  /* _IOC*, SIOCxxx */
#include <sys/protosw.h> /* PRU_CONTROL */
```

```
#include <sys/sockio.h>  /* SOIC* */

#include <sys/user.h>  /* u.* uid = u.u_uid, pid = u.u_procp->p_pid */

#include <netinet/in.h>  /* before st2.h -> st2_api.h; IPPROTO_* ,
    in_addr for netinet/if_ether.h*/
#include <net/if.h>  /* ifnet, ifqueue, aNetIf, aRV, aTV
    for netinet/if_ether.h */

/* Optional (ha, ha) Ethernet Support */

/* If you get an error here, the lines:
 * sunif/if_ie.c  optional ie INET device-driver
 * sunif/if_le.c  optional le INET device-driver
 * are missing from the file sys/sun<N>/conf/files.
 */

#include "ie.h"
#if defined (NIE)
#if NIE
#ifndef WANT_ETHERNET
#define WANT_ETHERNET 1
#endif WANT_ETHERNET
#endif NIE
#endif defined (NIE)

#include "le.h"
#if defined (NLE)
#if NLE
#ifndef WANT_ETHERNET
#define WANT_ETHERNET 1
#endif WANT_ETHERNET
#endif NLE
#endif defined (NLE)

#ifndef WANT_ETHERNET
#define WANT_ETHERNET 0
#endif WANT_ETHERNET


#if WANT_ETHERNET
#include <net/if_arp.h>  /* ether_family, for netinet/if_ether.h */
#include <netinet/if_ether.h> /* ether_addr, ether_header, wants net/if_arp.h,
    net/if.h, netinet/in.h */
#include <sys/errno.h>
#include <netinet/in.h>  /* for netinet/if_ether.h */
#include <net/if_arp.h>  /* ether_family, for netinet/if_ether.h */
#include <netinet/if_ether.h>

extern struct ether_addr  etherbroadcastaddr;
extern struct ether_family *ether_families;
extern int    ifqmaxlen;
```

```
extern int    tcif_ether_output ();
extern struct timeval   time;
#endif WANT_ETHERNET


/* Optional HSI/S Support */

/* If you get an error here, the line:
 * hsisdev/hsis.o  optional hsis device-driver
 * is missing from the file sys/sun<N>/conf/files.
 */

#include "hsis.h"

#if NHSIS
#include <net/ppp.h>
#endif NHSIS


#ifdef STII
#include <netinet/st2_api.h>
#include <netinet/st2.h>
#else !STII

/* Macros to call dbgstp when errors are detected
 *
S* BUGSTOP( m, errinfo, bug_id )
 *
S* BUGRETURN( m, errinfo, bug_id, return_type )
 *
S* BUGGOTO( m, errinfo, bug_id, label )
 */

#define BUGRETURN( m, code, where, type ) return ( (type) (code) );
#define BUGGOTO( m, code, where, label ) goto label;
#define BUGSTOP( m, code, where )  {;}

#endif STII


#ifndef AND
#define AND &&
#define OR ||
#define NOT !
#define EQ ==
#define NE !=
#endif AND


#ifndef Bcopy
#define Bcopy(srcp,dstp,len) \
 (void) bcopy ( (char *) (srcp), (char *) (dstp), (int) (len) )
```

```
#endif Bcopy


#ifndef Cat2
/* Build a symbol from three components */
#define Cat2(y,z)Ident(y)z
#endif Cat2


#ifndef DimensionOf
/* Find the dimension of an array */
#define DimensionOf(array) (sizeof (array) / sizeof (array[0]))
#endif DimensionOf


#ifndef Ident
/* Build a symbol from two components */
#define Ident(x)x
#endif Ident

#ifndef ExpAry

/*
D* Expanable Arrays
D*
D* An Expanable Array of "struct aXxx"s is composed of a list of
D* aXxxList structures, each holding a sub-array of struct aXxx.
D* The initial sub-array can be sized for the expected case with
D* the ability to handle overflow when necessary. Each sub-array
D* specifies the number of array elements allocated (allocated),
D* maximum used (maxused), and a pointer to the next sub-array
D* (nxtXxxp). A "selfp" pointer is provided so that the header
D* of a dynamically allocated block may be located to satisfy
D* the memory management routines.
D*
D* ExpAry (a,t,n)
d* "a" is the structure name prefix, generally "a".
d* "t" is the structure name base, Xxx.
d* "n" is the number of array elements allocated when the ExpAry is
d* instantiated.
d*
 */

#define ExpAry(a,t,n)       \
struct Cat2 (a,t)List {      \
 struct Cat2 (a,t)List *Cat2 (nxt,t)p;/* Ptr to next part of array */ \
 struct aPktDesc *selfp;  /* Ptr to aPktDesc of this struct */ \
 unsigned short  allocated, /* Number of possible entries here */ \
    nxtfree; /* Number of smallest free entry, */ \
     /* 0..allocated-1 */  \
 int    objId;  /* Type of array this is (ffs) */ \
 struct Ident (a)t  Ident (t)s[n]; /* Initial entries */  \
```

```
}

/*
D* InitaExpAry (alloc,objid)
d* Initializer for an Expanable Array.
d* "alloc" is the number of instantiated array elements.
d* "objid" is the array's object identifier.
d*
 */
#define InitaExpAry(alloc,objid) 0,0,alloc,0,objid

#endif ExpAry


#ifdef lint  /* stop "possible pointer alignment problem" */
#define Mkp(t,p,n) ((t)0+(int)(p)+(n))
#else !lint
#define Mkp(t,p,n) ((t) ((char *) (p) + (n)))
#endif lint


#define OffsetOf(x,t) ((int)&(((t*)0)->x))


#ifndef NULL
#define NULL 0
#endif NULL



/* Local Routines */

 caddr_t   gen_classify_func (/* gifcp, mp, pf,
     nethdrp, validlen */);
 void   gen_clockfast_func (/* gifcp */);
 int   gen_control_func (/* gifcp, op, datap, datalen */);
 struct mbuf *gen_dq_func (/* gifcp */);
 void   gen_drain_func (/* gifcp, state1p, state2p, state1ul,
     state2ul, npktsp, nbytesp */);
 int   gen_enforce_func (/* gifcp, mp, totlen, timevalp */);
 int   gen_init_func (/* gifcp */);
 void   gen_nq_func (/* gifcp, mp */);
 void   gen_quit_func (/* gifcp */);

#ifdef lint
 caddr_t   LintClassify (/* gifcp, mp, pf, nethdrp, validlen */);
 void   LintClockfast (/* gifcp */);
 int   LintControl (/* gifcp, cmd, datap, datalen */);
 struct mbuf *LintDq (/* gifcp */);
 void   LintDrain (/* gifcp, state1p, state2p,
    state1ul, state2ul, npktsp, nbytesp */);
 int   LintEnforce (/* gifcp, mp, totlen, timevalp */);
```

```
int   LintInit (/* gifcp */);
void   LintNq (/* gifcp, mp */);
void   LintQuit (/* gifcp */);
int   LintRmAlloc (/* gifcp, ridp, arsclpp, ratep, bytesp,
     fdp, destpp */);
int   LintRmCtrl (/* gifcp, cmd, datap, datalen */);
int   LintRmIdGet (/* gifcp, ridp */);
int   LintRmProbe (/* gifcp , TBD */);
int   LintRmIdRel (/* gifcp, ridp */);
int   LintRmRlse (/* gifcp, ridp, arsclpp, ratep, bytesp,
    fdp, destpp */);
int   LintRsrcAlloc (/* gifcp, ridp, arsclpp, ratep, bytesp,
     fdp, destpp */);
int   LintRsrcCtrl (/* gifcp, op, datap, datalen */);
int   LintRsrcGetid (/* gifcp, ridp */);
int   LintRsrcProbe (/* gifcp , TBD */);
int   LintRsrcRelid (/* gifcp, ridp */);
int   LintRsrcRlse (/* gifcp, ridp, arsclpp, ratep, bytesp,
     fdp, destpp */);
#endif lint

static int   tcif_AlgSwitch (/* gifcp, quit */);
 struct aNetIf *tfic_dev2gifcp (/* namep */);
 void   tcif_dq_func (/* ifqp, mpp */);
static int   tcif_dummy_output (/* ifnetp, pktp, ifsockadr */);
#if WANT_ETHERNET
static int   tcif_ether_output (/* acp, pktp, sockaddrp,
     fnc_start */);
#if NIE
/*static*/ int   tcif_ieoutput (/* acp, pktp, sockaddrp */);
#endif NIE
#endif WANT_ETHERNET
 struct aNetIf *tcif_ifp2gifcp (/* osifp */);
 void   tfic_init_gifcs (/* */);
 int   tcif_ioctl (/* cmd, argdatap, ifp */);
#if WANT_ETHERNET
#if NLE
/*static*/ int   tcif_leoutput (/* acp, pktp, sockaddrp */);
#endif NLE
#endif WANT_ETHERNET
 void   tcif_nq_func (/* ifqp, mp */);
static int   tcif_random_drop (/* ifqp */);



/* Local Data Structures */

/*
c* Number of generic network interfaces (DEF_GENIFS). Dummy plus one
c* per physical network interface plus one per Origin and Target on
c* local system.
 */
```

```
#ifdef STIIAPI
#define DEF_GENIFS (10 + 32)  /* Physical plus API pseudo */
#else
#define DEF_GENIFS (10)   /* Physical */
#endif STIIAPI

static struct ifnet  dummyif = { "dummy", 0, 576 }; /* name, unit, mtu */
#define TC_MAX 5   /* TC_NUN if C were less primitive */
static ExpAry (a,TrafficControl,TC_MAX*DEF_GENIFS)
   tcif_cache = { InitaExpAry (TC_MAX*DEF_GENIFS,TC_MAX) };
 ExpAry (a,NetIf,DEF_GENIFS)
   tfic_genifs = { InitaExpAry (DEF_GENIFS,0x9) };
 struct aNetIf *tcif_gifcheadp = NO_NETIFP;

struct _ovrlay {
 char   ifname[ IFNAMSIZ ]; /* ifr_name */
 char   data[1];
};


#ifdef STII
#define TcIfFlgOwnLE  ST2FlgOwnLE
#else !STII
#define ConfigFlag(x)  ((tcif_config & (x)) NE 0)
#define TcIfFlgOwnLE  (0x00100000)
static unsigned long  tcif_config = TcIfFlgOwnLE;
#endif STII


/* Tables for symbolic names */

struct aTCname {
 enum TC_Algorithm  value;  /* TC_xxx */
 char   name[16]; /* name */
}    tcif_TCnames[] =
 { /* @#$% cpp is too primative to allow \ in the formal parameter list */
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
  { Ident(TC_)id, "id" },
 aTcVectorList
#undef aTV
};


/*
 * Pre-defined Resource Management vectors
 */

extern int   tcif_BctAlloc (), tcif_BctRlse (), tcif_PtpAlloc (),
   tcif_PtpRlse (), tcif_RsIdGet (), tcif_RsIdRel ();
```

```
struct aRmVector {
 char   name[16];
 struct aResourceManagement rmf;
}    tcif_rmvectors[] =
 { /* "<--- name ----->" rsrcalloc rsrcctrl rsrcgetid rsrcprobe rsrcrelid
 * rsrcrelse */

#define aRV(id,name,aloc,ctrl,idget,probe,idrel,rlse)   \
 { "name", { aloc, ctrl, idget, probe, idrel, rlse } },

 aRmVectorList    /* Defined in st2_resource.h */

#undef aRV

 };


/*
 * Pre-defined TrafficControl vectors
 */

#ifdef SFQ_VC
extern int sfq_vc_init(/* gifcp */);
extern void sfq_vc_nq(/* gifcp, mp */);
extern struct mbuf *sfq_vc_dq(/* gifcp */);
extern void sfq_vc_drain(/* gifcp, state1p, state2p, state1ul, \
    state2ul, npktsp, nbytesp */);
extern caddr_t sfq_vc_classify(/*gifcp, mp, pf, nethdrp, validlen*/);
extern int sfq_vc_enforce(/* gifcp, mp, totlen, timevalp */);
#endif SFQ_VC

#ifdef VIRTUAL_CLOCK
extern int   vc_aloc_func (/* gifcp, arsc1p, bw, srcp, dstp, ctlp */);
extern int   vc_enf_func (/* gifcp, pktp, totlen, timevalp */);
extern int   vc_init_func (/* gifcp */);
extern void   vc_nq_func(/* gifcp, mp */);
#endif VIRTUAL_CLOCK

#ifdef FAIR_SHARE
extern int   fs_aloc_func (/* gifcp, arsc1p, bw,
     srcp, dstp, ctlp */);
extern caddr_t   fs_classify_func (/* gifcp, mp, PF_xxx,
     nethdrp, validlen */);
extern void   fs_clockfast_func (/* gifcp */);
extern int   fs_control_func (/* gifcp, op, xxxp, xxx1 */),
   fs_enforce_func (/* gifcp, mp, totlen, timevalp */),
   fs_init_func (/* gifcp */);
extern void   fs_nq1q_func (/* gifcp, mp */);
extern int   fs_rlse_func (/* gifcp, arsc1p, bw, srcp, dstp, ctlp */);
#endif FAIR_SHARE
```

```
struct aTcVector {
 char   name[16];
 struct aTrafficControl tcf;
}    tcif_tcvectors[] =
 { /* @#$% cpp is too primitive to allow \ in the formal parameter list */
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 { "name", { clsfy, fsclk, cntrl, dq, drain, enfrc, init, nq, quit, \
  aloc, cntrl, idget, probe, idrel, rlse } },

 aTcVectorList    /* Defined in st2_resource.h */

#undef aTV

};


/*
 * Bandwidth
 */

struct aBandwidthInfo {
 char   name[16];
 unsigned long  bw_conf,
    bw_load,
    bw_resv;
}    bws[] =
 { /*"<--- name ---->" conf load resv */
#define ENET_10MB 0
/* nfs complains ...
 { "ethernet_10mb", 1250000, 40000, 20000 },
*/
 { "ethernet_10mb", 1250000, 400000, 200000 },
#define HSIS_1344 1
 { "hsis_1344", 168000, 134400, 115000 },
 { "" }
 };


/* Driver Output Traffic Control Algorithms */

/*
S* gen_classify_func ( gifcp, mp, pf, nethdrp, validlen )
s*
s* Called to classify packet pointed to by mp, of protocol family
s* pf, whose network header is pointed to be nethdrp, where there
s* are validlen octets of network header. Returns a pointer to the
s* flow.
s*
 */
```

```
 caddr_t
gen_classify_func ( gifcp, mp, pf, nethdrp, validlen )
 struct aNetIf  *gifcp;
 struct mbuf  *mp;
 int    pf;
 caddr_t   nethdrp;
 int    validlen;
{
#ifdef lint
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 (void) clsfy ( gifcp, mp, pf, nethdrp, validlen );
 aTcVectorList;
#undef aTV
#endif lint

 return ( (caddr_t) NULL );
}


/*
S* gen_clockfast_func ( gifcp )
s*
s* Called to perform any Traffic Control timer activities.
s*
 */

 void
gen_clockfast_func ( gifcp )
 struct aNetIf  *gifcp;
{
#ifdef lint
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 fsclk ( gifcp );
 aTcVectorList;
#undef aTV
#endif lint

 return;
}


/*
S* gen_control_func ( gifcp, op, datap, datalen )
s*
s* Called to perform any Traffic Control control operations.
s*
 */
```

```
 int
gen_control_func ( gifcp, op, datap, datalen )
 struct aNetIf  *gifcp;
 int    op;
 caddr_t   datap;
 int    datalen;
{
#ifdef lint
 int    error;


#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 error = cntrl ( gifcp, op, datap, datalen );
 aTcVectorList;
#undef aTV
 error = error;
#endif lint

 return ( 0 );
}


/*
S* gen_dq_func ( gifcp )
s*
s* Called to get next packet to be sent.
s*
 */

 struct mbuf *
gen_dq_func ( gifcp )
 struct aNetIf  *gifcp;
{
 struct ifqueue *ifqp = &( gifcp->ifc_snd );
 struct mbuf  *mp;


#ifdef lint
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 mp = dq ( gifcp );
 aTcVectorList;
#undef aTV
#endif lint

 IF_DEQUEUE (ifqp, mp);

 return ( mp );
}
```

```
/*
S* gen_drain_func ( gifcp, state1p, state2p, state1ul, state2ul, npktsp,
nbytesp )
s*
s* Called to flush any packets queued for output. Returns the number
s* of dropped packets and dropped bytes.
s*
 */

 void
gen_drain_func ( gifcp, state1p, state2p, state1ul, state2ul, npktsp, nbytesp
)
 struct aNetIf  *gifcp;
 caddr_t   state1p, /* ARGSUSED1 */
    state2p;
 unsigned long   state1ul,
    state2ul,
   *npktsp,
   *nbytesp;
{
 struct ifqueue *ifqp = &( gifcp->ifc_snd );
 struct mbuf  *mp,
   *xp;
 int    oldpri;
 unsigned long  npkts = 0,
    nbytes = 0;


#ifdef lint
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 drain ( gifcp, state1p, state2p, state1ul, state2ul, &( npkts ), &( nbytes )
);
 aTcVectorList;
#undef aTV
#endif lint

 oldpri = splnet ();
 while ( (mp = ifqp->ifq_head) NE (struct mbuf *) NULL )
 {
 ifqp->ifq_head = mp->m_act;
 mp->m_act = (struct mbuf *) NULL;

 for ( xp = mp ; xp NE (struct mbuf *) NULL ; xp = xp->m_next )
  nbytes += xp->m_len;

 npkts++;
 m_freem ( mp );
 }

 ifqp->ifq_tail = (struct mbuf *) NULL;
```

```
 (void) splx ( oldpri );

 if ( npktsp NE (unsigned long *) NULL )
 *npktsp = npkts;

 if ( nbytesp NE (unsigned long *) NULL )
 *nbytesp = nbytes;

 return;
}


/*
S* gen_enforce_func ( gifcp, mp, totlen, timevalp )
s*
s* Called at splimp (or higher depending on driver).
s*
 */

 int
gen_enforce_func( gifcp, mp, totlen, timevalp )
 struct aNetIf  *gifcp;
 struct mbuf  *mp;
 unsigned int   totlen; /* ARGSUSED */
 struct timeval *timevalp;
{
 int    result = 0;


#ifdef lint
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 result = enfrc ( gifcp, mp, totlen, timevalp );
 aTcVectorList;
#undef aTV
#endif lint

 if ( IF_QFULL ( &( gifcp->ifc_snd ) ) )
 return ( -1 );

 return ( result );
}


/*
S* gen_init_func ( gifcp )
s*
s* Called to initialize Traffic Control data structures.
s*
 */
```

```
 int
gen_init_func ( gifcp )
 struct aNetIf  *gifcp;
{
 int    result = 0/*NoError*/;


#ifdef lint
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 result = init ( gifcp );
 aTcVectorList;
#undef aTV
#endif lint

 /* Nothing to do -- maybe set gifcp->osifp->if_snd.max_len */

 return ( result );
}


/*
S* gen_nq_func ( gifcp, mp )
s*
s* Called at splimp (or higher depending on driver).
s*
 */

 void
gen_nq_func( gifcp, mp )
 struct aNetIf  *gifcp;
 struct mbuf  *mp;
{
 struct ifqueue *ifqp = &( gifcp->ifc_snd );

#ifdef lint
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 nq ( gifcp, mp );
 aTcVectorList;
#undef aTV
#endif lint

 IF_ENQUEUE (ifqp, mp);

 return;
}


/*
```

```
S* gen_quit_func ( gifcp )
s*
s* Called to stop using Traffic Control algorithm.
s*
 */

 void
gen_quit_func ( gifcp )
 struct aNetIf  *gifcp;
{
 unsigned long  npkts,
    nbytes;

#ifdef lint
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 quit ( gifcp );
 aTcVectorList;
#undef aTV
#endif lint

 if ( gifcp->iftc_drain NE (void (*)()) NULL )
 (*gifcp->iftc_drain) ( gifcp, (caddr_t) NULL, (caddr_t) NULL,
    (unsigned long *) NULL, (unsigned long *) NULL,
    &( npkts ), &( nbytes ) );

 /* gifcp->osifp->if_snd.ifq_drops += npkts */

 return;
}


#ifdef lint

 caddr_t
LintClassify ( gifcp, mp, pf, nethdrp, validlen )
 struct aNetIf  *gifcp;
 struct mbuf  *mp;
 int    pf;
 caddr_t   nethdrp;
 int    validlen;
{
 return ( LintClassify ( gifcp, mp, pf, nethdrp, validlen ) );
}

 void
LintClockfast ( gifcp )
 struct aNetIf  *gifcp;
{
 LintClockfast ( gifcp );
 return;
```

```
}

 int
LintControl ( gifcp, cmd, datap, datalen )
 struct aNetIf  *gifcp;
 int    cmd;
 caddr_t   datap;
 int    datalen;
{
 return ( LintControl ( gifcp, cmd, datap, datalen ) );
}

 struct mbuf *
LintDq ( gifcp )
 struct aNetIf  *gifcp;
{
 return ( LintDq ( gifcp ) );
}

 void
LintDrain ( gifcp, state1p, state2p, state1ul, state2ul, npktsp, nbytesp )
 struct aNetIf  *gifcp;
 caddr_t   state1p,
    state2p;
 unsigned long   state1ul,
    state2ul,
   *npktsp,
   *nbytesp;
{
 LintDrain ( gifcp, state1p, state2p, state1ul, state2ul, npktsp, nbytesp );
 return;
}

 int
LintEnforce ( gifcp, mp, totlen, timevalp )
 struct aNetIf  *gifcp;
 struct mbuf  *mp;
 unsigned int   totlen;
 struct timeval *timevalp;
{
 return ( LintEnforce ( gifcp, mp, totlen, timevalp ) );
}

 int
LintInit ( gifcp )
 struct aNetIf  *gifcp;
{
 return ( LintInit ( gifcp ) );
}

 void
LintNq ( gifcp, mp )
```

```
 struct aNetIf  *gifcp;
 struct mbuf  *mp;
{
 LintNq ( gifcp, mp );
 return;
}

 void
LintQuit ( gifcp )
 struct aNetIf  *gifcp;
{
 LintQuit ( gifcp );
 return;
}

 int
LintRmAlloc ( gifcp, ridp, arsclpp, ratep, bytesp, fdp, destpp )
 struct aNetIf   *gifcp;
 struct aRsrcId  *ridp;
 struct aRsrcClnt **arsclpp;
 unsigned long   *ratep,
    *bytesp;
 struct aFlowDesc  *fdp;
 struct sockaddr **destpp;
{
 return ( LintRmAlloc ( gifcp, ridp, arsclpp, ratep, bytesp, fdp, destpp ) );
}

 int
LintRmCtrl ( gifcp, cmd, datap, datalen )
 struct aNetIf  *gifcp;
 int    cmd;
 caddr_t   datap;
 int    datalen;
{
 return ( LintRmCtrl ( gifcp, cmd, datap, datalen ) );
}

 int
LintRmIdGet ( gifcp, ridp )
 struct aNetIf   *gifcp;
 struct aRsrcId  *ridp;
{
 return ( LintRmIdGet ( gifcp, ridp ) );
}

 int
LintRmProbe ( gifcp /* , TBD */ )
 struct aNetIf   *gifcp;
{
 return ( LintRmProbe ( gifcp /* , TBD */ ) );
}
```

```
 int
LintRmIdRel ( gifcp, ridp )
 struct aNetIf   *gifcp;
 struct aRsrcId  *ridp;
{
 return ( LintRmIdRel ( gifcp, ridp ) );
}

 int
LintRmRlse ( gifcp, ridp, arsclpp, ratep, bytesp, fdp, destpp )
 struct aNetIf   *gifcp;
 struct aRsrcId  *ridp;
 struct aRsrcClnt **arsclpp;
 unsigned long   *ratep,
    *bytesp;
 struct aFlowDesc  *fdp;
 struct sockaddr **destpp;
{
 return ( LintRmRlse ( gifcp, ridp, arsclpp, ratep, bytesp, fdp, destpp ) );
}

 int
LintRsrcAlloc ( gifcp, ridp, arsclpp, ratep, bytesp, fdp, destpp )
 struct aNetIf   *gifcp;
 struct aRsrcId  *ridp;
 struct aRsrcClnt **arsclpp;
 unsigned long   *ratep,
    *bytesp;
 struct aFlowSpec  *fdp;
 struct sockaddr **destpp;
{
 return ( LintRsrcAlloc ( gifcp, ridp, arsclpp, ratep, bytesp,
    fdp, destpp ) );
}

 int
LintRsrcCtrl ( gifcp, op, datap, datalen )
 struct aNetIf  *gifcp;
 int    op;
 caddr_t   datap;
 int    datalen;
{
 return ( LintRsrcCtrl ( gifcp, op, datap, datalen ) );
}

 int
LintRsrcGetid ( gifcp, ridp )
 struct aNetIf   *gifcp;
 struct aRsrcId  *ridp;
{
 return ( LintRsrcGetid ( gifcp, ridp ) );
```

```
}

 int
LintRsrcProbe ( gifcp /* , TBD */ )
 struct aNetIf   *gifcp;
{
 return ( LintRsrcProbe ( gifcp /* , TBD */ ) );
}

 int
LintRsrcRelid ( gifcp, ridp )
 struct aNetIf   *gifcp;
 struct aRsrcId  *ridp;
{
 return ( LintRsrcRelid ( gifcp, ridp ) );
}

 int
LintRsrcRlse ( gifcp, ridp, arsclpp, ratep, bytesp, fdp, destpp )
 struct aNetIf   *gifcp;
 struct aRsrcId  *ridp;
 struct aRsrcClnt **arsclpp;
 unsigned long   *ratep,
    *bytesp;
 struct aFlowDesc  *fdp;
 struct sockaddr **destpp;
{
 return ( LintRsrcRlse ( gifcp, ridp, arsclpp, ratep, bytesp,
    fdp, destpp ) );
}

#endif lint


/*
S* tcif_AlgSwitch ( gifcp, quit )
s*
s* Switch to new traffic control algorithm on the specified interface.
s*
 */

 static int    /* Next bugid 0x71203 */
tcif_AlgSwitch( gifcp, quit )  /* from init_gifcs */
 struct aNetIf  *gifcp;
 int    quit;
{
 struct mbuf  *mp;
 int    alg = (int) gifcp->alg_next,
    oldpri;
 unsigned long  nbytes,
    npkts;
 struct aTrafficControl *tcp;
```

```
 /* Verify can switch before make changes */
 switch ( alg )
 {
 default: printf ( "tcif_AlgSwitch: Invalid Algorithm (%u)\n", alg );
  gifcp->alg_next = gifcp->alg;
  return ( EINVAL );

 case TC_FIFO: break;

#ifdef FAIR_SHARE
 case TC_FS1: break;
#endif FAIR_SHARE

 case TC_RD:  break;

#ifdef MY_FIFO
 case TC_MY_FIFO: break;
#endif MY_FIFO

/*
#ifdef SFQ
 case TC_SFQ: break;
#endif SFQ
*/

#ifdef SFQ_VC
 case TC_SFQ_VC: break;
#endif SFQ_VC

#ifdef VIRTUAL_CLOCK
 case TC_VC:  break;
#endif VIRTUAL_CLOCK
 } /* end of alg switch */


 if ( (int) TC_NUN > TC_MAX )
 {
 printf ( "tcif_AlgSwitch: TC_NUN (%u) must be <= TC_MAX (%u)\n",
   TC_NUN, TC_MAX );
 return ( ENOSR );
 }

 oldpri = spl4 ();  /* bring interface down, wait, ... */

 /* Locate cache area */
 if ( gifcp->per_alg EQ (caddr_t) NULL )
 {
 if ( tcif_cache.allocated < (tcif_cache.nxtfree + TC_MAX) )
  return ( ENOMEM );  /* ??? add more */
```

```
 tcp = &( tcif_cache.TrafficControls[ tcif_cache.nxtfree ] );
 tcif_cache.nxtfree += TC_MAX;
 bzero ( (char *) tcp, TC_MAX * sizeof (tcif_cache.TrafficControls[0]) );
 gifcp->per_alg = (caddr_t) tcp;
 }
 tcp = (struct aTrafficControl *) gifcp->per_alg; /* lint ppap */

#if 0
 /* Cache pervious algorithm's info */
 Bcopy ( &( gifcp->tcf ), /*->*/ (tcp + gifcp->iftc_alg), sizeof (* tcp) );
#endif 0

 /* Flush old queue(s) */
 if ( gifcp->iftc_drain NE (void (*)()) NULL )
 {
 (*gifcp->iftc_drain) ( gifcp, gifcp->iftc_state1p, gifcp->iftc_state2p,
    gifcp->iftc_state1ul, gifcp->iftc_state2ul,
    &npkts, &nbytes );

#ifdef lint
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 drain ( gifcp, gifcp->iftc_state1p, gifcp->iftc_state2p, \
  gifcp->iftc_state1ul, gifcp->iftc_state2ul, &npkts, &nbytes );
 aTcVectorList
#undef aTV
#endif lint
 }
 else
 {
 for (;;)
 {
  if ( gifcp->iftc_dq NE (struct mbuf *(*)()) NULL )
  mp = (*gifcp->iftc_dq) ( gifcp );
  else
  {
  IF_DEQUEUE ( &( gifcp->osifcp->if_snd ), mp );
  }

  if ( mp EQ (struct mbuf *) NULL )
  break;

#ifndef STII
  m_freem ( mp );
#else STII
  FreePkts ((struct aPktDesc *) mp);
#endif STII
 } /* end of forever loop */
 }

 if ( quit && (gifcp->iftc_quit NE (void (*)()) NULL) )
```

```
 (*gifcp->iftc_quit) ( gifcp );

#ifdef lint
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 quit ( gifcp );
 aTcVectorList;
#undef aTV
#endif lint

#if 1
 /* Cache pervious algorithm's info */
 Bcopy ( &( gifcp->tcf ), /*->*/ (tcp + gifcp->iftc_alg), sizeof (* tcp) );
#endif 1

 /* Switch algorithms */
 gifcp->alg = (short) alg;

 tcp += alg;    /* Restore previous parameters */
 Bcopy ( tcp, /*->*/ &( gifcp->tcf ), sizeof (gifcp->tcf) );

 if ( (gifcp->iftc_state1p EQ (caddr_t) NULL)
 AND (gifcp->iftc_state2p EQ (caddr_t) NULL)
 AND (gifcp->iftc_state1ul EQ 0)
 AND (gifcp->iftc_state2ul EQ 0) )
 {
 switch ( alg )
 {
 default:
  printf ( "tcif_AlgSwitch: Invalid Algorithm (%u); using FIFO\n",
   alg );
  gifcp->alg = (short) TC_FIFO;
  gifcp->alg_next = (short) TC_FIFO;
  gifcp->tcf = tcif_tcvectors[ (int) TC_FIFO ].tcf;
  break;

#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 case Ident(TC_)id:      \
  gifcp->tcf = tcif_tcvectors[ (int) Ident(TC_)id ].tcf; \
  break;

 aTcVectorList

#undef aTV
 } /* end of alg switch */
 }

 if ( gifcp->iftc_init NE (int (*)()) NULL )
 (void) (*gifcp->iftc_init) ( gifcp ); /* ??? check for error & undo */
```

```
 splx ( oldpri );  /* bring interface back up */

#if 0
 if ( gifcp->iftc_clockfast NE (void (*)()) NULL )
 (*gifcp->iftc_clockfast) ( gifcp );
#endif

 return ( 0 );
}


/*
S* tfic_dev2gifcp ( namep )
s*
s* Find generic interface given device name & unit string.
s*
 */

 struct aNetIf *
tfic_dev2gifcp( namep )
 char   *namep;
{
 struct aNetIf *gifcp = tcif_gifcheadp;

 while ( gifcp NE (struct aNetIf *) NULL )
 {
 if ( strcmp ( namep, &( gifcp->ifc_name[0] ) ) EQ 0 )
  return ( gifcp );

 gifcp = gifcp->ifc_nextp;
 }

 return ( (struct aNetIf *) NULL );
}


/* catch macro calls and convert into proper interface calls */

/*
S* tcif_dq_func( ifqp, mpp )
s*
s* Convergence routine from SunOS IF_DEQUEUE to Traffic Control
s* functions. Arguments are address if ifnet's ifqueue and address
s* of pointer for packet.
 *
s* Called at splimp (or higher, depending on driver).
s*
 */

 void
tcif_dq_func( ifqp, mpp )
 struct ifqueue  *ifqp;
```

```
 struct mbuf  **mpp;
{
 struct aNetIf *gifcp;
 struct mbuf  *mp,
   *(*dqfuncp)();


 /* Map ifqueue pointer back to ifnet, then to extended ifnet (aNetIf) */

 gifcp = tcif_ifp2gifcp( Mkp (caddr_t,ifqp,
     (-OffsetOf (if_snd,struct ifnet))) );
 /* dummyif0 / mp = NULL; ??? */

 /* If no specific dequeue function is supplied, use specified queue */

 if ( (dqfuncp = gifcp->iftc_dq) NE (struct mbuf * (*)()) NULL )
mp = (*dqfuncp) ( gifcp ); /* Use specific dequeue function */
 else    /* Use generic dequeue function */
 IF_DEQUEUE (ifqp, mp);

 *mpp = mp;    /* Return ptr to packet, if any */

 return;
}



 static int
tcif_dummy_output ( ifnetp, mp, sap )
 struct ifnet  *ifnetp; /* ARGSUSED */
 struct mbuf  *mp;  /* we free */
 struct sockaddr *sap;
 {
#ifndef STII
 m_freem ( mp );
#else STII
 FreePkts ((struct aPktDesc *) mp);
#endif STII

 ifnetp->if_oerrors++;
 return ( 0 );
}


#if WANT_ETHERNET

/*
S* tcif_ether_output ( acp, pktp, sockaddrp, fnc_start )
s*
s* Routine called to build ethernet packets and queue them for
s* transmission.
 *
```

```
 * Just because we don't have source to be able to recoompile
 * ether_output with a different IF_ENQUEUE macro in <net/if.h>
 * & fix randomdrop.
s*
 * Called indirectly via ifnet if_output dispatch.
 *
 */


 static int
tcif_ether_output( acp, pktp, sockaddrp, fnc_start )
 struct arpcom   *acp;  /* begins with struct ifnet */
 struct mbuf   *pktp;  /* packet to be sent, we dispose of it */
 struct sockaddr  *sockaddrp; /* sockaddr of some flavor */
 void   (*fnc_start) (); /* driver output-start function */
{
 unsigned short  ether_type; /* ether_type */
 int    len,  /* of packet */
    oldpri;
 struct ether_addr  dst_ea; /* an ethernet address */
 struct ether_family *efp;  /* to find appropriate dispatches */
 struct ether_header *ehp;  /* ehternet header */
 struct mbuf  *mh = pktp, /* mbuf that will have ether header */
    *bcstp = NULL, /* copy of packet if to be broadcast */
   *mp;  /* to scan packet to find length */
#ifdef TRAFFIC_CONTROL
 struct aNetIf *gifcp;
 caddr_t   flowp,
    nethdrp;
 int    validlen;
 unsigned long  key,
    rsrc;
 unsigned short  pf;
#endif TRAFFIC_CONTROL

 if ( (acp->ac_if.if_flags & (IFF_RUNNING | IFF_UP))
 NE (IFF_RUNNING | IFF_UP) )
 {
 m_freem ( pktp );
 return ( ENETDOWN );
 }

#ifdef TRAFFIC_CONTROL
 gifcp = tcif_ifp2gifcp( (caddr_t) acp ); /* dummyif0 ??? */

 pf = PF_UNSPEC;
 nethdrp = mtod ( pktp, caddr_t);
 validlen = pktp->m_len;

 if ( pktp->m_type EQ MT_TCDATA )
 {
 flowp = pktp->m_tcflowp;
```

```
rsrc = pktp->m_tcrsrc;
key = pktp->m_tckey;
}
else
{
flowp = (caddr_t) NULL;
rsrc = 0;
key = 0;
}
#endif TRAFFIC_CONTROL

/* Map protocol specific address in sockaddr to local network address */

switch ( sockaddrp->sa_family )
{
case AF_INET:   /* 2 IP packets */
oldpri = splimp ();
{
 if ( acp->ac_lastip.s_addr /* is translation in cache */
 NE ((struct sockaddr_in *) sockaddrp)->sin_addr.s_addr )
 {    /* no */
 acp->ac_lastip = ((struct sockaddr_in *) sockaddrp)->sin_addr;
 if ( NOT arpresolve ( acp, pktp ) ) /* look it up */
 {   /* not in table, arping it */
  acp->ac_lastip.s_addr = 0; /* no valid translation */
  splx ( oldpri );
  return ( 0 ); /* held til arp'd, if_output called */
 }
 }
#ifdef sparc    /* alignment is at least octet2 */
  * (short *) &( dst_ea.ether_addr_octet[0] ) /* lint ppap */
  = * (short *) &( acp->ac_lastarp.ether_addr_octet[0] );
  * (short *) &( dst_ea.ether_addr_octet[2] ) /* lint ppap */
  = * (short *) &( acp->ac_lastarp.ether_addr_octet[2] );
  * (short *) &( dst_ea.ether_addr_octet[4] ) /* lint ppap */
  = * (short *) &( acp->ac_lastarp.ether_addr_octet[4] );
#else !sparc
  dst_ea = acp->ac_lastarp;
#endif sparc
  ether_type = ETHERTYPE_IP;
#ifdef TRAFFIC_CONTROL
  pf = PF_INET;
#endif TRAFFIC_CONTROL
 }
 splx ( oldpri );
 break;

 case AF_UNSPEC:   /* 0 Ethernet packets */
    /* sockaddr has ethernet header */
 ehp = (struct ether_header *) &( sockaddrp->sa_data[0] ); /* lint ppap */

#ifdef sparc    /* alignment is at least octet2 */
```

```
 * (short *) &( dst_ea.ether_addr_octet[0] ) /* lint ppap */
  = * (short *) &( ehp->ether_dhost.ether_addr_octet[0] );
 * (short *) &( dst_ea.ether_addr_octet[2] ) /* lint ppap */
  = * (short *) &( ehp->ether_dhost.ether_addr_octet[2] );
 * (short *) &( dst_ea.ether_addr_octet[4] ) /* lint ppap */
  = * (short *) &( ehp->ether_dhost.ether_addr_octet[4] );
#else !sparc
 dst_ea = ehp->ether_dhost;
#endif sparc
 ether_type = ehp->ether_type; /* use specified ether_type */
#ifdef TRAFFIC_CONTROL
 if ( ether_type EQ ETHERTYPE_IP )
  pf = PF_INET;
#ifdef STII
 else if ( ether_type EQ ConfigParm (ethertype) )
  pf = PF_COIP;
#endif STII
#endif TRAFFIC_CONTROL
 break;

 default:    /* Lookup other types */
 efp = ether_families;  /* List of known types */
 while ( efp )
 {
  if ( efp->ef_family EQ sockaddrp->sa_family )
  break;   /* Found address family */

  efp = efp->ef_next;  /* try next */
 }

 if ( efp )   /* if found table entry */
 {
  if ( efp->ef_outfunc ) /* better have address translation */
  {
  if ( efp->ef_outfunc ( sockaddrp, pktp, acp, &( dst_ea ) ) )
   return ( 0 ); /* ef_outfunc disposed of pktp */
     /* classify, enforce, nq ??? */

  if ( efp->ef_ethertype EQ 1500 ) /* ? ETHERMTU */
  {
   len = 0;  /* find packet length */
   if ( mp = pktp )
   do
   {
    len += mp->m_len;
    mp = mp->m_next;
   } while ( mp );
   ether_type = len; /* use length as "ether_type" */
   /* leave pf = PF_UNSPEC */
   break;
  }
  ether_type = efp->ef_ethertype; /* ether_type from table */
```

```
#ifdef TRAFFIC_CONTROL
  pf = efp->ef_family;
#endif TRAFFIC_CONTROL
  break;
  }
 }

 identify ( acp );  /* unsuported ether_type, drop pkt */
 printf ( "can't handle AF 0x%x", sockaddrp->sa_family );
 m_freem ( pktp );
 return ( EAFNOSUPPORT );
 break;
 } /* end of sockaddrp->sa_family switch */


 /* Check if destined to the broadcast address */

 if ( ( * (short *) &( etherbroadcastaddr.ether_addr_octet[4] )
  EQ * (short *) &( dst_ea.ether_addr_octet[4] )) /* lint ppap */
 AND ( * (short *) &( etherbroadcastaddr.ether_addr_octet[2] )
  EQ * (short *) &( dst_ea.ether_addr_octet[2] )) /* lint ppap */
 AND ( * (short *) &( etherbroadcastaddr.ether_addr_octet[0] )
  EQ * (short *) &( dst_ea.ether_addr_octet[0] )) ) /* lint ppap */
 {
    /* yes, make copy for local delivery */
 bcstp = (struct mbuf *) m_copy ( pktp, 0, M_COPYALL );
 }


 /* Find space for ethernet header */

 if ( (pktp->m_off <
#ifdef TRAFFIC_CONTROL
    (unsigned long) OffsetOf (m_tcdat[0],struct mbuf)
#else !TRAFFIC_CONTROL
    MMINOFF
#endif TRAFFIC_CONTROL
    + sizeof (struct ether_header))
 OR ( M_HASCL( pktp )
#ifdef MCL_STATIC_HDR
  AND (pktp->m_cltype NE MCL_STATIC_HDR)
#endif MCL_STATIC_HDR
  ) )  /* no room in first mbuf, prepend another */
 {
#ifdef TRAFFIC_CONTROL
 mh = (struct mbuf *) m_get ( M_DONTWAIT, MT_TCDATA );
#else !TRAFFIC_CONTROL
 mh = (struct mbuf *) m_get ( M_DONTWAIT, MT_HEADER );
#endif TRAFFIC_CONTROL
 if ( ! mh )
 {
  m_freem ( pktp );
```

```
  m_freem ( bcstp );
  ether_error ( acp, "WARNING: no mbufs" );
  return ( ENOBUFS );
 }

#ifdef TRAFFIC_CONTROL   /* XXX 8 => mod x10 aligned */
 mh->m_off = OffsetOf (m_tcdat[8],struct mbuf);
#endif TRAFFIC_CONTROL
 mh->m_next = pktp;  /* prepend */
 mh->m_len = sizeof ( struct ether_header );
 }
 else    /* insert header */
 {
 pktp->m_off -= sizeof ( struct ether_header );
 pktp->m_len += sizeof ( struct ether_header );
 }


 /* Construct ethernet header */

 ehp = mtod ( mh, struct ether_header *);

#ifdef sparc    /* alignment is at least octet2 */
 * (short *) &( ehp->ether_dhost.ether_addr_octet[0] ) /* lint ppap */
 = * (short *) &( dst_ea.ether_addr_octet[0] );
 * (short *) &( ehp->ether_dhost.ether_addr_octet[2] ) /* lint ppap */
 = * (short *) &( dst_ea.ether_addr_octet[2] );
 * (short *) &( ehp->ether_dhost.ether_addr_octet[4] ) /* lint ppap */
 = * (short *) &( dst_ea.ether_addr_octet[4] );
#else !sparc
 ehp->ether_dhost = dst_ea;
#endif sparc

 ehp->ether_type = ether_type;


 /* Deliver local copy if broadcast */

 if ( bcstp )
 {
 mp = bcstp;
 len = 0;   /* find length */
 /* if ( bcstp ) */
 do
 {
  len += mp->m_len;
  mp = mp->m_next;
 } while ( mp );

#ifdef sparc    /* alignment is at least octet2 */
 * (short *) &( ehp->ether_shost.ether_addr_octet[0] ) /* lint ppap */
  = * (short *) &( acp->ac_enaddr.ether_addr_octet[0] );
```

```
 * (short *) &( ehp->ether_shost.ether_addr_octet[2] ) /* lint ppap */
  = * (short *) &( acp->ac_enaddr.ether_addr_octet[2] );
 * (short *) &( ehp->ether_shost.ether_addr_octet[4] ) /* lint ppap */
  = * (short *) &( acp->ac_enaddr.ether_addr_octet[4] );
#else !sparc
 ehp->ether_shost = acp->ac_enaddr;
#endif sparc

 /* deliver local copy */
 do_protocol ( ehp, bcstp, acp, len );
 }

#ifdef TRAFFIC_CONTROL
 len = mh->m_len;
 if ( (mp = mh->m_next) NE (struct mbuf *) NULL )
 do
 {
  len += mp->m_len;
 } while ( (mp = mp->m_next) NE (struct mbuf *) NULL );

 mh->m_tcrsrc = rsrc;
 mh->m_tckey = key;

 key = mh->m_type;   /* *** mbuf stats update below */
 mh->m_type = MT_TCDATA;

 if ( flowp EQ (caddr_t) NULL )
 {
 if ( gifcp->iftc_classify NE (caddr_t (*)()) NULL )
  flowp = (*gifcp->iftc_classify) ( gifcp, mh, pf, nethdrp, validlen );
 /* ??? else "flow 0" */
 }
 mh->m_tcflowp = flowp;
#endif TRAFFIC_CONTROL

 /* Enqueue packet for output & start driver */

 oldpri = splimp ();
 {
#ifdef TRAFFIC_CONTROL
 if ( key NE MT_TCDATA )  /* *** update mbuf usage stats now */
 {
  mbstat.m_mtypes[key]--;
  mbstat.m_mtypes[MT_TCDATA]++;
 }

 if ( (gifcp->iftc_enforce EQ (int (*)()) NULL)
  OR ((validlen = (*gifcp->iftc_enforce)
   ( gifcp, mh, len, (struct timeval *) NULL)) EQ -2) )
 {
  /* Packet not subject to traffic control */
```

```
    if ( validlen EQ -2 )
    {
    mh->m_type = MT_DATA; /* No-sort */
    mbstat.m_mtypes[ MT_DATA ]++;
    mbstat.m_mtypes[ MT_TCDATA ]--;
    }
    validlen = 0;

    if ( IF_QFULL( &( acp->ac_if.if_snd ) ) )
    {
    /* Queue is full, make room for this packet */

    ether_error ( acp, "WARNING: if_snd full" );
    /* drop a pkt */
    validlen = tcif_random_drop ( &( acp->ac_if.if_snd ) );
    }
   }

   /* Drop this packet */

   if ( validlen EQ -1 )
   {
    acp->ac_if.if_snd.ifq_drops++;
    /* update drop stats -- no good way to do keep accurate stats */
    /* bytes_dropped += len */
    /* bytes_queued += validlen - len */

    m_freem ( mh );
    splx ( oldpri );
    return ( ENOBUFS );  /* ??? better error */
   }

   if ( gifcp->iftc_nq NE (void (*)()) NULL )
   {
    (*gifcp->iftc_nq) ( gifcp, mh );
   }
   else
   {
    IF_ENQUEUE ( &( acp->ac_if.if_snd ), mh );
   }

   if ( validlen > 0 )
   {
    acp->ac_if.if_snd.ifq_drops++;
    /* update drop stats -- no good way to do keep accurate stats */
    /* bytes_dropped += len */
    /* bytes_queued += validlen - len */

    /* don't want to return an error as it may have higher layer
    ramifications for pkt instead of what was dropped */
   }
  #else !TRAFFIC_CONTROL
```

```
 if ( IF_QFULL( &( acp->ac_if.if_snd ) ) )
 {
  /* Queue is full, make room for this packet */
  ether_error ( acp, "WARNING: if_snd full" );

  len = tcif_random_drop ( &( acp->ac_if.if_snd ) ); /* drop a pkt */
  if ( len > 0 )
  {
  acp->ac_if.if_snd.ifq_drops++;
  /* update drop stats -- no good way to do keep accurate stats */
  /* bytes_dropped += len */
  /* bytes_queued += validlen - len */
  }
 }

 /* append new paacket */

 IF_ENQUEUE ( &( acp->ac_if.if_snd ), mh );
#endif TRAFFIC_CONTROL

 (*fnc_start) ( acp->ac_if.if_unit ); /* (re)start device output */
 }
 splx ( oldpri );

 return ( 0 );
}


#if NIE

/*
S* tcif_ieoutput ( ifp, pktp, sockaddrp )
s*
s* Routine called to send packets via the ethernet.
 *
 * Just because we don't have source to be able to recoompile
 * ether_output with a different IF_ENQUEUE macro in <net/if.h>
 * & fix randomdrop.
s*
 * Called via ifnet if_output dispatch.
 *
 */

extern void    iestartout ();


 /*static*/ int
tcif_ieoutput( acp, pktp, sockaddrp )
 struct arpcom  *acp;
 struct mbuf  *pktp;
 struct sockaddr *sockaddrp;
{
```

```
 return ( tcif_ether_output( acp, pktp, sockaddrp, iestartout ) );
}

#endif NIE


#if NLE

/*
S* tcif_leoutput ( ifp, pktp, sockaddrp )
s*
s* Routine called to send packets via the ethernet.
 *
 * Just because we don't have source to be able to recoompile
 * ether_output with a different IF_ENQUEUE macro in <net/if.h>
 * & fix randomdrop.
s*
 * Called via ifnet if_output dispatch.
 *
 */

extern void    lestart ();


 /*static*/ int    /* Next bugid 0x */
tcif_leoutput( acp, pktp, sockaddrp )
 struct arpcom  *acp;
 struct mbuf  *pktp;
 struct sockaddr *sockaddrp;
{
 return ( tcif_ether_output( acp, pktp, sockaddrp, lestart ) );
}

#endif NLE
#endif WANT_ETHERNET


/*
S* tcif_ifp2gifcp ( osifhandle ) from IF_Ext macro
s*
s* Returns the Generic Network Interface pointer corresponding to
s* the given OS handle (BSD: (struct ifnet *)).
s*
 */

 struct aNetIf *   /* Next bugid 0x71102 */
tcif_ifp2gifcp( osifp )   /* from */ /* (macro) */
 caddr_t   osifp;
{
 static int   gave_warning = 0;
 struct aNetIf *agip = tcif_gifcheadp;
```

```
 /* ??? Ought to have faster algorithm, but only a few interfaces. */

 for ( ; agip NE NO_NETIFP ; agip = agip->ifc_nextp )
 {
 if ( (caddr_t) agip->osifcp EQ osifp )
  return ( agip );  /* Found match */

 if ( osifp EQ (caddr_t) agip )
 {    /* be forgiving */
  if ( gave_warning EQ 0 )
  {
  printf ( "tcif_ifp2gifcp: ifnetp instead of gifcp (%x)\n",
    agip );
  gave_warning++;
  }
  else if ( gave_warning < 0 )
  panic ( "tcif_ifp2gifcp: ifnetp instead of gifcp\n" );

  return ( agip );  /* Found match */
 }

 } /* end of for loop */

 return ( &( tfic_genifs.NetIfs[0] ) ); /* ??? Send to blackhole, timeout */
}


/*
S* tfic_init_gifcs ()
s*
s* Routine to initialize generic network interfaces.
s*
 */

 void
tfic_init_gifcs()    /* from ??? */
{
 char   *sp;
 int    left,
    len;
 struct aNetIf  *agip = &( tfic_genifs.NetIfs[0] ),
   **tailpp = &( tcif_gifcheadp );
 extern struct ifnet  *ifnet; /* BSD's global list of interfaces */
 struct ifnet  *osifp = ifnet; /* BSD's global list of interfaces */


 if ( tcif_gifcheadp NE (struct aNetIf *) NULL )
 return;    /* already done */

 /* Create a dummy interface to try and fail gracefully if something is
 * broken.
```

```
*/

bzero( (char *) agip, sizeof (struct aNetIf) );
agip->ifc_nextp = NO_NETIFP;
agip->osifcp = &( dummyif );
agip->lclhdrlen = sizeof (((struct sockaddr *)0)->sa_data);
agip->ifc_output = tcif_dummy_output;
(void) strcpy ( &( agip->namebuf[0] ), /*<-*/ "dummyif0" );
agip++;
tfic_genifs.nxtfree++;


/* Create a Generic Network Interface for each OS interface */

for ( ; osifp NE (struct ifnet *) NULL ; osifp = osifp->if_next )
{
/* FYI: ifnet->if_addrlist not yet valid */

/* Logical end of addresses per interface loop */

if ( (tfic_genifs.nxtfree +1) >= tfic_genifs.allocated )
{
 printf ( "Too many network interfaces (%s%u); make DEF_GENIFS larger\n",
  osifp->if_name, osifp->if_unit );
 break;   /* out for next interface */
}

agip->osifcp = osifp;
agip->lclhdrlen = sizeof (((struct sockaddr *)0)->sa_data);

sp = &( agip->namebuf[0] );
left = sizeof (agip->namebuf) - 2;
(void) strncpy ( sp, osifp->if_name, left );
len = strlen ( sp );
sp += len;
left -= len;
if ( left < 5 )
 *sp++ = '?';
else
{
 len = osifp->if_unit & 0xFFFF;
 if ( len >= 10000 )
 *sp++ = '0' + (len / 10000), len %= 10000;
 if ( len >= 1000 )
 *sp++ = '0' + (len / 1000), len %= 1000;
 if ( len >= 100 )
 *sp++ = '0' + (len / 100), len %= 100;
 if ( len >= 10 )
 *sp++ = '0' + (len / 10), len %= 10;
 *sp++ = '0' + len;
}
*sp = '\0';
```

```
#if WANT_ETHERNET
 /* Don't have source access to change:
  * 1) IF_ENQUEUE / IF_DEQUEUE macros in ether_output
  * nor 2) random_drop
  */
#if NLE
 if ( strcmp ("le",osifp->if_name) EQ 0 )
 {
  if ( ConfigFlag (TcIfFlgOwnLE) )
  {
  extern int leoutput (/* ifnetp, pktp, sockaddrp */);

  agip->lclhdrlen = sizeof (struct ether_header);
  if ( osifp->if_output EQ leoutput )
   osifp->if_output = tcif_leoutput;
  }
/*
cwl ??? default TrafficControl */
  agip->bw_conf = bws[ (int) ENET_10MB ].bw_conf;
  agip->bw_load = bws[ (int) ENET_10MB ].bw_load;
  agip->bw_resv = bws[ (int) ENET_10MB ].bw_resv;

  agip->rmf = tcif_rmvectors[ (int) RM_BCST ].rmf;
#ifdef VIRTUAL_CLOCK
 /* agip->tcf = tcif_tcvectors[ (int) TC_VC ].tcf; */
  agip->alg_next = (short) TC_VC;
#else !VIRTUAL_CLOCK
  agip->alg_next = (short) TC_RD;
#endif VIRTUAL_CLOCK
 }
#endif NLE
#if NIE
 if ( strcmp ("ie",osifp->if_name) EQ 0 )
 {
  if ( ConfigFlag (TcIfFlgOwnLE) )
  {
  extern int ieoutput (/* ifnetp, pktp, sockaddrp */);

  agip->lclhdrlen = sizeof (struct ether_header);
  if ( osifp->if_output EQ ieoutput )
   osifp->if_output = tcif_ieoutput;
  }
 /*
 cwl ??? default TrafficControl */
  agip->bw_conf = bws[ (int) ENET_10MB ].bw_conf;
  agip->bw_load = bws[ (int) ENET_10MB ].bw_load;
  agip->bw_resv = bws[ (int) ENET_10MB ].bw_resv;

  agip->rmf = tcif_rmvectors[ (int) RM_BCST ].rmf;
 /* agip->tcf = tcif_tcvectors[ (int) TC_VC ].tcf; */
  agip->alg_next = (short) TC_VC;
```

```
 }
#endif NIE
#endif WANT_ETHERNET

#if NHSIS
 if ( strcmp ("hsis",osifp->if_name) EQ 0 )
 {
  agip->lclhdrlen = PPP_HDRSPACE;

/* #ifdef DARTNET */
  if ( (osifp->if_unit % 4) EQ 0 )
/* #endif DARTNET */
  {
/*
cwl ??? default TrafficControl */
  agip->bw_conf = bws[ (int) HSIS_1344 ].bw_conf;
  agip->bw_load = bws[ (int) HSIS_1344 ].bw_load;
  agip->bw_resv = bws[ (int) HSIS_1344 ].bw_resv;

  agip->rmf = tcif_rmvectors[ (int) RM_P2P ].rmf;
#ifdef VIRTUAL_CLOCK
  /* agip->tcf = tcif_tcvectors[ (int) TC_VC ].tcf; */
  agip->alg_next = (short) TC_VC;
#else !VIRTUAL_CLOCK
  agip->alg_next = (short) TC_FIFO;
#endif VIRTUAL_CLOCK
  }
 }
#endif NHSIS

/*
cwl ??? do "lo", too? */

/*
cwl ??? fill in default dq/nq/enf */

 tcif_AlgSwitch( agip, /*quit*/0 );

 *tailpp = agip;
 tailpp = &( agip->ifc_nextp );

 agip++;
 tfic_genifs.nxtfree++;

 } /* end of all interfaces */

 *tailpp = &( tfic_genifs.NetIfs[0] );

 return;
}
```

```
int
tcif_ioctl( cmd, argdatap, ifp )
 int    cmd;
 caddr_t   argdatap;
 struct ifnet  *ifp;
{
 struct _ovrlay *datap = (struct _ovrlay *) argdatap;
 struct aNetIf *gifcp = tfic_dev2gifcp( datap->ifname );
 int    alg;
 int   (*ctl_func) ();
 int    i;
 char  *algp;
 struct aTCname *tcp;


 if ( (gifcp EQ (struct aNetIf *) NULL)
 OR (gifcp->osifcp NE ifp) )
 return ( ENODEV );


 /* Find current traffic control algorithm name */
 alg = (int) gifcp->iftc_alg;
 algp = "";
 tcp = &( tcif_TCnames[0] );
 for ( i = DimensionOf (tcif_TCnames) - 1 ; i >= 0 ; i--, tcp++ )
 {
 if ( (int) tcp->value != alg )
  continue;

 algp = &( tcp->name[0] );
 break;
 }


 /* Process change of traffic control algorithm */
 if ( cmd EQ SIOCGTCALG )
 {
 strncpy ( (char *) &( datap->data[0] ), /*<-*/ algp,
   sizeof (tcp->name) );
 return ( 0 );
 }

 if ( cmd EQ SIOCSTCALG )
 {
 int   oldpri;
 short   if_flags;
 extern int  suser ();


 /* Find entry for specified traffic control algorithm name */
```

```
 algp = (char *) &( datap->data[0] );
 alg = 99999;
 tcp = &( tcif_TCnames[0] );
 for ( i = DimensionOf (tcif_TCnames) - 1 ; i >= 0 ; i--, tcp++ )
 {
  if ( strncmp ( tcp->name, algp, sizeof (tcp->name) ) NE 0 )
  continue;

  alg = (int) tcp->value;
  break;
 }

 if ( alg EQ 99999 )
  return ( EINVAL );

 if ( suser() EQ 0 )
  return ( u.u_error );

 oldpri = splnet ();

 if ( ((if_flags = gifcp->osifcp->if_flags) & IFF_UP) NE 0 )
 {
 /* gifcp->osifcp->if_flags &= ~ IFF_UP; /* in if_down */
  (void) if_down ( gifcp->osifcp );
 }

 gifcp->alg_next = (short) alg;  /* Set desired algorithm */
 tcif_AlgSwitch( gifcp, /*quit*/0 );

 if ( (if_flags & IFF_UP) NE 0 )
  gifcp->osifcp->if_flags |= IFF_UP;

 splx ( oldpri );

 return ( 0 );
 }

 /* Process interface specific operations */
 if ( (ctl_func = gifcp->iftc_control) EQ ( int (*)()) NULL )
 return ( EOPNOTSUPP );

 return ( (*ctl_func) ( gifcp, cmd, datap, ((cmd >> 16) & _IOCPARM_MASK) ) );
}



/*
S* tcif_nq_func ( ifqp, mp )
s*
s* Convergence routine from SunOS IF_ENQUEUE to Traffic Control
s* functions. Arguments are address if ifnet's ifqueue and pointer
s* to packet. Assumes that enforcement has already been performed.
```

```
 *
s* Called at splimp (or higher, depending on driver).
s*
 */

 void
tcif_nq_func( ifqp, mp )
 struct ifqueue *ifqp;
 struct mbuf  *mp;
{
 struct aNetIf *gifcp;
 void  (*nqfuncp)();


 mp->m_act = (struct mbuf *) NULL;

 /* Map ifqueue pointer back to ifnet, then to extended ifnet (aNetIf) */

 gifcp = tcif_ifp2gifcp( Mkp (caddr_t,ifqp,
     (- OffsetOf (if_snd,struct ifnet))) );
 /* dummyif0 /nq ??? */

 /* If no specific enqueue function is supplied, use specified queue */

 if ( (nqfuncp = gifcp->iftc_nq) NE (void (*)()) NULL )
 (*nqfuncp) ( gifcp, mp ); /* Use specific enqueue function */
 else
 IF_ENQUEUE (ifqp, mp);

 return;
}


/*
S* tcif_random_drop ( ifqp )
s*
s* Routine called to randomly drop one of "un-regulatedd"
s* packets from interface output queue.
 *
 * Just because we don't have source.
s*
 * Called from tcif_ether_output.
 *
 */


 static int
tcif_random_drop( ifqp )  /* from tcif_ether_output */
 struct ifqueue *ifqp;
{    /* splimp or higher */
 int    skip;
 struct mbuf  *dp,  /* packet to drop */
```

```
    *pp = 0; /* ptr before dp, or NULL */


   if ( (dp = ifqp->ifq_head) NE (struct mbuf *) NULL )
   return ( 0 );

   skip = ifqp->ifq_len;

  #ifdef MT_TCDATA
   if ( dp->m_type EQ MT_TCDATA )
   {
   do
   {
    pp = dp;
    skip--;
   } while ( ((dp = pp->m_act) NE (struct mbuf *) NULL)
     AND (dp->m_type EQ MT_TCDATA) );

   if ( (skip <= 0) OR (dp EQ (struct mbuf *) NULL) )
    return ( 0 );
   }
  #endif MT_TCDATA

   /* Find which packet to drop */
   skip = (time.tv_sec * time.tv_usec) % skip;

   for ( ; (skip > 0) AND (dp NE (struct mbuf *) NULL) ; skip-- )
   {
  pp = dp;
  dp = pp->m_act;
   }

   if ( (skip <= 0) OR (dp EQ (struct mbuf *) NULL) )
  return ( 0 );

   if ( pp EQ (struct mbuf *) NULL )
   {     /* drop first packet */
  if ( (ifqp->ifq_head = dp->m_act) EQ (struct mbuf *) NULL )
    ifqp->ifq_tail = pp; /* I.e., NULL */
  }
  else
  {
  if ( (pp->m_act = dp->m_act) EQ (struct mbuf *) NULL )
    ifqp->ifq_tail = pp;
  }

  if ( dp->m_type EQ MT_TCDATA )
  skip = dp->m_tcrsrc;
  else
  {
  skip = dp->m_len;
  if ( (pp = dp->m_next) NE (struct mbuf *) NULL )
```

```
 do
 {
 skip += pp->m_len;
 } while ( (pp = pp->m_next) NE (struct mbuf *) NULL );
}

ifqp->ifq_len--;   /* update queue length */
m_freem ( dp );   /* free packets */

return ( skip );
}


/*
S* tcif_udp_usrreq ( sop, req, mp, namep, rightsp )
s*
s* This routine replaces the udp_usrreq entry in the IPPROTO_UDP
s* entry of inetsw[]. It is used to intercept the new traffic
s* control and resource management ioctls.
s*
 */

 int
tcif_udp_usrreq( sop, req, mp, namep, rightsp )
 struct socket  *sop;
 int    req;
 struct mbuf  *mp,
   *namep,
   *rightsp;
{
 if ( req NE PRU_CONTROL )
 return ( udp_usrreq( sop, req, mp, namep, rightsp ) );

 if ( (((int) mp >> 8) & 0xFF) EQ 'i' )
 {
 switch ( (int) mp )
 {
#ifdef MROUTING
 case SIOCADDMULTI:
 case SIOCDELMULTI:
#endif MROUTING
 case SIOCDARP:
 case SIOCGARP:
 case SIOCSARP:
 case SIOCGIFCONF:
 case SIOCGIFFLAGS:
 case SIOCSIFFLAGS:
 case SIOCGIFMETRIC:
 case SIOCSIFMETRIC:
 case SIOCLOWER:
 case SIOCSIFMTU:
 case SIOCSPROMISC:
```

```
 case SIOCUPPER:
#ifdef TRAFFIC_CONTROL
 case SIOCSIFBWCONFIG:
 case SIOCSIFBWPKTOVRHD:
 case SIOCSIFCOSTS:
 case SIOCSIFCPU:
 case SIOCSIFDELAY:
 case SIOCSIFDELAYVAR:
 case SIOCSIFERRORS:
 case SIOCSIFPROPDELAY:
#endif TRAFFIC_CONTROL
  return ( tcifioctl( sop, (int) mp, (caddr_t) namep ) );

#ifdef TRAFFIC_CONTROL

 case SIOCGTCALG: /* Read traffic control algorithm */
 case SIOCSTCALG: /* Set traffic control algorithm */

  return ( tcif_ioctl( (int) mp, (caddr_t) namep,
     (struct ifnet *) rightsp ) );
#endif TRAFFIC_CONTROL

 default:
  break;
 } /* end of switch */
 }

 return ( in_control( sop, mp, namep, rightsp ) );
}


/*
S* tcifioctl ( sop, cmd, datap )
s*
s* Similar to ifioctl ().
s*
 */

 int
tcifioctl( sop, cmd, datap )
 struct socket  *sop;
 int    cmd;
 caddr_t   datap;
{
 struct ifnet *ifp,
   *if2p;
 struct ifreq *ifrp;
 int    oldpri;
#ifdef TRAFFIC_CONTROL
 struct aNetIf *nifp;

 extern int  st2_usrreq (/*sop, req, mp, namep, rightsp*/);
```

```
#endif TRAFFIC_CONTROL

 switch ( cmd )
 {
 case SIOCGIFCONF: return ( tcifconf( cmd, datap ) );

 case SIOCSARP:
 case SIOCDARP: if ( suser() == 0 ) return ( u.u_error );
   /* Fall through */

 case SIOCGARP: return ( arpioctl( cmd, datap ) );
 } /* end of first cmd switch */

 ifrp = (struct ifreq *) datap;
 if ( (ifp = ifunit( ifrp->ifr_name, sizeof (ifrp->ifr_name) ))
 == (struct ifnet *) NULL )
 return ( ENXIO );

 switch ( cmd )
 {
 case SIOCGIFFLAGS: ifrp->ifr_flags = ifp->if_flags; break;

 case SIOCGIFMETRIC: ifrp->ifr_metric = ifp->if_metric; break;

 case SIOCSIFFLAGS: if ( suser() == 0 ) return ( u.u_error );
   if ( ((ifp->if_flags & IFF_UP) != 0)
    && ((ifrp->ifr_flags & IFF_UP) == 0) )
   {
    oldpri = splimp();
    if_down( ifp );
    splx( oldpri );
   }
   if ( ifp->if_snd.ifq_maxlen == 0 )
    ifp->if_snd.ifq_maxlen = ifqmaxlen;
   ifp->if_flags = (ifp->if_flags & IFF_CANTCHANGE)
     | (ifrp->ifr_flags & ~ IFF_CANTCHANGE);
   if ( ifp->if_ioctl != (int (*)()) NULL )
    (void) (*ifp->if_ioctl) (ifp, cmd, datap );
   break;

 case SIOCSIFMETRIC: if ( suser() == 0 ) return ( u.u_error );
   ifp->if_metric = ifrp->ifr_metric;
   break;

 case SIOCSIFMTU: if ( suser() == 0 ) return ( u.u_error );
   ifp->if_metric = * (unsigned int *) &( ifrp->ifr_data[0] );
   break;

#ifdef TRAFFIC_CONTROL
 case SIOCGTCALG: /* Fall through */
 case SIOCSTCALG:
   return ( st2_usrreq ( sop, PRU_CONTROL, cmd, datap,
```

```
    ifp ) );

 case SIOCSIFBWCONFIG:
  if ( ((nifp = IF_Ext(ifp)) == (struct aNetIf *)0) ||
   (nifp->rsrcifp == (struct aRsrcNet *)0) )
   return ( EINVAL );
  nifp->rsrcifp->bw_cnfg = ifrp->ifr_rsrcnet_req.parm1;
  nifp->rsrcifp->bw_load = ifrp->ifr_rsrcnet_req.parm2;
  nifp->rsrcifp->bw_resv = ifrp->ifr_rsrcnet_req.parm3;
  break;

 case SIOCSIFBWPKTOVRHD:
  if ( ((nifp = IF_Ext(ifp)) == (struct aNetIf *)0) ||
   (nifp->rsrcifp == (struct aRsrcNet *)0) )
   return ( EINVAL );
  nifp->rsrcifp->bw_pkt = ifrp->ifr_rsrcnet_req.parm1;
  break;

 case SIOCSIFCOSTS:
  if ( ((nifp = IF_Ext(ifp)) == (struct aNetIf *)0) ||
   (nifp->rsrcifp == (struct aRsrcNet *)0) )
   return ( EINVAL );
  nifp->rsrcifp->cost = ifrp->ifr_rsrcnet_req.parm1;
  nifp->rsrcifp->cost_pkt = ifrp->ifr_rsrcnet_req.parm2;
  nifp->rsrcifp->cost_byte = ifrp->ifr_rsrcnet_req.parm3;
  nifp->rsrcifp->cost_msec = ifrp->ifr_rsrcnet_req.parm4;
  break;

 case SIOCSIFCPU:
  if ( ((nifp = IF_Ext(ifp)) == (struct aNetIf *)0) ||
   (nifp->rsrcifp == (struct aRsrcNet *)0) )
   return ( EINVAL );
  nifp->rsrcifp->cpu_in_byte =
   ifrp->ifr_rsrcnet_req.parm1;
  nifp->rsrcifp->cpu_in_pkt =
   ifrp->ifr_rsrcnet_req.parm2;
  nifp->rsrcifp->cpu_out_byte =
   ifrp->ifr_rsrcnet_req.parm3;
  nifp->rsrcifp->cpu_out_pkt =
   ifrp->ifr_rsrcnet_req.parm4;
  break;

 case SIOCSIFDELAY:
  if ( ((nifp = IF_Ext(ifp)) == (struct aNetIf *)0) ||
   (nifp->rsrcifp == (struct aRsrcNet *)0) )
   return ( EINVAL );
  nifp->rsrcifp->dly_in = ifrp->ifr_rsrcnet_req.parm1;
  nifp->rsrcifp->dly_out = ifrp->ifr_rsrcnet_req.parm2;
  nifp->rsrcifp->dly_que = ifrp->ifr_rsrcnet_req.parm3;
  break;

 case SIOCSIFDELAYVAR:
```

```
  if ( ((nifp = IF_Ext(ifp)) == (struct aNetIf *)0) ||
   (nifp->rsrcifp == (struct aRsrcNet *)0) )
   return ( EINVAL );
  nifp->rsrcifp->dly_in_var =
   ifrp->ifr_rsrcnet_req.parm1;
  nifp->rsrcifp->dly_out_var =
   ifrp->ifr_rsrcnet_req.parm2;
  nifp->rsrcifp->dly_que_var =
   ifrp->ifr_rsrcnet_req.parm3;
  break;

 case SIOCSIFERRORS:
  if ( ((nifp = IF_Ext(ifp)) == (struct aNetIf *)0) ||
   (nifp->rsrcifp == (struct aRsrcNet *)0) )
   return ( EINVAL );
  nifp->rsrcifp->ber = ifrp->ifr_rsrcnet_req.parm1;
  nifp->rsrcifp->droprate = ifrp->ifr_rsrcnet_req.parm2;
  break;

 case SIOCSIFPROPDELAY:
  if ( ((nifp = IF_Ext(ifp)) == (struct aNetIf *)0) ||
   (nifp->rsrcifp == (struct aRsrcNet *)0) )
   return ( EINVAL );
  nifp->rsrcifp->dly_prop = ifrp->ifr_rsrcnet_req.parm1;
  break;
#endif TRAFFIC_CONTROL

 case SIOCUPPER: if2p = ifunit( ifrp->ifr_oname,
    sizeof (ifrp->ifr_oname) );
  if ( if2p == (struct ifnet *) NULL )
   return ( ENXIO );
  if ( if2p->if_input == (int (*)()) NULL )
   return ( EINVAL );
  ifp->if_upper = if2p;
  break;

 case SIOCLOWER: if2p = ifunit( ifrp->ifr_oname,
    sizeof (ifrp->ifr_oname) );
  if ( if2p == (struct ifnet *) NULL )
   return ( ENXIO );
  if ( if2p->if_output == (int (*)()) NULL )
   return ( EINVAL );
  ifp->if_lower = if2p;
  break;

 case SIOCSPROMISC: if ( suser() == 0 ) return ( u.u_error );
  return ( ifpromisc( ifp, *(int *) datap ) );

/* #ifdef MROUTING */
 case SIOCADDMULTI:
 case SIOCDELMULTI: if ( suser() == 0 ) return ( u.u_error );
  /* Fall through */
```

```
/* #endif MROUTING */

 default:  if ( sop->so_proto == (struct protosw *) NULL )
    return ( EOPNOTSUPP );
   return ( (*sop->so_proto->pr_usrreq) ( sop, PRU_CONTROL,
       cmd, datap, ifp ) );
 } /* end of second cmd switch */

 return ( 0 );
}



 int
tcifconf( cmd, datap )
 int    cmd;
 caddr_t   datap;
{
 struct ifnet *ifp = ifnet;
 struct ifconf *icp = (struct ifconf *) datap;
 int    left = icp->ifc_len,
    retcod = 0;
 char  *cp,
   *up = (char *) icp->ifc_buf;
 struct ifreq  rsp;
#if IF_NADDR
 char  *ep = &( rsp.ifr_name[0] ) + sizeof (rsp.ifr_name) - 5;
    /* 4 = 3 digit unit + 1 '\0' + idx */
#else !IF_NADDR
 char  *ep = &( rsp.ifr_name[0] ) + sizeof (rsp.ifr_name) - 4;
    /* 4 = 3 digit unit + 1 '\0' */
#endif IF_NADDR
 struct ifaddr *ifap;


 for ( ; (left > sizeof (struct ifreq)) && (ifp != (struct ifnet *) NULL)
  ; ifp = ifp->if_next )
 {
 bcopy( ifp->if_name, &( rsp.ifr_name[0] ), sizeof (rsp.ifr_name) - 2 );

/* #ifndef sun3
 * for ( cp = &( rsp.ifr_name[0] ) ; cp < ep ; cp++ )
 * {
 *  if ( *cp == '\0' )
 *  break;
 * }
 * #else /* sun3 */
 cp = &( rsp.ifr_name[0] );
 while ( (cp < ep) && (*cp != '\0') )
  cp++;
/* #endif */
 if ( ifp->if_unit > 99 )
```

```
  *cp++ = (ifp->if_unit / 100) + '0';

 if ( ifp->if_unit > 9 )
  *cp++ = ((ifp->if_unit % 100) / 10) + '0';

 *cp++ = (ifp->if_unit % 10) + '0';
#if IF_NADDR
 *cp++ = '\0';   /* point to byte for address index */
 *cp = 0;
#else !IF_NADDR
 *cp = '\0';
#endif IF_NADDR

 if ( (ifap = ifp->if_addrlist) == (struct ifaddr *) NULL )
 {
  bzero( (char *) &( rsp.ifr_addr ), sizeof (rsp.ifr_addr) );

  if ( (retcod = copyout( (char *) &( rsp ), up, sizeof (rsp) ))
  != 0 )
  break;

  left -= sizeof (rsp);
  up += sizeof (rsp);
 }
 else
 {
  for ( ; (left > sizeof (rsp)) && (ifap != (struct ifaddr *) NULL)
   ; ifap = ifap->ifa_next )
  {
  rsp.ifr_addr = ifap->ifa_addr;
#if IF_NADDR
  *cp += 1;
#endif IF_NADDR
  if ( (retcod = copyout( (char *) &( rsp ), up, sizeof (rsp) ))
   != 0 )
   break;
  left -= sizeof (rsp);
  up += sizeof (rsp);
  } /* end of returning all of interface's addresses loop */
 }
 } /* end of processing all interfaces loop */

 icp->ifc_len -= left;

 return ( retcod );
}


#ifdef DOCUMENTATION
/*
S* tcif_output_func ( ifp, mp, dstp )
s*
```

```
s* Example of a driver output routine using Traffic Control
s* functions.
 *
s* Called at splimp (or higher, depending on driver).
s*
 */

 int
tcif_output_func ( ifp, mp, dstp )
 struct ifnet  *ifp;
 struct mbuf  *mp;
 struct sockaddr *dstp;
{
 struct aNetIf *gifcp;
 caddr_t   flowp,
   nethdrp;
 int    validlen;
 unsigned long  key,
   rsrc;
 unsigned short  pf;
 struct mbuf  *mhp = mp;
 struct local_header *lhp;
 void  (*nqfuncp)();


 if ( (ifp->if_flags & (IFF_RUNNING | IFF_UP))
 NE (IFF_RUNNING | IFF_UP) )
 {
 IF_DROP ( &( ifp->ifc_snd ) );
 /* update drop stats -- no good way to do keep accurate stats */
** /* bytes_dropped += len */

 m_freem ( mp );
 return ( ENETDOWN );
 }

 /* Map ifqueue pointer back to ifnet, then to extended ifnet (aNetIf) */

 /* Mkp ( unsigned long, ifqp, (- OffsetOf (if_snd,struct ifnet)) ) */
 gifcp = tcif_ifp2gifcp( ifp );  /* dummyif0 /nq ??? */

 pf = PF_UNSPEC;
 nethdrp = mtod ( mp, caddr_t);
 validlen = mp->m_len;

 if ( mp->m_type EQ MT_TCDATA )
 {
 flowp = mp->m_tcflowp;
 rsrc = mp->m_tcrsrc;
 key = mp->m_tckey;
 }
 else
```

```
{
flowp = (caddr_t) NULL;
rsrc = 0;
key = 0;
}

/* Map protocol specific address in sockaddr to local network address */

/* ...
pf = ...
*/

/* Find space for local network header */

if ( (mp->m_off < (unsigned long) ( OffsetOf (m_tcdat[0],struct mbuf)
    + sizeof (struct local_header)))
OR ( M_HASCL( mp )
#ifdef MCL_STATIC_HDR
  AND (mp->m_cltype NE MCL_STATIC_HDR)
#endif MCL_STATIC_HDR
  ) )  /* no room in first mbuf, prepend another */
{
mhp = (struct mbuf *) m_get ( M_DONTWAIT, MT_TCDATA );
if ( ! mhp )
{
  IF_DROP ( &( ifp->ifc_snd ) );
  /* update drop stats -- no good way to do keep accurate stats */
**  /* bytes_dropped += len */

  m_freem ( mp );
  return ( ENOBUFS );
}

    /* XXX 8 => mod x10 aligned */
mhp->m_off = OffsetOf (m_tcdat[8],struct mbuf);
mhp->m_next = pktp;  /* prepend */
mhp->m_len = sizeof ( struct local_header );
}
else    /* insert header */
{
mhp->m_off -= sizeof ( struct local_header );
mhp->m_len += sizeof ( struct local_header );
}

/* Construct local header */

lhp = mtod ( mhp, struct local_header *);
/* ... */


len = mhp->m_len;
if ( (mp = mhp->m_next) NE (struct mbuf *) NULL )
```

```
 do
 {
  len += mp->m_len;
** } while ( (mp = mp->m_next) NE (struct mbuf *) NULL );

 mhp->m_tcrsrc = rsrc;
 mhp->m_tckey = key;

 key = mhp->m_type;   /* *** mbuf stats update below */
 mhp->m_type = MT_TCDATA;

 if ( flowp EQ (caddr_t) NULL )
 {
 if ( gifcp->iftc_classify NE (caddr_t (*)()) NULL )
 {
  flowp = (*gifcp->iftc_classify) ( gifcp, mhp, pf,
      nethdrp, validlen );

#ifdef lint
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
  flowp = clsfy ( gifcp, mhp, pf, nethdrp, validlen );
  aTcVectorList;
#undef aTV
#endif lint
 }
 /* ??? else "flow 0" */
 }
 mhp->m_tcflowp = flowp;

 /* Enqueue packet for output & start driver */

 oldpri = splimp ();
 {
 if ( key NE MT_TCDATA )  /* *** update mbuf usage stats now */
 {
  mbstat.m_mtypes[key]--;
  mbstat.m_mtypes[MT_TCDATA]++;
 }

 if ( (gifcp->iftc_enforce EQ (int (*)()) NULL)
  OR ((validlen = (*gifcp->iftc_enforce)
   ( gifcp, mhp, len, (struct timeval *) NULL)) EQ -2) )
 {
  if ( validlen EQ -2 ) /* No enforcing */
  {
 mhp->m_type = PKT_DATA;
 mbstat.m_mtypes[ PKT_DATA ]++;
 mbstat.m_mtypes[ PKT_TCDATA ]--;
  }
```

```
 validlen = 0;
 if ( IF_QFULL( &( ifp->if_snd ) ) )
 {
 /* Queue is full */ /* drop a pkt */
 validlen = tcif_random_drop ( &( ifp->if_snd ) );
 }
}

if ( validlen EQ -1 )  /* drop this packet */
{
 IF_DROP ( &( ifp->ifc_snd ) );
 /* update drop stats -- no good way to do keep accurate stats */
 /* bytes_dropped += len */
 /* bytes_queued += validlen - len */

 m_freem ( mhp );
 splx ( oldpri );
 return ( ENOBUFS );
}

if ( gifcp->iftc_nq NE (void (*)()) NULL )
{
 (*gifcp->iftc_nq) ( gifcp, mhp );
}
else
{
 IF_ENQUEUE ( &( fip->if_snd ), mhp );
}

if ( validlen > 0 )
{
 IF_DROP ( &( ifp->ifc_snd ) );
 /* update drop stats -- no good way to do keep accurate stats */
 /* bytes_dropped += len */
 /* bytes_queued += validlen - len */

 /* don't want to return an error as it may have higher layer
 ramifications for pkt instead of what was dropped */
}

start_output ( ifp->if_unit ); /* (re)start device output */
}
splx ( oldpri );

return ( 0 );
}
#endif DOCUMENTATION
```

```
#ifndef _ST2_RESOURCE_H_
#define _ST2_RESOURCE_H_
#ifndef lint
static char rcsid_st2_resource_h[] = "\
@(#) $Header: st2_resource.h,v 1.98+ 93/04/08 18:00:00 clynn Exp $ \n";
/*--------------------------------------------------------------------*
 | Copyright (c) 1991-1993 by BBN Systems and Technologies,           |
 | A Division of Bolt Beranek and Newman Inc.                         |
 |                                                                    |
 | Permission to use, copy, modify, distribute, and sell this         |
 | software and its documentation for any purpose is hereby           |
 | granted without fee, provided that the above copyright notice      |
 | and this permission appear in all copies and in supporting         |
 | documentation, and that the name of Bolt Beranek and Newman        |
 | Inc. not be used in advertising or publicity pertaining to         |
 | distribution of the software without specific, written prior       |
 | permission. BBN makes no representations about the suitability     |
 | of this software for any purposes. It is provided "AS IS"          |
 | without express or implied warranties.                             |
 *--------------------------------------------------------------------*/
#endif lint


/*
M* st2_resource.h Definitions for ST-II Resource Management.
M*
 */

/*
m* Status:
m* Features:
m* Untested Features:
m* Restrictions/Bugs:
m* Things to do:
m*
 */


/* Module Revision History
 *
 * bugid 9,0
 * $Log: st2_resource.h,v $
 * Revision 1.11 92/04/03 18:31:35 clynn
 * Release for DARTNet. Virtual Clock enforcement & related changes.
 *
 * Revision 1.10 91/11/26 23:12:48 clynn
 * Fixed typo.
 *
 * Revision 1.9 91/11/04 09:23:00 clynn
 * Updated for Public Domain Release. Major changes: addition of Source
 * Routing, IP Encapsulation, HELLO protocol between neighbors, tracking
 * of neighbors, detection of component or agent failures & notification
```

```
 * to applications. More consistant naming and format, including making
 * all external names begin with "st2_". Moved some routines and data
 * structures to reduce external references.
 *
 * Minor documentation changes.
 *
 * Revision 1.8 91/05/28 16:17:20 clynn
 * New features: Add new targets from Application layer, UserData support,
 * basic bandwidth reservation for point-to-point links, more complete
 * state tables, added pcode parameter to InitaFlowSpec3, extended packet
 * buffer abstraction, added network interface abstraction; ststat utility.
 * Bug fixes: ADDR IN USE problem, data send problem, causes of some
 * crashes, cleanup of protocol control blocks.
 * Work around: DARTNET receive memory leak.
 * Eliminated several small modules to reduce globals; adeded Makefile.
 *
 * Revision 1.1 91/03/15 18:33:08 clynn
 * Initial revision
 */


#ifndef TRAFFIC_CONTROL

/* def/name/aloc getid probe relid rlse */

#define aRmVectorList       \
 aRV (P2P,point-to-point,tcif_PtpAlloc,NoRmCtrl,tcif_RsIdGet,  \
  NoRmProbe,tcif_RsIdRel,tcif_PtpRlse)   \
 aRV (BCST,broadcast,tcif_PtpAlloc,NoRmCtrl,tcif_RsIdGet,  \
  NoRmProbe,tcif_RsIdRel,tcif_PtpRlse)   \
 aRV (NUN,,NoRmAlloc,NoRmCtrl,NoRmIdGet,NoRmProbe,NoRmIdRel,NoRmRlse) \


/* def/name/classify clockfast control dq drain enforce init nq quit
 aloc ctrl getid probe relid rlse */

#ifdef FAIR_SHARE
#define MAYBE_FAIR_SHARE      \
 aTV (FS1,fair-share,fs_classify_func,fs_clockfast_func,fs_control_func,\
  gen_dq_func,NoDrain,fs_enforce_func,fs_init_func, \
  fs_nq1q_func,NoQuit,     \
  fs_aloc_func,NoRsrcCtrl,NoRsrcGetid,NoRsrcProbe, \
  NoRsrcRelid,fs_rlse_func)
#else
#define MAYBE_FAIR_SHARE
#endif FAIR_SHARE


#ifdef SFQ
#define MAYBE_SFQ       \
 aTV (SFQ,sfq,NoClassify,NoClockfast,NoControl,NoDq,NoDrain,  \
```

```
  NoEnforce,NoInit,NoNq,NoQuit,    \
  NoRsrcAlloc,NoRsrcCtrl,NoRsrcGetid,NoRsrcProbe,  \
  NoRsrcRelid,NoRsrcRlse)
#else
#define MAYBE_SFQ
#endif SFQ


#ifdef VIRTUAL_CLOCK
#define MAYBE_VIRTUAL_CLOCK     \
 aTV (VC,vc,NoClassify,NoClockfast,NoControl,gen_dq_func,NoDrain, \
  vc_enf_func,vc_init_func,vc_nq_func,NoQuit,  \
  vc_aloc_func,NoRsrcCtrl,NoRsrcGetid,NoRsrcProbe, \
  NoRsrcRelid,NoRsrcRlse)
#else
#define MAYBE_VIRTUAL_CLOCK
#endif VIRTUAL_CLOCK

   /* %^*% cpp! no #ifdefs in a #define */
#define aTcVectorList      \
 aTV (FIFO,fifo,NoClassify,NoClockfast,NoControl,gen_dq_func,NoDrain, \
  gen_enforce_func,NoInit,gen_nq_func,NoQuit,  \
  NoRsrcAlloc,NoRsrcCtrl,NoRsrcGetid,NoRsrcProbe,  \
  NoRsrcRelid,NoRsrcRlse)     \
 MAYBE_FAIR_SHARE      \
 aTV (RD,random-drop,NoClassify,NoClockfast,NoControl,gen_dq_func, \
  NoDrain,gen_enforce_func,NoInit,gen_nq_func,NoQuit, \
  NoRsrcAlloc,NoRsrcCtrl,NoRsrcGetid,NoRsrcProbe,  \
  NoRsrcRelid,NoRsrcRlse)     \
 MAYBE_SFQ       \
 MAYBE_VIRTUAL_CLOCK      \
 /* This must be last -- it has the largest enum value */  \
 aTV (NUN,,gen_classify_func,gen_clockfast_func,gen_control_func, \
  gen_dq_func,gen_drain_func,gen_enforce_func,  \
  gen_init_func,gen_nq_func,gen_quit_func,  \
  NoRsrcAlloc,NoRsrcCtrl,NoRsrcGetid,NoRsrcProbe,  \
  NoRsrcRelid,NoRsrcRlse)


enum RM_Strategy {
#define aRV(id,name,aloc,ctrl,idget,probe,idrel,rlse) Ident(RM_)id,
 aRmVectorList
#undef aRV
};


enum TC_Algorithm {
 /* @#$% cpp is too primative to allow \ in the formal parameter list */
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 Ident(TC_)id,
```

```
 aTcVectorList
#undef aTV
};


#ifdef lint

#define NoRmAlloc LintRmAlloc
#define NoRmCtrl LintRmCtrl
#define NoRmIdGet LintRmIdGet
#define NoRmProbe LintRmProbe
#define NoRmIdRel LintRmIdRel
#define NoRmRlse LintRmRlse

#define NoClassify LintClassify
#define NoClockfast LintClockfast
#define NoControl LintControl
#define NoDq  LintDq
#define NoDrain  LintDrain
#define NoEnforce LintEnforce
#define NoInit  LintInit
#define NoNq  LintNq
#define NoQuit  LintQuit
#define NoRsrcAlloc LintRsrcAlloc
#define NoRsrcCtrl LintCtrl
#define NoRsrcGetid LintRsrcGetid
#define NoRsrcProbe LintRsrcProbe
#define NoRsrcRelid LintRsrcRelid
#define NoRsrcRlse LintRsrcRlse

#else !lint

#define NoRmAlloc ((int (*)()) 0)
#define NoRmCtrl ((int (*)()) 0)
#define NoRmIdGet ((int (*)()) 0)
#define NoRmProbe ((int (*)()) 0)
#define NoRmIdRel ((int (*)()) 0)
#define NoRmRlse ((int (*)()) 0)

#define NoClassify ((caddr_t (*)()) 0)
#define NoClockfast ((void (*)()) 0)
#define NoControl ((int (*)()) 0)
#define NoDq  ((struct mbuf *(*)()) 0)
#define NoDrain  ((void (*)()) 0)
#define NoEnforce ((int (*)()) 0)
#define NoInit  ((int (*)()) 0)
#define NoNq  ((void (*)()) 0
#define NoQuit  ((void (*)()) 0)
#define NoRsrcAlloc ((int (*)()) 0)
#define NoRsrcCtrl ((int (*)()) 0)
#define NoRsrcGetid ((int (*)()) 0)
#define NoRsrcProbe ((int (*)()) 0)
```

```
#define NoRsrcRelid ((int (*)()) 0)
#define NoRsrcRlse ((int (*)()) 0)

#endif lint


#define aRV(id,name,aloc,ctrl,idget,probe,idrel,rlse)  \
extern int aloc (), ctrl (), idget (), \
  probe (), idrel (), rlse ();
 aRmVectorList
#undef aRV


#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
extern caddr_t clsfy (); extern void fsclk (); \
extern int cntrl (); extern struct mbuf *dq (); \
extern void drain (); extern int enfrc (); \
extern int init (); extern void nq ();  \
extern void quit ();     \
extern int aloc (), ctrl (), idget (), \
  probe (), idrel (), rlse ();
 aTcVectorList
#undef aTV

#endif TRAFFIC_CONTROL

#endif _ST2_RESOURCE_H_
```

```
#ifndef lint
static char rcsid[] =
 "@(#)$Header: hsis.c,v 1.98 1.98 93/03/24 00:00:00 clynn Rel $";
static char copyright[] =
 "Copyright (c) 1990 Regents of the University of California";
#endif

/*
 * Copyright (c) 1990 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that the above copyright notice and this paragraph are
 * duplicated in all such forms and that any documentation,
 * advertising materials, and other materials related to such
 * distribution and use acknowledge that the software was developed
 * by the University of California, Lawrence Berkeley Laboratory,
 * Berkeley, CA. The name of the University may not be used to
 * endorse or promote products derived from this software without
 * specific prior written permission.
 * THIS SOFTWARE IS PROVIDED ''AS IS'' AND WITHOUT ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
 * WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 */

#include "hsis.h"
#if NHSIS > 0
#include <sys/param.h>
#include <sys/mbuf.h>
#include <sys/socket.h>
#include <sys/socketvar.h>
#include <sys/errno.h>
#include <sys/systm.h>
#include <net/if.h>
#include <netinet/in.h>
#include <netinet/in_var.h>
#include <net/netisr.h>
#include <sys/stream.h>
#include <sys/ttycom.h>
#include <sys/tty.h>
#include <sys/time.h>

#include <sys/sockio.h>
#include <sundev/zsreg.h>
#include <sundev/mbvar.h>
#include <sun4c/intreg.h>

#include <sun4c/auxio.h>

#include "bpfilter.h"
#if NBPFILTER > 0
#include <net/bpf.h>
```

```
#endif

#include "z16c35.h"
#include "hsisreg.h"
#include "syncmode.h"
#include "syncstat.h"
#include "codecmode.h"
#include "hsiscom.h"
#include "ppp.h"

#ifndef AF_RAWSYNC
#define AF_RAWSYNC AF_IMPLINK
#endif

int hsidebug = 0;
extern int hz;
#ifdef HSIS_TRACE
int hsis_trace_lock = 0x7FFFFFFF;
#endif
#define dprintf(x) if(hsidebug)printf x;

/*
 * Like most things that have to deal with Zilog chips, this driver
 * requires two interrupt levels: a hardware level that should be as
 * high as possible (SBus level 5-7) and a software level that should
 * be at or below splimp (i.e., at level 4 or 6 on a sparc). Very little
 * time is spent at hardware interrupt level. Packet copies to and
 * from the board (<300 us, worst case) and almost all the device and
 * system stuff (20-100 us typically, 300 us if we have to call Sun's
 * incredibly slow Streams NIT) is done at the software interrupt level.
 */
#define splboard spl4
#define HARDINT_LEVEL 5

/*
 * Software interrupt level, state and status bits.
 */
#define SOFTINT_LEVEL 4
#define splsoft spl2

u_int hsis_isum;
#define RECV_DONE 1
#define XMIT_DONE 2
#define XMITDMA_DONE 4
#define SOFTINT(softc, event) (hsis_isum |= 1 << (sc)->sc_if.if_unit, \
    (sc)->sc_event |= (event))

/*
 * default set-up for scc (possibly modified by user-specified 'syncmode').
 * see hsis_init for how this is interpreted. Each entry is a particular
 * scc reg. The scc requires register be set in a particular order and
 * not all bits (in particular, the rx & tx enables) can be loaded from
```

```
 * here (see hsis_init). Be careful when changing this array.
 */
u_char hsis_scc_setup[] = {
 /* 0 */ 0,
#ifdef HSIS_EXTERNAL_RCVDONE
 /* 1 */ ZSWR1_SIE,
#else
 /* 1 */ ZSWR1_SIE | ZSWR1_RIE_SPECIAL_ONLY,
#endif
 /* 2 */ 0,
 /* 3 */ ZSWR3_RXCRC_ENABLE | ZSWR3_RX_8,
 /* 4 */ ZSWR4_SDLC,
 /* 5 */ ZSWR5_TXCRC_ENABLE | ZSWR5_RTS | ZSWR5_TX_8,
 /* 6 */ 0,
 /* 7 */ ZSWR7_SDLCFLAG,
 /* 8 */ 0,
 /* 9 */ ZSWR9_MASTER_IE | ZSWR9_NO_VECTOR | ZSWR9_VECTOR_INCL_STAT,
 /*10 */ ZSWR10_PRESET_ONES,
 /*11 */ 0,
 /*12 */ 0,
 /*13 */ 0,
 /*14 */ 0,
 /*15 */ ZSR15_SDLC_FIFO_ENA | ZSR15_TX_UNDER,
};

u_char hsis_scc_codec_setup[] = {
 /* 0 */ 0,
 /* 1 */ ZSWR1_SIE,
 /* 2 */ 0,
 /* 3 */ ZSWR3_HUNT | ZSWR3_RX_8,
 /* 4 */ 0,
 /* 5 */ ZSWR5_RTS | ZSWR5_TX_8,
 /* 6 */ 0,
 /* 7 */ 0,
 /* 8 */ 0,
 /* 9 */ ZSWR9_MASTER_IE | ZSWR9_NO_VECTOR | ZSWR9_VECTOR_INCL_STAT,
 /*10 */ 0,
 /*11 */ 0,
 /*12 */ 0,
 /*13 */ 0,
 /*14 */ 0,
 /*15 */ ZSR15_SYNC, /* XXX ZSR15_TX_UNDER needed for CLI? */
};

/*
 * default 'syncmode' (clock and loopback options) for scc.
 */
struct syncmode hsis_default_sm = {
 TXC_IS_BAUD, RXC_IS_BAUD, 1, 0, 9600, 0, 0
};
```

```
int hsisidentify(), hsisattach();
struct dev_ops hsis_ops = {
 1,
 hsisidentify,
 hsisattach,
};

int nhsisboard;   /* total # of hsis boards found */

struct hsiscom *hsiscom;
struct hsiscom *hsiscom_end;
struct hsis_softc *hsis_softc;
struct hsis_softc *hsis_softc_end;

int hsis_init(), hsis_output(), hsis_ioctl(), hsis_reset(), hsis_watchdog(),
 hsisintr(), hsissoftint();
void hsis_start();
void rawsyncinit();
void hsis_dmarecv_time();
void hsis_recv_done();

void
hsis_scc_attach(unit, hs, addr)
 register u_char unit;
 register struct hsiscom *hs;
 register u_char *addr;
{
 static int nhsischan;
 register struct zscc *zs = &hs->hs_zs[unit];
 register struct zdma *zd = &hs->hs_zd[unit >> 1];
 register u_char *zsdev = addr;
 register struct hsis_softc *sc = &hsis_softc[nhsischan];
 register struct ifnet *ifp = &sc->sc_if;

 zs->zs_unit = unit;
 zs->zs_addr = zsdev;

 /* disable rx, tx and interrupts. */
 SCC_WRITE(1, 0);
 SCC_WRITE(3, 0);
 SCC_WRITE(5, 0);

 /* disable external rcv done interrupt */
 hs->hs_board[unit + HSIS_EINT_ENA_A] = 0;

 /* set up the softc and make us known to the network code. */
 sc->sc_zs = zs;
 sc->sc_zd = zd;
 sc->sc_hs = hs;
 sc->sc_sm = hsis_default_sm;
 sc->sc_fifop = sc->sc_fifo;
 ifp->if_name = "hsis";
```

```
 ifp->if_unit = nhsischan++;
 ifp->if_mtu = HSIS_MTU;
#ifdef MULTICAST
 ifp->if_flags = IFF_POINTOPOINT | IFF_MULTICAST;
#else
 ifp->if_flags = IFF_POINTOPOINT;
#endif MULTICAST
 ifp->if_init = hsis_init;
 ifp->if_output = hsis_output;
 ifp->if_ioctl = hsis_ioctl;
 ifp->if_reset = hsis_reset;
 ifp->if_watchdog = hsis_watchdog;
 ifp->if_snd.ifq_maxlen = HSIS_MAX_SENDQ;
#if NBPFILTER > 0
 bpfattach(&sc->sc_bpf, ifp, DLT_PPP, PPP_HDRSPACE);
#endif
 if_attach(ifp);
}

void
hsis_dma_attach(unit, zd, addr)
 register u_char unit;
 register struct zdma *zd;
 register u_char *addr;
{
 register u_char *zddev = addr;

 zd->zd_addr = zddev;

 /*
  * Reset the dma section. Clear out the 'interrupt vector' address
  * so we get a simple value to switch on when interpreting intr.
  * Set the DCR to increment addresses rather than decrementing.
  * Set the ICR to enable master interrupt but leave individual
  * channels disabled (any HDLC event that would generate a dma intr
  * also generates an scc intr -- we waste enough time dealing with
  * this stupid chip and don't need to double all the interrupts),
  * don't put a vector on the bus when req. intr but do include
  * 'status' in the IVR on intr.
  * Zero out the dma address registers so we don't have to write
  * the high bytes when switching buffers.
  */
 DMA_WRITE0(ZSDMA_CCAR_DMA_RESET);
 DMA_WRITE0(ZSDMA_CCAR_RESET_IUS);
 DMA_WRITE(ZSDMA_IVR, 0);
 DMA_WRITE(ZSDMA_ICSR, 0);
 DMA_WRITE(ZSDMA_DCR, 0xf);
 DMA_WRITE(ZSDMA_ICR, ZSDMA_ICR_NV | ZSDMA_ICR_VIS | ZSDMA_ICR_MIE);

 DMA_WRITE(ZSDMA_RDARA, 0);
 DMA_WRITE(ZSDMA_RDARA+1, 0);
 DMA_WRITE(ZSDMA_RDARA+2, 0);
```

```
 DMA_WRITE(ZSDMA_RDARA+3, 0);

 DMA_WRITE(ZSDMA_TDARA, 0);
 DMA_WRITE(ZSDMA_TDARA+1, 0);
 DMA_WRITE(ZSDMA_TDARA+2, 0);
 DMA_WRITE(ZSDMA_TDARA+3, 0);

 DMA_WRITE(ZSDMA_RDARB, 0);
 DMA_WRITE(ZSDMA_RDARB+1, 0);
 DMA_WRITE(ZSDMA_RDARB+2, 0);
 DMA_WRITE(ZSDMA_RDARB+3, 0);

 DMA_WRITE(ZSDMA_TDARB, 0);
 DMA_WRITE(ZSDMA_TDARB+1, 0);
 DMA_WRITE(ZSDMA_TDARB+2, 0);
 DMA_WRITE(ZSDMA_TDARB+3, 0);
}

int
hsisidentify(name)
 char *name;
{
 if (strcmp(name, "HSI") == 0) {
  ++nhsisboard;
  return (1);
 } else
  return (0);
}

int
hsisattach(dev)
 register struct dev_info *dev;
{
 static int curhsis = 0;
 register struct hsiscom *hs;
 register u_char *hsboard;
 register int i;

 dev->devi_unit = curhsis;
 if (curhsis > sizeof(hsis_isum) * 8 / 4) {
  /* each hsis channel needs a bit in hsis_isum */
  printf("hsis: maximum number of channels exceeded.\n");
  return (-1);
 }
 if (hsiscom == NULL) {
  hsiscom = (struct hsiscom *)new_kmem_zalloc(
    (u_int)(nhsisboard * sizeof (struct hsiscom)),
    KMEM_SLEEP);
  if (hsiscom == NULL) {
   printf("hsis: no space for data structures.\n");
   return (-1);
  }
```

```
 hsiscom_end = &hsiscom[nhsisboard];

 hsis_softc = (struct hsis_softc *)new_kmem_zalloc(
   (u_int)(nhsisboard * 4 * sizeof (struct hsis_softc)),
   KMEM_SLEEP);
 if (hsis_softc == NULL) {
  printf("hsis: no space for data structures\n");
  return (-1);
 }
 hsis_softc_end = &hsis_softc[nhsisboard * 4];

 rawsyncinit(); /*XXX*/
}
hs = &hsiscom[dev->devi_unit];
hs->hs_dev = dev;

/*
 * register our interrupt handler, map the board into kernel memory,
 * then reset it. Note that the interrupt level must be <= splimp
 * (and, given the constraints imposed by the braindead Zilog dma,
 * the level should be as high as possible, e.g., splimp).
 */
addintr(dev->devi_intr->int_pri, hsisintr, dev->devi_name, curhsis);
addintr(SOFTINT_LEVEL, hsissoftint, "HSI-soft", curhsis);
hsboard = (u_char *)map_regs(dev->devi_reg->reg_addr,
    dev->devi_reg->reg_size,
    dev->devi_reg->reg_bustype);
hs->hs_board = hsboard;

/*
 * set up the free buffer list.
 */
for (i = HSIS_SRAM_SIZE; (i -= HSIS_BUFSIZE) >= 0; ) {
 register struct hsisbuf *bp;

 bp = (struct hsisbuf *)(hsboard + i + HSIS_SRAM);
 bp->next = hs->hs_free;
 hs->hs_free = bp;
}

/*
 * Initialize the two dma channels and 4 scc channels.
 * First reset each chip and set its bus configuration reg.
 * Then set up the software data structures.
 */
hsboard[HSIS_RST_ISSC0] = 0; /* reset chips */
hsboard[HSIS_RST_ISSC1] = 0;
DELAY(10);
hsboard[HSIS_SCC_A] = 0; /* clear BCR */
hsboard[HSIS_SCC_C] = 0;

hsis_dma_attach(0, &hs->hs_zd[0], &hsboard[HSIS_DMA_0]);
```

```
 hsis_dma_attach(1, &hs->hs_zd[1], &hsboard[HSIS_DMA_1]);

 hsis_scc_attach(0, hs, &hsboard[HSIS_SCC_A]);
 hsis_scc_attach(1, hs, &hsboard[HSIS_SCC_B]);
 hsis_scc_attach(2, hs, &hsboard[HSIS_SCC_C]);
 hsis_scc_attach(3, hs, &hsboard[HSIS_SCC_D]);

 report_dev(dev);
 ++curhsis;
 return (0);
}

int
copy_m_to_b(sc, m, cp)
 register struct hsis_softc *sc;
 register struct mbuf *m;
 register u_char *cp;
{
 register long len, totlen;
 register struct mbuf *m0 = m;

 totlen = 0;
 do {
  len = m->m_len;
  bcopy(mtod(m, caddr_t), (caddr_t)cp, (u_int)len);
  cp += len;
  totlen += len;
 } while (m = m->m_next);

 m_freem(m0);
 if (sc->sc_raw && (sc->sc_raw->so_snd.sb_flags & SB_WAIT ||
    sc->sc_raw->so_snd.sb_sel))
  sbwakeup(sc->sc_raw, &sc->sc_raw->so_snd);
 return (totlen);
}

hsis_output(ifp, m, dst)
 register struct ifnet *ifp;
 register struct mbuf *m;
 struct sockaddr *dst;
{
 register int s;
 register struct hsis_softc *sc;
#ifdef TRAFFIC_CONTROL
 struct aNetIf *gifcp;
 caddr_t   flowp,
    nethdrp;
 int   rsrclen,
    validlen;
 unsigned long  key,
    rsrc;
 unsigned short  pf;
```

```
 struct mbuf *mp,
   *(*dqfuncp) ();
#endif TRAFFIC_CONTROL

 sc = (struct hsis_softc *)ifp;
 TRACEL(T_OUTPUT, sc, 0)
#ifdef HSIS_TRACE
 /*
  * keep trace unlocked; ignore miniscule interrupt race.
  */
 if (hsis_trace_lock > 100) hsis_trace_lock = 0x7FFFFFFF;
#endif
 if ((ifp->if_flags & IFF_UP) == 0) {
  m_freem(m);
  return (ENETDOWN);
 }

#ifdef TRAFFIC_CONTROL
 /* Map ifnet pointer to extended ifnet (aNetIf) */

 gifcp = tcif_ifp2gifcp( (unsigned long) ifp );

 pf = PF_UNSPEC;
 nethdrp = mtod ( m, caddr_t);
 validlen = m->m_len;

 if ( m->m_type == MT_TCDATA )
 {
  flowp = m->m_tcflowp;
  rsrc = m->m_tcrsrc;
  key = m->m_tckey;
 }
 else
 {
  flowp = (caddr_t) NULL;
  rsrc = 0L;
  key = 0L;
 }
#endif TRAFFIC_CONTROL

 if (dst->sa_family != AF_UNSPEC) {
  /* Add a PPP header */
  register u_int off = m->m_off;

  if (off <
#ifdef TRAFFIC_CONTROL
    ((unsigned long) &(((struct mbuf*)0)->m_tcdat[0]))
#else !TRAFFIC_CONTROL
    MMINOFF
#endif TRAFFIC_CONTROL
     + PPP_HDRSPACE ||
   (off >= MSIZE && m->m_cltype != MCL_STATIC_HDR)) {
```

```
    /* need new mbuf for hdr (should really panic here
     * then fix whatever isn't leaving space for header) */
    register struct mbuf *m0;

#ifdef TRAFFIC_CONTROL
    MGET(m0, M_DONTWAIT, MT_TCDATA);
#else !TRAFFIC_CONTROL
    MGET(m0, M_DONTWAIT, MT_DATA);
#endif TRAFFIC_CONTROL
    if (m0 == (struct mbuf *)0) {
     m_freem(m);
     return (ENOBUFS);
    }
#ifdef TRAFFIC_CONTROL   /* XXX 8 => mod x10 aligned */
    m0->m_off = ((int)&(((struct mbuf*)0)->m_tcdat[8]));
#endif TRAFFIC_CONTROL
    m0->m_next = m;
    m0->m_len = PPP_HDRSPACE;
    m = m0;
   } else {
    m->m_off -= PPP_HDRSPACE;
    m->m_len += PPP_HDRSPACE;
   }
#ifdef STII
   if (dst->sa_family == AF_COIP)
   {
    *mtod(m, u_int *) = PPP_STII;
    pf = PF_COIP;
   }
   else if (dst->sa_family == (0x8000 | AF_COIP))
   {
    *mtod(m, u_int *) = PPP_SCMP;
    pf = PF_COIP;
   }
   else
#endif STII
#ifdef TRAFFIC_CONTROL
   {
    pf = PF_INET;
    *mtod(m, u_int *) = PPP_INET; /* XXX */
   }
#else !TRAFFIC_CONTROL
   *mtod(m, u_int *) = PPP_INET; /* XXX */
#endif TRAFFIC_CONTROL

#ifdef TRAFFIC_CONTROL
   rsrclen = m->m_len;
   if ( (mp = m->m_next) != (struct mbuf *) NULL )
    do
    {
    rsrclen += mp->m_len;
    } while ( (mp = mp->m_next) != (struct mbuf *) NULL );
```

```
m->m_tcrsrc = rsrc;
m->m_tckey = key;

if ( m->m_type != MT_TCDATA )
{
 s = splimp();
 mbstat.m_mtypes[m->m_type]--;
 mbstat.m_mtypes[MT_TCDATA]++;
 m->m_type = MT_TCDATA;
 splx( s );
}

/* If no packet has not been classified, try to do so now */

if ( flowp == (caddr_t) NULL )
{
 if ( gifcp->iftc_classify != (caddr_t (*)()) NULL )
 flowp = (*gifcp->iftc_classify) ( gifcp, m, pf, nethdrp, validlen );
 /* ??? else "flow 0" */
}
m->m_tcflowp = flowp;

s = splimp(); /* for mbufs / mbstat */
{
 /* Submit packet for enforcement */

 if ( (gifcp->iftc_enforce == (int (*)()) NULL)
 || ((validlen = (*gifcp->iftc_enforce)
  ( gifcp, m, rsrclen, (struct timeval *) NULL)) == -2) )
 {
 /* Packet not subject to traffic control */

 if ( validlen == -2 )
 {
  m->m_type = MT_DATA; /* No-sort */
  mbstat.m_mtypes[ MT_DATA ]++;
  mbstat.m_mtypes[ MT_TCDATA ]--;
 }
 validlen = 0;

#ifndef SFQ_VC
 if ( IF_QFULL( &( ifp->if_snd )) )
#else
 if ( (m->m_type == MT_TCDATA) && IF_QFULL( &( ifp->if_snd )) )
#endif
  validlen = -1; /* Ignore minor race reading if_qlen */
 }

 /* Drop this packet */

 if ( validlen == -1 )
```

```
    {
    IF_DROP( &( ifp->if_snd ) );
    TRACE(T_QFULL, sc, 0)
     /* update drop stats -- no good way to do keep accurate stats */
     /* bytes_dropped += rsrclen */
     /* bytes_queued += validlen? - rsrclen */

    m_freem ( m );
    splx ( s );
    return ( ENOBUFS ); /* ??? better error */
    }
   }
   splx( s );
   s = splboard();
   {
    /* If no specific enqueue function is supplied, use generic enqueue */

    if ( gifcp->iftc_nq != (void (*)()) NULL )
    (*gifcp->iftc_nq) ( gifcp, m );
    else
    IF_ENQUEUE ( &( ifp->if_snd ), m );

    m = (struct mbuf *) NULL; /* pkt enqueued */

#ifndef SFQ_VC
    if ( ifp->if_snd.ifq_len == 0 )
    {
    splx( s );
    return ( 0 );
    }
#endif
  }
  splx( s );
#endif TRAFFIC_CONTROL
 }

 /* Either raw (AF_UNSPEC) & m != 0, or m == 0 */

 if (sc->sc_ostate == 0) {
  register struct hsisbuf *bp = sc->sc_curout;
  register int len;

#ifdef TRAFFIC_CONTROL
  if ( m == (struct mbuf *) NULL )
  {
   s = splboard();

   /* If no specific dequeue function is supplied, use specified queue */

   if ( (dqfuncp = gifcp->iftc_dq) != (struct mbuf * (*)()) NULL )
   m = (*dqfuncp) ( gifcp ); /* Use specific dequeue function */
   else    /* Use generic dequeue function */
```

```
   IF_DEQUEUE(&sc->sc_if.if_snd, m);
   splx( s );

   if ( m == (struct mbuf *) NULL )
   return ( 0 );
  }
#endif TRAFFIC_CONTROL

  len = copy_m_to_b(sc, m, BUFtoCP(bp));
  s = splboard();
  TRACE(T_OUT_COPY, sc, (int)bp | len)
  bp->cnt = len;
  sc->sc_ostate = 2;
  hsis_start(sc);
#ifdef HSIS_EXTERNAL_RCVDONE
 } else if (sc->sc_if.if_snd.ifq_len == 0 && sc->sc_nextout->cnt == 0) {
#else
 } else if (sc->sc_codec.datalen &&
   sc->sc_if.if_snd.ifq_len == 0 && sc->sc_nextout->cnt == 0) {
#endif
  register struct hsisbuf *bp = sc->sc_nextout;
  register int len;

#ifdef TRAFFIC_CONTROL
  if ( m == (struct mbuf *) NULL )
  {
   s = splboard();

   /* If no specific dequeue function is supplied, use specified queue */

   if ( (dqfuncp = gifcp->iftc_dq) != (struct mbuf * (*)()) NULL )
   m = (*dqfuncp) ( gifcp ); /* Use specific dequeue function */
   else    /* Use generic dequeue function */
   IF_DEQUEUE(&sc->sc_if.if_snd, m);
   splx( s );

   if ( m == (struct mbuf *) NULL )
   return ( 0 );
  }
#endif TRAFFIC_CONTROL

  len = copy_m_to_b(sc, m, BUFtoCP(bp));
  s = splboard();
  TRACE(T_OUT_COPY, sc, (int)bp | len)
  bp->cnt = len;
  if (sc->sc_ostate == 0) {
   /*
    * last packet completed while we were doing copy --
    * flip buffers & restart output.
    */
   (sc->sc_nextout = sc->sc_curout)->cnt = 0;
   sc->sc_curout = bp;
```

```
    sc->sc_ostate = 2;
    hsis_start(sc);
   }
  } else {
   register struct ifqueue *ifq = &ifp->if_snd;

#ifdef TRAFFIC_CONTROL
   if ( m == (struct mbuf *) 0 )
    return ( 0 ); /* already enqueued */
#endif TRAFFIC_CONTROL
   s = splboard();
   if (IF_QFULL(ifq)) {
    IF_DROP(ifq);
    TRACE(T_QFULL, sc, 0)
    splx(s);
    m_freem(m);
    return (ENOBUFS);
   }
   IF_ENQUEUE(ifq, m);

   if (sc->sc_ostate == 0)
    hsis_start(sc);
  }
  splx(s);
  return (0);
}

/*
 * start new output operation. This routine *must* be called at splboard.
 */
void
hsis_start(sc)
 register struct hsis_softc *sc;
{
 register struct zscc *zs = sc->sc_zs;
 register u_char *zsdev = zs->zs_addr;
 register u_char *zddev = sc->sc_zd->zd_addr;
 register u_char unit = sc->sc_if.if_unit;
 register u_char *cp;
 register int len;
 register u_int baddr;
 register int s;
 register u_int resid;
 register int i;

 cp = BUFtoCP(sc->sc_curout);
 if (sc->sc_ostate == 3)  /* take handoff from hardware intr */
  sc->sc_ostate = 0;
 if (sc->sc_ostate == 0) {
  register struct hsisbuf *bp;

  if (len = (bp = sc->sc_nextout)->cnt) {
```

```
    /*
     * 'next' output buffer is full - swap current and
     * next. (The weird assignment below fools Sun-4 cc
     * into generating reasonable code -- maybe one day
     * Sun will discover ANSI C 'volatile' and this
     * crap can go away.)
     */
    (sc->sc_nextout = sc->sc_curout)->cnt = 0;
    sc->sc_curout = bp;
    cp = BUFtoCP(bp);
   } else {
    /*
     * on-board xmit buffer is empty. try to copy a new
     * packet to it. (we want to overlap the copy with
     * the scc sending the final crc and flag bytes to
     * avoid taking an extra xmit interrupt).
     */
    register struct mbuf *m;
#ifdef TRAFFIC_CONTROL
    struct aNetIf *gifcp;
    struct mbuf *(*dqfuncp) ();

    /* Map ifnet pointer to extended ifnet (aNetIf) */

    gifcp = tcif_ifp2gifcp( (unsigned long) &( sc->sc_if ) );

    /* If no specific dequeue function is supplied, use specified queue */

    if ( (dqfuncp = gifcp->iftc_dq) != (struct mbuf * (*)()) NULL )
     m = (*dqfuncp) ( gifcp ); /* Use specific dequeue function */
    else    /* Use generic dequeue function */
#endif TRAFFIC_CONTROL
    IF_DEQUEUE(&sc->sc_if.if_snd, m);

    if (m == NULL) {
     sc->sc_if.if_timer = 0;
     TRACE(T_EMPTY_OQ, sc, 0)
     return;
    }

    len = copy_m_to_b(sc, m, cp);
    TRACE(T_OUT_COPY, sc, (int)bp | len)
   }
  } else
   len = sc->sc_curout->cnt;

  sc->sc_if.if_timer = HSIS_WATCHDOG_TIME;
  if (!sc->sc_codec.datalen) {
   /*
    * if the transmit buffer is full it means that crc/flag
    * sending is still in progress. the stupid scc will jam
    * two packets together if we enable dma xmit so we have
```

```
         * to turn on the xmit interrupt (which should come when
         * the crc and flag have been sent) and exit waiting for
         * that interrupt. If the transmit buffer is empty, we
         * can just start the next packet (this should be the
         * usual case at T1 speeds).
         */
        SCC_READ0(resid);
        if ((resid & ZSRR0_TX_READY) == 0) {
         /*
          * we lose - wait for xmit intr.
          */
         SCC_BIS(1, ZSWR1_TIE);
         sc->sc_ostate = 2;
         sc->sc_curout->cnt = len;
         return;
        }
       }
       /*
        * Transmit buffer empty -- start up dma.
        */
       baddr = cp - sc->sc_hs->hs_board;
       SCC_WRITE0(ZSWR0_RESET_TXCRC);  /* no harm for codec mode */
       if (unit & 1) {
        /* B channel */
        register u_int c;

        DMA_READ(ZSDMA_TDCRB, resid);
        DMA_READ(ZSDMA_TDCRB + 1, c);
        resid |= c << 8;

        DMA_WRITE(ZSDMA_TDCRB, len);
        DMA_WRITE(ZSDMA_TDCRB+1, len >> 8);

        DMA_WRITE(ZSDMA_TDARB, baddr);
        DMA_WRITE(ZSDMA_TDARB+1, baddr >> 8);

        s = splhigh();
        DMA_WRITE(ZSDMA_CCAR, ZSDMA_CCAR_ENA_TX_B);
       } else {
        /* A channel */
        register u_int c;

        DMA_READ(ZSDMA_TDCRA, resid);
        DMA_READ(ZSDMA_TDCRA + 1, c);
        resid |= c << 8;

        DMA_WRITE(ZSDMA_TDCRA, len);
        DMA_WRITE(ZSDMA_TDCRA+1, len >> 8);

        DMA_WRITE(ZSDMA_TDARA, baddr);
        DMA_WRITE(ZSDMA_TDARA+1, baddr >> 8);
```

```
    s = splhigh();
    DMA_WRITE(ZSDMA_CCAR, ZSDMA_CCAR_ENA_TX_A);
   }
   if (!sc->sc_codec.datalen) {
    /*
     * Cretinous chip requires that EOM not be reset until dma has
     * loaded first data character into buffer but must be reset
     * before last character loaded into buffer. We've locked out
     * to prevent the obvious race so just wait until character
     * gets there.
     */
    for (i = 10; --i >= 0; ) {
     DELAY(2);
     SCC_READ0(baddr);
     if ((baddr & ZSRR0_TX_READY) == 0)
      goto rdy;
    }
    TRACE(T_NO_XMIT_RDY, sc, baddr)
    dprintf(("hsis%d: hsis_start: xmit didn't load (rr0=0x%x)\n",
     unit, baddr))
   rdy:
    SCC_WRITE0(ZSWR0_RESET_EOM);
   }
   TRACE(T_START_OUT, sc, (resid << 16) | len)
   sc->sc_ostate = 1;
   splx(s);
   if (resid) {
    ++sc->sc_if.if_oerrors;
    ++sc->sc_estats.sse_underrun;
   }
   ++sc->sc_if.if_opackets;
   ++sc->sc_dstats.ssd_opack;
   sc->sc_dstats.ssd_ochar += len;
#if NBPFILTER > 0
   if (sc->sc_bpf)
    bpf_tap(sc->sc_bpf, cp, len);
#endif
   /*
    * if there's more in the snd q, copy another packet to the
    * on-board 'next' buffer (we do it now to overlap the copy
    * with the send of the last packet to approximate back-to-back
    * output packets).
    */

#if defined (HSIS_EXTERNAL_RCVDONE) && !defined(SFQ_VC)
   if (sc->sc_if.if_snd.ifq_len)
#elif !defined(SFQ_VC) && !defined(HSIS_EXTERNAL_RCVDONE)
   if (sc->sc_codec.datalen && sc->sc_if.if_snd.ifq_len)
#endif
    {    /* Here to avoid confusing tgrind */
    register struct mbuf *m;
#ifdef TRAFFIC_CONTROL
```

```
  struct aNetIf *gifcp;
  struct mbuf *(*dqfuncp) ();

  /* Map ifnet pointer to extended ifnet (aNetIf) */

  gifcp = tcif_ifp2gifcp( (unsigned long) &( sc->sc_if ) );

  /* If no specific dequeue function is supplied, use specified queue */

  if ( (dqfuncp = gifcp->iftc_dq) != (struct mbuf * (*)()) NULL )
   m = (*dqfuncp) ( gifcp ); /* Use specific dequeue function */
  else   /* Use generic dequeue function */
#endif TRAFFIC_CONTROL
  IF_DEQUEUE(&sc->sc_if.if_snd, m);

  if (m) {
   register struct hsisbuf *bp = sc->sc_nextout;

   bp->cnt = len = copy_m_to_b(sc, m, BUFtoCP(bp));
   TRACE(T_OUT_COPY, sc, (int)bp | len)
  }
  }
}

int
hsis_start_dma_read(sc)
 register struct hsis_softc *sc;
{
 register u_char *zddev = sc->sc_zd->zd_addr;
 register int baddr = BUFtoCP(sc->sc_inbuf) - sc->sc_hs->hs_board;
 register u_char bl = baddr;
 register u_char bh = baddr >> 8;
 register int s = splhigh();
 register u_int rl;
 register u_char rh;

 /*
  * disable the dma channel, set it to xfer into sc_inbuf,
  * then re-enable. If there are back-to-back packets inbound,
  * we have to re-enable the dma channel before the 3 byte scc
  * rcv fifo overflows. E.g., at T1 (5.2us/byte) the time
  * from disable to enable should be no more than 10us. So,
  * we make sure nothing interrupts us during this window and
  * the code below is as fast as I can make it.
  * Data from the next packet probably got stuffed into the
  * buffer containing the current packet so we save and return
  * the current dma count reg. before overwriting it (so the
  * higher level routine can undo the damage done by this
  * braindead dma model).
  */
 if (sc->sc_zs->zs_unit & 1) {
  /* B channel */
```

```
/* if the TX DMA is just about finished, wait for it */
/* to avoid re-enabling it when disabling RX DMA */
DMA_READ(ZSDMA_TDCRB, rl);
DMA_READ(ZSDMA_TDCRB + 1, rh);
rl |= rh << 8;
if (rl > 0 && rl < 4) {
 do {
  DELAY(2);
  DMA_READ(ZSDMA_DER, rh);
 } while (rh & ZSDMA_DER_TX_B_ENABLE);
}

/* spin until the rcv fifo is empty */
for (rl = 10; --rl != 0; ) {
 register u_char *zsdev = sc->sc_zs->zs_addr;
 SCC_READ0(rh)
 if ((rh & ZSRR0_RX_READY) == 0)
  break;
 DELAY(2);
}
if (rl == 0)
 TRACE(T_FAIL, sc, rh);

/* repeat the disable until it works (harware bug) */
DMA_READ(ZSDMA_DER, rh);
do {
 DMA_WRITE(ZSDMA_DER, rh &~ ZSDMA_DER_RX_B_ENABLE);
 DMA_READ(ZSDMA_DER, rh);
} while (rh & ZSDMA_DER_RX_B_ENABLE);

DMA_READ(ZSDMA_RDARB, rl);
DMA_WRITE(ZSDMA_RDARB, bl);
DMA_READ(ZSDMA_RDARB+1, rh);
DMA_WRITE(ZSDMA_RDARB+1, bh);

DMA_WRITE0(ZSDMA_CCAR_ENA_RX_B);

/* set count after enabling to minimize time */
/* disabled; lsb first to prevent borrow from msb */
DMA_WRITE(ZSDMA_RDCRB, HSIS_MAXPACKET & 0xff);
DMA_WRITE(ZSDMA_RDCRB+1, HSIS_MAXPACKET >> 8);
} else {
/* A channel */
DMA_READ(ZSDMA_TDCRA, rl);
DMA_READ(ZSDMA_TDCRA + 1, rh);
rl |= rh << 8;
if (rl > 0 && rl < 4) {
 do {
  DELAY(2);
  DMA_READ(ZSDMA_DER, rh);
 } while (rh & ZSDMA_DER_TX_A_ENABLE);
```

```
    }

    for (rl = 10; --rl != 0; ) {
     register u_char *zsdev = sc->sc_zs->zs_addr;
     SCC_READ0(rh)
     if ((rh & ZSRR0_RX_READY) == 0)
      break;
     DELAY(2);
    }
    if (rl == 0)
     TRACE(T_FAIL, sc, rh);

    DMA_READ(ZSDMA_DER, rh);
    do {
     DMA_WRITE(ZSDMA_DER, rh &~ ZSDMA_DER_RX_A_ENABLE);
     DMA_READ(ZSDMA_DER, rh);
    } while (rh & ZSDMA_DER_RX_A_ENABLE);

    DMA_READ(ZSDMA_RDARA, rl);
    DMA_WRITE(ZSDMA_RDARA, bl);
    DMA_READ(ZSDMA_RDARA+1, rh);
    DMA_WRITE(ZSDMA_RDARA+1, bh);

    DMA_WRITE0(ZSDMA_CCAR_ENA_RX_A);

    DMA_WRITE(ZSDMA_RDCRA, HSIS_MAXPACKET & 0xff);
    DMA_WRITE(ZSDMA_RDCRA+1, HSIS_MAXPACKET >> 8);
   }
   rl |= rh << 8;
   TRACE(T_START_READ, sc, (((bh << 8) | bl) << 16) | rl)
   splx(s);
   return ((int)(rl - sizeof(struct hsisbuf)) & (HSIS_BUFSIZE - 1));
}

/*
 * This routine is called if an output operation takes longer than
 * HSIS_WATCHDOG_TIME (usually 2 minutes) to complete. Reset and
 * restart the channel (current output packet will be lost).
 */
hsis_watchdog(unit)
 register int unit;
{
 register struct hsis_softc *sc = &hsis_softc[unit];
 register u_int errcnt = sc->sc_oerrcnt;
 register int s;

 if (sc->sc_inbuf == NULL || (sc->sc_if.if_flags & IFF_UP))
  /* we have been manually turned offline or online */
  return;

 if (errcnt == 0)
  printf("hsis%d: watchdog timeout.\n", unit);
```

```
 s = splboard();
 sc->sc_oerrcnt = ++errcnt;
 hsis_reset(unit);
 hsis_init(unit);
 hsis_start(sc);
 splx(s);
}

void
hsis_ierr_timer(sc)
 register struct hsis_softc *sc;
{
 register int unit = sc->sc_if.if_unit;
 register int s;

 if (sc->sc_inbuf == NULL || (sc->sc_if.if_flags & IFF_UP))
  /* we have been manually turned offline or online */
  return;
 printf("hsis%d: reset and restarted.\n", unit);
 s = splboard();
 hsis_init(unit);
 hsis_start(sc);
 splx(s);
}

void
hsis_ierror(sc, zs, zsdev, msg)
 register struct hsis_softc *sc;
 register struct zscc *zs;
 register u_char *zsdev;
 register char *msg;
{
 register int unit = sc->sc_if.if_unit;
 register u_int errcnt = sc->sc_ierrcnt;

 sc->sc_ierrcnt = ++errcnt;
 if (errcnt >= HSIS_RESET_THRESH) {
#ifdef HSIS_TRACE
  if (hsis_trace_lock > 100) hsis_trace_lock = 100;
#endif
  hsis_reset(unit);
  if (errcnt >= HSIS_OFF_THRESH)
   timeout(hsis_ierr_timer, sc, HSIS_OFF_TIME);
  else
   hsis_ierr_timer(sc);
  return;
 }
 if (errcnt == 1)
  printf("hsis%d: %s (p#%d).\n", unit, msg,
   sc->sc_dstats.ssd_ipack);

 /*
```

```
     * book says we have to disable then re-enable
     * fifo to clear it. Then toss input queue in (vain) hope that
     * we'll end up with input stream and fifo in sync.
     */
    *(u_char *)AUXIO_REG = AUX_MBO|AUX_EJECT;
     SCC_BIC(3, ZSWR3_RX_ENABLE)
     SCC_BIC(15, ZSR15_SDLC_FIFO_ENA)
     ++sc->sc_if.if_ierrors;
     SCC_BIS(15, ZSR15_SDLC_FIFO_ENA)
     SCC_WRITE0(ZSWR0_RESET_ERRORS);
     (void) hsis_start_dma_read(sc);
     SCC_BIS(3, ZSWR3_RX_ENABLE)
     if (sc->sc_intail) {
      sc->sc_intail->next = sc->sc_hs->hs_free;
      sc->sc_hs->hs_free = sc->sc_inq;
      sc->sc_inq = NULL;
      sc->sc_intail = NULL;
     }
     bzero(sc->sc_fifo, sizeof(sc->sc_fifo));
     sc->sc_fifop = sc->sc_fifo;
     sc->sc_inoff = NULL;
    *(u_char *)AUXIO_REG = AUX_MBO|AUX_EJECT|AUX_LED;
    }

    void
    hsis_dmarecv_time(sc)
     register struct hsis_softc *sc;
    {
     register u_char *zddev = sc->sc_zd->zd_addr;
     register struct hsiscom *hs;
     register struct hsisbuf *bp;
     register struct hsisbuf *qp;
     register int *fp;
     register int cnt;
     register int did_something = 0;
     register int s = splboard();

     TRACE(T_RECVDMA_INT, sc, 0)
     hs = sc->sc_hs;
     if ((bp = sc->sc_inbuf) == 0 || !sc->sc_codec.datalen) {
      /* channel is offline or no longer codec mode */
      /* (void)hsis_start_dma_read(sc); DO NOTHING */
      splx(s);
      return;
     }
     if ((qp = hs->hs_free) == NULL) {
    #ifdef HSIS_TRACE
      if (hsis_trace_lock > 100) hsis_trace_lock = 100;
    #endif
      /* no free buffers - just toss input */
      (void)hsis_start_dma_read(sc);
      timeout(hsis_dmarecv_time, sc, sc->sc_ticks);
```

```
 TRACE(T_RCVINT_BUF, sc, 0)
 splx(s);
 return;
}
hs->hs_free = qp->next;
sc->sc_inbuf = qp;
cnt = hsis_start_dma_read(sc);
timeout(hsis_dmarecv_time, sc, sc->sc_ticks);
if ((bp->cnt = cnt) == 0) {
 bp->next = hs->hs_free;
 hs->hs_free = bp;
} else {
 bp->next = NULL;
 if (qp = sc->sc_intail)
  qp->next = bp;
 else
  sc->sc_inq = bp;
 sc->sc_intail = bp;

 sc->sc_bytesread += cnt;
 while (sc->sc_bytesread >= sc->sc_readlen) {
  sc->sc_bytesread -= sc->sc_readlen;
  fp = sc->sc_fifop;
  *fp++ = sc->sc_readlen;
  if (fp >= &sc->sc_fifo[HSIS_NFIFO])
   fp = sc->sc_fifo;
  sc->sc_fifop = fp;
  /*
   * Reset readlen to full length after first block.
   */
  sc->sc_readlen = sc->sc_codec.datalen;
  ++did_something;
 }
 if (did_something) {
  (void) splsoft();
  hsis_recv_done(sc);
 }
}
 splx(s);
}

void
hsis_dmaxmit_intr(sc)
 register struct hsis_softc *sc;
{
 register u_char *zddev = sc->sc_zd->zd_addr;
 register u_char unit = sc->sc_if.if_unit;
 register u_char *cp;
 register int len;
 register u_int baddr;
 register struct hsisbuf *bp;
```

```
 TRACE(T_XMITDMA_INT, sc, 0)
 DMA_WRITE(ZSDMA_ICSR, ZSDMA_ICSR_RESET_IP_IUS |
  (ZSDMA_ICSR_TX_A >> ((unit&1)<<1)));
 if ((len = (bp = sc->sc_nextout)->cnt) == 0) {
  sc->sc_ostate = 0; /* no 'next' buffer, go idle */
  return;
 }

 /*
  * 'next' output buffer is full - swap current and
  * next. (ditto 'volatile')
  */
 (sc->sc_nextout = sc->sc_curout)->cnt = 0;
 sc->sc_curout = bp;
 cp = BUFtoCP(bp);

 sc->sc_if.if_timer = HSIS_WATCHDOG_TIME;
 baddr = cp - sc->sc_hs->hs_board;
 if (unit & 1) {
  /* B channel */
  DMA_WRITE(ZSDMA_TDCRB, len);
  DMA_WRITE(ZSDMA_TDCRB+1, len >> 8);

  DMA_WRITE(ZSDMA_TDARB, baddr);
  DMA_WRITE(ZSDMA_TDARB+1, baddr >> 8);

  DMA_WRITE(ZSDMA_CCAR, ZSDMA_CCAR_ENA_TX_B);
 } else {
  /* A channel */
  DMA_WRITE(ZSDMA_TDCRA, len);
  DMA_WRITE(ZSDMA_TDCRA+1, len >> 8);

  DMA_WRITE(ZSDMA_TDARA, baddr);
  DMA_WRITE(ZSDMA_TDARA+1, baddr >> 8);

  DMA_WRITE(ZSDMA_CCAR, ZSDMA_CCAR_ENA_TX_A);
 }
 TRACE(T_START_OUT, sc, len)
 sc->sc_ostate = 1;
 ++sc->sc_if.if_opackets;
 ++sc->sc_dstats.ssd_opack;
 sc->sc_dstats.ssd_ochar += len;
 SOFTINT(sc,XMITDMA_DONE);
}

void
hsis_sccrecv_intr(sc, zs, zsdev)
 register struct hsis_softc *sc;
 register struct zscc *zs;
 register u_char *zsdev;
{
 register struct hsiscom *hs;
```

```
register struct hsisbuf *bp;
register struct hsisbuf *qp;
register int *fp;

/*
 * Stash the current contents of the sdlc fifo. This should
 * be done before we call 'start_dma_read' or we can end up
 * with the fifo & dma counts different: For god-only-knows
 * what reason, Zilog made the sdlc fifo count bytes that have
 * entered the 3 byte receive data fifo while the dma counts
 * bytes that have left the receive data fifo. Thus the fifo
 * can record the end of a packet that the dma hasn't finished.
 */
fp = sc->sc_fifop;
while (1) {
 register u_char rr1, rr6, rr7;
 register int i;

 SCC_READ(7, rr7)
 SCC_READ(6, rr6)
 SCC_READ(1, rr1)
 i = (((rr1 << 8) | rr7) << 8) | rr6;
 TRACE(T_RCVINT_FIFO, sc, i)
 if (rr1 & ZSRR1_DO) {
  /*
   * data overrun - the way the Zilog fifo works
   * we don't have a prayer of recovering so toss
   * everything in the fifo and input queue, put
   * the scc in hunt mode to skip to the start of
   * the next packet, then hope the fifo eventually
   * gets back in sync with the input stream.
   */
  hsis_ierror(sc, zs, zsdev, "receive data overrun");
  ++sc->sc_estats.sse_overrun;
  return;
 }
 switch (rr7 & 0xc0) {

 case 0x80:
 case 0xc0:
  /* fifo overflow */
  hsis_ierror(sc, zs, zsdev, "status fifo overflow");
  return;

 case 0x00:
  /* fifo empty */
  goto fifo_empty;
 }
 if (*fp) {
  hsis_ierror(sc, zs, zsdev, "status fifo array overflow");
  return;
 }
```

```
        *fp++ = i;
        if (fp >= &sc->sc_fifo[HSIS_NFIFO])
         fp = sc->sc_fifo;
       }
      fifo_empty:
       if (fp == sc->sc_fifop)
        /* did nothing (spurious read interrupt) */
        return;
       sc->sc_fifop = fp;

       hs = sc->sc_hs;
       if ((bp = sc->sc_inbuf) == 0) {
        /* channel is offline */
        (void)hsis_start_dma_read(sc);
        return;
       }
       if ((qp = hs->hs_free) == NULL) {
        /* no buffer - have to toss input or we lose sync with fifo */
        TRACE(T_RCVINT_BUF, sc, 0)
        hsis_ierror(sc, zs, zsdev, "no free bufs");
        return;
       }
       hs->hs_free = qp->next;
       sc->sc_inbuf = qp;
       if ((bp->cnt = hsis_start_dma_read(sc)) == 0) {
        bp->next = hs->hs_free;
        hs->hs_free = bp;
       } else {
        bp->next = NULL;
        if (qp = sc->sc_intail)
         qp->next = bp;
        else
         sc->sc_inq = bp;
        sc->sc_intail = bp;
       }
       SOFTINT(sc, RECV_DONE);
      }

      void
      hsis_stat_intr(sc, zs, zsdev)
       register struct hsis_softc *sc;
       register struct zscc *zs;
       register u_char *zsdev;
      {
       register u_int rr0;

       SCC_READ0(rr0);
       SCC_WRITE0(ZSWR0_RESET_STATUS);
       SCC_WRITE0(ZSWR0_CLR_INTR);
       TRACE(T_STAT_INT, sc, rr0)
       if ((rr0 & ZSRR0_TXUNDER) && sc->sc_ostate) {
        /* packet completed - start a new one if possible */
```

```
 sc->sc_oerrcnt = 0;
 sc->sc_ostate = 3; /* don't let hsis_output sneak in */
 SOFTINT(sc, XMIT_DONE);
} else if (rr0 & ZSRR0_BREAK) {
 /*
  * 'abort' received -- reset input (because scc fifo
  * state is messed up and we have no way to figure
  * out packet boundaries).
  */
 hsis_ierror(sc, zs, zsdev, "received 'abort'");
 ++sc->sc_estats.sse_abort;
} else if (!(rr0 & ZSRR0_SYNC)) {
 /*
  * In codec mode, have established sync so receive data
  * will now start to come in. Start the timer for
  * restarting receive DMA.
  */
 SCC_BIC(15, ZSR15_SYNC);
 if (sc->sc_codec.datalen)
  timeout(hsis_dmarecv_time, sc, sc->sc_ticks);
} else {
 /* XXX - should do something. */
 dprintf(("hsis%d: scc stat interrupt, rr0=0x%x.\n",
  sc->sc_if.if_unit, rr0))
}
}

void
hsis_sccxmit_intr(sc, zs, zsdev)
 register struct hsis_softc *sc;
 register struct zscc *zs;
 register u_char *zsdev;
{
 register u_int rr0;

 SCC_READ0(rr0);
 SCC_BIC(1, ZSWR1_TIE);
 SCC_WRITE0(ZSWR0_RESET_TXINT);
 SCC_WRITE0(ZSWR0_CLR_INTR);
 TRACE(T_XMIT_INT, sc, (sc->sc_ostate << 16) | rr0)
 if (sc->sc_ostate == 2) {
  /*
   * we were waiting for CRC/flag send to finish and it
   * has -- start next packet.
   */
  SOFTINT(sc, XMIT_DONE);
 } else {
  /*
   * since we never enabled xmit interrupt, something's weird.
   * it would be nice to print a warning message but there's
   * a small race in hsis_start where we TIE then find xmit is
   * done & turn it off. Zilog says the disable should clear
```

```
     * the interrupt but, of course, they lie & we can end
     * up here.
     */
    }
   }

#define ZS_A_INTR (ZSRR3_IP_A_STAT|ZSRR3_IP_A_RX|ZSRR3_IP_A_TX)
#define ZS_B_INTR (ZSRR3_IP_B_STAT|ZSRR3_IP_B_RX|ZSRR3_IP_B_TX)

#define ZINTSCAN(rmask, smask, tmask) { \
 if (rr3 & rmask) { \
  SCC_WRITE0(ZSWR0_RESET_ERRORS); \
  SCC_WRITE0(ZSWR0_CLR_INTR); \
  hsis_sccrecv_intr(sc, zs, zsdev); \
 } \
 if (rr3 & smask) \
  hsis_stat_intr(sc, zs, zsdev); \
 if (rr3 & tmask) \
  hsis_sccxmit_intr(sc, zs, zsdev); \
}

#define ZCHECK_INT(chan) ZINTSCAN(ZSRR3_IP_/**/chan/**/_RX, \
     ZSRR3_IP_/**/chan/**/_STAT, \
     ZSRR3_IP_/**/chan/**/_TX)
#ifndef HSIS_EXTERNAL_RCVDONE
#define ZCHECK_BOTH { \
 if (sc->sc_codec.datalen) { \
  register u_char *zddev = sc->sc_zd->zd_addr; \
  register u_char dma0; \
 \
  DMA_READ0(dma0); \
  if (dma0 & (ZSDMA_DSR_TX_A_ABORT|ZSDMA_DSR_TX_A_TERM)) { \
   ++did_something; \
   hsis_dmaxmit_intr(sc); \
  } \
  if (dma0 & (ZSDMA_DSR_TX_B_ABORT|ZSDMA_DSR_TX_B_TERM)) { \
   ++did_something; \
   hsis_dmaxmit_intr(sc + 1); \
  } \
 } \
 zs = sc->sc_zs; \
 zsdev = zs->zs_addr; \
 SCC_READ(3, rr3); \
 if (rr3 & ZS_A_INTR) { \
  ++did_something; \
  ZCHECK_INT(A) \
 } \
 ++sc; \
 if (rr3 & ZS_B_INTR) { \
  ++did_something; \
  zs = sc->sc_zs; \
  zsdev = zs->zs_addr; \
```

```
  ZCHECK_INT(B) \
 } \
 ++sc; \
}
#else
#define ZCHECK_BOTH { \
 if (sc->sc_codec.datalen) { \
  register u_char *zddev = sc->sc_zd->zd_addr; \
  register u_char dma0; \
 \
  DMA_READ0(dma0); \
  if (dma0 & (ZSDMA_DSR_TX_A_ABORT|ZSDMA_DSR_TX_A_TERM)) { \
   ++did_something; \
   hsis_dmaxmit_intr(sc); \
  } \
  if (dma0 & (ZSDMA_DSR_TX_B_ABORT|ZSDMA_DSR_TX_B_TERM)) { \
   ++did_something; \
   hsis_dmaxmit_intr(sc + 1); \
  } \
 } \
 zs = sc->sc_zs; \
 zsdev = zs->zs_addr; \
 if ((hsis[HSIS_EINT_STS] & rcvintmask) == 0) {\
  ++did_something; \
  hsis[zs->zs_unit + HSIS_EINT_CLR_A] = 0; \
  hsis_sccrecv_intr(sc, zs, zsdev); \
 } \
 rcvintmask <<= 1; \
 SCC_READ(3, rr3); \
 if (rr3 & ZS_A_INTR) { \
  ++did_something; \
  ZCHECK_INT(A) \
 } \
 ++sc; \
 if ((hsis[HSIS_EINT_STS] & rcvintmask) == 0) {\
  ++did_something; \
  zs = sc->sc_zs; \
  zsdev = zs->zs_addr; \
  hsis[zs->zs_unit + HSIS_EINT_CLR_A] = 0; \
  hsis_sccrecv_intr(sc, zs, zsdev); \
 } \
 rcvintmask <<= 1; \
 if (rr3 & ZS_B_INTR) { \
  ++did_something; \
  zs = sc->sc_zs; \
  zsdev = zs->zs_addr; \
  ZCHECK_INT(B) \
 } \
 ++sc; \
}
#endif
```

```
/*
 * Handle HSIS board(s) interrupt(s).
 */
int
hsisintr()
{
 register struct hsis_softc *sc;
 register struct hsis_softc *sc_end = hsis_softc_end;
 register struct zscc *zs;
 register u_char *zsdev, *hsis;
 register u_char rr3;
 register int did_something = 0, last_round;

 /*
  * It's costly to take an interrupt and likely that some other
  * channel finished while we were servicing the current channel.
  * We loop here until we make one full pass through the status
  * registers and don't find new work to do.
  */
 TRACE(T_INTR_ENTRY, hsis_softc, 0)
 do {
  last_round = did_something;
  /*
   * we can't reliably read the status registers on the
   * zilog chip so we look at the hsis board status register
   * to find interrupt requests. This means we loop over
   * four zilog channels at a time.
   * NOT TRUE ANYMORE per VJ 11-19-91
   */
  for (sc = hsis_softc; sc < sc_end; ) {
#ifdef HSIS_EXTERNAL_RCVDONE
   register u_char rcvintmask = 1;
#endif
   hsis = sc->sc_hs->hs_board;
   ZCHECK_BOTH
   ZCHECK_BOTH
  }
 } while (did_something != last_round);

 if (hsis_isum)
  set_intreg(IR_SOFT_INT4, 1);

 TRACE(T_INTR_EXIT, hsis_softc, did_something)
 return (did_something);
}

void
hsis_drop_input(sc, len, bufoff, buflen)
 register struct hsis_softc *sc;
 register int len, buflen;
 register u_char *bufoff;
{
```

```
/*
 * Note: To avoid races between the hardware & software
 * interrupt levels, it's important that the load of 'bp'
 * below be done *after* we are at splboard. In particular,
 * this means that the load of bp *cannot* be put in the
 * delay slot of the spl call. The statement order below
 * works with Sun's current compiler technology but may give
 * problems in the future (one day they'll discover "volatile"...)
 */
register int s = splboard();
register struct hsiscom *hs = sc->sc_hs;
register struct hsisbuf *bp = sc->sc_inq;

if (bp == NULL) {
 /*
  * timeout or hardware intr probably did a reset --
  * exit, making sure that input state stays clean.
  */
 sc->sc_inoff = 0;
 splx(s);
 return;
}
while (len) {
 if (buflen <= 0) {
  buflen = bp->cnt;
  bufoff = BUFtoCP(bp);
 }
 if (buflen > len) {
  buflen -= len;
  bufoff += len;
  break;
 } else {
  register struct hsisbuf *np;

  len -= buflen;
  buflen = 0;
  bufoff = NULL;

  np = bp->next;
  bp->next = hs->hs_free;
  hs->hs_free = bp;
  if ((sc->sc_inq = bp = np) == NULL) {
   sc->sc_intail = NULL;
   break;
  }
 }
}
if (buflen)
 bp->cnt = buflen;
sc->sc_inoff = bufoff;
splx(s);
}
```

```
/*
 * Following is mbuf offset to get nice alignment of packets.
 * Choice determined by:
 * 1. IP header *must* be aligned on a word (4 byte) boundary.
 * 2. bcopy will go much faster if data is cache aligned (i.e., at
 * 16-byte boundary).
 * 3. We need space to prepend a 14 byte ethernet header (if
 * forwarding packet).
 *
 * MMINOFF is 12 (thanks to Bill Joy for this horrible kludge) and the
 * next mult-of-16 above 12+14 is 32 so we offset 20 (= 32 - MMINOFF)
 * in an mbuf and 16 in a cluster.
 */
#define HDRSPACE (16)
#define MBUF_HDRSPACE (32 - MMINOFF)

/*
 * following macro drops 'len' input bytes (used on input errors).
 */
#define DROP_INPUT(len) { \
 hsis_drop_input(sc, len, bufoff, buflen); \
 if ((bp = sc->sc_inq) == NULL) \
  /* input was reset by timeout or error */ \
  return; \
 if (bufoff = sc->sc_inoff) \
  buflen = bp->cnt; \
 else \
  buflen = 0; \
}

/*
 * This routine goes through the software copy of the scc
 * status fifo (which records the lengths of incoming packets)
 * and sc_inq (the queue of unprocessed, incoming data) and
 * breaks the data up into packets then queues them for higher
 * level network processing. The loop structure is complicated
 * by the fact that packet boundaries are mapped randomly onto
 * the input queue (there may be more than one packet per input
 * buffer and packets may cross buffer boundaries).
 */
void
hsis_recv_done(sc)
 register struct hsis_softc *sc;
{
 register int *fp;
 register u_char *bufoff;
 register int buflen;
 register struct hsiscom *hs = sc->sc_hs;
 register struct hsisbuf *bp;

 /* find the first unprocessed packet in the fifo array */
```

```
for (fp = sc->sc_fifop - 1; ; --fp) {
 if (fp < sc->sc_fifo)
  fp = &sc->sc_fifo[HSIS_NFIFO - 1];
 if (*fp == 0)
  break;
}

if (bufoff = sc->sc_inoff) {
 if ((bp = sc->sc_inq) == NULL)
  return;
 buflen = bp->cnt;
} else
 buflen = 0;

while (1) {
 register struct mbuf *m;
 register u_char *op;
 register int sfifo;

 if (++fp >= &sc->sc_fifo[HSIS_NFIFO])
  fp = sc->sc_fifo;

 if ((sfifo = *fp) == 0)
  break;
 *fp = 0;
 TRACEL(T_RCVINT, sc, sfifo)

 if (sfifo & (ZSRR1_FE << 16)) {
  ++sc->sc_estats.sse_crc;
  ++sc->sc_if.if_ierrors;
  sfifo &= 0x3fff;
  DROP_INPUT(sfifo)
  continue;
 }
 sfifo &= 0x3fff;
 if (sfifo > MCLBYTES - HDRSPACE) {
  /* packet too big */
  ++sc->sc_if.if_ierrors;
  DROP_INPUT(sfifo)
  continue;
 }
 MGET(m, M_DONTWAIT, MT_DATA);
 if (m == (struct mbuf *)0) {
  DROP_INPUT(sfifo)
  continue;
 }
 if (sfifo <= MLEN - MBUF_HDRSPACE)
  m->m_off += MBUF_HDRSPACE;
 else {
  /* too big for mbuf - use cluster */
  MCLGET(m);
  if (m->m_len != MCLBYTES) {
```

```
        /* no clusters - drop this packet */
        m_freem(m);
        DROP_INPUT(sfifo)
        continue;
       }
       m->m_off += HDRSPACE;
      }
      /*
       * if we're in codec mode and this is the first
       * packet after syncing up, prepend the sync byte(s)
       * that the 8530 stripped.
       */
      if (sc->sc_codec.datalen) {
       m->m_len = sfifo;
       op = mtod(m, u_char *);
       if (sfifo < sc->sc_codec.datalen) {
        if (sc->sc_codec.synclen == 8) {
         m->m_len += 1;
         m->m_off -= 1;
         op[-1] = sc->sc_codec.syncpat;
        } else {
         m->m_len += 2;
         m->m_off -= 2;
         op[-2] = sc->sc_codec.syncpat;
         op[-1] = sc->sc_codec.syncpat >> 8;
        }
       }
      } else {
       if (! sc->sc_raw)
        m->m_off -= PPP_HDRSPACE;
       m->m_len = sfifo - 2;
       op = mtod(m, u_char *);
      }
      sc->sc_dstats.ssd_ichar += sfifo;
      ++sc->sc_dstats.ssd_ipack;
      ++sc->sc_if.if_ipackets;
      while (sfifo) {
       if (buflen <= 0) {
        bp = sc->sc_inq;
        if (bp == NULL) {
         /*
          * nothing on input queue - probably
          * had an overrun at hardware intr
          * level. just bail.
          */
         m_freem(m);
         return;
        }
        buflen = bp->cnt;
        bufoff = BUFtoCP(bp);
       }
       if (buflen > sfifo) {
```

```
    bcopy(bufoff, op, sfifo);
    buflen -= sfifo;
    bufoff += sfifo;
    break;
   } else {
    register int s;
    register struct hsisbuf *bp;

    bcopy(bufoff, op, buflen);
    sfifo -= buflen;
    op += buflen;
    buflen = 0;
    bufoff = NULL;

    s = splboard();
    if ((bp = sc->sc_inq) == NULL) {
     splx(s);
     m_freem(m);
     return;
    }
    if ((sc->sc_inq = bp->next) == NULL)
     sc->sc_intail = NULL;
    bp->next = hs->hs_free;
    hs->hs_free = bp;
    splx(s);
   }
  }
  /*
   * queue the packet to the appropriate network protocol.
   */
#if NBPFILTER > 0
  if (sc->sc_bpf) {
   register u_char c;

   op = mtod(m, u_char *);
   c = *op;
   *op = 0; /* 'in' direction */
   bpf_tap(sc->sc_bpf, op, m->m_len);
   *op = c;
  }
#endif
  if (sc->sc_raw) {
   register struct socket *so = sc->sc_raw;

   if (m->m_len > sbspace(&so->so_rcv)) {
    ++sc->sc_idrops;
    m_freem(m);
   } else {
    register struct mbuf *n = so->so_rcv.sb_mb;

    if (n) {
     while (n->m_next)
```

```
            n = n->m_next;
            n->m_next = m;
           } else
            so->so_rcv.sb_mb = m;

           so->so_rcv.sb_cc += m->m_len;
           sorwakeup(so);
          }
         } else {
          register struct ifqueue *inq;
          register int s;
#ifdef STII
          struct ifqueue *st2_fromhsis();
          int st2_intr();
#endif STII

          /*
           * Note: the protocol input routines all require
           * an ifp at the front of the buffer. We make
           * use of the fact that a PPP header is the same
           * size as an ifp & just overwrite the 4 bytes
           * of header without allocating new space.
           */
          s = splimp();
          op = mtod(m, u_char *);
          switch (*(u_int *)op) {

          case PPP_INET:
           inq = &ipintrq;
           schednetisr(NETISR_IP);
           break;
#ifdef STII
          case PPP_SCMP:
          case PPP_STII:
           inq = st2_fromhsis( &sc->sc_if, m, op );
           if ( ! inq ) {
            splx(s);
            sc->sc_ierrcnt = 0;
            continue;
           }
           softcall(st2_intr, (caddr_t)0);
           break;
#endif STII
          default:
           splx(s);
           ++sc->sc_ibadtype;
           m_freem(m);
           continue;
          }
          if (IF_QFULL(inq)) {
           ++sc->sc_idrops;
           m_freem(m);
```

```
    } else {
     *(struct ifnet **)op = &sc->sc_if;
     IF_ENQUEUE(inq, m);
    }
    splx(s);
   }
   sc->sc_ierrcnt = 0;
  }
  /*
   * done with fifo -- if there's data left in the current buffer,
   * remember where we are.
   */
  if (bufoff && (bp = sc->sc_inq))
   bp->cnt = buflen;
  sc->sc_inoff = bufoff;
}

void
hsis_xmit_done(sc)
 register struct hsis_softc *sc;
{
 register int len;
 register int s;

 /*
  * if there's more in the snd q, copy another packet to the
  * on-board 'next' buffer.
  */
 s = splboard();    /* %%% necessary? */
#ifndef SFQ_VC
 if (sc->sc_if.if_snd.ifq_len)
#endif
 {
  register struct mbuf *m;
#ifdef TRAFFIC_CONTROL
  struct aNetIf *gifcp;
  struct mbuf *(*dqfuncp) ();

  /* Map ifnet pointer to extended ifnet (aNetIf) */

  gifcp = tcif_ifp2gifcp( (unsigned long) &( sc->sc_if ) );

  /* If no specific dequeue function is supplied, use specified queue */

  if ( (dqfuncp = gifcp->iftc_dq) != (struct mbuf * (*)()) NULL )
   m = (*dqfuncp) ( gifcp ); /* Use specific dequeue function */
  else    /* Use generic dequeue function */
#endif TRAFFIC_CONTROL
  IF_DEQUEUE(&sc->sc_if.if_snd, m);

  if (m) {
   register struct hsisbuf *bp = sc->sc_nextout;
```

```
    bp->cnt = len = copy_m_to_b(sc, m, BUFtoCP(bp));
    TRACE(T_OUT_COPY, sc, (int)bp | len)
   }
  }
  splx(s);
}

int
hsissoftint()
{
 register u_int isum;
 register int did_something = 0;

 TRACE(T_SOFTINT, hsis_softc, hsis_isum)
 while (1) {
  register struct hsis_softc *sc;
  register struct hsis_softc *sc_end = hsis_softc_end;
  register int s;
  register u_int event;

  s = splboard();
  isum = hsis_isum;
  hsis_isum = 0;
  splx(s);
  if (isum == 0)
   break;

  ++did_something;
  sc = hsis_softc;
  for ( ; isum && sc < sc_end; ++sc) {
   if (isum & 1) {
    s = splboard();
    event = sc->sc_event;
    sc->sc_event = 0;
    splx(s);

    if (event & XMIT_DONE) {
     s = splboard();
     hsis_start(sc);
     splx(s);
    }
    if (event & RECV_DONE)
     hsis_recv_done(sc);
    if (event & XMITDMA_DONE)
     hsis_xmit_done(sc);
   }
   isum >>= 1;
  }
 }
 TRACE(T_SOFTINT_EXIT, hsis_softc, did_something)
 return (did_something);
```

```
}

int
hsis_ioctl(ifp, cmd, data)
 register struct ifnet *ifp;
 register int cmd;
 register caddr_t data;
{
 register int unit = ifp->if_unit;
 register struct hsis_softc *sc = &hsis_softc[unit];
 register struct ifreq *ifr = (struct ifreq *)data;
 register int s = splboard(), error = 0;

 TRACE(T_IOCTL, sc, cmd)
 switch (cmd) {

 case SIOCSIFADDR:
  bzero((caddr_t)&sc->sc_dstats, sizeof(sc->sc_dstats));
  bzero((caddr_t)&sc->sc_estats, sizeof(sc->sc_estats));
  error = hsis_init(unit);
  break;

 case SIOCSIFDSTADDR:
  break;

 case SIOCSIFFLAGS:
  switch (ifp->if_flags & (IFF_UP|IFF_RUNNING)) {

  case IFF_UP:
   /* down interface just marked up */
   bzero((caddr_t)&sc->sc_dstats, sizeof(sc->sc_dstats));
   bzero((caddr_t)&sc->sc_estats, sizeof(sc->sc_estats));
   error = hsis_init(unit);
   hsis_start(sc);
   break;

  case IFF_RUNNING:
   /* up interface just marked down */
   hsis_offline(unit);
   break;
  }
  break;

 case SIOCSSDSTATS:
  *(struct ss_dstats *)ifr->ifr_data = sc->sc_dstats;
  break;

 case SIOCSSESTATS:
  *(struct ss_estats *)ifr->ifr_data = sc->sc_estats;
  break;

 case SIOCGETSYNC:
```

```
        *(struct syncmode *)ifr->ifr_data = sc->sc_sm;
        break;

       case SIOCSETSYNC:
        if (ifp->if_flags & IFF_RUNNING)
         hsis_offline(unit);
        sc->sc_sm = *(struct syncmode *)ifr->ifr_data;
        if (sc->sc_sm.sm_baudrate != 0) {
         bzero((caddr_t)&sc->sc_dstats, sizeof(sc->sc_dstats));
         bzero((caddr_t)&sc->sc_estats, sizeof(sc->sc_estats));
         error = hsis_init(unit);
         hsis_start(sc);
        }
        break;

       case SIOCSETCODEC:
        if (! sc->sc_raw)
         /* must be raw socket to use codec mode */
         error = EINVAL;
        else {
         struct codecmode *cm =
          (struct codecmode *)ifr->ifr_data;

         /*
          * Note that sc->sc_codec.datalen != 0 is the
          * driver's internal flag indicating codec mode.
          */

         if ((u_int)cm->datalen > HSIS_MTU ||
          sc->sc_sm.sm_baudrate == 0 ||
          (cm->synclen != 8 && cm->synclen != 16))
          error = EINVAL;
         else {
          if (ifp->if_flags & IFF_RUNNING)
           hsis_offline(unit);
          sc->sc_codec = *cm;
          sc->sc_ticks = hz * cm->datalen * 8 /
           sc->sc_sm.sm_baudrate;
          if (sc->sc_ticks == 0)
           sc->sc_ticks = 1;
          error = hsis_init(unit);
         }
        }
        break;

      #ifdef MULTICAST
       case SIOCADDMULTI:
       case SIOCDELMULTI:
        switch (ifr->ifr_addr.sa_family) {
      #ifdef INET
        case AF_INET:
         break;
```

```
#endif INET
  default:
   error = EAFNOSUPPORT;
   break;
  }
  break;
#endif MULTICAST

 default:
  error = EINVAL;
 }
 splx(s);
 return (error);
}

int
hsis_init(unit)
 int unit;
{
 register struct hsis_softc *sc = &hsis_softc[unit];
 register struct zscc *zs = sc->sc_zs;
 register u_char *zsdev = zs->zs_addr;
 register u_char *zddev = sc->sc_zd->zd_addr;
 u_char wreg[sizeof(zs->zs_wreg)];
 register int s;
 register int clk = 0, txin = 0;

 TRACEL(T_INIT, sc, sc->sc_if.if_flags)
 if (sc->sc_if.if_flags & IFF_RUNNING)
  return (0);

 /* set appropriate defaults for scc */
 if (sc->sc_codec.datalen)
  bcopy((caddr_t)hsis_scc_codec_setup, (caddr_t)wreg, sizeof(wreg));
 else
  bcopy((caddr_t)hsis_scc_setup, (caddr_t)wreg, sizeof(wreg));

 /*
  * modify defaults as per Sun's 'syncmode' (hsi compatibility).
  */
 switch (sc->sc_sm.sm_txclock) {

 case TXC_IS_TXC:
  wreg[11] |= ZSWR11_TXCLK_TRXC;
  txin = 1;
  break;

 case TXC_IS_RXC:
  wreg[11] |= ZSWR11_TXCLK_RTXC;
  break;

 case TXC_IS_BAUD:
```

```
 wreg[11] |= ZSWR11_TXCLK_BAUD;
 clk = 1;
 break;

case TXC_IS_PLL:
 wreg[11] |= ZSWR11_TXCLK_DPLL;
 clk = 2;
 break;

default:
 return (EINVAL);
}
switch (sc->sc_sm.sm_rxclock) {

case RXC_IS_RXC:
 wreg[11] |= ZSWR11_RXCLK_RTXC;
 break;

case RXC_IS_TXC:
 wreg[11] |= ZSWR11_RXCLK_TRXC;
 txin = 1;
 break;

case RXC_IS_BAUD:
 wreg[11] |= ZSWR11_RXCLK_BAUD;
 clk |= 1;
 break;

case RXC_IS_PLL:
 wreg[11] |= ZSWR11_RXCLK_DPLL;
 clk |= 2;
 break;

default:
 return (EINVAL);
}
if (clk) {
 register long tconst;

 if (clk > 2)
  return (EINVAL);
 tconst = sc->sc_sm.sm_baudrate;
 if (clk == 2) {
  if (! sc->sc_sm.sm_nrzi)
   return (EINVAL);
  tconst <<= 5;
 }
 tconst = HSIS_PCLK / (tconst * 2) - 2;
 if (tconst == 0)
  return (EINVAL);

 sc->sc_sm.sm_baudrate = (HSIS_PCLK / 2) / (tconst + 2);
```

```
 wreg[12] = tconst;
 wreg[13] = tconst >> 8;
 wreg[14] |= ZSWR14_BAUD_FROM_PCLK | ZSWR14_BAUD_ENA;
}
if (! txin)
 wreg[11] |= ZSWR11_TRXC_OUT_ENA | ZSWR11_TRXC_XMIT;
if (sc->sc_sm.sm_loopback)
 wreg[14] |= ZSWR14_LOCAL_LOOPBACK;
if (sc->sc_sm.sm_nrzi)
 wreg[10] |= ZSWR10_NRZI;

/*
 * if we don't have read and write buffers yet, get them.
 */
s = splboard();
if (sc->sc_inbuf == NULL) {
 if ((sc->sc_inbuf = sc->sc_hs->hs_free) == NULL) {
  printf("hsis%d: hsis_init: no free bufs.\n",
   sc->sc_if.if_unit);
  splx(s);
  return (ENOBUFS);
 }
 sc->sc_hs->hs_free = sc->sc_inbuf->next;
}
if (sc->sc_curout == NULL) {
 if ((sc->sc_curout = sc->sc_hs->hs_free) == NULL) {
  printf("hsis%d: hsis_init: no free bufs\n",
   sc->sc_if.if_unit);
  splx(s);
  return (ENOBUFS);
 }
 sc->sc_hs->hs_free = sc->sc_curout->next;
 sc->sc_curout->cnt = 0;
}
if (sc->sc_nextout == NULL) {
 if ((sc->sc_nextout = sc->sc_hs->hs_free) == NULL) {
  printf("hsis%d: hsis_init: no free bufs\n",
   sc->sc_if.if_unit);
  splx(s);
  return (ENOBUFS);
 }
 sc->sc_hs->hs_free = sc->sc_nextout->next;
 sc->sc_nextout->cnt = 0;
}

/*
 * Set sync mode and sync pattern for codec mode.
 */
if (sc->sc_codec.datalen) {
 wreg[6] = sc->sc_codec.syncpat & 0xFF;
 if (sc->sc_codec.synclen == 16) {
```

```
    wreg[4] |= ZSWR4_BISYNC;
    wreg[7] = (sc->sc_codec.syncpat >> 8) & 0xFF;
   }
  }

  /*
   * If we got here, sm contents must be reasonable and wreg contains
   * the new scc configuration. Disable rcvr & xmitter, load in new
   * modes then re-enable.
   */

  sc->sc_hs->hs_board[zs->zs_unit + HSIS_INT_CLK_A] = 0;
#ifdef HSIS_EXTERNAL_RCVDONE
  /*
   * if there's an external receive clock, use the external
   * done interrupt. Otherwise, enable the chip's receive intr.
   */
  if (!sc->sc_codec.datalen) {
   if (sc->sc_sm.sm_rxclock == RXC_IS_RXC)
    sc->sc_hs->hs_board[zs->zs_unit + HSIS_EINT_ENA_A] = 0xff;
   else {
    sc->sc_hs->hs_board[zs->zs_unit + HSIS_EINT_ENA_A] = 0;
    wreg[1] |= ZSWR1_RIE_SPECIAL_ONLY;
   }
  }
#endif
  SCC_BIC(3, ZSWR3_RX_ENABLE);
  SCC_BIC(5, ZSWR5_TX_ENABLE);

  SCC_WRITE(4, wreg[4]);
  SCC_WRITE(10, wreg[10]);
  SCC_WRITE(6, wreg[6]);
  SCC_WRITE(7, wreg[7]);
  SCC_WRITE(3, wreg[3]);
  SCC_WRITE(5, wreg[5]);
  SCC_WRITE(1, wreg[1]);
  SCC_WRITE(9, wreg[9]);
  SCC_WRITE(11, wreg[11]);
  SCC_WRITE(12, wreg[12]);
  SCC_WRITE(13, wreg[13]);
  if (clk == 2) {
   SCC_WRITE(14, ZSWR14_DPLL_SRC_BAUD);
   SCC_WRITE(14, ZSWR14_DPLL_NRZI);
  } else
   SCC_WRITE(14, ZSWR14_DPLL_DISABLE);
  SCC_WRITE(14, wreg[14]);
  SCC_WRITE(15, wreg[15]);

  if (txin) {
   /*
    * TRxC pin is input (i.e., we're using an external clock).
    * Set latch that allows clock to get to the pin.
```

```
 */
 register u_char *board = sc->sc_hs->hs_board + zs->zs_unit;

 board[HSIS_INT_CLK_A] = 1;
 /* XXX - ext. clocks are inverted */
 board[HSIS_RX_CLK_A] = 1;
 board[HSIS_TX_CLK_A] = 1;
}
SCC_WRITE0(ZSWR0_RESET_ERRORS);
SCC_WRITE0(ZSWR0_RESET_STATUS);
SCC_WRITE0(ZSWR0_RESET_TXCRC);

/*
 * Everything should be configured. Start a dma read then
 * enable recv dma and the receiver.
 *
 * For codec mode, first dma will be missing sync bytes -
 * set first readlen to account for this, start dma for
 * first read then enable receiver (which puts us in
 * hunt mode). readlen will be reset by hsis_dmarecv_time
 * to the full blocksize after first block is read.
 */
if (sc->sc_codec.datalen) {
 sc->sc_readlen = sc->sc_codec.datalen - sc->sc_codec.synclen/8;
 sc->sc_bytesread = 0;
 (void)hsis_start_dma_read(sc);
 if (unit & 1) {
  DMA_BIS(ZSDMA_ICR, ZSDMA_ICR_TX_B_ENA);
 } else {
  DMA_BIS(ZSDMA_ICR, ZSDMA_ICR_TX_A_ENA);
 }
} else {
 (void)hsis_start_dma_read(sc);
}
SCC_BIS(1, ZSWR1_REQ_ENABLE)
SCC_BIS(3, ZSWR3_RX_ENABLE);
/*
 * Enable the transmitter and turn on DTR.
 */
SCC_BIS(5, ZSWR5_TX_ENABLE|ZSWR5_DTR);

SCC_WRITE0(ZSWR0_RESET_ERRORS);
SCC_WRITE0(ZSWR0_RESET_STATUS);
SCC_WRITE0(ZSWR0_RESET_TXCRC);

sc->sc_ostate = 0;
sc->sc_if.if_flags |= IFF_UP|IFF_RUNNING;
TRACE(T_INIT_DONE, sc, *zsdev)
splx(s);
return (0);
}
```

```
hsis_reset(unit)
 register int unit;
{
 register int s = splboard();
 register struct hsis_softc *sc = &hsis_softc[unit];
 register struct zscc *zs = sc->sc_zs;
 register u_char *zsdev = zs->zs_addr;
 register u_char *zddev = sc->sc_zd->zd_addr;
 register struct hsisbuf *bp;

 TRACE(T_RESET, sc, *zsdev)
 SCC_WRITE(3, 0);
 SCC_WRITE(5, 0);
 SCC_WRITE(15, 0);
#ifdef HSIS_EXTERNAL_RCVDONE
 sc->sc_hs->hs_board[zs->zs_unit + HSIS_EINT_ENA_A] = 0;
#endif
 untimeout(hsis_dmarecv_time, sc);
 (void) hsis_start_dma_read(sc);
 if (zs->zs_unit & 1) {
  DMA_BIC(ZSDMA_DER, ZSDMA_DER_RX_B_ENABLE);
  DMA_BIC(ZSDMA_ICR, ZSDMA_ICR_TX_B_ENA);
 } else {
  DMA_BIC(ZSDMA_DER, ZSDMA_DER_RX_A_ENABLE);
  DMA_BIC(ZSDMA_ICR, ZSDMA_ICR_TX_A_ENA);
 }
 (void) hsis_start_dma_read(sc);
 if (sc->sc_intail) {
  sc->sc_intail->next = sc->sc_hs->hs_free;
  sc->sc_hs->hs_free = sc->sc_inq;
  sc->sc_inq = NULL;
  sc->sc_intail = NULL;
 }
 bzero(sc->sc_fifo, sizeof(sc->sc_fifo));
 sc->sc_fifop = sc->sc_fifo;
 sc->sc_inoff = NULL;

 sc->sc_ostate = 0;
 if (bp = sc->sc_curout)
  bp->cnt = 0;
 if (bp = sc->sc_nextout)
  bp->cnt = 0;
 sc->sc_if.if_flags &=~ (IFF_UP|IFF_RUNNING);
 splx(s);
}

hsis_offline(unit)
 register int unit;
{
 register int s = splboard();
 register struct hsis_softc *sc = &hsis_softc[unit];
 register struct hsisbuf *bp;
```

```
 /* reset channel, then free any resources it holds */

 hsis_reset(unit);
 if (bp = sc->sc_inbuf) {
  sc->sc_inbuf = NULL;
  bp->next = sc->sc_hs->hs_free;
  sc->sc_hs->hs_free = bp;
 }
 if (bp = sc->sc_curout) {
  sc->sc_curout = NULL;
  bp->next = sc->sc_hs->hs_free;
  sc->sc_hs->hs_free = bp;
 }
 if (bp = sc->sc_nextout) {
  sc->sc_nextout = NULL;
  bp->next = sc->sc_hs->hs_free;
  sc->sc_hs->hs_free = bp;
 }
 splx(s);
}

/*
 * Remainder of this code is support for 'raw sync' protocol domain.
 * It should probably be in a separate file since it (should not be)
 * hsis specific but I'm too lazy to set that up just now.
 */

#include <sys/protosw.h>
#include <sys/domain.h>
#include <sys/user.h>
#include <sys/uio.h>

int rawsync_usrsend(), rawsync_usrrecv(), rawsync_usrreq();

extern struct domain rawsyncdomain;

struct protosw rawsyncsw[] = {
{ SOCK_RAW, &rawsyncdomain, 0,  PR_ATOMIC,
 0,  0,  0,  0,
 rawsync_usrreq,
 0,  0,  0,  0,
 rawsync_usrsend, rawsync_usrrecv},
};

struct domain rawsyncdomain =
 { AF_RAWSYNC, "rawsync", 0, 0, 0,
 rawsyncsw, &rawsyncsw[sizeof(rawsyncsw)/sizeof(rawsyncsw[0])] };

void
rawsyncinit()
{
```

```
 register struct domain *dp;

 rawsyncdomain.dom_next = domains;
 domains = &rawsyncdomain;
}

int
rawsync_usrsend(so, nam, uio, flags, rights)
 register struct socket *so;
 struct mbuf *nam;
 register struct uio *uio;
 int flags;
 struct mbuf *rights;
{
 register int len;
 register int error = 0;
 register int s;
 register struct hsis_softc *sc;
 register struct mbuf *m;
 static struct sockaddr dst = { AF_UNSPEC };

#ifdef lint
 nam = nam; rights = rights;
#endif lint
 if ((sc = (struct hsis_softc *)so->so_pcb) == NULL ||
  so != sc->sc_raw)
  return (EINVAL);

 len = uio->uio_resid;
 if (len <= 0 || len > sc->sc_if.if_mtu)
  return (EMSGSIZE);

 if (so->so_state & SS_CANTSENDMORE) {
  psignal(u.u_procp, SIGPIPE);
  return (EPIPE);
 }
 while (1) {
  s = splboard();
  if (! IF_QFULL(&sc->sc_if.if_snd))
   break;

  sbwait(&so->so_snd);
  splx(s);
 }
 (void) splnet();
 MGET(m, M_DONTWAIT, MT_DATA);
 if (m == (struct mbuf *)0) {
  error = ENOBUFS;
  goto out;
 }
 if (len <= MLEN - MBUF_HDRSPACE)
  m->m_off += MBUF_HDRSPACE;
```

```
 else {
  /* too big for mbuf - use cluster */
  MCLGET(m);
  if (m->m_len != MCLBYTES) {
   /* no clusters - drop this packet */
   m_freem(m);
   error = ENOBUFS;
   goto out;
  }
  m->m_off += HDRSPACE;
 }
 m->m_len = len;
 error = uiomove(mtod(m, caddr_t), len, UIO_WRITE, uio);
 if (! error)
  error = hsis_output(&sc->sc_if, m, &dst);
out:
 splx(s);
 return (error);
}

int
rawsync_usrrecv(so, anam, uio, flags, arights)
 register struct socket *so;
 struct mbuf **anam;
 register struct uio *uio;
 int flags;
 struct mbuf **arights;
{
 register int len, mlen;
 register int error = 0;
 register int s;
 register struct hsis_softc *sc;
 register struct mbuf *m, *n;
 register struct sockbuf *sb;

 if (anam)
  *anam = NULL;
 if (arights)
  *arights = NULL;

 if ((sc = (struct hsis_softc *)so->so_pcb) == NULL ||
  so != sc->sc_raw)
  return (EINVAL);

 len = uio->uio_resid;
 if (len <= 0) {
  error = EMSGSIZE;
  goto out;
 }
 if (so->so_state & SS_CANTRCVMORE)
  goto out;
```

```
 sb = &so->so_rcv;
 sblock(so, sb);
 s = splboard();
 if (sb->sb_cc == 0) {
  if (so->so_error) {
   error = so->so_error;
   so->so_error = 0;
   goto release;
  }
  if (so->so_state & SS_NBIO) {
   error = EWOULDBLOCK;
   goto release;
  }
  while (sb->sb_cc == 0) {
   sbunlock(so, sb);
   sbwait(sb);
   if (so->so_error) {
    error = so->so_error;
    so->so_error = 0;
    goto release;
   }
  }
 }
 m = sb->sb_mb;
 sb->sb_mb = m->m_next;
 mlen = m->m_len;
 sb->sb_cc -= mlen;
 splx(s);
 sbunlock(so, sb);
 m->m_next = NULL;
 if (mlen > len) {
  error = EMSGSIZE;
  mlen = len;
 }
 error = uiomove(mtod(m, caddr_t), mlen, UIO_READ, uio);
 MFREE(m, n);
out:
 return (error);
release:
 splx(s);
 sbunlock(so, sb);
 return (error);
}

/*ARGSUSED*/
rawsync_usrreq(so, req, m, nam, rights)
 struct socket *so;
 int req;
 struct mbuf *m, *nam, *rights;
{
 register struct hsis_softc *sc;
 register int error = 0;
```

```
if (req == PRU_CONTROL) {
 int cmd = (int)m;
 caddr_t data = (caddr_t)nam;
 register struct ifnet *ifp = (struct ifnet *)rights;

 if (ifp == 0 || ifp->if_ioctl == 0)
  return (EOPNOTSUPP);
 return ((*ifp->if_ioctl)(ifp, cmd, data));
}

if (rights && rights->m_len) {
 error = EOPNOTSUPP;
 goto release;
}
sc = (struct hsis_softc *)so->so_pcb;
if (sc != NULL && so != sc->sc_raw) {
 error = EINVAL;
 goto release;
}
switch (req) {

case PRU_ATTACH:
 if ((so->so_state & SS_PRIV) == 0)
  error = EACCES;
 else if (sc)
  error = EINVAL;
 else if (sbreserve(&so->so_snd, 4096) == 0)
  error = ENOBUFS;
 else if (sbreserve(&so->so_rcv, 4096) == 0) {
  sbrelease(&so->so_snd);
  error = ENOBUFS;
 }
 break;

/*
 * Destroy state just before socket deallocation.
 * Flush data or not depending on the options.
 */
case PRU_DETACH:
 if (sc == 0)
  error = ENOTCONN;
 else {
  register int s = splboard();

  if (sc->sc_codec.datalen) {
   /* turn off codec mode if left on */
   hsis_offline(sc->sc_if.if_unit);
   sc->sc_codec.datalen = 0;
  }
  sc->sc_raw = NULL;
  so->so_pcb = NULL;
```

```
        sofree(so);
        splx(s);
       }
       break;

      case PRU_BIND:
       if (sc)
        error = EISCONN;
       else {
        register struct sockaddr *addr =
          mtod(nam, struct sockaddr *);
        if (addr->sa_family != AF_RAWSYNC) {
         error = EINVAL;
         break;
        }
        sc = (struct hsis_softc *)ifunit(addr->sa_data, MLEN);
        if (sc == NULL)
         error = EADDRNOTAVAIL;
        else if (sc->sc_raw)
         error = EADDRINUSE;
        else {
         register int s = splboard();

         sc->sc_raw = so;
         so->so_pcb = (caddr_t)sc;
         splx(s);
        }
       }
       break;

      /*
       * Mark the connection as being incapable of further input.
       */
      case PRU_SHUTDOWN:
       socantsendmore(so);
       break;

      case PRU_ABORT:
       if (sc != NULL && (so->so_state & SS_NOFDREF)) {
        register int s = splboard();

        if (sc->sc_codec.datalen) {
         /* turn off codec mode if left on */
         hsis_offline(sc->sc_if.if_unit);
         sc->sc_codec.datalen = 0;
        }
        sc->sc_raw = NULL;
        so->so_pcb = NULL;
        splx(s);
       }
       sofree(so);
       soisdisconnected(so);
```

```
 break;

 case PRU_SENSE:
  /*
   * stat: don't bother with a blocksize.
   */
  return (0);

 /*
  * Not supported.
  */

 case PRU_CONNECT:
 case PRU_CONNECT2:
 case PRU_DISCONNECT:

 case PRU_RCVOOB:
 case PRU_RCVD:

 case PRU_LISTEN:
 case PRU_ACCEPT:
 case PRU_SENDOOB:

 case PRU_SOCKADDR:
 case PRU_PEERADDR:
  error = EOPNOTSUPP;
  break;

 default:
  panic("rawsync_usrreq");
 }
release:
 if (m != NULL)
  m_freem(m);
 return (error);
}

#endif
```

```
#ifndef lint
static char rcsid_st2_proto_c[] = "\
@(#) $Header: st2_proto.c,v 1.98+ 93/04/08 18:00:00 clynn Exp $ \n";
/*--------------------------------------------------------------------*
 | Copyright (c) 1991-1993 by BBN Systems and Technologies,           |
 | A Division of Bolt Beranek and Newman Inc.                         |
 |                                                                    |
 | Permission to use, copy, modify, distribute, and sell this         |
 | software and its documentation for any purpose is hereby           |
 | granted without fee, provided that the above copyright notice      |
 | and this permission appear in all copies and in supporting         |
 | documentation, and that the name of Bolt Beranek and Newman        |
 | Inc. not be used in advertising or publicity pertaining to         |
 | distribution of the software without specific, written prior       |
 | permission. BBN makes no representations about the suitability     |
 | of this software for any purposes. It is provided "AS IS"          |
 | without express or implied warranties.                             |
 *--------------------------------------------------------------------*/
#endif lint


/*
M* st2_proto.c ST-II Configuration and Site-dependent Routines.
M*
M* This module contains the Configuration Databases and Initialization
M* Routines for ST-II. It also serves as a common point for access to
M* and from the underlying OS.
M*
 */

/*
m* Status:
m* Features:
m* IP Encapsulation.
m* Untested Features:
m* Portability: nothing's portable until you've ported it!
m* Restrictions/Bugs:
m* Some interaction with DARTNet mbufs causes them to be dropped;
m* offending code in st2_fromXXX disabled.
m* Things to do:
m* Hooks for other protocols over ST-II.
m* ST-II from IP (version 5).
m*
 */


/* Module Revision History
 *
 * $Log: st2_proto.c,v $
 * Revision 1.11 92/04/03 18:31:23 clynn
 * Release for DARTNet. Virtual Clock enforcement & related changes.
 *
```

```
* Revision 1.10 91/11/26 23:12:12 clynn
* Make DEF_TIMEOUTFACTOR independent of STIIDEBUG option.
*
* Revision 1.9 91/11/04 09:22:28 clynn
* Updated for Public Domain Release. Major changes: addition of Source
* Routing, IP Encapsulation, HELLO protocol between neighbors, tracking
* of neighbors, detection of component or agent failures & notification
* to applications. More consistant naming and format, including making
* all external names begin with "st2_". Moved some routines and data
* structures to reduce external references.
*
* Simplified since everybody gets the full source.
*
* Revision 1.8 91/05/28 16:17:55 clynn
* New features: Add new targets from Application layer, UserData support,
* basic bandwidth reservation for point-to-point links, more complete
* state tables, added pcode parameter to InitaFlowSpec3, extended packet
* buffer abstraction, added network interface abstraction; ststat utility.
* Bug fixes: ADDR IN USE problem, data send problem, causes of some
* crashes, cleanup of protocol control blocks.
* Work around: DARTNET receive memory leak.
* Eliminated several small modules to reduce globals; adeded Makefile.
*
* Revision 1.1 91/03/15 18:33:26 clynn
* Initial revision
*/

#define _FILE_ Strpro   /* Module bugid 7,13 */



/* External definitions */

#include <errno.h>  /* ENODEV EOPNOTSUPP */
#include <sys/param.h>  /* includes <machine/param.h> */
#include <sys/mbuf.h>  /* needs sys/types.h & sys/param.h */
#include <sys/protosw.h>
#include <sys/domain.h>
#include <sys/time.h>  /* struct timeval */
#include <sys/types.h>  /* for netinet/in.h; caddr_t u_xxx for
    net/if.h */

#include <sys/user.h>  /* u. */

#include <sys/socket.h>  /* sockaddr for net/if.h */
#include <netinet/in.h>  /* before st2.h -> st2_api.h; IPPROTO_* ,
    in_addr */
#include <net/if.h>  /* ifqueue */


#include "st2_api.h"  /* Global Definitions, before st2.h */
#include "st2.h"  /* ST-II Implementation definitions,
```

```
     * includes st2_api.h, before st2_cmp.h */


#define NUM_NEXTHOP 64 /* number of nexthops for all streams */
   /* (used in st2_cmp.h) */

#include "st2_cmp.h"  /* SCM Protocol definitions */
#include "st2_resource.h" /* RM_*, aRmVectorList, TC_*, aTcVectorList */


extern int   arpresolve (/* acp, mbufp */);
extern void   st2_RsrcInit (/* */);

#ifdef TRAFFIC_CONTROL
extern ExpAry (a,NetIf,1) tfic_genifs;
extern struct aNetIf *tcif_gifcheadp;
#endif TRAFFIC_CONTROL

/* Local routines */

static void   init_STTables (/* */);
 void   st2_dbgstp (/* stopflag, errcode, unique_id, string,
    file_id, line_number */);
static void   st2_at_check ();
static void   st2_at_stop ();
static int   st2_ClkFast (/* */);
static struct ifqueue *st2_fromether (/* ifnetp, pktp, ehp */);
 unsigned char *st2_GetSpace (/* pktpp, a1, a2, a3 */);
 int   st2_init (/* */);
 boolean   st2_IPAdrFunc (/* fnc,ipadrp */);
static int   st2_toether (/* sockaddrp, pktp, ifnetp, eap */);



#include <sys/ioccom.h>  /* _IOC*, SIOCxxx */
#if 0
#include <sys/types.h>  /* for sys/socket.h */
#include "sys/socket.h"  /* for net/if.h */
#include "net/if.h"  /* IFNAMSIZ */

#ifndef TRAFFIC_CONTROL
#ifdef DOCUMENTATION
<sys/ioccom.h>

#define _IOCPARM_MASK 0xff  /* parameters must be < 256 bytes */
#define _IOC_VOID 0x20000000 /* no parameters */
#define _IOC_OUT 0x40000000 /* copy out parameters */
#define _IOC_IN  0x80000000 /* copy in parameters */
#define _IOC_INOUT (_IOC_IN|_IOC_OUT)
/* the 0x20000000 is so we can distinguish new ioctl's from old */
#define _IO(x,y) (_IOC_VOID|('x'<<8)|y)
```

```
#define _IOR(x,y,t) (_IOC_OUT|((sizeof(t)&_IOCPARM_MASK)<<16)|('x'<<8)|y)
#define _IORN(x,y,t) (_IOC_OUT|(((t)&_IOCPARM_MASK)<<16)|('x'<<8)|y)
#define _IOW(x,y,t) (_IOC_IN|((sizeof(t)&_IOCPARM_MASK)<<16)|('x'<<8)|y)
#define _IOWN(x,y,t) (_IOC_IN|(((t)&_IOCPARM_MASK)<<16)|('x'<<8)|y)
/* this should be _IORW, but stdio got there first */
#define _IOWR(x,y,t) (_IOC_INOUT|((sizeof(t)&_IOCPARM_MASK)<<16)|('x'<<8)|y)
#define _IOWRN(x,y,t) (_IOC_INOUT|(((t)&_IOCPARM_MASK)<<16)|('x'<<8)|y)

<sys/sockio.h>
/* socket i/o controls */
#define SIOCSHIWAT _IOW(s, 0, int)  /* set high watermark */
#define SIOCGHIWAT _IOR(s, 1, int)  /* get high watermark */
#define SIOCSLOWAT _IOW(s, 2, int)  /* set low watermark */
#define SIOCGLOWAT _IOR(s, 3, int)  /* get low watermark */
#define SIOCATMARK _IOR(s, 7, int)  /* at oob mark? */
#define SIOCSPGRP _IOW(s, 8, int)  /* set process group */
#define SIOCGPGRP _IOR(s, 9, int)  /* get process group */

#define SIOCADDRT _IOW(r, 10, struct rtentry) /* add route */
#define SIOCDELRT _IOW(r, 11, struct rtentry) /* delete route */
#define SIOCSETRTINFO _IOWR(r, 12, struct fullrtentry) /* change aux info */
#define SIOCGETRTINFO _IOWR(r, 13, struct fullrtentry) /* read aux info */

#define SIOCSIFADDR _IOW(i, 12, struct ifreq) /* set ifnet address */
#define SIOCGIFADDR _IOWR(i,13, struct ifreq) /* get ifnet address */
#define SIOCSIFDSTADDR _IOW(i, 14, struct ifreq) /* set p-p address */
#define SIOCGIFDSTADDR _IOWR(i,15, struct ifreq) /* get p-p address */
#define SIOCSIFFLAGS _IOW(i, 16, struct ifreq) /* set ifnet flags */
#define SIOCGIFFLAGS _IOWR(i,17, struct ifreq) /* get ifnet flags */
#define SIOCSIFMEM _IOW(i, 18, struct ifreq) /* set interface mem */
#define SIOCGIFMEM _IOWR(i,19, struct ifreq) /* get interface mem */
#define SIOCGIFCONF _IOWR(i,20, struct ifconf) /* get ifnet list */
#define SIOCSIFMTU _IOW(i, 21, struct ifreq) /* set if_mtu */
#define SIOCGIFMTU _IOWR(i,22, struct ifreq) /* get if_mtu */

 /* from 4.3BSD */
#define SIOCGIFBRDADDR _IOWR(i,23, struct ifreq) /* get broadcast addr */
#define SIOCSIFBRDADDR _IOW(i,24, struct ifreq) /* set broadcast addr */
#define SIOCGIFNETMASK _IOWR(i,25, struct ifreq) /* get net addr mask */
#define SIOCSIFNETMASK _IOW(i,26, struct ifreq) /* set net addr mask */
#define SIOCGIFMETRIC _IOWR(i,27, struct ifreq) /* get IF metric */
#define SIOCSIFMETRIC _IOW(i,28, struct ifreq) /* set IF metric */

#define SIOCSARP _IOW(i, 30, struct arpreq) /* set arp entry */
#define SIOCGARP _IOWR(i,31, struct arpreq) /* get arp entry */
#define SIOCDARP _IOW(i, 32, struct arpreq) /* delete arp entry */
#define SIOCUPPER _IOW(i, 40, struct ifreq) /* attach upper layer */
#define SIOCLOWER _IOW(i, 41, struct ifreq) /* attach lower layer */
#define SIOCSETSYNC _IOW(i, 44, struct ifreq) /* set syncmode */
#define SIOCGETSYNC _IOWR(i, 45, struct ifreq) /* get syncmode */
#define SIOCSSDSTATS _IOWR(i, 46, struct ifreq) /* sync data stats */
#define SIOCSSESTATS _IOWR(i, 47, struct ifreq) /* sync error stats */
```

```
#define SIOCSPROMISC _IOW(i, 48, int)  /* request promisc mode
       on/off */
#define SIOCADDMULTI _IOW(i, 49, struct ifreq) /* set m/c address */
#define SIOCDELMULTI _IOW(i, 50, struct ifreq) /* clr m/c address */

/* FDDI controls */
#define SIOCFDRESET _IOW(i, 51, struct ifreq) /* Reset FDDI */
#define SIOCFDSLEEP _IOW(i, 52, struct ifreq) /* Sleep until next dnld req */
#define SIOCSTRTFMWAR _IOW(i, 53, struct ifreq) /* Start FW at an addr */
#define SIOCLDNSTRTFW _IOW(i, 54, struct ifreq) /* Load the shared memory */
#define SIOCGETFDSTAT _IOW(i, 55, struct ifreq) /* Get FDDI stats */
#define SIOCFDNMIINT _IOW(i, 56, struct ifreq) /* NMI to fddi */
#define SIOCFDEXUSER _IOW(i, 57, struct ifreq) /* Exec in user mode */
#define SIOCFDGNETMAP _IOW(i, 58, struct ifreq) /* Get a netmap entry */
#define SIOCFDGIOCTL _IOW(i, 59, struct ifreq)  /* Generic ioctl for fddi */

/* protocol i/o controls */
#define SIOCSNIT _IOW(p, 0, struct nit_ioc) /* set nit modes */
#define SIOCGNIT _IOWR(p, 1, struct nit_ioc) /* get nit modes */

#endif DOCUMENTATION
#endif 0

/* for <sys/sockio.h> */
/* <net/if.h> has struct ifreq */

#define SIOCSETTCALG _IOWR(i, 8, struct ifreq) /* set next Traffic Control
Algorithm */
#define SIOCGETTCALG _IOWR(i, 9, struct ifreq) /* read current Traffic Control
Algorithm */


#define SIOCTCFSSTATUS _IOW(i, 90, struct ifreq)
#define SIOCTCFSINTFC _IOW(i, 91, struct ifreq)
#define SIOCTCFSADD _IOW(i, 92, struct ifreq)
#define SIOCTCFSINS _IOW(i, 93, struct ifreq)
#define SIOCTCFSMVADD _IOW(i, 94, struct ifreq)
#define SIOCTCFSMVINS _IOW(i, 95, struct ifreq)
#define SIOCTCFSMOD _IOW(i, 96, struct ifreq)
#define SIOCTCFSREM _IOW(i, 97, struct ifreq)
#define SIOCTCFSSHOW _IOW(i, 98, struct ifreq)
#define SIOCTCFSEPARMS _IOW(i, 99, struct ifreq)
#define SIOCTCFSCPARMS _IOW(i,100, struct ifreq)
#define SIOCTCFSQPARMS _IOW(i,101, struct ifreq)
#define SIOCTCFSDPARMS _IOW(i,102, struct ifreq)
#define SIOCTCFSSTATS _IOW(i,103, struct ifreq)
#define SIOCTCFSTRAFFIC _IOW(i,104, struct ifreq)
#define SIOCTCFSTPARMS _IOW(i,105, struct ifreq)

#endif TRAFFIC_CONTROL
```

```
/* ST2 Permanent Data Structures */

/*
 * Site-specific configuration information
 */

#define CTLFLG (ST2FlgOwnLE)

#ifdef STIIDEBUG
#define DBGFLG 0 /*DBG_PRINT1*/
#else !STIIDEBUG
#define DBGFLG 0
#endif STIIDEBUG

#define MAX_TRAPS 256

/*
m* Received packets are dispatched (queued) by the receiving interface
m* to one of several protocol families, based on information in the
m* link-level header. This may take one of two forms: ST packets
m* delivered to the IP family, using IP Version 5, or, for testing
m* purposes, through a specific link-level identifier that specifies
m* ST directly. For now, the packets will be queued for ST
m* (st2_intrq); unresolved is the issue of whether the call to do the
m* enqueueing must return quickley in order not to loose the next
m* packet, or whether it can push a data packet through to the output
m* queue before returning. When removed from the queue (by st2_intr),
m* the ST Header will be validated and the connection id (HID)
m* extracted. The HID will be anded with ConfigParm (hid_mask) to
m* generate an HID_BITS-bit index into the st2_ConHsh table. That
m* entry will contain the full HID and a pointer to the associated
m* control block (aST2pcb). The associated control block has all of
m* the connection-specific information.
m*
m* st2_ConHsh[ HID & ConfigParm (hid_mask) ]<HID,cp> -> aST2pcb<...>
m*
 */


/*
m* HIDs are used to bind ST-II data packets to their stream, on a
m* hop-by-hop basis. A HID is approved by the receiving ST Agent and
m* given to the previous-hop Agent for it to place into data packets
m* that is sends to the receiving Agent. When network-level multicast
m* is used, all of the receiving Agents must approve the same HID.
m* HIDs contain 16 bits, of which we use the HID_BITS least
m* significant bits to index into the st2_ConHsh table to find the
m* ST-II protocol control block (aka aST2pcb) for the associated
m* stream. HID 0 is used for SCMP (control) messages. HID 1 may be
m* used by HELLO messages that are to be treated as "data" packets by
m* are also passed to SCMP. HIDs 2-3 are reserved. Each stream may
```

```
m* have up to MAX_HIDS_PER_STREAM associated with it (multiple
m* st2_ConHsh entries pointing to the same aST2pcb). (2 ** HID_BITS)
m* should be greater than (MAX_STREAMS * average number of HIDs per
m* stream), where (average number of HIDs per stream) is <=
m* (MAX_HIDS_PER_STREAM). (2 ** VLNK_BITS) should be >= (1 + average
m* number of next-hop agents per stream) * MAX_STREAMS.
 */

/*
c* Maximum number of simultaneous streams (MAX_STREAMS), and
c* HID bits used (HID_BITS).
 */

# define MAX_STREAMS (64)  /* Configured to less than 64 streams */
# define HID_BITS (8)  /* Number of LSB in HID we use */


/*
c* Maximum number of targets placed into a target list (TARG_LISTS) and
c* maximum number of target lists per SCMP message (TARG_PER_LIST).
 */

# define TARG_LISTS 1
# define TARG_PER_LIST 1


#ifndef TRAFFIC_CONTROL
/*
c* Number of generic network interfaces (DEF_GENIFS). Dummy plus one
c* per physical network interface plus one per Origin and Target on
c* local system.
 */

#ifdef STIIAPI
#define DEF_GENIFS (10 + 32)  /* Physical plus API pseudo */
#else
#define DEF_GENIFS (10)   /* Physical */
#endif STIIAPI

#else TRAFFIC_CONTROL

#define DEF_GENIFS 99999  /* It's in if_aux.c */

#endif TRAFFIC_CONTROL


/*
d* ST-II packet Input Queue (st2_intrq).
 */

struct ifqueue st2_intrq = {  /* queue of input packets from
     networks, splimp */
```

```
 /* ifq_head ifq_tail ifq_len ifq_maxlen ifq_drops */
 0, 0, 0, 50,  0 };


    /* Masked net-order HID -> Con idx */
struct aConHshEntry  st2_ConHsh[ (1 << HID_BITS) ]; /* splnet */

struct aST2pcb  *st2_ConTblFreep, /* List of free st2_ConTbl entries,
     splnet */
   *st2_ConTblTailp,
    st2_ConTbl[ MAX_STREAMS ];

char    st2_fsver3, /* Init. to min legnth of flowspec */
    st2_fsver4,
    st2_fsver5,
    st2_fsver6;


struct NextHopTbl  st2_nexthops;
struct aVLinkTbl  st2_vlinks;


/* ST-II Experimenters */

enum Experimenters {
#define aExp(value,site,contact) site = value,
 ExperimenterList
#undef aExp
};


/* ST-II Configuration Paramteres */

struct aConfigBlock  st2_config = {
#define aCfgParm(name,cv,nv,xv,cs,fo,desc) (unsigned long) (cv),
 ConfigParmList
#undef aCfgParm
};


/* ST-II Statistics counters */

struct aStatsBlock  st2_stats = {
 sizeof (struct aStatsBlock), Experimenter|ST2StatsVersion
};


/*
 * Tables for debugging information.
 */

struct aStateName
```

```
#define aState(name,desc) { (int) name, 0, "name", desc },
    st2_pcb_states[] = { PCB_STATE_LIST },
    st2_hid_states[] = { HID_STATE_LIST },
    st2_fwdr_states[] = { FWDR_STATE_LIST },
    st2_nhop_states[] = { NHOP_STATE_LIST },
#undef aState
#define aTState(name,side,desc) { (int) name, (int) side, "name", desc },
    st2_targ_states[] = { TargetStateList };
#undef aTState



int    st2_npcb_states = DimensionOf (st2_pcb_states),
    st2_nhid_states = DimensionOf (st2_hid_states),
    st2_nfwdr_states = DimensionOf (st2_fwdr_states),
    st2_ntarg_states = DimensionOf (st2_targ_states),
    st2_nnhop_states = DimensionOf (st2_nhop_states);


/*
 * Debugging Strings.
 */

char   *st2_dbgstrings[] =
 {
#define aString(name,text) text,
 StringList
#undef aString
};


char  *(mtype_names[ PKT_DIM ]) =
 { "FREE", "DATA", "HEADER", "SOCKET", "PCB", "RTABLE", "HTABLE", "ATABLE",
 "SONAME", "ZOMBIE", "SOOPTS", "FTABLE", "CONTROL", "IF", "IPMOPTS",
"IPMADDR",
 "IFMADDR", "MRTABLE", "TIMESTAMP", "19", "20", "21", "22", "23",
 "24", "25", "26", "27", "28", "29", "30", "31",
 "TCDATA", "NEIGHBOR", "NETIF", "NXTHOP", "PARSEDSCM", "TARGET" };


struct atrap   st2_traptable[ MAX_TRAPS ];



/*
 * Tranlate internal error codes to offical ST-II code, loosing information.
 */

struct aErrorXlate  st2_reasontable[ MAX_ST_ERRORS ] = {
#define Q(x) | (unsigned int) (1 << ((8*sizeof (int) )-1-(int)x))
#define anError(vi,vx,n,f,def) { vx, n, (unsigned int) 1 f },
    /* "1" for init_STTables () */
```

```
 ST2ErrorList
 ImplErrorList
#undef anError
#undef Q
};


/*
D* BSD-style Protocol Configuration & Dispatch Table (protosw coipsw).
d*
d * Currently, no protocols are defined above ST-II.
d*
 */

#ifndef STIIAPI

 static int    /* Next bugid 0x70102 */
nosubr ()
{
 BUGRETURN (-,45/*EOPNOTSUPP*/,0x70101,int);
}
#endif STIIAPI


extern struct domain  coipdomain; /* Forward reference */

static struct protosw coipsw[] = { /* Protocols supported by COIP */
    /* ST2 */
 { SOCK_RAW, &coipdomain, IPPROTO_ST, PR_ATOMIC | PR_ADDR | PR_RIGHTS,
 /*st2_input*/ 0, /*st2_output*/ 0, /*st2_ctlinput*/ 0, /*st2_ctloutput*/ 0,
#ifndef STIIAPI
 nosubr,    /* Gateway-only */
#else
 st2_usrreq,    /* Local applications */
#endif STIIAPI
 st2_init, st2_ClkFast, st2_ClkSlow, /*st2_drain*/ 0,
 },

};


/*
D* BSD-style Protocol Domain Table (coipdomain).
 */

STATIC struct domain coipdomain = /* Define COIP Protocol Family/domain */
 { PF_COIP, "COIP", 0, 0, 0,  /* ??? is it a PF or an AF, or assumption they are
== */
 coipsw, &coipsw[DimensionOf (coipsw)] };


/*
```

```
S* coipdomaininit ( count )
 */

 void
coipdomaininit( count )
 int    count;  /* ARGSUSED */
{
 extern int   protocol_family ();


 (void) protocol_family( &coipdomain, /*sthash*/ NULL, /*stnetmatch*/ NULL );
 return;
}



/* Optional (ha, ha) Ethernet Support */

/* If you get an error here, the lines:
 * sunif/if_ie.c  optional ie INET device-driver
 * sunif/if_le.c  optional le INET device-driver
 * are missing from the file sys/sun<N>/conf/files.
 */

#include "ie.h"
#if defined (NIE)
#if NIE
#ifndef WANT_ETHERNET
#define WANT_ETHERNET 1
#endif WANT_ETHERNET
#endif NIE
#endif defined (NIE)

#include "le.h"
#if defined (NLE)
#if NLE
#ifndef WANT_ETHERNET
#define WANT_ETHERNET 1
#endif WANT_ETHERNET
#endif NLE
#endif defined (NLE)

#ifndef WANT_ETHERNET
#define WANT_ETHERNET 0
#endif WANT_ETHERNET


#if WANT_ETHERNET
#include <net/if_arp.h>  /* ether_family, for netinet/if_ether.h */
#include <netinet/if_ether.h> /* ether_addr, ether_header, wants
    net/if.h, netinet/in.h */
```

```
/* Routine to register our ethertype */
extern void ether_register ();

/* Ethernet callable dispatch routine */
extern struct ifqueue *st2_fromether ();

/* AF_COIP ethernet address resolution routine in ethernet context */
extern int   st2_toether ();

/* Linkage from ethernet for an ethertype */
static struct ether_family st2_ether_family =
 /* vvvvvv gets replaced by ConfigParm (ethertype) on startup */
 { AF_COIP, 0x5354, st2_fromether, st2_toether, st2_intr, 0 },
   st2_ether_family_in =
 { AF_COIP, 0x5354, st2_fromether, st2_toether, st2_intr, 0 };



/*
S* st2_fromether ( ifnetp, pktp, ehp )
s*
s* Routine to save ethernet header information.
s*
 * Ethernet layer (do_protocol) calls this routine via the st2_ether_family
 * structure when an ST-II ethertype packet is received. It may dispose of
 * the packet directly (and return NULL), or it may return a pointer to the
 * ifqueue structure to which the packet should be queued (and a softcall to
 * st2_intr will be scheduled). This is the last chance to see the actual
 * ethernet header -- e.g., destination & source.
 *
 * Since its an interrupt routine, it may be called while rest of code is
 * running.
 *
 */

 static struct ifqueue *  /* Next bugid 0x70200 */
st2_fromether( ifnetp, pktp, ehp ) /* from */
 struct ifnet  *ifnetp; /* Receiving interface structure */
 struct aPktDesc *pktp;  /* Network level data (m_len may */
    /* include padding trash) */
    /* If return NULL, we dispose of it */
    /* If return ifqueue*, caller queues */
 struct ether_header *ehp;  /* Ethernet header */
{
 int    oldpri;
 struct aPktDesc *hdrp;
 struct LocalNetInfo *lnhp;


 hdrp = NO_PKTDSCP;

 oldpri = splimp ();   /* Lock out other drivers */
```

```
if ( IF_QFULL ( &( st2_intrq ) ) )
{    /* If our queue is full, not doing */
   /* reservations very well! */
IF_DROP ( &( st2_intrq ) );
Count (net_in_drop_pkts,1);
/*Count (net_in_drop_bytes,n);*/
FreePkts (pktp);  /* Queue full, flush packet */
}
else
{
if ( ConfigFlag (ST2FlgNoSrcChking) )
{
 hdrp = pktp;  /* Do not want header */
}
else
{
 lnhp = GetSpace (struct LocalNetInfo *, &( hdrp ), PKT_HEADER,
   PKT_MD_NEW, PKT_XA_PS,
   0,0, sizeof (struct LocalNetInfo) ,0,0);
 if ( lnhp EQ (struct LocalNetInfo *) NULL )
 {
 hdrp = pktp;  /* Continue w/o header */
 }
 else   /* Remember local net info */
 {
 hdrp->pkt_nxtmsgp = NO_PKTDSCP;
 uniqtime ( &( lnhp->rcvdts ) );
 lnhp->gifcp = IF_Ext (ifnetp); /* ??? ethertype too */
 lnhp->linkmux = (ehp->ether_type NE 0x800) ? ehp->ether_type : 0;
 lnhp->len = sizeof (struct ether_addr);

 /* WARNING: make sure that:
  * sizeof (struct ether_addr) <= ST2_MAX_LNH_ADRLEN
  * I'd use an #if if cpp could evaluate sizeof()
  */

 /* LocalNetInfo assumes len (src) EQ len (dst) */
 Bcopy (&( ehp->ether_dhost.ether_addr_octet[0] ),
  &( lnhp->dst[0] ),lnhp->len);
 Bcopy (&( ehp->ether_shost.ether_addr_octet[0] ),
  &( lnhp->src[0] ),lnhp->len);

 hdrp->nextp = pktp;
 }
}

/* Put packet in queue */
IF_ENQUEUE ( &( st2_intrq ), ((struct mbuf *) hdrp) );
}

(void) splx ( oldpri );
```

```
 if ( hdrp NE NO_PKTDSCP )
 softcall ( (int (*) ()) st2_intr, 0 ); /* Go process it later */

 return ( (struct ifqueue *) NULL ); /* pktp disposed */

#ifdef NOTYET
|  /* queue pktp in st2_intrq, softcall ( st2_intr ) */
|  return ( &st2_intrq );
#endif NOTYET

} /* end of st2_fromether */


/*
S* st2_toether ( sockaddrp, pktp, ifnetp, eap )
s*
s* Routine called to do ST2 to ethernet address translation.
s*
 * Called by ethernet layer when AF of destination sockaddr does not
 * contain either AF_UNSPEC (raw ethernet header) or AF_INET for et = 800.
 *
 */


 static int    /* Next bugid 0x70302 */
st2_toether( sockaddrp, pktp, ifnetp, eap ) /* from */
 struct sockaddr *sockaddrp; /* our structure in aNxtHop */
 struct aPktDesc *pktp;
 struct ifnet  *ifnetp;
 struct ether_addr *eap;
{
 struct arpcom *acp = (struct arpcom *) ifnetp;
 struct ether_header *ehp;


 /* if AF_COIP, do arp lookup & update nxthop */

 if ( sockaddrp->sa_family NE ConfigParm (coip_family) )
 {
 Count (net_out_drop_pkts,1);
 /*Count (net_out_drop_bytes,n);*/
 FreePkts (pktp);
 BUGSTOP (-,sockaddrp->sa_family,0x70301);
 return ( 1 );   /* failed, we keep pkt */
 }

 if ( (Mkp (struct in_addr *, &( sockaddrp->sa_data[2] ), 0))->s_addr
 NE acp->ac_lastip.s_addr )
 {
 acp->ac_lastip.s_addr =
  (Mkp (struct in_addr *, &( sockaddrp->sa_data[2] ), 0))->s_addr;
```

```
    /* ??? make a trash pkt to send, as pkt initiating arp may be lost
     * (sent with IP ethertype)
     */

    if ( arpresolve ( acp, (struct mbuf *) pktp ) EQ 0 )
    {
     acp->ac_lastip.s_addr = 0; /* not valid */
     return ( 1 );  /* failed, arp keeps pkt */
        /* NB: lost if not retransmitted */
    }
    }

    /* Record ethernet address in our aNxtHop & switch to AF_UNSPEC
    to avoid future translations (paths are fixed in ST) */

    ehp = Mkp (struct ether_header *, &( sockaddrp->sa_data[0] ), 0);

    Bcopy (&( acp->ac_lastarp ),&( ehp->ether_dhost ),
       sizeof (struct ether_addr));
    ehp->ether_type = ConfigParm (ethertype); /*??? per interface*/
    sockaddrp->sa_family = AF_UNSPEC; /* length SB 14 */

    Bcopy (&( acp->ac_lastarp ),eap,sizeof (struct ether_addr));

    return ( 0 );   /* go send pkt */

} /* end of st2_toether */



#ifndef TRAFFIC_CONTROL
#include <sys/errno.h>
#include <netinet/in.h>  /* for netinet/if_ether.h */
#include <net/if_arp.h>  /* ether_family, for netinet/if_ether.h */
#include <netinet/if_ether.h>

extern struct ether_addr  etherbroadcastaddr;
extern struct ether_family *ether_families;

extern int    tcif_ether_output ();
extern struct timeval   time;



#if NIE

/*
S* tcif_ieoutput ( ifp, pktp, sockaddrp )
s*
s* Routine called to send packets via the ethernet.
 *
```

```
 * Just because we don't have source to be able to recoompile
 * ether_output with a different IF_ENQUEUE macro in <net/if.h>
 * & fix randomdrop.
s*
 * Called via ifnet if_output dispatch.
 *
 */

extern void    iestartout ();


 /*static*/ int   /* Next bugid 0x */
tcif_ieoutput( acp, pktp, sockaddrp )
 struct arpcom  *acp;
 struct mbuf  *pktp;
 struct sockaddr *sockaddrp;
{
 return ( tcif_ether_output( acp, pktp, sockaddrp, iestartout ) );
}

#endif NIE



#if NLE

/*
S* tcif_leoutput ( ifp, pktp, sockaddrp )
s*
s* Routine called to send packets via the ethernet.
 *
 * Just because we don't have source to be able to recoompile
 * ether_output with a different IF_ENQUEUE macro in <net/if.h>
 * & fix randomdrop.
s*
 * Called via ifnet if_output dispatch.
 *
 */

extern void    lestart ();


 /*static*/ int    /* Next bugid 0x */
tcif_leoutput( acp, pktp, sockaddrp )
 struct arpcom  *acp;
 struct mbuf  *pktp;
 struct sockaddr *sockaddrp;
{
 return ( tcif_ether_output( acp, pktp, sockaddrp, lestart ) );
}

#endif NLE
```

```
 /*
 S* tcif_ether_output ( acp, pktp, sockaddrp, fnc_start )
 s*
 s* Routine called to build ethernet packets and queue them for
 s* transmission.
  *
  * Just because we don't have source to be able to recoompile
  * ether_output with a different IF_ENQUEUE macro in <net/if.h>
  * & fix randomdrop.
 s*
  * Called indirectly via ifnet if_output dispatch.
  *
  */


 static int
tcif_ether_output( acp, pktp, sockaddrp, fnc_start )
 struct arpcom   *acp;  /* begins with struct ifnet */
 struct mbuf   *pktp;  /* packet to be sent, we dispose of it */
 struct sockaddr  *sockaddrp; /* sockaddr of some flavor */
 void   (*fnc_start) (); /* driver output-start function */
{
 unsigned short  ether_type; /* ether_type */
 int    len,  /* of packet */
    oldpri;
 struct ether_addr  dst_ea; /* an ethernet address */
 struct ether_family *efp;  /* to find appropriate dispatches */
 struct ether_header *ehp;  /* ehternet header */
 struct mbuf  *mh = pktp, /* mbuf that will have ether header */
    *bcstp = NULL, /* copy of packet if to be broadcast */
   *mp;  /* to scan packet to find length */
#ifdef TRAFFIC_CONTROL
 struct aNetIf *gifcp;
 caddr_t   flowp,
    nethdrp;
 int    validlen;
 unsigned long  key,
    rsrc;
 unsigned short  pf;
#endif TRAFFIC_CONTROL

 if ( (acp->ac_if.if_flags & (IFF_RUNNING | IFF_UP))
NE (IFF_RUNNING | IFF_UP) )
 {
m_freem ( pktp );
return ( ENETDOWN );
 }

#ifdef TRAFFIC_CONTROL
```

```
gifcp = tcif_ifp2gifcp( (caddr_t) acp ); /* dummyif0 ??? */

pf = PF_UNSPEC;
nethdrp = mtod ( pktp, caddr_t);
validlen = pktp->m_len;

if ( pktp->m_type EQ MT_TCDATA )
{
flowp = pktp->m_tcflowp;
rsrc = pktp->m_tcrsrc;
key = pktp->m_tckey;
}
else
{
flowp = (caddr_t) NULL;
rsrc = 0;
key = 0;
}
#endif TRAFFIC_CONTROL

/* Map protocol specific address in sockaddr to local network address */

switch ( sockaddrp->sa_family )
{
case AF_INET:   /* 2 IP packets */
oldpri = splimp ();
{
 if ( acp->ac_lastip.s_addr /* is translation in cache */
 NE ((struct sockaddr_in *) sockaddrp)->sin_addr.s_addr )
 {   /* no */
 acp->ac_lastip = ((struct sockaddr_in *) sockaddrp)->sin_addr;
 if ( NOT arpresolve ( acp, pktp ) ) /* look it up */
 {   /* not in table, arping it */
  acp->ac_lastip.s_addr = 0; /* no valid translation */
  splx ( oldpri );
  return ( 0 ); /* held til arp'd, if_output called */
 }
 }
#ifdef sparc    /* alignment is at least octet2 */
  * (short *) &( dst_ea.ether_addr_octet[0] ) /* lint ppap */
  = * (short *) &( acp->ac_lastarp.ether_addr_octet[0] );
  * (short *) &( dst_ea.ether_addr_octet[2] ) /* lint ppap */
  = * (short *) &( acp->ac_lastarp.ether_addr_octet[2] );
  * (short *) &( dst_ea.ether_addr_octet[4] ) /* lint ppap */
  = * (short *) &( acp->ac_lastarp.ether_addr_octet[4] );
#else !sparc
  dst_ea = acp->ac_lastarp;
#endif sparc
  ether_type = ETHERTYPE_IP;
#ifdef TRAFFIC_CONTROL
  pf = PF_INET;
#endif TRAFFIC_CONTROL
```

```
 }
 splx ( oldpri );
 break;

 case AF_UNSPEC:   /* 0 Ethernet packets */
    /* sockaddr has ethernet header */
 ehp = (struct ether_header *) &( sockaddrp->sa_data[0] ); /* lint ppap */

#ifdef sparc    /* alignment is at least octet2 */
 * (short *) &( dst_ea.ether_addr_octet[0] ) /* lint ppap */
  = * (short *) &( ehp->ether_dhost.ether_addr_octet[0] );
 * (short *) &( dst_ea.ether_addr_octet[2] ) /* lint ppap */
  = * (short *) &( ehp->ether_dhost.ether_addr_octet[2] );
 * (short *) &( dst_ea.ether_addr_octet[4] ) /* lint ppap */
  = * (short *) &( ehp->ether_dhost.ether_addr_octet[4] );
#else !sparc
 dst_ea = ehp->ether_dhost;
#endif sparc
 ether_type = ehp->ether_type; /* use specified ether_type */
#ifdef TRAFFIC_CONTROL
 if ( ether_type EQ ETHERTYPE_IP )
  pf = PF_INET;
#ifdef STII
 else if ( ether_type EQ ConfigParm (ethertype) )
  pf = PF_COIP;
#endif STII
#endif TRAFFIC_CONTROL
 break;

 default:    /* Lookup other types */
 efp = ether_families;  /* List of known types */
 while ( efp )
 {
  if ( efp->ef_family EQ sockaddrp->sa_family )
  break;   /* Found address family */

  efp = efp->ef_next;  /* try next */
 }

 if ( efp )   /* if found table entry */
 {
  if ( efp->ef_outfunc ) /* better have address translation */
  {
  if ( efp->ef_outfunc ( sockaddrp, pktp, acp, &( dst_ea ) ) )
   return ( 0 ); /* ef_outfunc disposed of pktp */
     /* classify, enforce, nq ??? */

  if ( efp->ef_ethertype EQ 1500 ) /* ? ETHERMTU */
  {
   len = 0;  /* find packet length */
   if ( mp = pktp )
   do
```

```
  {
   len += mp->m_len;
   mp = mp->m_next;
  } while ( mp );
  ether_type = len; /* use length as "ether_type" */
  /* leave pf = PF_UNSPEC */
  break;
 }
 ether_type = efp->ef_ethertype; /* ether_type from table */
#ifdef TRAFFIC_CONTROL
 pf = efp->ef_family;
#endif TRAFFIC_CONTROL
 break;
 }
}

identify ( acp );  /* unsuported ether_type, drop pkt */
printf ( "can't handle AF 0x%x", sockaddrp->sa_family );
m_freem ( pktp );
return ( EAFNOSUPPORT );
break;
} /* end of sockaddrp->sa_family switch */


/* Check if destined to the broadcast address */

if ( ( * (short *) &( etherbroadcastaddr.ether_addr_octet[4] )
 EQ * (short *) &( dst_ea.ether_addr_octet[4] )) /* lint ppap */
AND ( * (short *) &( etherbroadcastaddr.ether_addr_octet[2] )
 EQ * (short *) &( dst_ea.ether_addr_octet[2] )) /* lint ppap */
AND ( * (short *) &( etherbroadcastaddr.ether_addr_octet[0] )
 EQ * (short *) &( dst_ea.ether_addr_octet[0] )) ) /* lint ppap */
{
   /* yes, make copy for local delivery */
bcstp = (struct mbuf *) m_copy ( pktp, 0, M_COPYALL );
}


/* Find space for ethernet header */

if ( (pktp->m_off <
#ifdef TRAFFIC_CONTROL
   (unsigned long) OffsetOf (m_tcdat[0],struct mbuf)
#else !TRAFFIC_CONTROL
   MMINOFF
#endif TRAFFIC_CONTROL
   + sizeof (struct ether_header))
 OR ( M_HASCL( pktp )
#ifdef MCL_STATIC_HDR
  AND (pktp->m_cltype NE MCL_STATIC_HDR)
#endif MCL_STATIC_HDR
  ) )  /* no room in first mbuf, prepend another */
```

```
  {
 #ifdef TRAFFIC_CONTROL
  mh = (struct mbuf *) m_get ( M_DONTWAIT, MT_TCDATA );
 #else !TRAFFIC_CONTROL
  mh = (struct mbuf *) m_get ( M_DONTWAIT, MT_HEADER );
 #endif TRAFFIC_CONTROL
  if ( ! mh )
  {
   m_freem ( pktp );
   m_freem ( bcstp );
   ether_error ( acp, "WARNING: no mbufs" );
   return ( ENOBUFS );
  }

 #ifdef TRAFFIC_CONTROL   /* XXX 8 => mod x10 aligned */
  mh->m_off = OffsetOf (m_tcdat[8],struct mbuf);
 #endif TRAFFIC_CONTROL
  mh->m_next = pktp;  /* prepend */
  mh->m_len = sizeof ( struct ether_header );
  }
  else    /* insert header */
  {
  pktp->m_off -= sizeof ( struct ether_header );
  pktp->m_len += sizeof ( struct ether_header );
  }


  /* Construct ethernet header */

  ehp = mtod ( mh, struct ether_header *);

 #ifdef sparc    /* alignment is at least octet2 */
  * (short *) &( ehp->ether_dhost.ether_addr_octet[0] ) /* lint ppap */
  = * (short *) &( dst_ea.ether_addr_octet[0] );
  * (short *) &( ehp->ether_dhost.ether_addr_octet[2] ) /* lint ppap */
  = * (short *) &( dst_ea.ether_addr_octet[2] );
  * (short *) &( ehp->ether_dhost.ether_addr_octet[4] ) /* lint ppap */
  = * (short *) &( dst_ea.ether_addr_octet[4] );
 #else !sparc
  ehp->ether_dhost = dst_ea;
 #endif sparc

  ehp->ether_type = ether_type;


  /* Deliver local copy if broadcast */

  if ( bcstp )
  {
 mp = bcstp;
 len = 0;   /* find length */
 /* if ( bcstp ) */
```

```
do
{
 len += mp->m_len;
 mp = mp->m_next;
} while ( mp );

#ifdef sparc    /* alignment is at least octet2 */
 * (short *) &( ehp->ether_shost.ether_addr_octet[0] ) /* lint ppap */
  = * (short *) &( acp->ac_enaddr.ether_addr_octet[0] );
 * (short *) &( ehp->ether_shost.ether_addr_octet[2] ) /* lint ppap */
  = * (short *) &( acp->ac_enaddr.ether_addr_octet[2] );
 * (short *) &( ehp->ether_shost.ether_addr_octet[4] ) /* lint ppap */
  = * (short *) &( acp->ac_enaddr.ether_addr_octet[4] );
#else !sparc
 ehp->ether_shost = acp->ac_enaddr;
#endif sparc

 /* deliver local copy */
 do_protocol ( ehp, bcstp, acp, len );
 }

#ifdef TRAFFIC_CONTROL
 len = mh->m_len;
 if ( (mp = mh->m_next) NE (struct mbuf *) NULL )
 do
 {
  len += mp->m_len;
 } while ( (mp = mp->m_next) NE (struct mbuf *) NULL );

 mh->m_tcrsrc = rsrc;
 mh->m_tckey = key;

 key = mh->m_type;   /* *** mbuf stats update below */
 mh->m_type = MT_TCDATA;

 if ( flowp EQ (caddr_t) NULL )
 {
 if ( gifcp->iftc_classify NE (caddr_t (*)()) NULL )
  flowp = (*gifcp->iftc_classify) ( gifcp, mh, pf, nethdrp, validlen );
 /* ??? else "flow 0" */
 }
 mh->m_tcflowp = flowp;
#endif TRAFFIC_CONTROL

 /* Enqueue packet for output & start driver */

 oldpri = splimp ();
 {
#ifdef TRAFFIC_CONTROL
 if ( key NE MT_TCDATA )  /* *** update mbuf usage stats now */
 {
  mbstat.m_mtypes[key]--;
```

```
    mbstat.m_mtypes[MT_TCDATA]++;
   }

   if ( (gifcp->iftc_enforce EQ (int (*)()) NULL)
    OR ((validlen = (*gifcp->iftc_enforce)
     ( gifcp, mh, len, (struct timeval *) NULL)) EQ -2) )
   {
    /* Packet not subject to traffic control */

    if ( validlen EQ -2 )
    {
   mh->m_type = PKT_DATA; /* No-sort */
   mbstat.m_mtypes[ PKT_DATA ]++;
   mbstat.m_mtypes[ PKT_TCDATA ]--;
    }
    validlen = 0;

#ifndef SFQ_VC
   if ( IF_QFULL( &( acp->ac_if.if_snd ) ) )
#else
  if ( (m->m_type == MT_TCDATA) && IF_QFULL( &( acp->ac_if.if_snd )) )
#endif SFQ_VC
   {
   /* Queue is full, make room for this packet */

   ether_error ( acp, "WARNING: if_snd full" );
   /* drop a pkt */
   validlen = tcif_random_drop ( &( acp->ac_if.if_snd ) );
   }
  }

  /* Drop this packet */

  if ( validlen EQ -1 )
  {
   acp->ac_if.if_snd.ifq_drops++;
   /* update drop stats -- no good way to do keep accurate stats */
   /* bytes_dropped += len */
   /* bytes_queued += validlen - len */

   m_freem ( mh );
   splx ( oldpri );
   return ( ENOBUFS );  /* ??? better error */
  }

  if ( gifcp->iftc_nq NE (void (*)()) NULL )
  {
   (*gifcp->iftc_nq) ( gifcp, mh );
  }
  else
  {
   IF_ENQUEUE ( &( acp->ac_if.if_snd ), mh );
```

```
}

if ( validlen > 0 )
{
 acp->ac_if.if_snd.ifq_drops++;
 /* update drop stats -- no good way to do keep accurate stats */
 /* bytes_dropped += len */
 /* bytes_queued += validlen - len */

 /* don't want to return an error as it may have higher layer
 ramifications for pkt instead of what was dropped */
}
#else !TRAFFIC_CONTROL
 if ( IF_QFULL( &( acp->ac_if.if_snd ) ) )
 {
  /* Queue is full, make room for this packet */
  ether_error ( acp, "WARNING: if_snd full" );

  /* drop a pkt */
  len = tcif_random_drop ( &( acp->ac_if.if_snd ) );
  if ( len > 0 )
  {
  acp->ac_if.if_snd.ifq_drops++;
  /* update drop stats -- no good way to do keep accurate stats */
  /* bytes_dropped += len */
  /* bytes_queued += validlen - len */
  }
 }

 /* append new paacket */

 IF_ENQUEUE ( &( acp->ac_if.if_snd ), mh );
#endif TRAFFIC_CONTROL

 (*fnc_start) ( acp->ac_if.if_unit ); /* (re)start device output */
 }
 splx ( oldpri );

 return ( 0 );
}

#endif WANT_ETHERNET
#endif TRAFFIC_CONTROL


/* Optional HSI/S Support */

/* If you get an error here, the line:
 * hsisdev/hsis.o  optional hsis device-driver
 * is missing from the file sys/sun<N>/conf/files.
 */
```

```
#include "hsis.h"
#if NHSIS

#include <net/ppp.h>   /* PPP_ definitions for ST_II */

/* HSI/S callable dispatch routine */
 struct ifqueue *st2_fromhsis (/* ifnetp, pktp, ppphdrp */);

/* AF_COIP PPP address resolution routine */
static int   st2_tohsis (/* sockaddrp, pktp, ifnetp, ppphdrp */);

#ifdef NOTWANTED
extern void hsis_register ();  /* Routine to register our PPP type */

/* Linkage from hsis for an PPP type */
static struct register_family  st2_stii_family =
/*net_family, net_ppptype, net_infunc, net_outfunc, net_netisr, net_next*/
 { AF_COIP, PPP_STII_PROTO, st2_fromhsis, st2_tohsis, st2_intr, 0 },
   st2_scmp_family =
 { 0x8000 | AF_COIP, PPP_SCMP_PROTO, st2_fromhsis, st2_tohsis, st2_intr, 0 };
#endif NOTWANTED


/*
S* st2_fromhsis ( ifnetp, pktp, ppphdrp )
s*
s* Routine to save HSI/S header information.
s*
 */

 struct ifqueue *   /* Next bugid 0x70400 */
st2_fromhsis( ifnetp, pktp, ppphdrp ) /* from */
 struct ifnet  *ifnetp; /* Receiving interface structure */
 struct aPktDesc *pktp;  /* Network level data (pkt_length may */
    /* include padding trash) */
    /* If return NULL, we dispose of it */
    /* If return ifqueue*, caller queues */
 caddr_t   ppphdrp; /* PPP header */ /* ARGSUSED */
{
 int    oldpri;
 struct aPktDesc *hdrp;
 struct LocalNetInfo *lnhp;
 struct ifaddr *ifaddrp;
 unsigned short  linkmux = *(((unsigned short *) ppphdrp) + 1); /* lint ppap */

 hdrp = NO_PKTDSCP;

 oldpri = splimp ();   /* Lock out other drivers */

 if ( IF_QFULL ( &( st2_intrq ) ) )
 {    /* If our queue is full, not doing */
    /* reservations very well! */
```

```
IF_DROP ( &( st2_intrq ) );
Count (net_in_drop_pkts,1);
/*Count (net_in_drop_bytes,n);*/
FreePkts (pktp);  /* Queue full, flush packet */
}
else
{
/* We are required to supply the ifnet pointer as the first four
 bytes of the first data packet. Since the PPP header is 4
 bytes long, we just overwrite it with the ifnet pointer. */

/* ??? NOTYET: check if data or control & process data,
 * queue control
 */

*pkt2data (pktp, struct ifnet **) = ifnetp;

if ( ConfigFlag (ST2FlgNoSrcChking) )
{
 hdrp = pktp;  /* Do not want header */
}
else
{
 lnhp = GetSpace (struct LocalNetInfo *, &( hdrp ), PKT_HEADER,
   PKT_MD_NEW, PKT_XA_PS,
   0,0, sizeof (struct LocalNetInfo) ,0,0);
 if ( lnhp EQ (struct LocalNetInfo *) NULL )
 {
 hdrp = pktp;  /* Continue w/o header */
 }
 else   /* Remember local net info */
 {
 uniqtime ( &( lnhp->rcvdts ) );
 ifaddrp = ifnetp->if_addrlist; /* ??? look for IP entry */
 hdrp->pkt_nxtmsgp = NO_PKTDSCP;
 lnhp->gifcp = IF_Ext (ifnetp); /* ??? PPP type too */
 lnhp->linkmux = (linkmux NE PPP_STII_PROTO) ? linkmux : 0;
 lnhp->len = sizeof (INETADDR);

 /* WARNING: make sure that:
  sizeof (INETADDR) <= ST2_MAX_LNH_ADRLEN
  I'd use an #if if cpp could evaluate sizeof() */

 /* ??? always assume len (src) EQ len (dst) */
 Bcopy (&( ifaddrp->ifa_addr.sa_data[2] ),
  &( lnhp->dst[0] ),lnhp->len);
 Bcopy (&( ifaddrp->ifa_dstaddr.sa_data[2] ),
  &( lnhp->src[0] ),lnhp->len);

 hdrp->pkt_nxtbufp = pktp;
 }
}
```

```
    /* Put packet in queue */
    IF_ENQUEUE ( &( st2_intrq ), ((struct mbuf *) hdrp) );
    }

    (void) splx ( oldpri );

    if ( hdrp NE NO_PKTDSCP )
    softcall ( (int (*) ()) st2_intr, 0 ); /* Go process it later */

    return ( (struct ifqueue *) NULL ); /* pktp disposed */

  #ifdef NOTYET
  | /* queue pktp in st2_intrq, softcall ( st2_intr ) */
  |  return ( &st2_intrq );
  #endif NOTYET

  } /* end of st2_fromhsis */


  /*
  S* st2_tohsis ( sockaddrp, mbufp, ifnetp, ppphdrp )
  s*
  s* Routine called to do ST2 to PPP address translation.
  s*
   * Called by link layer when AF of destination sockaddr does not
   * contain AF_UNSPEC (raw link layer header)
   */

   static int    /* Next bugid 0x70502 */
  st2_tohsis( sockaddrp, pktp, ifnetp, ppphdrp ) /* from */
   struct sockaddr *sockaddrp;
   struct aPktDesc *pktp;
   struct ifnet  *ifnetp; /* ARGSUSED */
   caddr_t   ppphdrp;
  {
   if ( sockaddrp->sa_family NE ConfigParm (coip_family) )
   {
  Count (net_out_drop_pkts,1);
  /*Count (net_out_drop_bytes,n);*/
  FreePkts (pktp);
  BUGSTOP (-,sockaddrp->sa_family,0x70501);
  return ( 1 );   /* failed, we keep pkt */
   }

   return ( 0 );   /* go send pkt */

 } /* end of st2_tohsis */

#endif NHSIS
```

```
#ifndef TRAFFIC_CONTROL
/*
 * Pre-defined Resource Management vectors
 */

extern int   tcif_BctAlloc (), tcif_BctRlse (), tcif_PtpAlloc (),
   tcif_PtpRlse (), tcif_RsIdGet (), tcif_RsIdRel ();

struct aRmVector {
 char   name[16];
 struct aResourceManagement rmf;
}    tcif_rmvectors[] =
 { /* "<--- name ----->" rsrcalloc rsrcctrl rsrcgetid rsrcprobe
 * rsrcrelid rsrcrelse */

#define aRV(id,name,aloc,ctrl,idget,probe,idrel,rlse)   \
 { "name", { aloc, ctrl, idget, probe, idrel, rlse } },

 aRmVectorList    /* Defined in st2_resource.h */

#undef aRV

 };


/*
 * Pre-defined TrafficControl vectors
 */

#ifdef VIRTUAL_CLOCK
extern int   vc_aloc_func (/* gifcp, arsclp, bw, srcp, dstp, ctlp */);
extern int   vc_enf_func (/* gifcp, pktp, totlen, timevalp */);
extern int   vc_init_func (/* gifcp */);
extern void   vc_nq_func(/* gifcp, mp */);
#endif VIRTUAL_CLOCK

#ifdef FAIR_SHARE
extern caddr_t   fs_classify_func (/* gifcp, mp, PF_xxx, nethdrp, validlen */);
extern int   fs_aloc_func (/* gifcp, arsclp, bw, srcp, dstp, ctlp */),
    fs_control_func (/* gifcp, op, xxxp, xxxl */),
    fs_enforce_func (/* gifcp, mp, totlen, timevalp */),
    fs_init_func (/* gifcp */);
extern void   fs_clockfast_func (/* gifcp */),
    fs_nqlq_func (/* gifcp, mp */);
extern int   fs_rlse_func (/* gifcp, arsclp, bw, srcp, dstp, ctlp */);
#endif FAIR_SHARE


struct aTcVector {
 char   name[16];
 struct aTrafficControl tcf;
}    tcif_tcvectors[] =
```

```
 { /* @#$% cpp is too primitive to allow \ in the formal parameter list */
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 { "name", { clsfy, fsclk, cntrl, dq, drain, enfrc, init, nq, quit, \
  aloc, cntrl, idget, probe, idrel, rlse } },

 aTcVectorList    /* Defined in st2_resource.h */

#undef aTV

 };

/*
 * Bandwidth
 */

struct aBandwidthInfo {
 char   name[16];
 unsigned long  bw_conf,
    bw_load,
    bw_resv;
}    bws[] =
 { /*"<--- name ---->" conf load resv */
#define ENET_10MB 0
/* nfs complains ...
 { "ethernet_10mb", 1250000, 40000, 20000 },
*/
 { "ethernet_10mb", 1250000, 400000, 200000 },
#define HSIS_1344 1
 { "hsis_1344", 168000, 134400, 115000 },
 { "" }
 };
#endif TRAFFIC_CONTROL


#ifndef TRAFFIC_CONTROL
/*
S* tcif_AlgSwitch ( gifcp, quit )
s*
s* Switch to new traffic control algorithm on the specified interface.
s*
 */

 static int    /* Next bugid 0x71203 */
tcif_AlgSwitch( gifcp, quit )  /* from init_gifcs */
 struct aNetIf  *gifcp;
 int    quit;
{
 struct mbuf  *mp;
 int    alg = (int) gifcp->alg_next,
    oldpri;
```

```
unsigned long  nbytes,
   npkts;
struct aTrafficControl *tcp;


/* Verify can switch before make changes */
switch ( alg )
{
default:  BUGSTOP (-,alg,0x71200);
  gifcp->alg_next = gifcp->alg;
  return ( EINVAL );

case TC_FIFO: break;

#ifdef FAIR_SHARE
 case TC_FS1: break;
#endif FAIR_SHARE

 case TC_RD:  break;

#ifdef MY_FIFO
 case TC_MY_FIFO: break;
#endif MY_FIFO

/*
#ifdef SFQ
 case TC_SFQ: break;
#endif SFQ
*/
#ifdef SFQ_VC
 case TC_SFQ_VC: break;
#endif SFQ_VC

#ifdef VIRTUAL_CLOCK
 case TC_VC:  break;
#endif VIRTUAL_CLOCK
 } /* end of alg switch */


 if ( (int) TC_NUN > TC_MAX )
 {
 BUGSTOP (-,TC_NUN<<16|TC_MAX,0x71201);
 return ( ENOSR );
 }

 oldpri = spl4 ();  /* bring interface down, wait, ... */

 /* Locate cache area */
 if ( gifcp->per_alg EQ (caddr_t) NULL )
 {
 if ( tcif_cache.allocated < (tcif_cache.nxtfree + TC_MAX) )
  return ( ENOMEM );  /* ??? add more */
```

```
 tcp = &( tcif_cache.TrafficControls[ tcif_cache.nxtfree ] );
 tcif_cache.nxtfree += TC_MAX;
 bzero ( (char *) tcp, TC_MAX * sizeof (tcif_cache.TrafficControls[0]) );
 gifcp->per_alg = (caddr_t) tcp;
 }
 tcp = (struct aTrafficControl *) gifcp->per_alg; /* lint ppap */

#if 0
 /* Cache pervious algorithm's info */
 Bcopy ( &( gifcp->tcf ), /*->*/ (tcp + gifcp->iftc_alg), sizeof (* tcp) );
#endif 0

 /* Flush old queue(s) */
 if ( gifcp->iftc_drain NE (void (*)()) NULL )
 {
 (*gifcp->iftc_drain) ( gifcp, gifcp->iftc_state1p, gifcp->iftc_state2p,
   gifcp->iftc_state1ul, gifcp->iftc_state2ul,
   &npkts, &nbytes );

#ifdef lint
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 drain ( gifcp, gifcp->iftc_state1p, gifcp->iftc_state2p, \
  gifcp->iftc_state1ul, gifcp->iftc_state2ul, &npkts, &nbytes );
 aTcVectorList
#undef aTV
#endif lint
 }
 else
 {
 for (;;)
 {
  if ( gifcp->iftc_dq NE (struct mbuf *(*)()) NULL )
  mp = (*gifcp->iftc_dq) ( gifcp );
  else
  {
  IF_DEQUEUE ( &( gifcp->osifcp->if_snd ), mp );
  }

  if ( mp EQ (struct mbuf *) NULL )
  break;

  FreePkts ((struct aPktDesc *) mp);
 } /* end of forever loop */
 }

 if ( quit && (gifcp->iftc_quit NE (void (*)()) NULL) )
 (*gifcp->iftc_quit) ( gifcp );

#ifdef lint
```

```
#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 quit ( gifcp );
 aTcVectorList;
#undef aTV
#endif lint

#if 1
 /* Cache pervious algorithm's info */
 Bcopy ( &( gifcp->tcf ), /*->*/ (tcp + gifcp->iftc_alg), sizeof (* tcp) );
#endif 1

 /* Switch algorithms */
 gifcp->alg = (short) alg;

 tcp += alg;    /* Restore previous parameters */
 Bcopy ( tcp, /*->*/ &( gifcp->tcf ), sizeof (gifcp->tcf) );

 if ( (gifcp->iftc_state1p EQ (caddr_t) NULL)
 AND (gifcp->iftc_state2p EQ (caddr_t) NULL)
 AND (gifcp->iftc_state1ul EQ 0)
 AND (gifcp->iftc_state2ul EQ 0) )
 {
 switch ( alg )
 {
 default:
  BUGSTOP (-,alg,0x71202);
  gifcp->alg = (short) TC_FIFO;
  gifcp->alg_next = (short) TC_FIFO;
  gifcp->tcf = tcif_tcvectors[ (int) TC_FIFO ].tcf;
  break;

#define
aTV(id,name,clsfy,fsclk,cntrl,dq,drain,enfrc,init,nq,quit,aloc,ctrl,idget,prob
e,idrel,rlse) \
 case Ident(TC_)id:     \
  gifcp->tcf = tcif_tcvectors[ (int) Ident(TC_)id ].tcf; \
  break;

 aTcVectorList

#undef aTV
 } /* end of alg switch */
 }

 if ( gifcp->iftc_init NE (int (*)()) NULL )
 (void) (*gifcp->iftc_init) ( gifcp ); /* ??? check for error & undo */

 splx ( oldpri );  /* bring interface back up */

#if 0
 if ( gifcp->iftc_clockfast NE (void (*)()) NULL )
```

```
 (*gifcp->iftc_clockfast) ( gifcp );
#endif

 return ( 0 );
}
#endif TRAFFIC_CONTROL


#ifndef TRAFFIC_CONTROL
/*
S* tfic_init_gifcs ()
s*
s* Routine to initialize generic network interfaces.
s*
 */

 static void    /* Next bugid 0x70602 */
tfic_init_gifcs()    /* from st2_init */
{
 char   *sp;
 int    left,
    len;
 struct aNetIf  *agip = &( tfic_genifs.NetIfs[0] ),
  **tailpp = &( tcif_gifcheadp );
 extern struct ifnet  *ifnet; /* BSD's global list of interfaces */
 struct ifnet  *osifp = ifnet; /* BSD's global list of interfaces */


 /* Create a dummy interface to try and fail gracefully if something is
 * broken.
 */

 bzero( (char *) agip, sizeof (struct aNetIf) );
 agip->ifc_nextp = NO_NETIFP;
 agip->osifcp = &( dummyif );
 agip->lclhdrlen = sizeof (((struct sockaddr *)0)->sa_data);
 agip->ifc_output = tcif_dummy_output;
 (void) strcpy ( &( agip->namebuf[0] ), /*<-*/ "dummyif0" );
 agip++;
 tfic_genifs.nxtfree++;


 /* Create a Generic Network Interface for each OS interface */

 for ( ; osifp NE (struct ifnet *) NULL ; osifp = osifp->if_next )
 {
 /* FYI: ifnet->if_addrlist not yet valid */

 /* Logical end of addresses per interface loop */

 if ( (tfic_genifs.nxtfree +1) >= tfic_genifs.allocated )
 {
```

```
 BUGSTOP (-,Misconfiguration,0x70601);
 break;   /* out for next interface */
}

agip->osifcp = osifp;
agip->lclhdrlen = sizeof (((struct sockaddr *)0)->sa_data);

sp = &( agip->namebuf[0] );
left = sizeof (agip->namebuf) - 2;
(void) strncpy ( sp, osifp->if_name, left );
len = strlen ( sp );
sp += len;
left -= len;
if ( left < 5 )
 *sp++ = '?';
else
{
 len = osifp->if_unit & 0xFFFF;
 if ( len >= 10000 )
 *sp++ = '0' + (len / 10000), len %= 10000;
 if ( len >= 1000 )
 *sp++ = '0' + (len / 1000), len %= 1000;
 if ( len >= 100 )
 *sp++ = '0' + (len / 100), len %= 100;
 if ( len >= 10 )
 *sp++ = '0' + (len / 10), len %= 10;
 *sp++ = '0' + len;
}
*sp = '\0';

#if WANT_ETHERNET
 /* Don't have source access to change:
  * 1) IF_ENQUEUE / IF_DEQUEUE macros in ether_output
  * nor 2) random_drop
  */
#if NLE
 if ( strcmp ("le",osifp->if_name) EQ 0 )
 {
  if ( ConfigFlag (ST2FlgOwnLE) )
  {
  extern int leoutput (/* ifnetp, pktp, sockaddrp */);

  agip->lclhdrlen = sizeof (struct ether_header);
  if ( osifp->if_output EQ leoutput )
   osifp->if_output = tcif_leoutput;
  }
/*
cwl ??? default TrafficControl */
  agip->bw_conf = bws[ (int) ENET_10MB ].bw_conf;
  agip->bw_load = bws[ (int) ENET_10MB ].bw_load;
  agip->bw_resv = bws[ (int) ENET_10MB ].bw_resv;
```

```
  agip->rmf = tcif_rmvectors[ (int) RM_BCST ].rmf;
#ifdef VIRTUAL_CLOCK
 /* agip->tcf = tcif_tcvectors[ (int) TC_VC ].tcf; */
  agip->alg_next = (short) TC_VC;
#else !VIRTUAL_CLOCK
  agip->alg_next = (short) TC_RD;
#endif VIRTUAL_CLOCK
 }
#endif NLE
#if NIE
 if ( strcmp ("ie",osifp->if_name) EQ 0 )
 {
  if ( ConfigFlag (ST2FlgOwnLE) )
  {
  extern int ieoutput (/* ifnetp, pktp, sockaddrp */);

  agip->lclhdrlen = sizeof (struct ether_header);
  if ( osifp->if_output EQ ieoutput )
   osifp->if_output = tcif_ieoutput;
  }
/*
cwl ??? default TrafficControl */
  agip->bw_conf = bws[ (int) ENET_10MB ].bw_conf;
  agip->bw_load = bws[ (int) ENET_10MB ].bw_load;
  agip->bw_resv = bws[ (int) ENET_10MB ].bw_resv;

  agip->rmf = tcif_rmvectors[ (int) RM_BCST ].rmf;
 /* agip->tcf = tcif_tcvectors[ (int) TC_VC ].tcf; */
  agip->alg_next = (short) TC_VC;
 }
#endif NIE
#endif WANT_ETHERNET

#if NHSIS
 if ( strcmp ("hsis",osifp->if_name) EQ 0 )
 {
  agip->lclhdrlen = PPP_HDRSPACE;

/* #ifdef DARTNET */
  if ( (osifp->if_unit % 4) EQ 0 )
/* #endif DARTNET */
  {
/*
cwl ??? default TrafficControl */
  agip->bw_conf = bws[ (int) HSIS_1344 ].bw_conf;
  agip->bw_load = bws[ (int) HSIS_1344 ].bw_load;
  agip->bw_resv = bws[ (int) HSIS_1344 ].bw_resv;

  agip->rmf = tcif_rmvectors[ (int) RM_P2P ].rmf;
#ifdef VIRTUAL_CLOCK
  /* agip->tcf = tcif_tcvectors[ (int) TC_VC ].tcf; */
  agip->alg_next = (short) TC_VC;
```

```
#else !VIRTUAL_CLOCK
  agip->alg_next = (short) TC_FIFO;
#endif VIRTUAL_CLOCK
  }
 }
#endif NHSIS

/*
cwl ??? do "lo", too? */

/*
cwl ??? fill in default dq/nq/enf */

 tcif_AlgSwitch( agip, /*quit*/0 );

 *tailpp = agip;
 tailpp = &( agip->ifc_nextp );

 agip++;
 tfic_genifs.nxtfree++;

 } /* end of all interfaces */

 *tailpp = &( tfic_genifs.NetIfs[0] );

 return;
}
#endif TRAFFIC_CONTROL


/*
 * Debugging hook.
 */

static int st2_stop = 0;

 static void
st2_at_stop () {};

 static void
st2_at_check () {};


/*
S* st2_dbgstp ( stopflag, errcode, unique_id, string, file_id, line_number )
s*
s* STIIDEBUG routine to print error message
s*
 */

 void    /* Next bugid 0x70700 */
st2_dbgstp( stopflag, errcode, unique_id, string, file_id, line_number ) /*
```

```
from */
 int    stopflag,
    errcode,
    unique_id;
 char   *string;
 enum aDbgString  file_id;
 int    line_number;
{
 struct atrap *trapp;
 int    oldpri;


 oldpri = splimp ();
 {
 trapp = (struct atrap *) st2_stats.trapnext;
 uniqtime ( &( trapp->time ) );
 trapp->trapid = (unsigned long) unique_id;
 trapp->trapinfo1 = (unsigned long) errcode;
 trapp->trapinfo4 = (unsigned long) file_id;
 trapp->trapinfo5 = (unsigned long) line_number;
 if ( ++trapp >= &( st2_traptable[ ConfigParm (ntraps) ] ) )
  st2_stats.trapnext = (unsigned long) &( st2_traptable[0] );
 else
  st2_stats.trapnext = (unsigned long) trapp;
 Count(trapcount,1);
 }
 splx ( oldpri );

 if ( ((ConfigParm (dbgflg) & (unsigned long) DBG_PRINT2) NE 0)
 OR ( (stopflag NE 0)
  AND ((ConfigParm (dbgflg) & (unsigned long) DBG_PRINT1) NE 0)) )
 {
 if ( string NE (char *) NULL )
  (void) printf ( "*** BUGCHECK :condition %s failed\n", string );

 (void) printf ( "*** BUGCHECK:%x reason %d (0x%x)",
   unique_id, errcode, errcode );
 (void) printf ( STStr (LineNo), line_number, STStr (file_id) );
 }

 if ( unique_id EQ st2_stop )
 st2_at_stop ();

 if ( stopflag < 0 )
 st2_at_check ();

 return;

} /* end of st2_dbgstp */


/*
```

```
S* st2_GetSpace ( pktpp, a1, a2, a3 )
s*
s* General routine to get memory space, accessed via the GetSpace &
s* AddSpace macros.
s*
 */

 unsigned char *   /* Next bugid 0x70815 */
st2_GetSpace( pktpp, a1, a2, a3 ) /* from */
 struct aPktDesc **pktpp; /* Only updated when PKT_MD_NEW */
 int    a1,
    a2,
    a3;  /* VARARGS3 */
{
 int    flag,
    provhd,
    pr,
    space,
    ps,
    psovhd,
    len;
 unsigned short *xap;
 struct aPktDesc *pp;
 struct mbuf  *mp;
 unsigned char *ucp,
   *xcp;


 if ( pktpp EQ (struct aPktDesc **) NULL )
 {
 IfLog (DBG_MEM_ALO)
  (void) printf ( "st2_GetSpace (invalid return pointer)\n" );

 BUGSTOP (-,InvalidPointer,0x70801);
 return ( (unsigned char *) NULL );
 }

 if ( (PKT_MD_BITS & a1) EQ PKT_MD_NEW ) /* New space */
 {
 flag = a2 | a3;
 flag |= (flag | a1) >> 16;
 provhd = (((0x0FFF & (a1 >> 16)) + ALIGNMENT) & ~ ALIGNMENT);
 pr = (((0x0FFF & (a2 >> 16)) + ALIGNMENT) & ~ ALIGNMENT);
 space = (0x0FFF & a2);
 ps = (((0x0FFF & (a3 >> 16)) + ALIGNMENT) & ~ ALIGNMENT);
 psovhd = (((0x0FFF & a3) + ALIGNMENT) & ~ ALIGNMENT);
 len = provhd + pr + space + ps + psovhd;
 len = (len + ALIGNMENT) & ~ ALIGNMENT;
 }
 else   /* Adding space */
 {
 len = (0x0FFF & a2);
```

```
 if ( ((PKT_MD_BITS & a1) EQ PKT_MD_PROVHD)
  OR ((PKT_MD_BITS & a1) EQ PKT_MD_PSOVHD) )
 {
  if ( (len & ALIGNMENT) NE 0 )
  BUGSTOP (-,len,0x70802 + ALIGNMENT);
  len = (len + ALIGNMENT) & ~ ALIGNMENT;
 }
 pp = *pktpp;
 if ( pp EQ NO_PKTDSCP )
 {
  IfLog (DBG_MEM_ALO)
  (void) printf ( "st2_GetSpace (%d) no existing buffer\n", len );
  Count (scmp_nobuf0,1); /* count no buf to use */
  BUGSTOP (-,InvalidPointer,0x70806);
  return ( (unsigned char *) NULL );
 }
 }

 if ( len > MCLBYTES )
 {
 IfLog (DBG_MEM_ALO)
  (void) printf ( "st2_GetSpace (%d too big for buffer (%u))\n",
   len, MCLBYTES );
 Count (scmp_nobuf3,1);  /* no buf big enough */
 BUGSTOP (-,((len << 16) | MCLBYTES),0x70807);
 return ( (unsigned char *) NULL );
 }


 switch ( PKT_MD_BITS & a1 )
 {
 default:
 IfLog (DBG_MEM_ALO)
  (void) printf ( "st2_GetSpace (Invalid request mode (%d)\n",
   PKT_MD_BITS & a1 );
 Count (scmp_nobufmod,1); /* count invalid GetSpace mode */
 BUGSTOP (-,Inconsistency,0x70808);
 return ( (unsigned char *) NULL );


 case PKT_MD_PROVHD:   /* 0x0100 */
 if ( len > pp->pkt_prepad )
 {
  IfLog (DBG_MEM_ALO)
  (void) printf ( "st2_GetSpace (cannot get %d OVHD bytes (%u))\n",
   len, pp->pkt_prepad );
  Count (scmp_nobfxo1,1); /* count pre-overhead too small */
  BUGSTOP (-,CantGetResrc,0x70809);
  return ( (unsigned char *) NULL );
 }
 ucp = (unsigned char *) pp + pp->pkt_offset - pp->pkt_prepad;
 pp->pkt_preovhd += len;
```

```
pp->pkt_prepad -= len;
xcp = (unsigned char *) pp + pp->pkt_offset - pp->pkt_prepad
    - pp->pkt_preovhd;
break;


case PKT_MD_HDR:   /* 0x0200 */
if ( len > pp->pkt_prepad )
{
 IfLog (DBG_MEM_ALO)
 (void) printf ( "st2_GetSpace (cannot get %d HDR bytes (%u))\n",
   len, pp->pkt_prepad );
 Count (scmp_nobfxhd,1); /* count packet header too small */
 BUGSTOP (-,CantGetResrc,0x70810);
 return ( (unsigned char *) NULL );
}
pp->pkt_offset -= len;
pp->pkt_prepad -= len;
pp->pkt_length += len;
ucp = (unsigned char *) pp + pp->pkt_offset;
xcp = (unsigned char *) pp + pp->pkt_offset - pp->pkt_prepad
    - pp->pkt_preovhd;
break;


case PKT_MD_PKT:   /* 0x0300 */
if ( len > pp->pkt_postpad )
{
 IfLog (DBG_MEM_ALO)
 (void) printf ( "st2_GetSpace (cannot extend PKT %d bytes (%u))\n",
   len, pp->pkt_postpad );
 Count (scmp_nobfxpk,1); /* count packet too small */
 BUGSTOP (-,CantGetResrc,0x70811);
 return ( (unsigned char *) NULL );
}
ucp = (unsigned char *) pp + pp->pkt_offset + pp->pkt_length;
pp->pkt_length += len;
pp->pkt_postpad -= len;
xcp = (unsigned char *) pp + pp->pkt_offset - pp->pkt_prepad
    - pp->pkt_preovhd;
break;


case PKT_MD_PSOVHD:   /* 0x0400 */
if ( len > pp->pkt_postpad )
{
 IfLog (DBG_MEM_ALO)
 (void) printf ( "st2_GetSpace (cannot get %d trailer bytes (%u))\n",
   len, pp->pkt_postpad );
 Count (scmp_nobfxo2,1); /* count post-overhead too small */
 BUGSTOP (-,CantGetResrc,0x70812);
 return ( (unsigned char *) NULL );
```

```
}
pp->pkt_postpad -= len;
pp->pkt_postovhd += len;
ucp = (unsigned char *) pp + pp->pkt_offset + pp->pkt_length
    + pp->pkt_postpad;
xcp = (unsigned char *) pp + pp->pkt_offset - pp->pkt_prepad
    - pp->pkt_preovhd;
break;


case PKT_MD_NEW:   /* 0x0500 */

MGET ( mp, M_DONTWAIT, (0xFF & a1) );
if ( mp EQ (struct mbuf *) NULL )
{
 IfLog (DBG_MEM_ALO)
 (void) printf ( "st2_GetSpace (no free mbufs)\n" );
 Count (scmp_nobuf1,1); /* count no small bufs */
 BUGSTOP (-,CantGetResrc,0x70813);
 return ( (unsigned char *) NULL );
}

IfLog (DBG_MEM_ALO)
 (void) printf ( "MBUF: get m %8x st2_getspace type %2u %s\n", mp, mp->m_type,
 (mp->m_type < DimensionOf (mtype_names)) ? mtype_names[ mp->m_type ] : "" );

pp = (struct aPktDesc *) mp;
pp->pkt_nxtmsgp = NO_PKTDSCP;
pp->pkt_nxtbufp = NO_PKTDSCP;
pp->pkt_preovhd = provhd;
pp->pkt_prepad = pr;
pp->pkt_length = space;
pp->pkt_postpad = ps;
pp->pkt_postovhd = psovhd;
pp->pkt_usecnt = 1;
if ( pp->pkt_type EQ PKT_TCDATA )
{
 pp->pkt_tcflowp = (caddr_t) NULL;
 pp->pkt_tcrsrc = 0;
 pp->pkt_tckey = 0;
}


switch ( PKT_XA_BITS & a1 )
{
case PKT_XA_PROVHD:  /* 0x1800 */
 xap = &( pp->pkt_preovhd );
 break;

case PKT_XA_PR:   /* 0x0800 */
 xap = &( pp->pkt_prepad );
 break;
```

```
default:
case PKT_XA_PS:   /* 0x0000 */
 xap = &( pp->pkt_postpad );
 break;

case PKT_XA_PSOVHD:  /* 0x1000 */
 xap = &( pp->pkt_postovhd );
 break;

} /* end of extra switch */


if ( (len <= sizeof (pp->pkt_intdata))
 AND ((flag & PKT_LARGE) EQ 0) )
{
#if 1  /* BSD mbuf */
 pp->pkt_size = sizeof (struct aPktDesc);
 *xap += sizeof (pp->pkt_intdata) - len;
 ucp = &( pp->pkt_intdata[ pp->pkt_preovhd + pp->pkt_prepad ] );
 pp->pkt_offset = (int) ucp - (int) pp;
 pp->pkt_length = space;
 /* correct for info before and after pktdata in BSD model */
 pp->pkt_preovhd += (int) &( pp->pkt_intdata[0] ) - (int) pp;
 pp->pkt_postovhd += sizeof (pp->pkt_nxtmsgp);
#endif 1 /* BSD mbuf */

 pp->pkt_nxtbufp = *pktpp;
 *pktpp = pp;

 xcp = (unsigned char *) NULL;
 break;
}


MCLGET ( mp );
if ( pp->pkt_length NE MCLBYTES )
{
 (void) m_free ( (struct mbuf *) pp );

 IfLog (DBG_MEM_ALO)
 (void) printf ( "st2_GetSpace (no free cluster mbufs)\n" );
 Count (scmp_nobuf2,1); /* count no large bufs */
 BUGSTOP (-,CantGetResrc,0x70814);
 return ( (unsigned char *) NULL );
}

xcp = (unsigned char *) pp + pp->pkt_offset;

IfLog (DBG_MEM_ALO)
 (void) printf ( "MBUF: get cl %8x st2_getspace type %2u %s for %x\n", xcp,
mp->m_type,
```

```
    (mp->m_type < DimensionOf (mtype_names)) ? mtype_names[ mp->m_type ] : "", mp
  );

  pp->pkt_size = pp->pkt_length;
  *xap += pp->pkt_length - len;
  pp->pkt_offset += pp->pkt_preovhd + pp->pkt_prepad;
  pp->pkt_length = space;

  pp->pkt_nxtbufp = *pktpp;
  *pktpp = pp;

  ucp = (unsigned char *) pp + pp->pkt_offset;
  break;

  } /* end of mode switch */

  IfLog (DBG_MEM_ALO)
  (void) printf ( "st2_GetSpace (%d bytes at %x in %x/%x)\n",
    len, ucp, pp, xcp );
  return ( ucp );
}


/*
S* st2_init ()
s*
s* COIP Protocol Family initialization routine. Initialize
s* st2_ConTbl, st2_ConHsh, st2_nexthops, st2_vlinks, Generic
s* Network Interface, and Resource Management structures.
s* Register our ethertype/PPPtype with the ethernet/HSI/S
s* drivers.
s*
 */

/* HID_CONTROL 0 control */
/* HID_HELLO 1 HELLOs */
#define NSPECIALHIDS 2


 int     /* Next bugid 0x70900 */
st2_init()    /* from OS */
{
 int    i;
 struct aConHshEntry *conHashp;
 struct aST2pcb *cp;
 struct timeval  tv;
 static boolean  init_done = FALSE;


 /* Only do this once. */
 if ( init_done )
 return ( 0 );
```

```
 CheckAlignment1;

 st2_stats.trapnext = (unsigned long) &( st2_traptable[0] );

 printf ( "ST-II Software Copyright (c) 1991-1993 by BBN Systems and
Technologies\n" );

 uniqtime ( &tv );
 st2_stats.butSeconds = tv.tv_sec; /* too soon to have tod value */
 st2_stats.butFract = tv.tv_usec;

 /* do what a good compiler/linker ought to do ... ---------------------- */
  /* @#$% C cannot initialize specific array entries @#$% */
 {
 int   i,  /* Slot being verified */
    j;  /* Desired slot, to be vacated &
      replaced */
 struct aErrorXlate *eip,  /* Ptr to slot being verified */
   *ejp,  /* Ptr to desired slot */
    ej,  /* Contents for desired slot */
    ex;  /* Contents removed from desired slot */

 i = DimensionOf (st2_reasontable) - 1;
 eip = &( st2_reasontable[ i ] ); /* Bottom up should be fastest */
 for ( ; i >= 0 ; i--, eip-- )
 {
  if ( eip->flags EQ 0 )
  continue;  /* Unused slot, except NoError */

  ex = *eip;   /* Save error info */
  eip->flags = 0;  /* Free the slot */

  while ( ex.flags NE 0 )
  {
  ej = ex;  /* Info to be inserted */
  j = (int) ej.internal; /* Desired slot for it */
  ejp = &( st2_reasontable[ j ] );
  ex = *ejp;  /* Save contents of slot */
  *ejp = ej;  /* Put info in desired slot */

  } /* end of while loop */
 } /* end of for loop */
 }

 /* ------------------------------------------------------------------- */
 /* ??? get incarnation # from disk?? */
 /* ??? static int st2_active = 0;
 * and all st2 entry points check it
 */

 /* Record supported versions ------------------------------------------ */
```

```
{ octet4   flowspec_versions[1];

flowspec_versions[0] = 0;

if ( st2_fsver3 NE 0 )
 BitSet32 (flowspec_versions, 3);
if ( st2_fsver4 NE 0 )
 BitSet32 (flowspec_versions, 4);
if ( st2_fsver5 NE 0 )
 BitSet32 (flowspec_versions, 5);
if ( st2_fsver6 NE 0 )
 BitSet32 (flowspec_versions, 6);

ConfigParm (flowspec_versions) = flowspec_versions[0];
}

/* Initialize connection table ---------------------------------------- */

st2_ConTblTailp = &( st2_ConTbl[ MAX_STREAMS - 1 ] );
st2_ConTblTailp->nextp = NO_PCBP;
st2_ConTblFreep = st2_ConTblTailp;

for ( i = MAX_STREAMS - NSPECIALHIDS ; i > 0 ; --i )
{
cp = st2_ConTblFreep--;
st2_ConTblFreep->nextp = cp;
}

/* Install pseudo connection 0 for control ---------------------------- */

cp = &( st2_ConTbl[0] );
Bzero (cp,sizeof (*cp));
SetState (PCB,cp,pcb_state,CINS_PREOPEN);
cp->dspin = st2_CMPInput;  /* "Connection 0" EQ HID 0 EQ scmp */
cp->nextp = &( st2_ConTbl[0] ); /* Initialize circular queue */
cp->prevp = &( st2_ConTbl[0] );


/* Initialize connection hash table ----------------------------------- */

conHashp = &( st2_ConHsh[0] );
for ( i = (1 << HID_BITS) ; i > 0 ; --i )
{
conHashp->hid = 0;
(conHashp++)->cp = NO_PCBP;
}

/* Install pseudo connections 0 for control & 1 for HELLOs ------------ */

st2_ConHsh[ HID_CONTROL ].hid = HID_CONTROL;
st2_ConHsh[ HID_CONTROL ].cp = &( st2_ConTbl[0] );
```

```
 st2_ConHsh[ HID_HELLO ].hid = HID_HELLO;
 st2_ConHsh[ HID_HELLO ].cp = &( st2_ConTbl[0] );


 /* Call for initialization of each object class ---------------------- */

 { struct NextHop *next_hopp;
 unsigned long xstkdebug = ConfigParm (dbgflg);
 ConfigParm (dbgflg) = 0;

 /*lock st2_nexthops*/
 {
  /* initialize each table entry to empty */
  next_hopp = &( st2_nexthops.table[0] );
  for ( i = 0 ; i < DimensionOf (st2_nexthops.table)
   ; i++, next_hopp++ )
  {
   Bzero (next_hopp,sizeof (*next_hopp));
   SetState (NHP,next_hopp,nhop_state,FREE_HOP);

   next_hopp->nextp = st2_nexthops.freep;
   st2_nexthops.freep = next_hopp;
  }
 }/*unlock st2_nexthops*/

 ConfigParm (dbgflg) = xstkdebug;
 }

 { int link_index;

 /*lock st2_vlinks*/
 {
  /* Empty the free list for reconstruction. */
  st2_vlinks.first_freep = NO_VLINKP;
  st2_vlinks.last_freep = NO_VLINKP;

  /* Scan link table using link_entry. */
  for ( link_index = 0 ; link_index < (ConfigParm (vlnk_mask) + 1)
   ; link_index++ )
  {
 struct aVLink *link_entry = &( st2_vlinks.table[ link_index ] );

 /* Mark as not in use. */
 st2_VLinkReset( link_entry, (VLID)link_index );
 /* ??? error */
    /* ??? use incarnation # */
 if ( st2_vlinks.first_freep EQ NO_VLINKP )
  st2_vlinks.first_freep = link_entry;
 else
  st2_vlinks.last_freep->nextp = link_entry;
 st2_vlinks.last_freep = link_entry;
```

```
    }
   }/*unlock st2_vlinks*/
   }

   /* Estimate delay and other FlowSpec adjustments for each device */

   /* Initialize generic network interfaces ------------------------------ */
  #ifndef TRAFFIC_CONTROL

   tfic_init_gifcs();   /* everything should be ready */

  #else TRAFFIC_CONTROL

   if ( tcif_gifcheadp EQ (struct aNetIf *) NULL )
   tfic_init_gifcs();   /* do it for them */

   ConfigParm(def_genifs) = tfic_genifs.allocated;

  #endif TRAFFIC_CONTROL

   /* Reset resource allocation tables for each device ------------------- */
  #ifdef OBS
  | st2_RsrcInit();
  #endif OBS


  #ifdef NOTWANTED
  #if NHSIS
   /* Register our PPP type ---------------------------------------------- */

   hsis_register( &( st2_scmp_family ) );
   hsis_register( &( st2_stii_family ) );
  #endif NHSIS
  #endif NOTWANTED

  #if WANT_ETHERNET
   /* Register our ethertype --------------------------------------------- */

   st2_ether_family.ef_ethertype = ConfigParm (ethertype);

   if ( st2_ether_family.ef_ethertype NE st2_ether_family_in.ef_ethertype )
   ether_register( &( st2_ether_family_in ) );

   ether_register( &( st2_ether_family ) );
  #endif WANT_ETHERNET

   /* set active flag ---------------------------------------------------- */
   init_done = TRUE;

   return ( 0 );

  } /* end of st2_init */
```

```
/*
S* st2_IPAdrFunc ( fnc,ipadrp )
s*
s* Routine to try and encapsulate deatils of native OS network interfaces.
s* IPAdrFuncValidate: validate given IPAddress as one of ours.
s*
 */

 boolean    /* Next bugid 0x71000 */
st2_IPAdrFunc( fnc,ipadrp )  /* from api_1bind st2_PCBUpdate */
 enum IPAdrFuncs  fnc;
 INETADDR  *ipadrp;
{
 struct aNetIf *ifp;
 struct ifaddr *wifaddrp;
 boolean   go = TRUE;

 switch ( fnc )
 {
 default:
 case IPAdrFuncScan:
 return ( FALSE );
 break;

 case IPAdrFuncValidate:
 ForAllGifc (ifp, tcif_gifcheadp, go AND)
 {
  for ( wifaddrp = ifp->osifcp->if_addrlist
   ; go AND (wifaddrp NE (struct ifaddr *) NULL)
   ; wifaddrp = wifaddrp->ifa_next )
  {
  if ( wifaddrp->ifa_addr.sa_family EQ AF_INET )
  {
   if ( (octet4) ((struct sockaddr_in *)
    &( wifaddrp->ifa_addr ))->sin_addr.s_addr EQ *ipadrp )
   {
   go = FALSE; /* Found match, stop looping */
   }
  }
  }
 }

 return ( (go) ? FALSE : TRUE );
 break;
 } /* end of switch */

 return ( FALSE );
}
```