NAG 2-638

Electrical Engineering Department University of Delaware
Technical Report 90-9-3 September 1990

Highball: a High Speed, Reserved-Access, Wide Area Network

David L. Mills, Charles G. Boncelet, John G. Elias,
Paul A. Schragger and Alden W. Jackson

NASA-AMES

Abstract

This document describes a network architecture called Highball and a preliminary design for a prototype, wide-area data network designed to operate at speeds of 1 Gbps and beyond. It is intended for applications requiring high speed burst transmissions where some latency between requesting a transmission and granting the request can be anticipated and tolerated. Examples include real-time video and disk-disk transfers, national filestore access, remote sensing and similar applications. The network nodes include an intelligent crossbar switch, but have no buffering capabilities; thus, data must be queued at the end nodes. There are no restrictions on the network topology, link speeds or end-end protocols. The end systems, nodes and links can operate at any speed up to the limits imposed by the physical facilities.

This document presents an overview of an initial design approach and is intended as a benchmark upon which a detailed design can be developed. It describes the network architecture and proposed access protocols, as well as functional descriptions of the hardware and software components that could be used in a prototype implementation. It concludes with a discussion of additional issues to be resolve in continuing stages of this project.

Keywords: supercomputer networks, gigabit networks, reservation-TDMA, Highball architecture.

Sponsored by: Defense Advanced Research Projects Agency under NASA Ames Research Center contract number NAG 2-638 and National Science Foundation grant number NCR-89-13623.

Table of Contents

List of Figures

List of Tables

1. Introduction

Local-area networks operating at speeds approaching or exceeding 1 Gbps are not uncommon in supercomputer centers and other institutions where very high speed communications are required. Even the S-bus interface used in current Sun Microsystems products can operate at a peak rate of 800 Mbps. The fact that data paths can be wide and propagation delays small leads to straightforward architectures using centralized control and multi-wire, point-to-point transmission links. An example of current fast LAN technology is the High Speed Point to Point Interconnect (HPPI) architecture and protocol [ANS90]. In HPPI data are transmitted in blocks of 32- or 64-bit words using a connection-mode protocol operating at speeds up to 1.6 Gbps.

Wide area networks are usually constructed over dedicated or common-carrier transmission facilities. Whether satellite or terrestrial, digital networks are today built using a switching fabric of 1.5 Mbps (T1) or 45 Mbps (T3) channels and a 125-μs frame interval. These channels are then cross-connected and aggregated over multiplexed transmission facilities operating at rates to 2.4 Gbps. In future, it is likely that long distance, high speed digital networks will be constructed directly from the transmission facilities and interconnected with switching equipment using SONET [BEL88] interfaces operating at discrete rates from 51 Mbps (OC-1) to 2.4 Gbps (OC-48) in increments of 51 Mbps.

Current common-carrier digital networks operate in synchronous mode, where all network clocks are synchronized at the same 125-μs frame interval. Future networks may employ the Synchronous Digital Hierarchy (SDH) [ASA90], which uses the SONET interface. It may well happen that high speed, wide area networks (WAN) can be constructed using SDH transmission facilities, SONET interfaces and high speed switches co-located with the SONET termination equipment in the central office or user premises. SONET interfaces operate in a fixed configuration (i.e., not switched on either a circuit or a frame basis). At low and medium speeds up to perhaps 150 Mbps (STS-3) for a single user, packet-switching access technologies, such as Asynchronous Transfer Mode (ATM) or Synchronous Metropolitan Data System (SMDS) [HEM88], are appropriate for conventional interactive-access and bulk-transfer network services.

For unconventional applications at speeds much above 150 Mbps for a single user, packet-switching technologies may not be the best choice. This report presents the rationale for an alternative network architecture called *Highball* and a strawman design for a high speed network capable of operation at end-to-end transmission rates of 1 Gbps or greater over significant geographical distances. It is intended for applications involving very high traffic rates typical of supercomputing and space-science applications. The report describes the approach and rationale for a particular architecture and design involving hardware, software and protocol components for a prototype Highball network to be built and evaluated in the near future. It begins with a discussion of the assumptions made about the physical nature of the network and the implied constraints on its operation. It then summarizes various approaches for network architecture and design and develops a strawman model based on a reservation-time-division multiple access (R-TDMA) protocol and a high speed, externally scheduled crossbar switch.

The report continues with an overview of a specific plan for a candidate design and prototype Highball network to be built for demonstration and evaluation. The network links are constructed from cable or fiber operating at 100 Mbps and switched using a bitslice VLSI crossbar switch which can be configured for speeds from 25 Mbps to 1 Gbps by simply adding bitslices. The switches are externally actuated by a node controller and an ancillary processor using a distributed reservation protocol and scheduling algorithm. The report then describes the principal hardware and software components and how reservations are made, schedules are constructed and nodes are synchronized. The report concludes with a discussion of issues to be studied and resolved in future work.

While the technology discussed in this report is believed new and subject to review and change as development proceeds, it should be recognized that many of the ideas embodied in the scheduling and synchronization algorithms have evolved from prior work, most notably the DARPA Packet Satellite Projects (SATNET and WIDEBAND [HA86]), Multiple Satellite System (MSS), the NASA Advanced Communications Technology Satellite (ACTS) and the Network Time Protocol (NTP) [MIL90c].

1.1. Overview

The Highball Project is a research effort designed to specify and ultimately build a fast, wide area network. It is designed to operate at gigabit speeds or higher, although nothing in the design specifies a particular speed. Connections between nodes are either multi-conductor cable or fiber optic, usually occurring in pairs, one cable or fiber for each direction. Nodes are crossbar switches, capable of connecting any input to any or to many outputs.

The network is designed for high demand, bursty users. Typical users include supercomputers, real-time disk-to-disk transfers, interactive video, and visualization on remote workstations. These users can individually demand a gigabit or more, but do so for brief, bursty periods. In contrast, conventional high speed networks are usually designed for highly multiplexed aggregates of low rate users. The statistics are very different for bursty users than for a multiplexed aggregate. A network designed for multiplexed, low rate users is ill-suited for bursty, high rate users.

Because of the high speeds involved, communication bursts are 'short' compared to the size of the network. For instance, even a megabit burst at 1 Gbps is only a millisecond long and flies across the country in about 30 ms. Instantiating a virtual circuit for a burst is inefficient. On the other hand, store and forward networks with acknowledgments are infeasible at these speeds.

The scheduling problem is perhaps best understood in terms of an analogy to a railroad of high speed trains. The trains have finite size and are small compared with the distance across the network. We desire to dispatch trains without collisions and without stopping a train once started. Furthermore, unlike real train networks, we cannot signal ahead because our trains are traveling as fast as the signals. The scheduling problem is when to start the trains and how and when to set the switches along the way. One complication is that we desire to transmit many trains concurrently, each going in different directions. This scheduling analogy also gives the Highball Project its name. In the vernacular of trains, ***highball*** means that the train has priority and can travel as fast as the tracks allow. We are trying to allow our trains to travel as fast as their tracks allow.

Time synchronization is a significant hurdle for this network. Bursts propagate at about two-thirds of the speed of light and are switched on-the-fly. The entire network must agree on the clock phase to within about one microsecond, while delays between nodes must be known as accurately. We believe the network clocks can be made this accurate using the methodology of NTP [MIL90a], which has recently been extended for just this purpose.

1.2. Traffic Models

Most traffic carried on existing interconnected networks such as the Internet consists of some variants of interactive terminal, bulk-data transfer or remote-procedure-call applications. The key observation here is that the traffic generated by any single application is usually much less than the capacity of a single network link, so that traffic from many applications can be aggregated for transmission over that link. As the degree of aggregation increases, the behavior of the resulting traffic flow more closely matches well known probabilistic models of network behavior. Systems such as SMDS and ATM, which are designed for packet-switched voice and video services at rates to 150-Mbps for individual users, are examples of conventional approaches to the multiplexing problem.

However, recent scientific applications of high speed networking have tended to fall in the areas of supercomputing, remote sensing and video applications of various kinds. The typical application involves a massive file transfer between two supercomputer sites or between a site and a remote workstation, often in the form of full-motion video. An uncompressed 1024 x 1024 x 24 raster requires over three megabytes of storage and, refreshed at 60 frames per second, a sustained data rate of about 1.5 Gbps. Supercomputers, as well as some workstations and other devices are capable of burst transfers at 1 Gbps and beyond. For instance, the S-bus used in Sun Microsystems SPARCstations can operate for short burst rates of one 32-bit word every 40-ns clock cycle, or 800 Mbps, while available high speed shift registers provide a serial/deserial function at over 2 Gbps.

When these applications are mounted on a WAN, considerable performance degradation can occur due to large end-end delays. For instance, a 5000 km path over a fiber facility with a propagation constant of 0.6 incurs a one-way delay of about 28 ms, or about 3.5 megabytes at 1 Gbps. The latency due to this delay usually dominates all other processing and data-transfer latencies in the application. If retransmission (ARQ) methods are used, rather than forward-error-correction (FEC) methods, a sender may have to buffer over seven megabytes of data between initial transmission and arrival of the acknowledgement. Proposed FEC transmission methods [MCA90] take advantage of the fact that fiber digital networks may suffer far fewer errors than older technologies; however, protocols designed for LAN internetworking, such as SMDS, are designed to drop packets under conditions of heavy load, which might easily overwhelm these methods.

On the other hand, it is not likely that the users of a switched, high speed WAN will be using interactive services in the usual sense, but will need service in the form of giant bursts of information where delay and delay dispersion are most important. For example, consider a bulk file transfer between two supercomputers equipped with parallel-access disks rotating at 3600 rpm (16.7 ms per revolution). At an 800-Mbps transfer rate as in HPPI, each revolution delivers over 1.6 megabytes, which means that well over one full revolution can be in flight across the continent. However, most disks require up to a few tens of milliseconds to perform a seek operation and wait for the next sector to arrive, so that the average data rate may be somewhat less than the peak data rate.

As another example, consider a frame buffer for a high resolution, motion-video workstation designed for medical imaging or pilot training. Depending on compression technique and frame-update rate, network rates exceeding 1 Gbps may be required. While at frame rates of 60 Hz it is possible that an entire frame may be in transit over the network, it is predictable when these frames will occur. Both this application and the previous one are examples of applications requiring timely access in high speed bursts to a dedicated medium and suggests the use of R-TDMA technology.

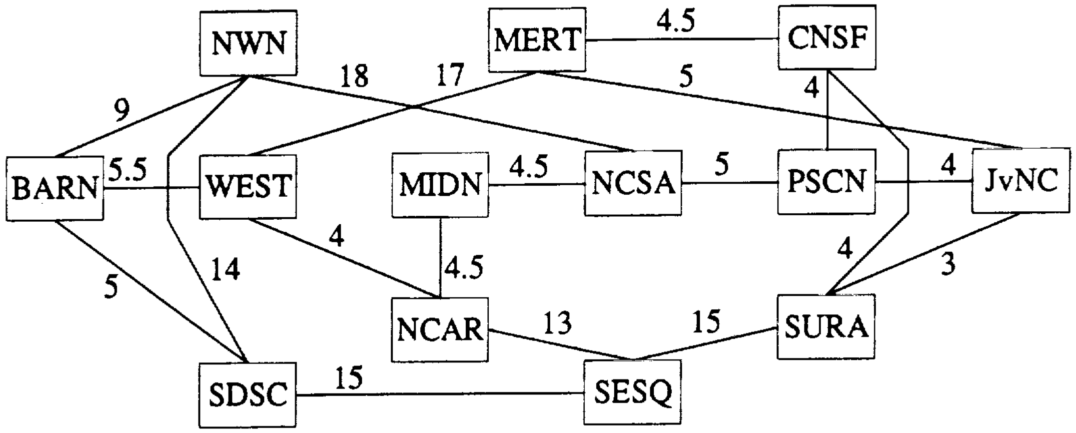


Figure 1. NSFNET Backbone Network

Other examples in which R-TDMA technology may be used effectively include space science and surveillance. The Advanced Communications Technology Satellite (ACTS), which uses satellite-switched TDMA (SS-TDMA) and transponders that operate up to several hundred Mbps, would appear to be an ideal candidate for this technology. The Space Station [SMI90] is to operate using a 300-Mbps space-ground link multiplexed for telemetry, video and voice. As the reported plans include a manual patchboard for the various video feeds on the station, it would seem a safe bet that delays for reservations in a R-TDMA system would not be a problem. In other systems used for surveillance, astronomy and land-resources management a R-TDMA approach would be equally effective.

The architecture and design proposed herein for a high speed WAN has been specialized for high speed burst applications and a modest user population. A prototype Highball network might at deployment consist of from a few to a few tens of fiber trunks and include nodes at supercomputer centers and participating workstation sites. Figure 1 shows how such an overlay might be added to the present NSFNET Backbone network, for example. The nodes shown correspond to NSFNET regional gateways, while the connecting links operate at 1.5 Mbps or 45 Mbps speeds. If all the links were upgraded to use 1-Gbps fiber facilities, for example, the link delays, in milliseconds, would correspond to the labels shown for each link. The primary service of this network would be to deliver very large quantities of data over very long distances as fast as possible and schedule network access on a fixed or dynamic basis. It would be capable of several bulk-reservation modes appropriate for bulk-data transfer and sustained motion-video refresh in both unicast and multicast modes.

1.3. Design Approach

In conventional packet-switched systems data are packetized and sent from node to node to the destination. In many designs each node holds a packet until an acknowledgement is received indicating the packet has been successfully received at the next node and queued for onward transmission. The total end-end delay is the sum of the individual propagation delays on the links between the nodes, plus the delay for the packet to be received and stored in a buffer, plus the queueing time at each node. In the case of ARQ systems each node must buffer data for at least the roundtrip time to the next node, plus the receiving and queueing time for the packet and its acknowledgement, plus various processing times. Summing the buffering requirements over a 5000 km path, for example, reveals that at least seven megabytes of storage will be required, a portion of that at each node, and this storage and the algorithms controlling it must operate at 1 Gbps. While it is in principle possible to implement a 1-Gbps packet-switched network based on this technology, the network could become enormously expensive, especially as the anticipated number of users of such a network might not be large.

As the network speed approaches 1 Gbps and its size approaches transcontinental proportions, traditional sliding-window transport protocols become most awkward. While at T0 speeds (64 kbps) and typical roundtrip transmission delays of 300 ms the window need be only a couple of kilobytes, at T1 speeds and roundtrip delays of 100 ms the window size must be at least 20 kilobytes and at 1 Gbps and roundtrip delays of 56 ms the window size must be several megabytes. With this in mind and the expectation that transmission errors are likely to be very low with fiber transmission, future transport protocols are likely to use FEC for primary error control, selective, delayed-retransmission for secondary error control and reserved or rate-based flow control. Such designs are well suited to R-TDMA network architectures.

In conventional packet-switch designs a header on each packet is interpreted to determine the output port for the next link on the path and the packet queued for transmission on that port. The algorithm to determine the output port typically involves parsing the header and then searching a table, perhaps assisted by hashing methods, to find the port number. These operations can be computationally intensive to the point of impracticality at 1 Gbps. Recent designs for high speed packet switches involve either a self-routing or externally switched architecture. In self-routing designs the switch fabric itself interprets an address in the header of each packet, indexes into a routing table in high-speed local memory and selects the outgoing port and next address. With sufficient buffering and parallel processing, sustained nonblocking throughputs of many Gbps are possible, but with considerable switch complexity and protocol specificity. In principle, self-routing switch designs could be developed for operation at 1 Gbps for a single user, perhaps by paralleling the switching fabric and performing a serial-parallel conversion at both ends of each link. However, the size, power and expense of such a switch appears prohibitive with present technology.

An alternative approach is to use an externally switched architecture, in which the timing and routing of the packets are determined by precomputed scheduling tables in high speed local memory at each node. The actual switching fabric takes the form of a crossbar switch driven by a system of counters which precisely determines the switch configurations and dwell times from the scheduling tables. The tables are computed using a synchronized, distributed algorithm which runs in a dedicated processor at each node. Since the nodes need not inspect the packet headers or data formats used for end-end traffic, the network can support a variety of protocols, packet formats and encoding methods. In addition, while other methods can be adapted for multicast delivery, in the externally switched crossbar approach a multicast capability is inherent.

The externally switched approach appears to be ideally suited for a high speed WAN. For reasons of switching speed, cost and availability at the present state of development, electronic crossbar switches are more practical and represent the technology of choice for a prototype Highball network. However, the approach may be particularly attractive should appropriate photonic switches become available, in which case the network speeds are limited only by the characteristics of the fiber and switch itself and no electronics need appear in the data paths.

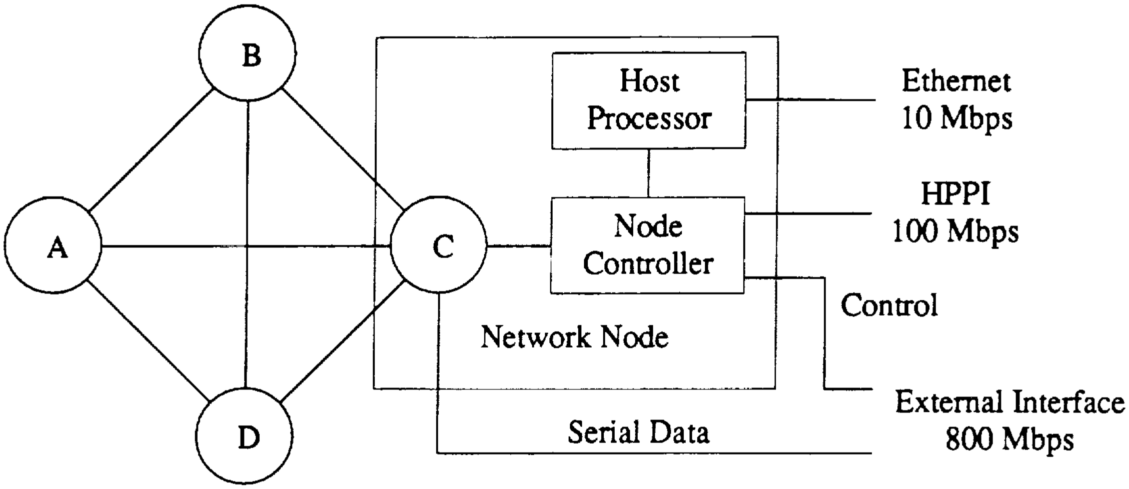


Figure 2. Reservation TDMA Network

The network design described in this report is based on an electronic crossbar switch assembled using specially designed VLSI chips fabricated using MOSIS and inexpensive 2-micron technology [DAV90]. While these devices are presently limited to 25 Mbps, it is expected that devices operating at speeds beyond 100 Mbps can be readily fabricated using other technologies at acceptable cost. A bitslice design is used in which the basic switch is one bit wide (25 Mbps) and increasing in powers of two to 32 bits (1 Gbps). Appropriate buffering and serial/deserial facilities are required at the fiber termination points. The network is operated in a synchronous, R-TDMA mode with a 40-ns symbol time and up to four bits per symbol. The design includes the VLSI switch, a dedicated RISC-based node processor with supporting logic and a general-purpose workstation which supports the system and experiments which may be run on it.

2. Network Architecture

Figure 2 shows the overall organization of a Highball network node and data interface arrangement. The network consists of a number of nodes interconnected by high speed links operating at some factor W times the basic rate of the switching fabric, where W is the number of bits per symbol. A node consists of a crossbar switch, node controller and host processor. For speeds up to 10 Mbps data can be transferred using the standard Ethernet connection provided by the host processor. For speeds up to 100 Mbps data can be transferred directly to the node controller using either a twisted-pair, ribbon cable or HPPI adapter, which might be implemented as part of the node controller circuitry. For the highest speeds supported by the crossbar switch and internode links an appropriate interface is required at the external processor supporting the application. Input/output transfers are enabled by the node controller, but the timing and formatting of the data burst itself is the responsibility of the external interface.

In the prototype Highball network the switching fabric operates with four-bit symbols, one bit for each bitslice, a 40-ns symbol time and a 32-bit word for an aggregate rate of 100 Mbps. In principle, higher speeds can be handled by increasing the number of bits per symbol and doing serial-parallel conversions on the link interfaces. Ordinarily, this requires an isochronous network; that is, one where clocks in all network nodes are synchronized to within less than the symbol time, in this case 40 ns, or some amount of elastic storage must be provided on each link. For the prototype Highball network, where programmable shift registers are used to simulate link delays, the elastic storage is built-in. Alternatively, the W bitslices can be switched independently, providing W separate 25-Mbps subnetworks, each with one bit per symbol. There is no intrinsic requirement to operate at the full network data rate, only an agreement to clock the transfers to and from the network and to agree with the receiving end on protocol and data rate. This design is well suited for space telemetry and application data, which are often encoded separately by function and transmitted over a single path.

The 40-ns switching fabric is compatible with both FDDI and HPPI and can support network speeds in principle to over 1 Gbps with appropriate choice of W. With W bits transmitted per symbol and a 40-ns symbol time, appropriate values of W are 1 for a 25-Mbps demonstration network, 4 for a 100-Mbps network of FDDI capacity and 32 for a 800-Mbps network of HPPI capacity. The Highball prototype network is designed initially for 100-Mbps aggregate link speeds, a 4-bit switching fabric and four nodes located in near proximity. The links may use ordinary twisted-pair ribbon cable, with link delays simulated by special-purpose hardware. Later development stages may include the use of the campus fiber plant, higher crossbar speeds and larger values of W. However, the intent is eventually to increase the switching speed in order to reduce the size of the crossbar switch and to reduce the complexity of the serial/deserial operations required. The present design allows the use of available FDDI, TAXI and FOXI encoder-decoder chipsets made by Advanced Micro Devices and others operating at speeds to 100 Mbps.

A network node consists of a node controller and host processor, as shown in Figure 2. Nodes are of two types, intermediate nodes and end nodes. Intermediate nodes originate no traffic other than hello bursts, which are used for synchronization and data-burst scheduling. They have no capacity to store switched data and operate only in a cut-through mode. End nodes send data bursts and in addition send reservations in the hello bursts. Since the network itself provides no data storage, end nodes must queue the data pending permission of the distributed scheduling algorithm to send it. For flexibility in the prototype Highball network, all nodes can operate both as intermediate and end nodes.

All nodes, including intermediate and end nodes, contain a node controller and host processor to synchronize the network and run the scheduling algorithm. The node controller is designed as a programmed peripheral for a general-purpose workstation with a bus architecture capable of speeds at least that of the network links. It consists of link interfaces, a crossbar switch, switch controller, node processor, input/output controller, master clock and processor interface to be described later. The host processor consists of a general-purpose workstation using RISC technology and the Unix operating system. In the proposed design the host processor is a Sun Microsystems SPARCstation, which includes the S bus, a fast backplane interconnect that can operate at burst rates to 800 Mbps. The workstations serve multiple purposes as host processor, development platform and application testing environment.

Nodes communicate among themselves using a signalling protocol and data transceiver included in the node controller. The data transceiver includes the necessary encoding-decoding and clocking functions common to other systems such as FDDI, so that chipsets for these systems may be used. While the data transceivers are necessary for signalling between the nodes of the network, they can in principle also be used to carry ordinary end-end data that originates in the node itself. This would occur if the node were used as an Ethernet gateway or monitoring platform, for example. However, it is also possible for the actual data burst to originate in some other processor attached to the switch, such as an HPPI interface. Appropriate means to do this are reserved for further study.

The basic switching fabric is constructed of an N x N x W asynchronous crossbar switch in which each of N W-bit input lines can be connected simultaneously to any combination of N output lines or to none, but no output line can be connected to more than one input line. For N = 8, as in the prototype VLSI switch, the configuration is described by a vector of eight four-bit components, one component associated with each output line, with the value of the component indicating which input line is connected to it. The prototype VLSI switch operates at a data rate of 25 Mbps with a configuration dwell time of at least 10 μs and an interburst guard time at least 1 μs.

For the prototype 100-Mbps Highball network, four 8 x 8 VLSI switch bitslices are used, with each bitslice operating at a 40-ns symbol time and having an independent configuration and dwell-time capability. Thus, the network can be configured as one 100-Mbps network, two 50-Mbps subnetworks or four 25-Mbps subnetworks, with the dwells of each subnetwork independently scheduled. This design supports the fully connected configuration shown in Figure 2, as well as additional inputs from external devices and exotic configurations including storage rings. While the switch presently operates in binary mode and has been implemented using technology which is not the most advanced available, there is no compelling reason why it could not be realized in analog form or operated at much higher speeds and used to switch analog video, for example.

2.1. Network Protocols

One of the most interesting aspects of this project is the design of the algorithms for synchronizing the nodes and computing the transmission schedules used by the network switches and client hosts. There is a spectrum of algorithms that could be used from nondeterministic distributed ALOHA algorithms to receiver-directed algorithms such as proposed for MSS and deterministic centralized reservation algorithms similar to those used in SATNET/WIDEBAND [JAC77]. It is even possible to operate the network on a store-and-forward basis by staging transmissions in the node controllers themselves. However, the most interesting and useful algorithms are distributed R-TDMA algorithms similar to the PODA algorithm used in the SATNET and WIDEBAND satellite systems. Approaches leading to useful Highball designs are presented in following sections.

2.1.1. Slotted-ALOHA Mode

Slotted ALOHA (S-ALOHA) usually refers to a system involving a single contention bus where all nodes are synchronized to a common slot clock and each node can transmit in a slot a single packet to any or all other nodes. To simplify the exposition here, the link delays are assumed a multiple of the slot interval, so that all switch schedules in the network can be aligned to a slot boundary. For each of the N output lines of an N x N crossbar switch, one of N input lines may be selected; therefore, the switch configuration can be described as a vector of N $\log_2 N$ bits. In a network of M nodes, the network configuration can be described as the vector consisting of the concatenation of the M switch vectors. For instance, the present NSFNET Backbone network shown in Figure 1 can be constructed from thirteen 4 x 4 crossbar switches and represented by a network configuration vector of 104 bits. Note that N has been increased by one over the three shown in Figure 2 to support the node itself. This vector completely describes the state of the network at each slot interval.

An interesting and useful exercise is to explore modes of operation which require little information to be known beforehand. In S-ALOHA the senders are synchronized to the slot clock, but know nothing about the network topology, link delays or configuration vectors. In order to provide fair service for all end-end paths in the network, both point-point and multicast, it is necessary to generate a repetitive sequence of most or all values of the network configuration vector in which each value occurs at approximately the same frequency. Ideally, and unless some stochastic conspiracy exists to the contrary, all possible end-end paths should occur and the mean intervals between occurrences should be roughly proportional to path length.

A simple way to do this is to generate a deterministic, pseudo-random (PR) sequence assumed known by all nodes in advance. The sequence should have good statistical properties; that is, its period should be very long compared to the length of the network configuration vector and have a sharply peaked autocorrelation function with low sidelobes and have a low cross-correlation function relative to all subsequences. For instance, a PR sequence can be generated by a shift-register generator (SRG) which performs multiplications over the Galois field of 2^n elements, where n is the length of the network configuration vector. Using a primitive polynomial as the basis, the SRG would have n stages and would generate all $2^n - 1$ nonzero values, which also establishes the sequence period. For the NSFNET example, n = 104, which even for slots as small as 40 ns, is much greater than the known age of the universe. While there will in general be many values of this 104-bit quantity that support a particular end-end path, waiting for a particular value, such as all ones, would not be recommended. Note that using a PR sequence generated in this way is not feasible, unless a way were provided to permute the vector to reduce the correlation effects induced by the shift register. Another way to do this may be using the DES algorithm as a code generator.

It is necessary to map a sequence produced in this way to the switch schedules. This is easily done by segmenting the sequence into subsequences matching the switch configuration vectors. This results in each vector taking on all possible values and thus producing all possible switch configurations for both point-point and multicast modes, although not all configurations may be interesting and some might result in loops between nodes (such loops might be useful). Each node maintains a sequence generator, but the generators in different nodes do not need to be started at the same time and do not in principle even have to generate the same sequence.

In S-ALOHA a sender simply emits a packet in a convenient slot or slots and retransmits if no acknowledgement is heard. Since by construction no switch schedule can result in the same output line connected to more than one input line, collisions in the network are impossible; however, there may be an uncomfortably high probability that during any particular slot no complete end-end path between the sender and its intended receiver exists. If not, or a packet is delivered to the wrong receiver, the packet is simply discarded. The remedy for this may be to send the same packet in more than one slot; or, for that matter, send it in every slot (perhaps in round-robin fashion with other packets waiting in queue) until acknowledged. Strategies for doing this will not be examined further, since better methods are available, as described below.

2.1.2. Fixed-TDMA Mode

As in other nondeterministic contention systems, S-ALOHA transmissions may not succeed in reaching the intended receiver and may have to be retransmitted, possibly many times. This could lead to unacceptable packet delays and network inefficiencies. However, it is possible to operate the network in a deterministic, fixed-TDMA (F-TDMA) mode. In F-TDMA all nodes are synchronized by slot, the PR sequence used by each node is known and all sequences are started at the same global time. In principle, each sender can determine the configuration of all switches in the network and use only those slots for which a complete end-end path exists.

A considerable amount of computation may be necessary to determine if a complete end-end path exists. However, since the origin of the PR sequence is known at each node along with the delays (in slot intervals) of all links, a list of slot numbers and destinations to which paths exist can be precomputed and preserved in node local storage. One way to do this is to compute from the given topology a number of likely paths from each sender to each receiver and group of receivers (for multicast). For each node on a path the slot time relative to the sender can be computed from the link delays and used to define a vector consisting of ones at the computed slot times and zeros elsewhere. It is a simple matter to match these vectors against successive shifts of the PR sequence to determine the slot number when one or more of the paths become available. For networks enjoying more than a modest size and rich connectivity, the list so constructed is likely to grow quite large. Even for networks of the size of the NSFNET example, the list would probably be so large as to be impractical.

An F-TDMA network constructed using the above methods is not likely to be very efficient when the offered traffic is not generally uniform among all the nodes. It does not seem feasible to tune the network for nonuniform traffic by artfully crafting the PR sequence. A better approach to this problem may be using reservation modes and a synthesized schedule, as discussed in the next section.

2.1.3. R-TDMA Mode

While F-TDMA packets are sent only when a complete end-end path exists and so are not normally lost, there may be a latency lasting up to a complete cycle of the PR sequence when no such path exists. In addition, the method described for generating the PR sequence itself makes no provision for topological optimization or for capacity control. These functions require the synthesis of a dynamic switch schedule as a function of network topology and offered traffic. A natural way to do this is using a reservation-TDMA (R-TDMA) mode.

R-TDMA requires that slot reservations be broadcast to all node controllers and other hosts in the network. Very likely it will prove more efficient to have the nodes themselves do this on behalf of the LAN hosts, perhaps communicating with the hosts using an out-of-band path such as shown in Figure 2. Since the network can easily be configured for broadcast, this requires a prior-reserved interval for each node in which the network configures itself as a minimum spanning tree routed on each node in turn and sustains each configuration for a prior-agreed time during which reservations can be sent. The scheduling problem itself represents an interesting and challenging problem similar to scheduling multiple trains on a richly connected railroad, which lends the architecture its name, while allowing for propagation delay, avoiding collisions and without stopping the trains.

There are several types of services that might be supported by a comprehensive R-TDMA protocol. The simplest is a bulk reservation for a specified quantity of data, transmission rate and scheduling constraints such as deadline for completion, priority, etc. The distributed algorithm responds with a list of times and durations in which the contents can be sent. The list is entered in the scheduling table and the interface is enabled to transmit at the specified time(s). For stream data such as motion video, the reservation specifies the minimum rate and maximum jitter and the algorithm responds with a similar list and recurrence rate. This reduces the latency from twice across the network to only once, as well as the resources to compute the schedule. An intrinsically useful property of this architecture is a broadcast capability which might be very useful for video teleconferencing.

The entire network must be synchronized to within some small number of 40-ns clock cycles. However, the expected delay, jitter and wander caused by temperature variations, etc., may be much larger than that. Although the expected long-term wander should occur over periods which can be reliably tracked and compensated, it is unclear what the effects of the short-term jitter will be. An appropriate mechanism for synchronizing the network may be NTP, which involves the exchange of timestamps over the network and the adjustment of precision frequency and time references [MIL90b].

2.2. Scheduling Algorithms

We are investigating many paradigms for scheduling bursts and believe that two are most promising. The first, called reservation-TDMA or R-TDMA, operates in the following manner: When a source node decides to transmit a burst, it sends a request to all other nodes on a special reservations channel or burst. After a suitable waiting time for the reservation to propagate across the network, all nodes schedule the request. Since all nodes implement the same algorithm on the same data, they all arrive at the same schedule. At the appropriate time, the source transmits the burst and switches along the way are set 'just in time' to allow this burst to traverse through to the destination without colliding with other bursts. As demonstrated later in this document, R-TDMA scheduling efficiencies can be very high, exceeding 80% throughput. The biggest disadvantages of this mode are that the computational burden of scheduling individual bursts is high and that bursts must wait for the reservations to propagate before being sent. As described later in this report, we are working on mitigating both problems.

The second paradigm involves incrementally adjusting a repeating schedule in the following manner: Each node has a repeating schedule of when it can transmit to each other node. A node queues bursts directed to a given destination until the appropriate slot is available, then transmits them. When a node finds itself generating too much traffic to a given destination, it sends out a requests for more capacity to that specific destination. All these requests are received by all nodes, arbitrated and new schedules computed. This paradigm incurs little delay and the computational burden is low, since only changes must be computed. The disadvantage is that, in short time periods, the schedule may not reflect immediate demand.

No matter which paradigm is followed, the actual scheduling problem is quite difficult. Several seemingly easy variations are NP-complete. As a result, we are concentrating our efforts of finding and developing good heuristics. One heuristic described in detail later in this document is to freeze the current schedule as computed, then schedule new reservations within open times, generally in a greedy fashion. It is not obvious what choices to make; however, the algorithms we have developed can choose among several criteria, including earliest-arrival-time, earliest-starting-time, least-hops, shortest-transit-time, or a combination of these. When multiple reservations occur, the best sequence in which to schedule them is not at all obvious. We are investigating several choices, including earliest-reservation-origination-time, longest-burst-first, longest-expected-path-first, most-expected-hops-first, or a combination of these.

2.2.1. Breadth-First Algorithms

One promising scheduling algorithm developed in this research is a variant of breadth-first search. In simple networks, breadth-first search is an easy way of finding shortest paths from a given node to all other nodes. The scheduling algorithm finds shortest paths from the source node to all other nodes at all possible times.

The breadth-first algorithm operates in the following way. Each link maintains a timeline with four states: NO, meaning that the link is busy at this time; NOSTAR, the link is not busy, but the burst cannot fit in the available time slot; MAYBE, the link is not busy and the burst can fit, but it is unknown if the burst can arrive here at this time; and YES, the link is available, the burst can fit, and a path does exist. Initially, all timelines are in the NO and MAYBE states. The first step is to mark all the NOSTAR's. NOSTAR's are treated exactly like NO's except that at the termination of the scheduler, they are returned to MAYBE's.

The goal of the breadth-first algorithm is to change MAYBE's into YES's. The source link is marked first and put on a FIFO. Until finished, the algorithm pops a link, l, off the FIFO and examines its successor links (those that can be connected to it by the switch). Call a successor s. At a time t, if the first link's timeline is YES and its successor's at time t + delay(l) is MAYBE, then change the MAYBE to YES and add s to the FIFO. Eventually, when s is popped off the top, its successors will be checked and possibly updated.

Assuming the search is terminated at some finite time, we can guarantee that this algorithm finds a path if one exists, and terminates in finite time. By carrying along more data in parallel to the timeline, other performance criteria can easily be accommodated. For instance, it is a simple matter to count the number of hops and choose the least-hop path.

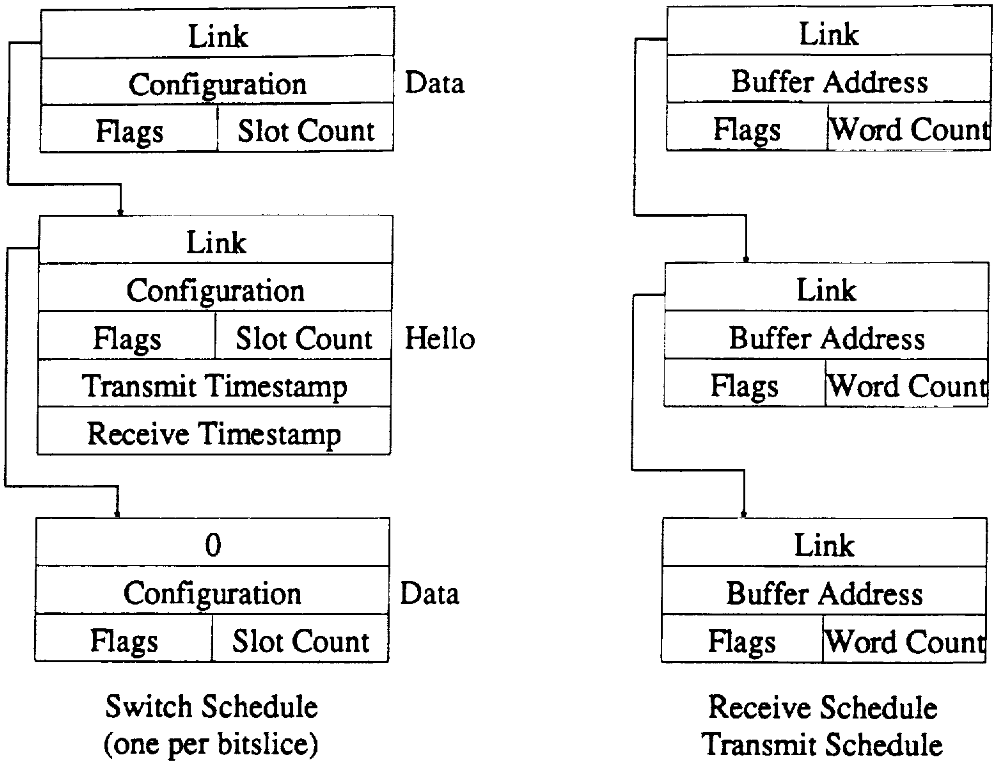
Simulation results are encouraging. On a simulation of the NSF Backbone network (13 nodes and 19 links) with 1 ms bursts, we found scheduling efficiencies of 80-90%. On average, each node was sending and receiving 80-90% of the time. A drawback with the algorithm is that computing a schedule for a single burst requires about 5 ms on a Sun SPARCstation, our intended computation engine. This is too long and would imply inordinately long bursts (60 Mbytes/s). Fortunately, we have some ideas to speed the computation and are currently testing their efficacy.

2.2.2. Incremental Algorithms

An alternative to full R-TDMA is an incrementally changed, repeating schedule. The idea is that a repeating schedule is determined a priori. As demand changes, the schedule is slowly adjusted to match. These changes may persist for a relatively long time, such as for a video conference. The computational burden of R-TDMA would be largely avoided, since schedule changes should be far less frequent than individual bursts.

The breadth-first scheduling algorithm above can solve this problem with slight changes. Treat the timelines as circular arrays, with a repeating period, p, in units of the burst length. Guess p and try to schedule a burst from each node to every other. If successful, reduce p and repeat. Clearly, p cannot be less than $n - 1$. The simulation results indicate that p close to n should be possible.

After a repeating schedule is set up and operating, nodes can request more or less bandwidth to specific destinations. All nodes receive these requests, arbitrate them, and modify the schedule accordingly. A potential disadvantage of this paradigm is that the scheduler may be too slow to react



Switch Schedule
(one per bitslice)

Receive Schedule
Transmit Schedule

Figure 3. Burst Schedules

to the short term effects. These issues arose also in the design of the SATNET and WIDEBAND systems, from which valuable lessons can be learned.

2.2.3. Other Algorithms

Two other specific scheduling algorithms developed so far are the Trailblazer and Pathfinder algorithms [SCH91]. These algorithms are described in detail in a later section of this document. Trailblazer optimally calculates schedules via enumeration of the non-intersecting trails on a slotted network. Since this problem is NP-Hard, Trailblazer requires exponential time in the number of requests to be processed and in the size of the network. Pathfinder may not always produce optimal schedules, but is designed to run much faster than Trailblazer. Pathfinder applies a greedy assignment algorithm to an unslotted network. Simulations indicate that Pathfinder is effective at finding paths; however, additional enhancements may be necessary in order to cope with short packets at the full 1-Gbps rate.

2.3. Burst Formatting and Framing

The prototype Highball network is to operate in a time-division, multiple-access mode where capacity is assigned in the form of variable length dwells in each of which a single burst can be transmitted. Four bits per 40-ns slot, one for each bitslice, can be transmitted, received and switched in each dwell. In order to transmit a burst consisting of a number of 32-bit host-processor words, a node broadcasts a dwell reservation specifying this number of words to all other nodes in the network. As reservations arrive at a node, the local switch schedule, transmit schedule and receive schedule are determined.

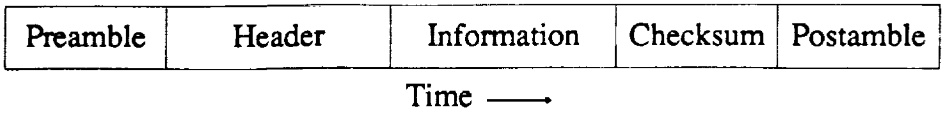


Figure 4. Burst Format

As illustrated in Figure 3, each of the three schedules is structured as a linked list of entries, with the last link indicated by zero. The switch schedule consists of the configuration and timing information necessary to control the crossbar switch operation. Each entry of the switch schedule contains three or five 32-bit words (96 or 160 bits), including 32 bits for configuration information, 16 bits for dwell duration (in words), 16 bits for flags and 32 bits for the link. Entries corresponding to hello bursts include two additional 32-bit timestamp fields as well. The transmit schedule consists of a list of buffer addresses and counts for each scheduled burst originated by the node, while the receive schedule consists of a similar list for each expected burst destined for the node. Each entry of these schedules contains three 32-bit words (96 bits), including 32 bits for buffer address, 16 bits for word count, 16 bits for flags and 32 bits for the link. The switch controller processes these lists in real time and controls the crossbar switch and processor input/output interface as required.

The goal of the scheduling algorithm is to guarantee that collisions, where more than one dwell is assigned at the same time on a particular link, do not occur. This is done using a globally synchronized database which establishes the network topology, link delays and reservation backlog. The resulting schedules insure that dwell times and guard times are respected so that the crossbar switch configuration upon arrival of a burst has completely settled. While avoiding collisions, the algorithm attempts to optimize the use of network resources and respect the delay and throughput constraints requested in the reservation. The manner in which service issues are engineered is for further study.

In order to conveniently implement scheduling operations, the network schedule is organized on the basis of frames up to 65536 symbols (about 2.62 ms with 40-ns symbol times) in length. Frame timing at all nodes is maintained with respect to global time to less than 1 μs, as determined by the synchronization algorithm. Within each frame a node can originate, receive or relay one or more bursts on each link, but bursts may not overlap in time on any link. While originated bursts may not overlap two frames, a burst originated in one frame by a node may appear split between frames at other nodes.

As shown in Figure 4, a burst consists of the preamble, header, information, checksum and postamble fields. The preamble field, which consists of some number of bit-synchronizing symbols followed by a start delimiter, is necessary for reliable burst-clock recovery. The header field includes control information, such as burst type, node addresses, timestamp and message identifier. Since all transmissions are scheduled using a common, distributed algorithm, node addresses and timestamps are in principle required only in hello bursts. Data bursts require only a unique message identifier to serve as a check on the scheduling operation. The information field can be absent or include up to 2048 32-bit words. The timestamp and checksum subfields are computed by the hardware as the burst is transmitted. The postamble field consists of an end delimiter followed by an idle interval or guard time in the order of 1 μs.

In the prototype Highball network the links can operate either in symbol-parallel mode or in symbol-serial mode. In symbol-parallel mode each bit of a symbol is transmitted over a separate wire, while in symbol-serial mode the bits are serialized and transmitted over a single wire or fiber. In the prototype Highball network, the serial encoding operation is provided by TAXI chipsets, which encode 4-bit data symbols in 5-bit channel symbols at 125 Mbps. While a burst can have any length less than 65536 words, the actual length is rounded up to a multiple of 32 bits when transmitted to allow for pipeline flushing. A minimum dwell time in the order of 10 μs is required by the design of the prototype VLSI crossbar switch. A guard time in the order of 1 μs is required to account for synchronization tolerance and skew. Thus, the minimum burst length, including guard and dwell time, is about 275 bits for W = 1 and about 8800 bits for W = 32. For example, an 8192-bit (256 32-bit words with W = 32) HPPI packet burst fits and is transmitted in 11.24 μs for a net data rate of 730 Mbps.

Each node is allocated one hello burst per frame. For end nodes the burst may include a list of reservations requesting a number of 32-bit words and possibly related information such as priority, deadline and so forth. The switch schedule constructed by the scheduling algorithm for the hello burst consists of a broadcast spanning tree, so that every hello burst will be received by every other node in the network. Other information is included in the hello burst for purposes of link, node, network and schedule synchronization.

2.4. Scheduling Operations

In the following discussion the network topology and link delays are assumed known and the switch controllers synchronized in time to well within the guard time. The network has been initialized by some prior procedure that first establishes an ordering of the nodes and then runs the scheduling algorithm to produce the initial switch, transmit and receive schedules for the hello bursts broadcast by each node. The algorithm is initialized by these schedules when it is run at the beginning of each frame to generate the schedule for the following frame. For end systems, sufficient space is allocated in the hello burst for a fixed maximum number of reservations. At the beginning of each frame the schedule computed during the previous frame is executed to control the switch, generate the hello bursts and create the dwells for the data bursts.

The switch schedule consists of separate lists of entries for each bitslice of the crossbar switch, each entry including the configuration vector together with dwell time, transmit bit, receive bit and other control bits as necessary. In order to simplify VLSI switch implementation, dwell time is specified in units of 16-symbol slots. As each entry is processed a configuration vector is loaded into the switch bitslice and the slot counter, which is part of the bitslice, is initialized with the dwell time for the burst. A master oscillator, which is part of the node controller, generates slot pulses which decrement the counters. When a switch bitslice counter reaches zero the next entry in its schedule is processed in turn. When the last entry is processed, indicated by a special bit in the entry, the host processor is interrupted. Note that the host processor has already determined the dwells and buffers for the next frame and inserted them in the schedules by this time.

The switch controller runs continuously as directed by the switch schedule and interrupts the host processor at the end of each frame. Each entry includes the transmit-enable and receive-enable bits as described previously. When a configuration is loaded in the switch bitslice, these bits are inspected to determine whether to enable the local transmit or receive DMA system. The number of transmit and receive bits in any frame must correspond to the number of buffers on the transmit and receive schedules, respectively, for that frame. This mode of operation assumes the input/output architecture of the node controller is capable of scatter-gather operation and that the buffering and interleaving design permits burst-mode memory transfers at rates from 3.125 to 100 megabytes per second (for W from 1 to 32.

The prototype Highball network includes only one data transceiver per node, although four switch bitslices are provided. When the configuration is partitioned into subnets of one or two bitslices each, the intent is that hello bursts be confined to one of the subnets, although the schedule and data bursts generated for each subnet could be different. This provides a great deal of flexibility in the design of reservation and scheduling techniques. For instance, one subnet could be operated in a demand-assigned mode for reliability-sensitive traffic and another in an ALOHA mode for delay-sensitive traffic.

2.5. Node Synchronization

It is vital that the master oscillators at each node be synchronized to each other within a tolerance less than the guard time, currently estimated at 1 μs. This is not believed to be a severe problem using the principles described in the Network Time Protocol Version 3 [MIL90b], which was specially designed for these applications. NTP assumes a local-clock model based on an adaptive-parameter, type-II phase-lock loop. For the purposes required here, the NTP local-clock model can be applied directly, with incremental phase adjustments made by adjusting the master clock frequency. While this will require recalculation of the gain constants, etc., the parameters of the NTP local-clock model should remain valid for this design as well as the conventional Internet design.

In order for the synchronization algorithms to work properly, it is necessary to exchange precision timestamps from time to time. The requirement that all nodes have local databases giving the delay along all links in the network requires some extra overhead not present in ordinary NTP messages. However, since hello and data bursts occur much more frequently than in ordinary NTP operation, the system bandwidth to do this should not be intrusive. It is vital that the timestamps used to synchronize the master clocks be precise to an order less than the slot time, preferably to the order of the symbol time (40 ns). This may be an ambitious goal for a network of national scope and indeed, if possible, this in itself would be a valuable service to synchronize digital telephone networks and local exchanges.

It is necessary to generate timestamps directly at the hardware level as bursts are transmitted and received. The hardware inserts the timestamp directly in the burst upon transmission and leaves a copy in the controlling transmit schedule entry in processor memory. The timestamp is inserted in the burst at the beginning in order that the timing be independent of the burst length; however, the jitter introduced by the encoding hardware on the timestamp itself may degrade the accuracy somewhat. Upon reception the receive timestamp is inserted in the controlling receive schedule entry and the transmit timestamp extracted from the burst and inserted in the same entry where it will be found later by the processor. Note that the transmit/receive data itself may reside in a user buffer in the host processor memory and it is highly desirable to avoid unnecessary per-packet processing, so nothing but user data need or should be in the user buffer.

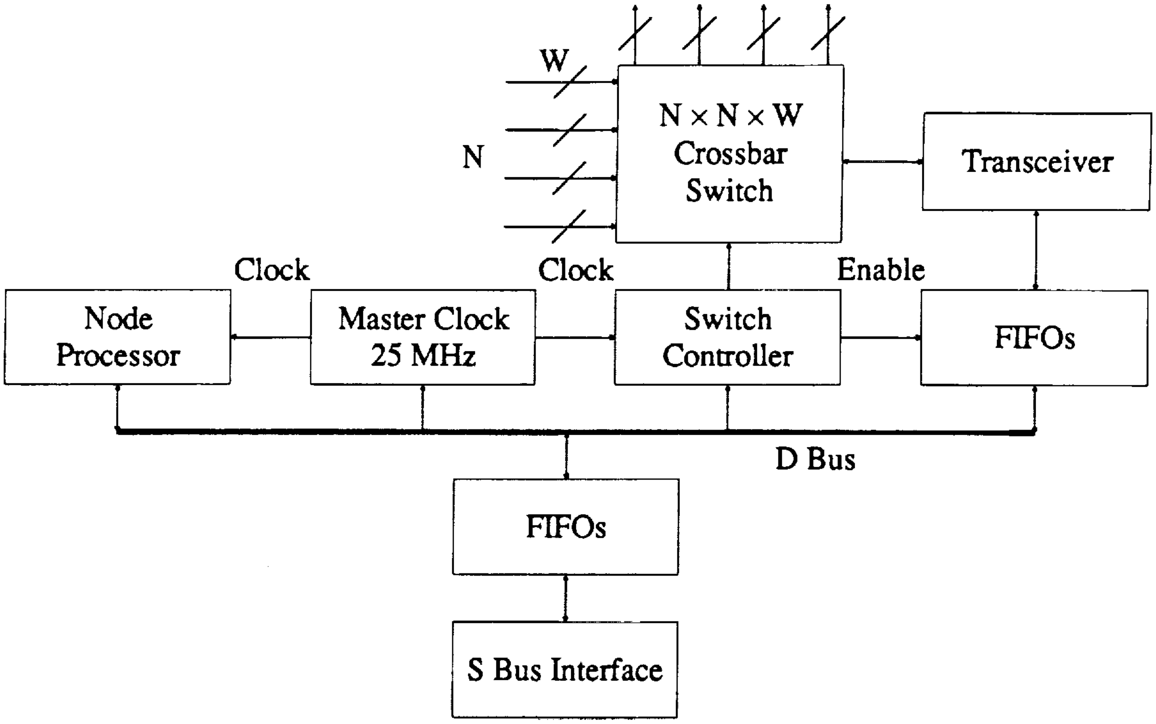


Figure 5. Node Controller

3. Hardware Overview

In order to assess the feasibility of the architecture described in the previous sections, it is useful to work through a strawman hardware design using presently available technology. The hardware design described in following sections is appropriate for the prototype 100-Mbps Highball network and for even higher speed networks with the addition of an external interface to an application processor capable of data rates to and beyond 1 Gbps.

In following sections a candidate hardware design for the node controller is described. This design is preliminary and may change as the detailed design is refined. The overall organization of the node controller is shown in Figure 5. It consists of the VLSI crossbar switch together with link interfaces, switch controller, transceiver and input/output controller, master clock, RISC-based node processor and host-processor input/output bus interface. The particular host-processor bus design has yet to be finalized; however, the S-bus interface as used in some Sun Microsystems products appears to be a good choice.

3.1. Crossbar Switch, Controller and Interfaces

The crossbar switch itself is an 8 x 8 x 4 asynchronous binary design implemented in VLSI. It is controlled by the configuration register and various buffer and shift-register circuity shown in Figure 6. Besides holding configuration data, the configuration vector determines the transmit and receive bits which are included in the switch schedule entry described previously. The design is such that a given configuration can be held during a single dwell while the next configuration vector is shifted into the buffer. The dwell time is controlled by the slot counter, which decrements at the slot rate of 1⁄16 times the symbol frequency or 1.5625 MHz. Upon reaching zero the counter generates a pulse which loads the configuration register and requests the next configuration vector to be loaded into

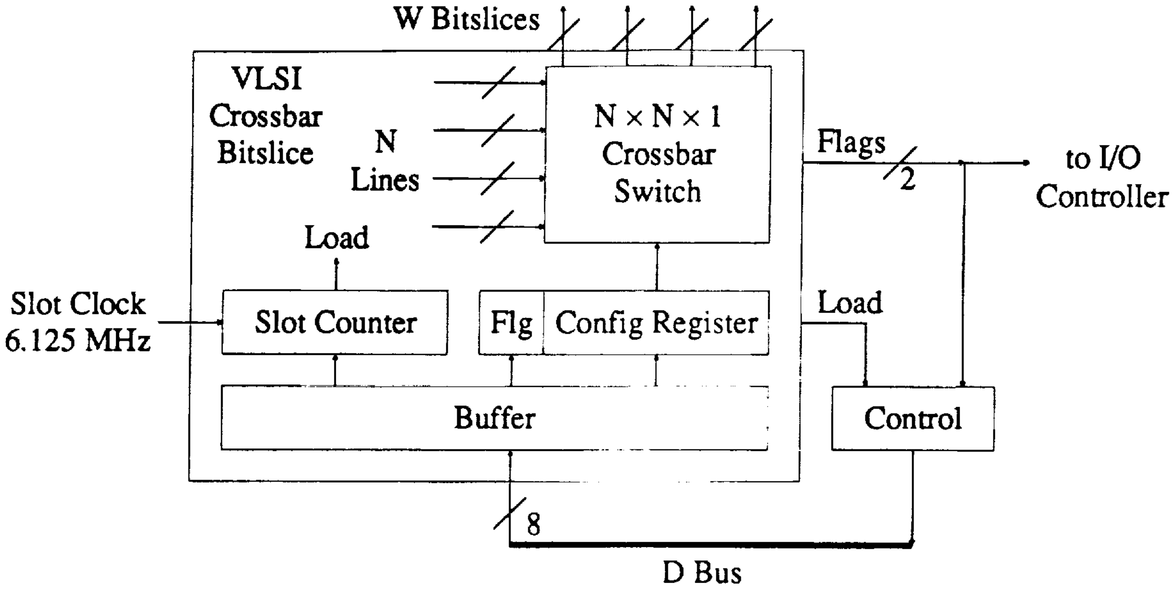


Figure 6. Switch Controller

the buffer. The transmit and receive bits are extracted from the configuration vector and sent to the input/output controller at this time.

The node processor supervises the transfer of data from the switch schedule computed by the scheduling algorithm and placed in host-processor memory. It prefetches each entry of the switch schedule in turn and stores it in an onboard FIFO register, then supplies the data to the switch controller as required. Control bits which are part of each entry determine the transmit, receive and interrupt bits which are interpreted by the switch controller and input/output controller. The interrupt bit causes a processor interrupt, which collects the timestamps left behind and returns the expired schedule resources for later use. Additional bits may be used for frame synchronization to be discussed later.

The operations of the switch controller require a low-latency path to processor memory, which must be prioritized relative to the input/output controller data paths. However, it is likely that the switch controller operations will not significantly interfere with the other paths, since the input/output data transfers occur wholly within the dwells produced by the switch controller. Thus, the switch controller will normally request the next switch schedule entry immediately upon completion of the data transfer and the actual processor memory fetch for the next entry can occur during the guard time.

3.2. Data Transceiver and Controller

In the prototype Highball network a bidirectional link consists of two unidirectional cables of four twisted-pair or coaxial bitslice lines with standard line driver and receiver devices. While it would be possible in principle to multiplex the four bitslices onto a single 100-Mbps line using either copper or fiber media, in fully deployed networks this would require word retiming and elastic buffering at each node, such as provided internally by the AMD FDDI chipset. Whether to adapt the node controller to include this capability is a matter for future study.

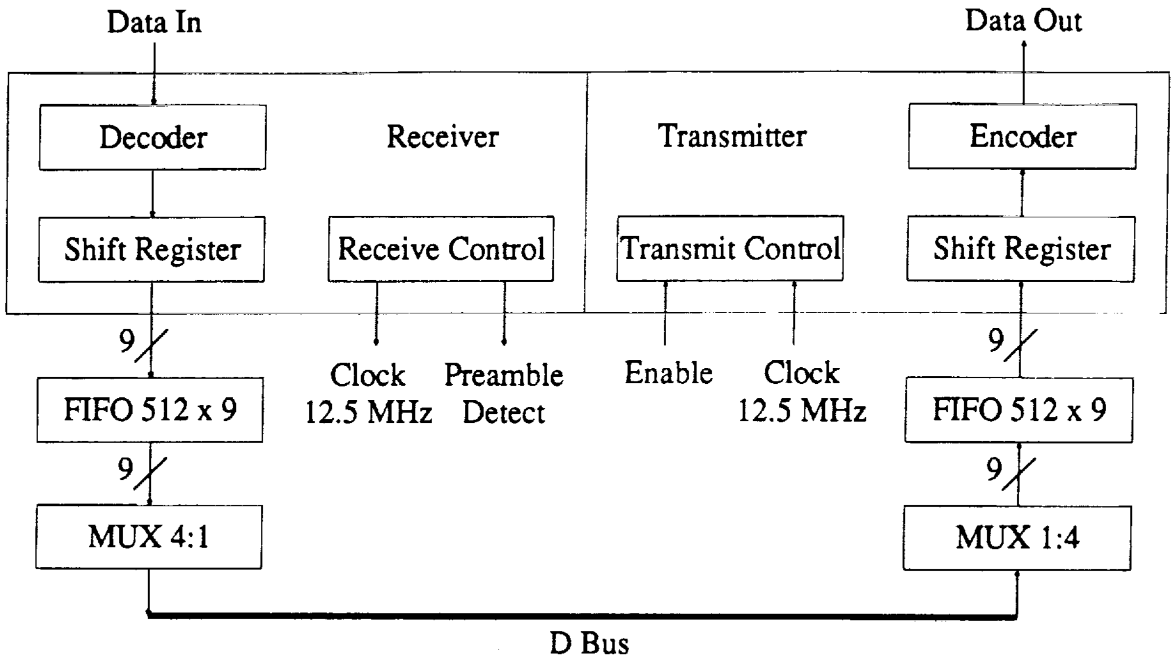


Figure 7. Transceivers and Input/Output Controller

The data transceiver, shown in Figure 7, includes the encoding, decoding and clock-recovery functions common to other link-level systems such as FDDI. In this figure it is assumed that the links are implemented using 100-Mbps fiber links, rather than twisted-pair cable. It is anticipated that off-shelf silicon components such as the AMD 7968/7969 TAXI chipsets can be adapted for use in this application. The transmit and receive signals provided by the switch controller are connected to the transmit-enable and receive-enable inputs of the transceiver. The clock and control signals are connected to the input/output controller to control the movement of data along the data paths and input/output bus.

The burst format shown in Figure 3 includes the preamble, header, information, checksum and postamble fields. The transceiver is responsible only for byte clock recovery and burst segmentation, which includes detecting the start delimiter and end delimiter on receive, as well as providing byte clock and other control signals to other subsystems. The coding is such that the beginning and end of the burst can be determined from the raw data and without interpreting count fields or calculating the checksum. The 4/5 channel encoding used by the TAXI chipsets provides an integrated serial/deserial capability, as well as convenient burst delimiters and other control codes which cannot be mimicked by data patterns. With a 25-Mbps channel rate, 4/5 encoding and 8-bit data byte, for example, the transceiver data rate is 2.5 Mbytes/s, well within the capabilities of conventional microcontrollers and host processor interfaces such as SCSI.

The input/output controller, also shown in Figure 7, supervises the data paths between the transceiver and node-processor data bus. It includes FIFOs to buffer data transferred between the links and registers of the node processor and bridge the latency of host-processor memory transfers. The system is capable of simultaneous input and output transfers driven by the input and output schedules computed by the scheduling algorithm. The input and output operations operate independently of each other, since the transmitter can be originating a burst on an output line while the receiver is

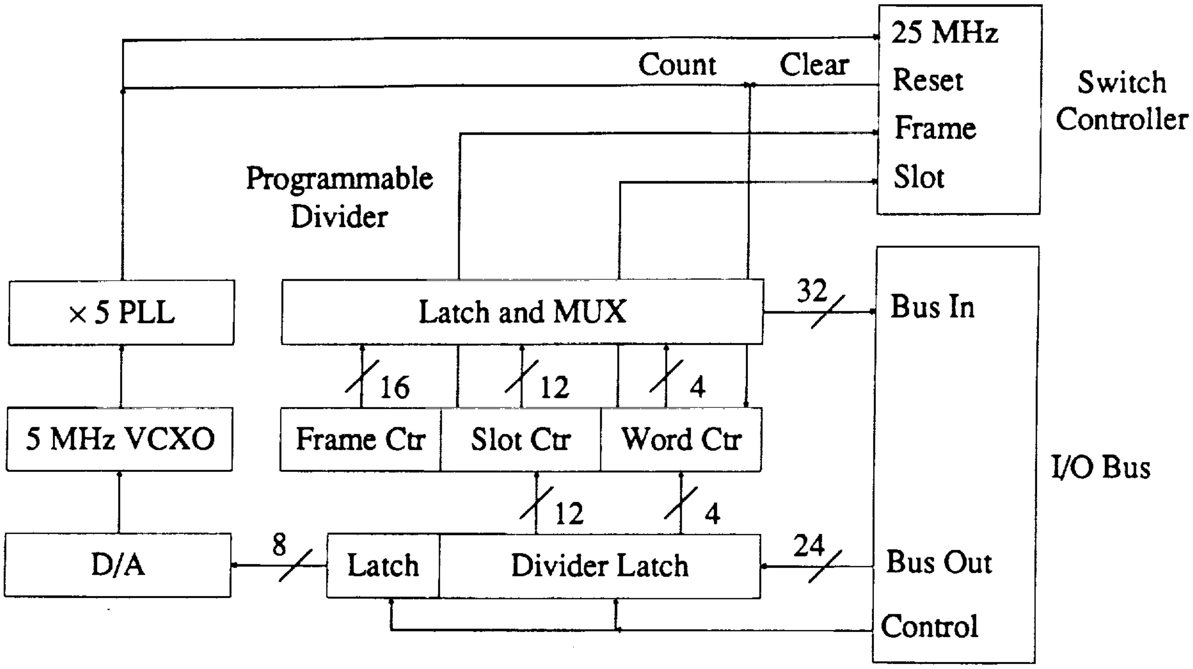


Figure 8. Master Clock

receiving (and simultaneously relaying) a burst on an input line. The principle components include shift registers in order to assemble and disassemble 32-bit words for transfer to and from processor memory, together with various buffers and control circuitry.

The scatter-gather function is controlled by the entries on the input and output schedules. For receive, The controller prefetches the next entry on the input schedule, which designates the address and word count for the operation, along with various control bits including an interrupt bit. When the first byte arrives from the transceiver a control function indicates whether the burst is of hello or data type. If data, the bytes are placed in processor memory, the registers updated and checksum updated, as metered by the receive clock. If hello, the transmit timestamp is compiled from the first 32-bit word and the receive timestamp determined from the local clock. Both quantities are placed in the receive schedule entry for later processing. Bytes following the transmit timestamp in the hello burst are processed as a data burst. Finally, various status bits are determined and inserted in the schedule entry and, if the interrupt bit is set, the host processor is interrupted to process the data and timestamp information and return the schedule resources for later use.

For output operations the procedure is reversed. In general, there is always an output operation in progress, since schedules are determined one frame in advance of transmission. The controller prefetches the next entry on the schedule and the first word of the data, then waits for the transmit enable. When enabled, the controller shifts bytes into the transceiver, as metered by the transmit clock. If a hello burst, the transmit timestamp is determined from the local clock and inserted with the appropriate control function ahead of the data, as well as inserted in the schedule entry. Finally, various status bits are determined and inserted in the schedule entry and, if the interrupt bit is set, the host processor is interrupted to process the data and timestamp information and return the schedule resources for later use.

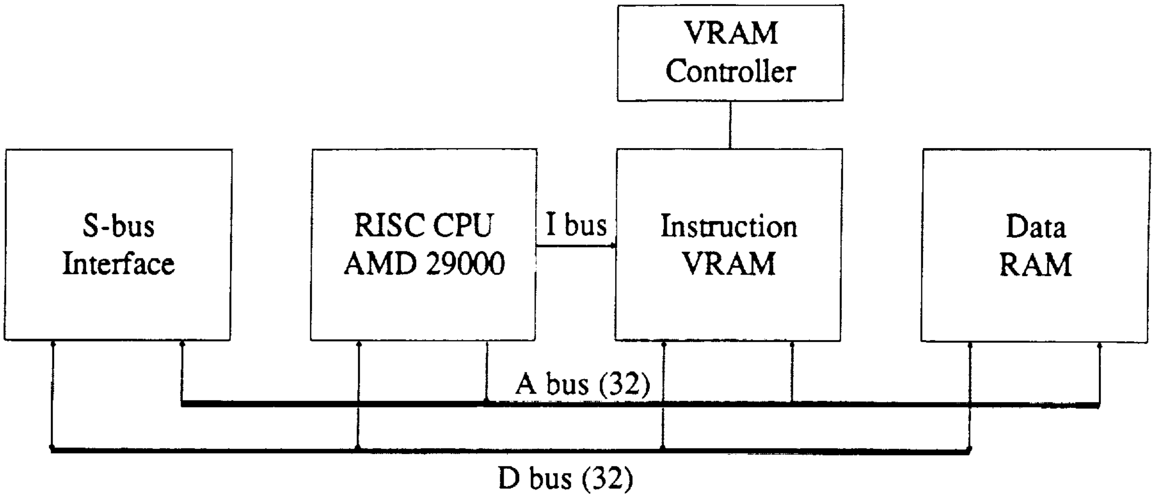


Figure 9. Node Processor

3.3. Master Clock

The master clock, shown in Figure 8, controls all node controller functions except the transfer of data within a burst. It consists of a 5-MHz temperature-compensated quartz oscillator and various multiplier and divider components. In order to maintain overall network synchronization, it is necessary that the master clock frequency be controlled over a small range by either the node processor or host processor. The most convenient way to do this is using a voltage-controlled oscillator and D/A converter. Devices appropriate for this application are readily available.

The heart of the master clock is a programmable System Timing Controller (STC) based on the AMD 9513A chip. This chip contains five 16-bit counters, together with various compare, buffer and auxiliary counter functions. These are connected externally and programmed internally as a 4-bit word counter, 12-bit slot counter and 16-bit frame counter. The word counter can be programmed to shift the phase of the slot and frame counters over the entire frame interval for initial synchronization. The slot counter modulus and D/A voltage can be latched from the host processor. The frame counter is incremented on overflow of the slot counter. The values of all three counters can be read by the node or host processors. The word and slot counters can also be cleared by the node processor.

The clock circuitry also provides precision timestamps used by the synchronization algorithm and input/output controller. The timestamp is in the form of a 32-bit number, with the low-order 4 bits taken from the word counter, the next 12 bits from the slot counter and the high-order 16 bits from the frame counter. This provides unique timestamps for all network operations with maximum lifetimes to about 344 seconds.

3.4. Node Processor

The programmed node processor shown in Figure 9 coordinates all operations of the node controller. Its principal components are an AMD 29000 RISC processor, separate instruction and data RAM and S-bus interface. The 29000 operates at a clock speed of 40 ns, which matches the rates used elsewhere in the system. It has one 32-bit address bus and two data buses, one for instruction fetch

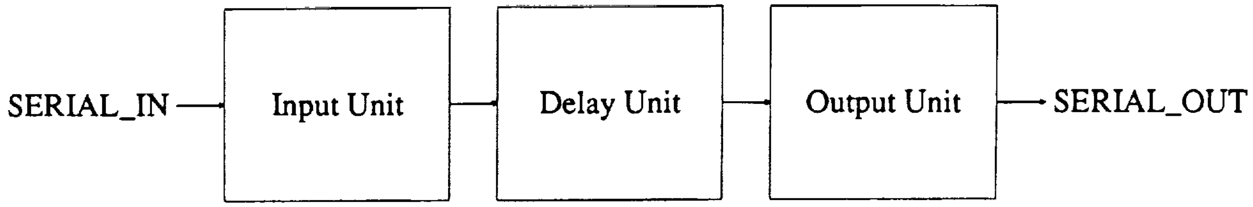


Figure 10. Link Simulator

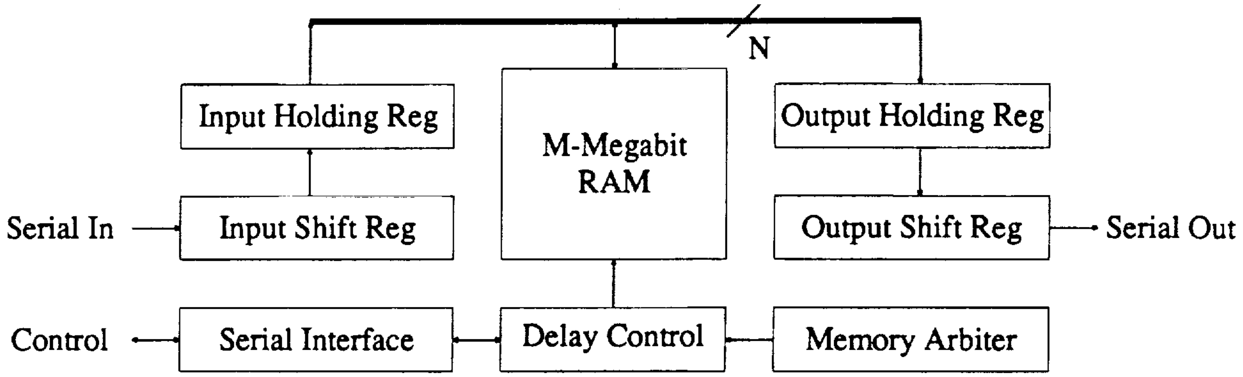


Figure 11. Link Simulator Detail

and the other for data read/write. The program code and initialized data are downloaded from the host processor.

3.5. Link Simulator

A thorough evaluation of the design and technology discussed in this report requires extensive simulation and testing of the scheduling algorithm and hardware design. To facilitate this, a test system must incorporate realistic link delays without having to actually install long high speed lines at great cost. A subsystem, the link simulator, has been designed to simulate the various link delays found in wide-area networks. The design is capable of simulating a coast-to-coast link operating at 125 Mbps and is shown in Figure 10. It consists of an input unit to condition the high speed source data stream, a delay unit which operates under computer control and simulates link delays of a few microseconds to hundreds of milliseconds, and an output unit to transmit the delayed data stream to its destination.

The operation of the link simulator, illustrated in Figure 11, is described follows. Serial data arriving from a source node is clocked into the serial_in port of a high speed shift register. This input shift register is N bits long and has an output pin for each register. This provides parallel access to the last N bits in the incoming data stream. The purpose of an N-bit wide data path is to reduce the bandwidth requirements for the delay unit, which decreases cost and increases available link time delay.

The parallel outputs of the input shift register are clocked into the input holding register on the rising edge of the clock following the Nth datum bit entering the input shift register. These input holding register data remain constant for the next N rising clock edges. During this period the data are written in parallel into the next delay slot of the delay unit.

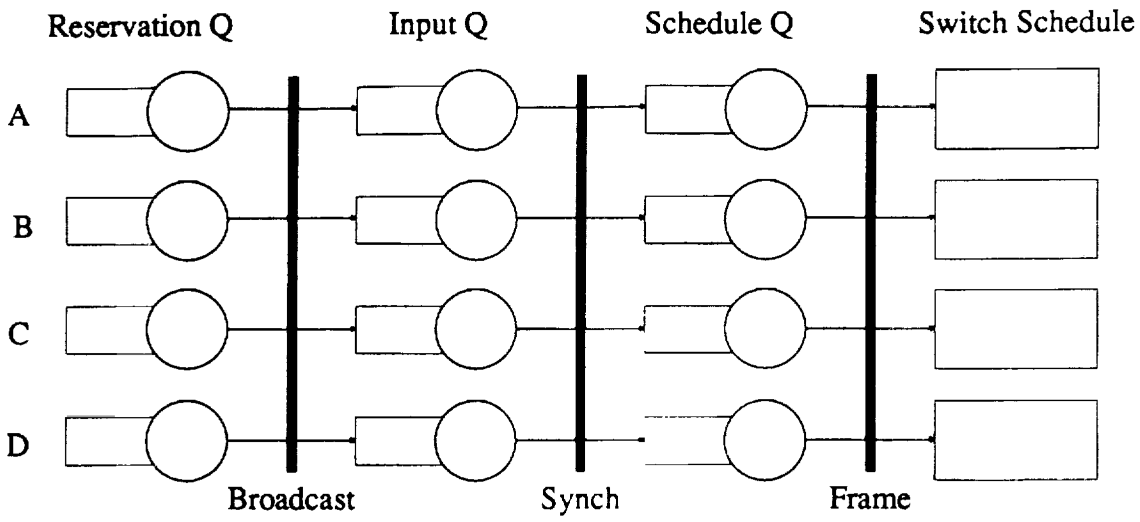


Figure 12. Software Queues

The delay unit comprises a delay controller, a memory arbiter, a serial communications port for link delay programming, and N*M megabits of dynamic RAM (DRAM) organized as an array N bits wide and M megabits deep. The minimum value of N is determined by the serial data stream rate and the memory cycle time. During each N-clock period incoming data are written to DRAM and outbound data are read from DRAM. At irregular intervals during an N-clock period a third operation, memory refresh, may also occur. The occurrence of these three operations during any given N-clock period places limits on the minimum value of N. In a future design we will investigate the possibility of hiding refresh cycles inside read and write operations to lower the minimum value for N.

The delay controller and memory arbiter will be implemented in state machines. The controller's main function is to produce the DRAM addresses for both the incoming and outbound data streams. The arbiter's role is to ensure proper sequencing of memory cycle requests. Outbound data are accessed from the delay unit as an N-bit wide word and stored in the output holding register for a maximum of N clock periods. The output shift register is loaded with these data at the appropriate time and outputs each bit on its serial_out port at the rising edge of clock. The loading of the output shift register with data from the output holding register occurs on the $N+1$th clock edge following the previous load.

The entire link simulator has been modeled, simulated, and verified at the behavior level using the Verilog hardware description language.

4. Software Overview

The above discussion assumes the network topology and link delays are known and that the network is synchronized and the node databases consistent. It is of some interest to explore how the network gets started and how it adapts when nodes and links fail and are restored. The basic mechanism of discovery is the hello burst. By design, a hello burst from every node is heard once per frame by all nodes in the network. In general, all such bursts must be heard reliably by all nodes; otherwise, not all nodes may compute the same schedule and synchronization will be lost.

4.1. Normal Operation

The overall organization of a strawman software implementation is shown in Figure 12. Each node includes a reservation queue, input queue, switch queue and the switch, transmit and receive schedules. Reservations are generated by a node in response to an onboard application program or externally, such as a received Ethernet packet, for example. Upon arrival, the burst size, priority and deadline are determined along with an identifier later used to determine which buffer is to be transmitted (or which external interface is to be enabled). The composite reservation is inserted in the reservation queue for later transmission and also in the input queue for later processing by the scheduling algorithm. The reservation queue is processed when the hello burst schedule is constructed for transmission in the next following frame, (called Broadcast in Figure 11).

Reservations received in hello bursts broadcast from other nodes are placed in the input queue pending processing by the scheduling algorithm. The scheduling algorithm runs at the beginning of each frame, triggered by a real-time interrupt from the switch controller (called Synch in Figure 11). The algorithm inspects the reservations on the input queue to determine which are eligible for processing. To be eligible for processing, the reservation must have arrived at all other nodes in the network. This is determined by calculating the overall flight time from the originator of the reservation to all other nodes using the known topology and computed delays provided by the synchronization algorithm. Included in the schedules are the hello bursts themselves, which are always processed before any others.

The scheduling algorithm produces the burst schedules which eventually will drive the switch and input/output controllers. In principle, each node constructs the same burst schedule for all nodes, since each has the same database and processes the same reservations on the input queue at the same global time. However, the burst schedule produced by a single reservation may involve switch and input/output schedule entries many frames in the future, while the later schedules are computed only one frame in advance to allow for possible preemption.

At the beginning of each frame (called Frame in Figure 11) the burst schedule at each node is processed only for the entries that will be executed in the next following frame, but only those entries that involve that node will result in switch and input/output entries for that node. For end nodes the message identifier assigned when the reservation was transmitted is used to identify which buffer to use and the receive and transmit schedule constructed at that time.

4.2. Link and Node Synchronization

In order to compute a consistent schedule, it is necessary first to know the network topology and the delays for all links. This can be determined using the hello bursts, which are exchanged over each link consisting of an outgoing line and an associated incoming line. Since every node broadcasts a hello burst at least once during each frame and the broadcast spanning tree must touch every neighbor node of the broadcaster, at least for real delays and real networks, timestamps are exchanged between a node and each of its neighbors once in every frame. However, there may be a delay of many frames between the transmission of a timestamp on a line and its reception at the other end of the line. The problem is to calculate the precise delay on the line and synchronize the local clocks in each node to established global time.

If a link is synchronized and the neighbor is participating in the scheduling algorithm, the node has already enabled its receiver at the proper time for each hello burst expected on the link, as determined by the scheduling algorithm. If not, and the node senses by other means (such as configuration information) that another node is potentially reachable via the link, it transmits its hello burst to that node as usual, but privately schedules its receiver to listen at times when no received bursts are expected on the synchronized links, while equalizing the listening time among the unsynchronized ones. While other nodes need not know about the private schedule, the distributed scheduling algorithm must insure that every node has a modest amount of cracks in the schedule, so that some receiver free time is available for this purpose at every node.

A node coming up has no idea of global time, network topology, link delays or reservation backlog. It first listens for a few frames on each link in order to receive the hello bursts transmitted by its neighbors, who at this point are unaware of its activity. These received bursts allow the node to calculate roughly when to transmit its own bursts in order to announce its presence, determine the link delays to its neighbors and synchronize its local clock. This may require some amount of hunting, since the neighbor node may listen on the receive line only a short time during a frame and the link delays can vary up to 30 ms. In addition, many frames can be in propagation on the link, so it is necessary to disambiguate which one is to be used for the delay calculation.

An appropriate link-synchronization algorithm for a node coming up is first to assume the link delay is zero, then calculate the apparent global time and transmit burst offset within the frame from the received burst timestamps. Starting from a random offset relative to the apparent frame, it transmits hello bursts, then listens for confirmation that these bursts have been heard by the neighbor, while slewing the offset in increments of approximately the guard time for each succeeding frame. Once confirmed, the node and its neighbor exchange timestamps and determine the correct global time, local-clock offset and link delay as described in the NTP specification. In the worst case, this process can take up to about 2,620 frames or 6.9 s; however, each link is independent and can be processed in parallel.

4.3. Network Synchronization

Once link and node synchronization have been achieved, the network topology can be determined in ways familiar in packet-network engineering. Since the scheduling algorithm needs to know the complete link-delay matrix, a link-state spanning tree algorithm is used, which requires each link to broadcast delays for all adjacent links to all other nodes. This is conveniently done by using the hello burst; however, in order to minimize network overheads, the burst format is structured so that the information for any particular link is broadcast on a round-robin basis and then infrequently, like once per second. In this way, it should take no more than a few seconds for knowledge of a new node or link to percolate to all nodes.

The link delay measurement process operates simultaneously with the link discovery process. This allows each node to learn about all other nodes and the link delays between them within a few seconds. By virtue of the spanning tree nature of the hello broadcasts, all nodes will assume the same topology and delays; however, they will not all learn this at the same time or even in the same frame. Therefore, it is necessary to wait a predetermined hold-down interval after reception of a hello burst and before executing a topology change in the database until all nodes have received the burst and all agree on the precise time of execution. Since nodes and links are not expected to come and go frequently, the simplest approach is to add a fixed interval, like 30 ms, to the global time a hello burst was sent (indicated by a field in the burst) and then execute the change.

Each time a topological change is made, it is necessary to recalculate the delay matrix and broadcast spanning trees and, should this be necessary for resource limitations, recompute the initial switch, transmit and receive schedules. Since it may take a number of frames to do this, a second hold-down interval may be necessary to allow for the computations to complete. The need for and nature of these hold-down provisions remains to be studied.

4.4. Schedule Synchronization

It is important that the scheduling algorithm have a bounded convergence time; that is, no entry on a scheduling queue can depend on data with an age, according to current global time, older than a bounded number of frames. Thus, a new node joining the network need only receive that number of frames in order to be assured that its local state variables are consistent with the remaining nodes in the network. It does this by processing reservations as received, but suppressing the production of schedule entries other than those necessary to sustain the hello burst. However, it is necessary that all nodes agree when a new node has achieved schedule synchronization, since otherwise they might attempt to reserve a data burst to it. This can be avoided using a hold-down in the same fashion as for network synchronization.

Having once achieved link, node, network and schedule synchronization, the node can broadcast reservations and transmit traffic as required. The nodes continue to monitor link delays and send hello bursts. If a node determines that an adjacent link goes down, it simply stops broadcasting that. If a node loses synchronization for some reason, it immediately reverts to the link-synchronization state described above. If a node goes down, its neighbors will notice the adjacent links going down and, when all links are down, will declare the node itself down. As before, a synchronized hold-down delay is necessary in order that all nodes retain a consistent data base.

Without external notification, it is not possible for a node to discover that scheduled outband transmissions, such as from an external HPPI interface, have collided due to inconsistent databases. In principle, it is possible to, for instance, include a database checksum in the hello bursts, but this might not be necessary on the assumption that inconsistencies in the database should quickly become apparent in the timing of the hello bursts themselves, the expected result being that a hello burst from a particular node arrives at a time other than predicted or during a time a link receiver is not enabled and therefore not receiving at all.

An inherent problem with this design is that schedule synchronization is vulnerable to transmission errors. A single garbled hello burst can in principle bring down the entire network, although this would result only in situations unlikely to occur in practice. Nevertheless, a very important part of the detailed design must be to minimize the incidence and extent of instability propagation due to transmission errors. For instance, if an error occurs on a hello burst from a particular node (checksum failure, DMA word-count mismatch, etc.), it is not likely that the network topology, link delays or synchronization information has changed in significant ways since the last hello burst and in any case can be verified in subsequent bursts before database changes must be executed; however, it is of course not known what reservations may have been included in the burst.

An obvious way to deal with the problem of lost reservations is to transmit them (and related timestamps) two or more times. Duplicates can be suppressed upon entry to the input queue by inspection of the timestamps. However, this technique requires an additional reservation delay of at least one frame, as well as an increased hello burst length. Another way to deal with this problem is to require that schedules be non-preemptive; that is, new reservations will not dislodge scheduling entries computed from prior reservations. These are issues for further study.

5. Scheduling Algorithms

A number of scheduling algorithms have been proposed to meet the requirements of the Highball architecture. The primary objective of a scheduling algorithm is to provide collision-free schedules in response to user requests. Secondary objectives, such as minimizing waiting times, minimizing average communication delays, maximizing link utilization, or maximizing network throughput, should be evaluated with respect to realistic constraints of available resources. In addition to generating collision-free schedules, the algorithm must complete processing of all requests received in a frame within a small number, preferably one, of frames.

The *Trailblazer* algorithm [SCH91] was developed to explore the possibility of calculating minimum delay schedules via enumeration of the non-intersecting paths or trails generated for a slotted network. Algorithms that generate such schedules require exhaustive enumeration of all possible solution sets, which in general is NP-hard. Therefore it is not surprising to find that the complexity of the Trailblazer algorithm grows exponentially with the number of requests to be processed and the size of the network.

Heuristic algorithm approaches, although usually suboptimal, were then explored to generate schedules where the maximum computation time must be bounded. The *Pathfinder* heuristic algorithm [SCH91] creates a collision-free schedule by processing one reservation request at a time without preempting the current schedule and applying a best fit, greedy assignment of resources. The subsequent schedule may not provide the minimum delay schedule over all possible schedules, but it does provide the minimum delay schedule under the constraint that the existing schedule cannot be changed.

The Pathfinder algorithm assumes that the network topology and link delays are known. From the network graph all possible non-looping paths are generated. For each distinct source and destination pair a list of paths is constructed and ordered by the overall delay of the path. Pointers from the links to the paths, and back pointers from the paths to the links are created. For each link and switch an empty schedule of time blocks is initialized. A time block has a beginning time and an end time signifying either utilization or availability depending on the type of schedule.

When a reservation request becomes valid for processing, the list of paths for the required source-destination pair is found. The minimum delay path is constructed using the following algorithm:

1. Construct the path list for the reservation to be processed, as determined by the source and destination nodes.
2. Set a variable DELAY to infinity.
3. Set the current path to the first path on the path list.
4. Calculate the schedule of available blocks of time for the current path.
5. Find the first available time block that has a duration greater than or equal to the requested duration.
6. Calculate the total delay of the data burst for the first fit time block.

	Host Processor		
Compiler	IBM RS6000	Sun SPARC-1	Sun SPARC-1+
Native C	6.86	12.17	9.12
Optimized C	4.47	7.69	5.94
gcc		8.19	

Table 1. Scheduling Times

7. If the total delay is less than DELAY, then the present path becomes the selected path for the reservation and set the variable DELAY to the present path's total delay.
8. If DELAY is greater than the next path's delay then evaluate the next path as the current path at 4.
9. Update all links and switch schedules affected by the selected path.

After a path is selected for the request, the link schedules are updated in the order encountered along the path. When a link schedule is updated the associated switch schedule is updated. The link and switch schedules show the use of the link by data burst. Each time block on a schedule is associated with a particular data burst. The burst's reserved time block is inserted into the link schedule according to its arrival time at the link. The arrival time is calculated using the start time of transmission on the first link and the sum of all the link delays on the path up to the link being scheduled.

A number of functions are required to support the Pathfinder algorithm. The available time schedule of a path is calculated by a function that traverses the path from link to link and logically AND's the time blocks of each link schedule. A function on the list of path lists is provided that returns a pointer to the first path of the linked list of paths associated with a specified source and destination node. To find the first available time block on a path schedule, another function traverses the link list of available time blocks in the path schedule, compares the duration of the request to the available blocks, and returns the time of the first block with a duration greater than or equal to the request's duration.

The Pathfinder algorithm was coded in C and tested using simulated traffic sources on the NSFNET Backbone Network shown in Figure 1. The simulations assume Poisson arrivals, exponential message lengths and bidirectional links operating at 1 Gbps. The execution time required to find a single path on the network depends on the host processor and compiler. Table 1 profiles the execution times in milliseconds for various processors and compilers.

6. Simulation

A key element in the development strategy leading to a successful demonstration project is the capability to simulate the network operations with a high degree of precision and fidelity. There are three areas in which simulation is important: algorithm development and performance verification, network and node synchronization parameters and system guard times and stability analysis.

A special-purpose algorithm simulator is necessary in order to verify the design concepts and forecast performance in various implementation strategies. A suitable simulator has been built which models the performance of the network as a whole driven by source-destination node and interarrival distributions drawn from a various model processes. The simulator would incorporate the various queues shown in Figure 12 and build the scheduling tables as indicated, then model the actual switch configurations and search for conflicts and various errors. In this simulation the network configuration and link delays are assumed known and the network correctly synchronized.

For the most precise and reliable network synchronization, the system must be modelled as a system of mutually coupled control-feedback loops and the stability assessed both analytically and by simulation. Analytic models are readily available drawn from the technology of synchronizing digital telephone systems; however, the accuracy required in a fully deployed R-TDMA network is at least an order beyond that required in a digital telephone network (about 18 μs). In addition, the fact that synchronization must be achieved over switched links, instead of over DS1 or DS3 framing, requires that the dynamics of the system must be simulated in order to verify that unusual configurations, such as failures of links and nodes, do not compromise the synchronization accuracy.

A synchronization simulator can be adapted from one built to explore the operation of NTP with various control-feedback loop parameters. Basically, this amounts to replicating the node-specific algorithms designed for that simulator once for each new node and modelling the link delays on the basis of previously measured link delay variations. Construction of such a simulator is for further work.

Fine-tuning the prototype Highball network for the minimum guard times and other parameter selections consistent with high system reliability and low error rates requires careful analysis and probably detailed simulation of the link jitter, switching delays memory-access latencies. Link jitter is usually dominated by the encoding/decoding and clock jitter, while switching delays are due to the particular VLSI conductor geometry and path through the switch. Both of these can be predicted from theory and simulated using existing tools, such as MAGIC and SPICE.

Delays due to memory-access latencies are much harder to predict, since they depend not only on the fact memory in both the node processor and host processor is shared for a number of functions, but the movement paths over which the data are transferred from host processor memory via various intermediate staging registers and internal bus paths to the links involves specific sequences of machine code produced by an optimizing compiler. Simulation can be used as a tool to explore various implementation strategies, such as the depth of a FIFO and length of a block move.

6.1. A Special-Purpose Simulator

A special-purpose simulator has been constructed to verify the design concept and aid in predicting the performance of various implementation strategies. Examining the network at a macro level, we see a network topology and interarrival distributions at the source nodes that drive the simulation. Upon closer examination however, we recognize that there is a complex interaction of states within the network. For example, a burst is ready to be scheduled only after it has risen to the top of the scheduling queue. The scheduler can only run at certain intervals and takes a finite amount of time to run. To get into the scheduling queue, the request must have been valid, i.e., received by all the nodes in the network. Thus, the local node must queue any requests until valid. Since the hello burst only occurs at well-defined intervals, the source node must queue any outstanding requests for future hello bursts. All of this assumes that the corresponding crossbar switches at each node are properly set and no data errors occur in the hello bursts.

Since the system we are trying to model has complex and nested interaction modes, we decided to develop the simulator in the object-oriented language C++. Using C++ allows us to build an

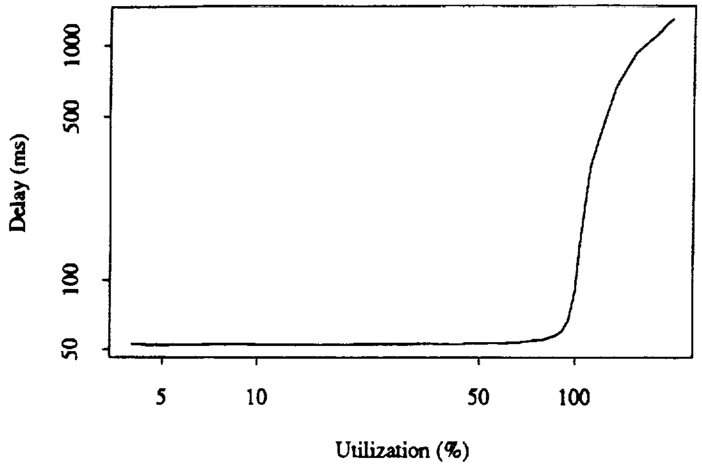


Figure 13. Utilization-Delay

event-driven simulator with each *event* being a type of interaction. While this can be done in ordinary C, the advantage of C++ is that we can make the events have properties which are inherited from other events. Inheritance is just one of the properties of C++ that allows us to create modular models of the interaction processes. These properties keep the simulation code simple and easy to modify. For example, since the simulator is event-driven, all the events are timestamped. The interaction events are created as children of a *time event*. Thus, the basic simulator engine only manipulates a queue of time events, eliminating the need to differentiate between different types of events in the event queue.

Since the simulator models the system from the source-destination level down to the configuration of a particular crossbar switch, the types of analysis possible is varied. We can examine the delay and utilization characteristics of this technology on various topologies. We can also examine the anticipated performance of different implementations of scheduling algorithms and control channel techniques. In the future, we wish to use the simulator to evaluate timing, synchronization and error-recovery algorithms.

6.2. Preliminary Simulation Results

We have some very preliminary results using the simulator and designed to verify the simulator operates as intended. Figure 13 displays the delay with respect to the interarrival time. The parameters used in the generation of this curve is:

1. Time for the scheduling algorithm to run = 2.5 ms.

2. Average size of burst (exponentially distributed) = 1.6 ms. At 1 Gbps, this gives an average burst size of 1.6 Mb or 0.2 MB.

3. Frame length = 2 ms.

4. The reservation channel is out-of-band.

5. Current NFSNET topology with links operating at 1 Gbps. Destinations are selected uniformly over all other nodes.

The interarrival process for the bursts is exponentially distributed. Recall that a network interarrival time of less than 2 ms is greater than one arrival per frame. The delay curve shown in Figure 13 exhibits a flat delay of approximately 50 ms, which is characteristic of the reservation method used to schedule the bursts, but does not necessarily reflect the final method used. The delay remains flat until the mean interarrival time λ approaches the frame size, i.e., when λ approaches the service time μ.

The link utilization determined by the simulator given the present reservation method is almost flat and in the range 90-95% when the mean interarrival time is less than the mean service time. This indicates the particular reservation and scheduling algorithm, in this case based on the Pathfinder algorithm described previously, are very effective in finding useful paths and, in fact, suggests that innovative heuristics may produce simpler, faster algorithms with acceptable compromise in additional delay.

7. Development Beyond the Initial Prototype Network

The prototype Highball network is not intended as a fixed target, but rather as a development platform supporting a gradual evolution from the capabilities represented in the initial design proposed in this document to much greater capabilities anticipated in future enhancements. These fall into two principal areas: increasing the speed of the network and enhancing the capabilities of the controlling algorithms.

7.1. Getting Fast

The most intractable obstacle to higher transmission rates in the present design is the VLSI crossbar switch, which with current technology is limited to about 25 Mbps. There appears to be no insurmountable technical obstacles to increasing the rates to 100 Mbps through use of alternative proven VLSI technology, such as reworking the design for 1.1-micron technology, other than cost. However, at the present stage of development it is believed prudent to continue work toward demonstration of a 25-Mbps network before investing in a faster switch.

The use of an all-photonic switching fabric remains the ultimate goal of many projects, including this one. While there are devices which provide electronically controlled optical switching, these devices are either physically unwieldy and expensive or require very long times (milliseconds) to reconfigure. Nevertheless, it is the intent in the design of the prototype Highball network that photonic switches can easily replace the electronic ones should they become available.

The most inviting way of getting fast, given presently available technology, is to increase the number of bitslices in the network. Based on the technology developed for HPPI, expanding the number of bitslices to match the number of bits in a host-processor word, 32 or 64 bits, would increase the transmission capability of the network to 0.8 or 1.6 Mbps without significantly altering the design of the node controller or scheduling algorithms. However, it is not feasible, either technically or financially, to procure data transport service in 32-bit bundles of low-skew long-haul transmission facilities. Therefore, it would be necessary to provide high speed serial/deserial functions in the node controller for each input/output port.

While provisioning of high speed serial/deserial functions is not viewed as technically difficult, and in fact the necessary logic components are now available, incorporating these functions in the design of the prototype Highball network is not a valued goal in the development program. The main goal of the program is to establish technology and demonstrate a prototype of an architecture and design for a network capable of speeds well in excess of 1 Gbps, but not necessarily building and deploying a network of nationwide span, which is certainly a most expensive undertaking.

7.2. Reducing Processing Requirements

Preliminary analysis and experience with the types of scheduling algorithms likely to be useful in the prototype Highball network raise some concern that processing times are likely to be a limiting factor in switching reconfiguration speeds. In order to sustain a peak reconfiguration rate of one new reservation for every frame from each of the 13 NSFNET Backbone network nodes, which is unlikely in practice, present candidate algorithms require at least a frame time (about 2.62 ms) on a SPARCstation 1 to process. Obviously, there is a concern for scalability here.

There are several approaches to mitigating the scalability problem. The most obvious is to reduce the rate of new reservations, perhaps by creating streams as in SATNET and WIDEBAND, in which the switch configuration remains the same for successive frames in particular slots. This strategy relies on the fact that it takes far longer to compute the schedule than it does to execute it and actually control the switch. A stream could be created as the result of explicit client request, which would specify the recurrence interval and allowable jitter, for instance, or implicitly by the scheduling algorithm in the fashion of the 'stutter stream' capability of the late-SATNET era.

Another feature that might be exploited is a technique similar to that used for the NASA ACTS system, in which slots in a recurring configuration are 'borrowed' by incrementally adjusting the dwells assigned to various sources as the result of explicit or implicit traffic flows. One way this could be done is to presume that a reservation, once granted in one frame, will recur in the next frame, but with a slightly less duration. In effect, this creates a stutter stream with continually decreasing duration. If a new reservation is received in the following frame, it causes schedule recomputation only if it is for a duration longer than that already allocated. Even if an increased duration is required, a wholesale recomputation might be avoided by inspecting those dwells adjacent to the reserved one which have been allocated, but no reservation has been received and reallocating the burst to the requested one. There appears to be numerous opportunities for clever ideas in this area.

While detailed discussion and evaluation of the candidate scheduling algorithms developed for this project are given elsewhere, it is safe to say that their performance in reservations-per-second can probably be improved to a considerable extent, their basically enumerative nature will eventually limit an indefinite expansion in network size. The present algorithms provide very efficient schedules using exhaustive or very near exhaustive searches of the scheduling space with only sparse use of heuristics. There appears little doubt that the running times can be much reduced through use of more heuristics at only minor decrease in efficiency. In addition, while some of the present algorithms employ an incremental approach which avoids the necessity of recomputing all routes when a new reservation arrives, there may be merit in exploring ways in which the database and incremental update process can be made more efficient. The effort in accomplishing this goal may profit from the lessons learned in the evolution of the ARPANET SPF algorithms, in which the Dijkstra algorithm was revamped for efficient incremental update.

7.3. Reducing Reservation Delays

The requirement that all nodes receive a reservation before acting on it places a lower bound on the overall transmission delay, which includes components due to the frame time, reservation propagation time, schedule computation time and the actual data propagation time. Effectively, this nearly doubles the waiting time from one nominal hop across the network to two. The most important contribution to waiting time is the reservation propagation time, especially if the communicating end nodes enjoy relatively small delays between them compared to the maximum delays on the reservation spanning trees.

One approach to reducing the waiting time is the use of streams, as described above. As long as the new request can fit in the existing stream allocation (and that the extent of borrowing is known), there is no need to wait for a new reservation to be processed. This is the approach chosen in the design of the SATNET and WIDEBAND scheduling algorithms to mitigate the effect of the space-segment delay. However, in these systems the space-segment delay is much longer than the 30-ms terrestrial delays assumed here; furthermore, the delay was substantially the same for all earth stations. In a Highball network most or all link delays will be different and all will be less than 30 ms for a transcontinental network. It does not seem reasonable to wait for 30 ms in order to schedule a burst to a nearby node less than a millisecond away.

An interesting way to deal with this problem in a Highball network is to restrict the scope of a reservation based on a-priori analysis of the spanning trees and delay matrix. For example, if it is known at all nodes that a particular reservation can be processed and a collision-free schedule completed before any other reservation can affect it, then it is not necessary to wait for all other nodes to hear the reservation. This might occur in the case of a number of node clusters, each consisting of a number of closely packed nodes, where each cluster is connected to others over relatively long links. For transmissions between the cluster nodes, there is no need to wait to propagate the reservations to all nodes in the network and nodes outside the cluster can simply ignore the intracluster reservations.

Another way to reduce waiting times might be the limited use of nondeterministic scheduling, in effect the introduction of ALOHA techniques. This can be done as an extension to the above deterministic evaluation of the scheduling configuration. It may be that, rather than an exhaustive evaluation to find possible collisions created by presumptive reservations that have not been heard yet, that a feasible approach is to allocate the burst anyway. There will of course be a small chance that a reservation not heard yet will result in a collision, but this will appear indistinguishable from the case where not all nodes hear the reservation due a local error. The need for and benefit from such adventures as this remain to be evaluated.

8. References

[ANS90] American National Standards Institute. *High-Performance Parallel Interface - Mechanical, Electrical and Signalling Protocol Specification.* ANSI Document X3T9/88-127, February 1990.

[ASA88] K. Asatani, K.R. Harrison, R. Ballart. CCITT standardization of network node interface of synchronous digital hierarchy. *IEEE Communications Magazine 28, 8* (August 1990), 15-20.

[BEL88] Bellcore. *SONET Digital Switch Intervace.* Bellcore Technical Advisory TA-TSY-000782, March 1988.

[DAV90] Davis, M., K. Monington, E. Perkins. 8 x 8 scheduled crossbar switch. Electrical Engineering Department Report 90-10-1, University of Delaware, October 1990.

[HA86] Ha, T. *Digital Satellite Communications*. MacMillan, New York, NY, 1986.

[HEM88] Hemrick, C.F., R.W. Klessig and J.M. McRoberts. Switched multimegabit data service and early availability via MAN technolgy. *IEEE Communications Magazine 26, 4* (April 1988), 9-15.

[JAC77] Jacobs, I., et al. CPODA - a demand assignment protocol for SATNET. *Proc. Fifth Data Communications Symposium* (1977), 2.5-2.9.

[MCA90] McAuley, A.J. Reliable broadband communication using a burst erasure correcting code. *Proc. ACM SIGCOM Symposium* (September 1990), 297-306.

[MIL90a] Mills, D.L. On the accuracy and stability of clocks synchronized by the Network Time Protocol in the Internet system. *ACM Computer Communication Review 20, 1* (January 1990), 65-75.

[MIL90b] Mills, D.L. Network Time Protocol (Version 3) specification, implementation and analysis. Electrical Engineering Department Report 90-6-1, University of Delaware, June 1990.

[MIL90c] Mills, D.L. Internet time synchronization: the Network Time Protocol. *IEEE Trans. Communications* (to appear).

[SCH90c] Schragger, P.A. Scheduling algorithms for burst reservations on high-speed wide area networks. *Proc. IEEE Infocom 91* (to appear).

[SMI90] Smith, J.F., D. Willett and S. Paul. Overview of the Space Station communications networks. *IEEE Network 4, 5* (September 1990), 22-28.