SYSTEM ANALYSIS
FOR THE
HUNTSVILLE OPERATIONAL SUPPORT CENTER
DISTRIBUTED COMPUTER SYSTEMS

SUMMARY

The Huntsville Operations Support Center (HOSC) is a distributed computer system used to provide real time data acquisition, analysis and display during NASA space missions and to perform simulation and study activities during non-mission times. The primary purpose of the research is to provide a HOSC system simulation model that may be used to investigate the effects of various HOSC system configurations. Such a model would be valuable in planning the future growth of HOSC and in ascertaining the effects of data rate variations, update table broadcasting and smart display terminal data requirements on the HOSC HYPERchannel network system.

A simulation model has been developed in PASCAL and results of the simulation model for various system configurations have been obtained. In this report a tutorial of the model is presented and the results of simulation runs are presented. Some very high data rate situations were simulated to observe the effects of the HYPERchannel switch over from contention to priority mode under high channel loading.
A computer tape was transported to NASA Huntsville and the simulation program installed on the NASA computer. Appendix II is a listing of the simulation code.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>SECTIONS</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUMMARY</td>
<td>iii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF ACRONYMS</td>
<td>x</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 HOSC System Overview</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Research Objective</td>
<td>2</td>
</tr>
<tr>
<td>2. HOSC SYSTEM DESCRIPTION</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Typical HOSC System Activities</td>
<td>5</td>
</tr>
<tr>
<td>2.2 HOSC System Components</td>
<td>9</td>
</tr>
<tr>
<td>3. HOSC MODEL DESCRIPTION</td>
<td>41</td>
</tr>
<tr>
<td>3.1 The Real System</td>
<td>41</td>
</tr>
<tr>
<td>3.2 Experimental Framework for the Base Model</td>
<td>41</td>
</tr>
<tr>
<td>3.3 Informal Description of the Base Model</td>
<td>43</td>
</tr>
<tr>
<td>4. SOFTWARE DESIGN DESCRIPTION</td>
<td>46</td>
</tr>
<tr>
<td>4.1 Scope</td>
<td>46</td>
</tr>
<tr>
<td>4.1.1 Major Software Functions</td>
<td>46</td>
</tr>
</tbody>
</table>
### TABLE OF CONTENTS (Continued)

<table>
<thead>
<tr>
<th>SECTIONS</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.2 System Files</td>
<td>47</td>
</tr>
<tr>
<td>4.1.3 Major Design Constraints</td>
<td>48</td>
</tr>
<tr>
<td><strong>4.2 Design Specification</strong></td>
<td></td>
</tr>
<tr>
<td>4.2.1 Derived Software Structure</td>
<td>48</td>
</tr>
<tr>
<td>4.2.2 Interfaces within Structure</td>
<td>53</td>
</tr>
<tr>
<td><strong>4.3 Module Description</strong></td>
<td></td>
</tr>
<tr>
<td>4.3.1 Mod3c</td>
<td>54</td>
</tr>
<tr>
<td>4.3.2 Initialize</td>
<td>59</td>
</tr>
<tr>
<td>4.3.3 CharacterizeNetwork</td>
<td>62</td>
</tr>
<tr>
<td>4.3.4 PrintNetDescription</td>
<td>66</td>
</tr>
<tr>
<td>4.3.5 FindNext</td>
<td>66</td>
</tr>
<tr>
<td>4.3.6 Picka</td>
<td>68</td>
</tr>
<tr>
<td>4.3.7 ACollision</td>
<td>70</td>
</tr>
<tr>
<td>4.3.8 UpdateClocks</td>
<td>71</td>
</tr>
<tr>
<td>4.3.9 PrintNetworkStats</td>
<td>76</td>
</tr>
<tr>
<td>4.3.10 ActivitySummary</td>
<td>78</td>
</tr>
</tbody>
</table>

For each module:

- 4.3.X.1 Processing Narrative
- 4.3.X.2 Interface Description
- 4.3.X.3 Design Description
- 4.3.X.4 Modules Used

**4.4 Validation Criteria**

79
TABLE OF CONTENTS (Continued)

<table>
<thead>
<tr>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5. HOSC SYSTEM ANALYSIS AND CONCLUSIONS</td>
<td>80</td>
</tr>
<tr>
<td>5.1 Performance Bounds</td>
<td>80</td>
</tr>
<tr>
<td>5.2 Correlation with Simulation Results</td>
<td>85</td>
</tr>
<tr>
<td>6. REFERENCES AND BIBLIOGRAPHY</td>
<td>92</td>
</tr>
<tr>
<td>APPENDIX I SOFTWARE SOURCE LISTING</td>
<td>95</td>
</tr>
<tr>
<td>APPENDIX II SUMMARY OF ISO-OSI MODEL OF DISTRIBUTED NETWORK</td>
<td>96</td>
</tr>
</tbody>
</table>
## LIST OF TABLES

<table>
<thead>
<tr>
<th>TABLES</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 HOSC Data Transfers</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Summary of DEC VAX 11/780 I/O Characteristics</td>
<td>17</td>
</tr>
<tr>
<td>2.3 Summary of PE 3240 I/O Characteristics</td>
<td>20</td>
</tr>
<tr>
<td>2.4 HYPERchannel Protocol Frames</td>
<td>37</td>
</tr>
<tr>
<td>4.1 Interactive Queries Posed by Initialize</td>
<td>60</td>
</tr>
<tr>
<td>4.2 Prompts for Network Characterization</td>
<td>63</td>
</tr>
<tr>
<td>5.1 Theoretical Upper Bounds on HYPERchannel Trunk</td>
<td>84</td>
</tr>
<tr>
<td>5.2 Cumulative Probability of Transmission Matrix</td>
<td>87</td>
</tr>
<tr>
<td>Device to Device</td>
<td></td>
</tr>
<tr>
<td>5.3 Configuration Parameters For the Multiple Simulation Runs</td>
<td>88</td>
</tr>
<tr>
<td>5.4 Statistical Results For The Multiple Simulation Runs</td>
<td>89</td>
</tr>
<tr>
<td>5.5 Comparisons of Theoretical Parameters S and U Versus Measured Results</td>
<td>91</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURES</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Proposed HYPERchannel Network Configuration</td>
<td>6</td>
</tr>
<tr>
<td>2.2 Block Diagram of VAX 11/780 Computer</td>
<td>12</td>
</tr>
<tr>
<td>2.3 Basic Bus Configuration of VAX 11/780</td>
<td>13</td>
</tr>
<tr>
<td>2.4 Block Diagram of PE 3244 Computer</td>
<td>19</td>
</tr>
<tr>
<td>2.5 Multiple Trunking System</td>
<td>22</td>
</tr>
<tr>
<td>2.6 Delay Time Events</td>
<td>26</td>
</tr>
<tr>
<td>2.7 HYPERchannel Adapter Block Diagram</td>
<td>29</td>
</tr>
<tr>
<td>2.8 HYPERchannel Frame Formats</td>
<td>31</td>
</tr>
<tr>
<td>2.9 Message Only Frame Sequence</td>
<td>32</td>
</tr>
<tr>
<td>2.10 Message with Data Frame Sequence</td>
<td>33</td>
</tr>
<tr>
<td>4.1 Full Scale Model</td>
<td>49</td>
</tr>
<tr>
<td>4.2 First Level Refinement</td>
<td>49</td>
</tr>
<tr>
<td>4.3 Second Level Refinement</td>
<td>51</td>
</tr>
<tr>
<td>4.4 Third Level Refinement</td>
<td>52</td>
</tr>
<tr>
<td>5.1 Multiple Simulation Run Configuration</td>
<td>86</td>
</tr>
</tbody>
</table>
LIST OF ACRONYMS

ASCI  American Standard Code for Information Interchange
BDP   Buffered Data Path
BR    Bus Request
CATV  Community Area Television
CPU   Central Processing Unit
CSMA  Carrier Sense Multiple Access
DDP   Direct Data Path
DEC   Digital Equipment Corporation
DMA   Direct Memory Access
ECIO  Experiment Computer Input/Output
ET    External Tanks
FEP   Front End Processor
I/O   Input/Output
ISO   International Standards Organization
HSLN  High Speed Local Network
HOSCI Huntsville Operation Support Center
JSC   Johnson Space Center
KSC   Kennedy Space Center
LPS   Launch Processing System
Mbps  Megabits per second
MEO   Main Engine Cut Off
MER   Mission Evaluation Room
MPS   Main Propulsion System
MSFC  Marshall Space Flight Center
NASA  National Aeronautics and Space Administration
NPR   Non-Processor Request
NSC   Network Systems Corporation
OD    Operational Data
OSI   Open Systems Interconnection
PE    Perkin Elmer
POCC  Payload Operations Control Center
RSS   Range Safety System
SBI   Synchronous Backplane Interface
SRB   Solid Rocket Booster
SSME  Space Shuttle Main Engine
STS   Shuttle Transport System
UBA   UNIBUS Adapter
UBUS  UNIBUS
μsec  Microsecond
SECTION 1

Introduction

1.1 HOSC System Overview

Marshall Space Flight Center (MSFC), Huntsville, Alabama, has implemented the Huntsville Operations Support Center (HOSC) to provide real-time acquisition, analysis, and display of data during National Aeronautics and Space Administration (NASA) space missions. The HOSC is a distributed computer system composed of large minicomputers and various peripheral equipment. Primarily designed to provide support for the Space Shuttle, Space Telescope, and Space Laboratory missions, HOSC has the inherent flexibility to be expanded and reconfigured to meet the needs of future missions, and also to provide MSFC with a large computer resource that can be used to support non-mission activities.

During missions, the network computers are semidedicated to performing specific mission tasks. For example, one processor is responsible for the space shuttle main engine data analysis while a separate processor provides backup capabilities for the same data. The data arrives via satellite communication links or direct ground links from the Kennedy Space Center Firing Room at NASA's Kennedy Space Center (KSC) in Florida or from NASA's Johnson Space Center (JSC) in Texas. During a mission, the computers provide data analysis and presentation services for HOSC mission support teams. In addition to the mission responsibilities, HOSC also provides
resource support for routine non-mission activities such as the Payload Operations Control Center (POCC) preplanning activity. Future non-mission activities may include computer resource support for such diverse tasks as the Digital Equipment Corporation's interactive graphics database, IGDS, and the XEROX Corporation's SIGMA text-processing operation.

1.2 Research Objective

The diversity of tasks performed during mission and non-mission operations at HOSC should be easy to perceive since the facility requires information-sharing between multiple, independent processors. Based on the requirements of each mission support team, the data traffic on the HOSC computer network may be composed of short bursts of high-rate data transfers or of sustained periods of high-rate data transfers. Furthermore, HOSC is a dynamic facility that must possess the ability to adapt and change its hardware and software resources as particular mission requirements dictate.

In order to document the flexibility and efficiency of HOSC, an analysis of the capabilities of the existing computer facility was needed. Ideally, the analysis would not only offer documentation of existing resources, but would also provide insight into future HOSC operations by predicting performance characteristics in various hypothetical mission scenarios. With this in mind, the primary goal of the research effort was the development of a simulation model of the HOSC network facility that could be manipulated to predict and characterize the behavior of HOSC in a variety of situations.
The first step in this analysis was the definition of a set of resource configurations and mission scenarios that might represent plausible growth stages of HOSC. One of these configurations is outlined in Section 2 of this document. Included in the outline are the available and projected hardware components along with their operating characteristics and functions. The second step of the analysis, detailed in Section 3 and 4, involved the development of a system model of HOSC and a software simulation of this model. In this software simulation, system parameters were varied to simulate the changes in the operating environment and the scope of responsibilities of the HOSC. Finally, the results and conclusions of these simulations were compiled and analyzed to provide a summary of current operating characteristics, as well as of the projected operating characteristics, of HOSC.
SECTION 2

HOSC System Description

As implied in the Introduction, a primary task of HOSC is to provide NASA engineers at MSFC with a near real-time data base of critical information during missions of the Space Shuttle. This information provides MSFC engineers and contractor personnel the data they need as they act in a support capacity to other mission specialists at other mission control sites such as KSC and JSC. HOSC provides pre-launch support for the Space Shuttle Main Engine (SSME), the External Tanks (ET), the Solid Rocket Boosters (SRBs), the Main Propulsion System (MPS), and the Range Safety System (RSS). Concurrent with mission responsibilities, HOSC provides resource support for mission experiments designed by MSFC specialists.

NASA has established the Flight Operations Support functions of HOSC as follows:

During powered flight, the HOSC will receive only data which is in the LPS (Launch Processing System) at KSC. The Shuttle support team will be in the HOSC during this phase of the mission and will be the point of contact with the JSC Mission Evaluation Room (MER) for problem discussion and resolution as required and will be on call during orbital operations when applicable.

Following completion of the active Shuttle vehicle support activities, data is recalled as required for more detailed analysis, and initial preparation is made to provide support to postflight evaluation [NASA82].
HOSC is located in the west end of A-wing. Building 4663 at MSFC. Figure 2.1 shows the functional components of the HOSC system at the time of this writing.

2.1 Typical HOSC System Activities

Activities performed at HOSC can be grouped into two broad categories, routine daily activities and mission/launch activities. Table 2.1 provides a summary of the activities by category and also provides the details of some typical data transfers. In addition to the activities described below and in the table are the experimental activities directed by the HOSC mission specialists. Since they change from mission to mission, the activities are not included in the following descriptions but must be characterized and considered in the specific mission scenarios required for the total system analysis.

POCC simulation is an ongoing simulation activity that utilizes the computing resources of HOSC. This activity is not associated directly with the real-time responsibilities of the network and is thus considered to be a routine daily activity.

The impact of POCC simulation on the overall performance of the network stems from the requirement of continuous data transfers between network computers. This data is currently being transferred between the MPS Primary Computer (VAX4) and the MPS Backup Computer (VAX1). During each twenty four hour period, this activity requires the transfer of 150,000 512 Byte blocks of data. This data is transferred in two components; six times each day, an 8344-Byte block
Table 2.1. HOSC Data Transfers.

I. ROUTINE DAILY ACTIVITIES

A. POCC Activity
   Resources Involved:
   MPS Primary (VAX4).
   MPS Backup (VAX1).
   Quantity of Data: 150,000 512-Byte blocks in two components. 6 8344-Byte blocks and 100,000 512-Byte blocks.

B. ECIO Data Stream (Generated by POCC)
   Resources Involved: MIPS Backup (VAX1).
   Spacelab 8/32.
   Quantity of Data: 51.2 kilobyte per second stream concurrent with POCC.

C. IGDS/SIGMA Activities (Proposed)
   Resources Involved: To be determined.
   Quantity of Data: Unknown.

II. ROUTINE LAUNCH ACTIVITIES

A. Routine Daily Activities (See Item I.)

B. Main Engine Data
   Resources Involved: STS primary (PE 3244).
   MPS Backup (VAX1).
   Quantity of Data: 50 kilobit per second stream (launch minus 8 hours until MECO).

C. OD Data Stream
   Resources Involved: FEP SSME (PE 3244).
   CSO Computer.
   STS Primary (PE 3244).
   MPS Backup (VAX1).
   Quantity of Data: 1923 kilobit per second stream into FEP and then to CSO with transfer of 40 percent to STS Primary and MPS Backup (launch minus 9 secs until MECO).

D. Engineering Display Changes
   Resources Involved: STS Primary (PE 3244).
   STS Backup (PE 8/32).
   Spacelab 8/32 (PE 8/32).
   Quantity of Data: Insignificant.
is transferred (50,000 Bytes cumulative), with another 100,000 512-Byte blocks transmitted randomly but distributed evenly throughout the day.

POCC also generates a continuous 51.2 kilobit per second data stream known as the Experiment Computer Input/Output (ECIO) data stream. This data stream is ongoing and concurrent with POCC activity. ECIO data is transferred from the MPS Backup Computer (VAX1) to the Spacelab Computer (PE 8/32c).

The routine components of the launch day data activities are the Main Engine Data Stream, the Operational Data (OD) Stream, and the Engineering Display Change requests. The Main Engine Data is collected and disseminated on launch day only. Data is funneled through the network to the MPS backup Computer (VAX1) from the Shuttle Transport System (STS) Computer (PE 3244). From eight hours before launch until about twelve minutes after launch at Main Engine Cut Off (MECO), STS Primary accepts a continuous 50 kilobit per second data stream directly from the KSC Firing Room and transfers about 24 percent (12 kilobits per second) of the total stream to MPS Backup.

The OD Stream is a 192 kilobit per second data stream arriving at the Front End Processor Space Shuttle Main Engine (FEP SSME) Computer (PE 3244) on launch day concurrently with some of the Main Engine Data (launch minus 9 seconds until MECO). This data arrives from Goddard Space Flight Center, Maryland and is transferred over the network to the CSO facility.
The Engineering Display Change activity is an almost insignificant addition to the total network traffic. This activity involves a transfer from STS Primary to STS Backup and the Spacelab 8/32. This transfer consists of the name of each engineering console format in the support facility that is changed during this prelaunch and launch period (launch minus 9 seconds to MECO).

2.2 HOSC System Components

HOSC may be thought of as simply a communications channel (Local Area Network) for connection of computer resources. The basic components of the system are the computers and the communications processors that receive the transmitted data from telephone lines and microwave links and the computer networking hardware that provides the interconnection between the computers. The HOSC network hardware is a high speed, local network manufactured by Network System Corporation (NSC) called HYPERchannel. HYPERchannel provides communications services between computers from a variety of manufacturers such as Digital Equipment Corporation (DEC), Perkin-Elmer (PE), and UNIVAC. In order to provide a clearer understanding of the resources involved at HOSC, a description of HYPERchannel is given, along with a cursory look at two of the network computers that provide the bulk of the processing power for the network: the DEC VAX and the PE 3240. These machines are examined for sake of information only, since from the point of view of the subsequent analysis all network users are considered to be nothing more than data sources with fixed data generating attributes. An in-depth look
is then given to HYPERchannel network since the success of the simulation model will rest directly on a clear and accurate understanding of it.

The Digital Equipment Corporation’s VAX series of computers supports a 32-bit word architecture that establishes a virtual address space of 4.3 billion bytes of user-addressable memory. Throughput of data in the system is optimized by using a 32-bit high speed data structure that ties together the central processor, main memory, UNIBUS subsystem, MASSBUS subsystem and the DR780 high speed direct memory access subsystem. This high speed data structure is known as the Synchronous Backplane Interface (SBI) and a device linked to it is known as a NEXUS. Each NEXUS receives every SBI transfer; electronic logic in the NEXUS determines whether it is the designated receiver for the ongoing transfer. Data transfers can occur from CPU to memory, from I/O controller to memory subsystem, or from CPU to I/O controller, with maximum aggregate transfer rates of 13.3 megabytes per second. These transfer rates are limited by the following factors:

1. 200 nanoseconds/cycle = 5 million cycles per second
2. each cycle can carry an entire byte of data or address representing a memory request
3. one cycle is used to request eight bytes of data to be read or written, and two cycles are used to carry data at four bytes per cycle
4. 5 million cycles/second * 4 bytes/cycle = 20 million bytes/second
\[ 20 \times \frac{2}{3} \text{ (1 of every 3 cycles is an address)} = 13.3 \text{ million bytes per second} \ [\text{DEC80}] \]

As is shown in Figure 2.2, a VAX computer may interface with more than one memory subsystem. In the case of a two-controller, interleaved memory configuration, the computer would also have two NEXUS memory controllers on the SBI.

The UNIBUS Subsystem (UBUS) is a high speed, asynchronous data system that allows communication between peripheral hardware and the VAX computer. The VAX 11/780 is capable of supporting four UBUSes: one is standard, but three more are optional (Figure 2.3). The UBUS is connected to the SBI through a UBUS Adapter (UBA) which performs several important services that are transparent to the user. The UBA provides:

- access to UBUS address space from the SBI
- mapping of the UBUS addresses to SBI addresses for the direct memory access (DMA) transfers to the system memory
- data transfer paths for UNIBUS device access to random SBI memory addresses and high speed transfer for devices that transfer to consecutive, increasing addresses
- UNIBUS interrupt fielding
- UNIBUS priority arbitration
Figure 2.2 Block Diagram of VAX 11/780 Computer
The address-mapping function is especially important because the UBUS has only eighteen data lines providing an apparent memory-addressing capability of only 200 kilobytes. Through its address mapping abilities, the UBA provides the capability of mapping the UNIBUS addresses into the SBI addresses so that the full system memory of 16 array boards of 256 kilobytes each (4 gigabytes total) can be accessed.

The UBA accepts two forms of input from the UBUS: hardware-generated interrupts and direct memory access requests. Each device connected to the UBUS may use one of five priority levels for requesting bus service. A non-processor request (NPR), which is the highest priority request, may be used when the device requests an operation that does not require processor intervention, such as a direct memory access transfer to memory or to some other device. A bus request (BR) is thus generated when the device wishes to interrupt the UBA for service. Such service might be a CPU-directed data transfer or a flag indicating the existence of an error condition at the peripheral. Since there are only five priority levels and more than one device may be connected to a specific request level, if more than one device makes the same request, the device that is electrically closest to the UBA receives the highest priority.

The NPR request for direct memory access is a very important feature of the UBUS subsystem. These DMA transfers can be divided into two groups: random access of noncontiguous addresses and sequential access of sequentially increasing addresses. For random
access, each UBUS transfer is made through the Direct Data Path (DDP, one per UBUS) and is mapped into an SBI transfer. This procedure allows only one word of data to be transferred during a single SBI cycle. For devices capable of requesting sequential access services, use is made of the Buffered Data Path (BDP). Each UNIBUS provides fifteen such BDPs which store the data so that four UBUS transfers are performed for each SBI transfer.

The DDP must be used by devices not transferring to consecutively increasing addresses or by devices that mix read and write functions. The maximum throughput via the DDP is about 425 kilowords per second for write operations and about 316 kilowords per second for each read operation. These rates will decrease as other SBI activity increases [DEC80].

Maximum published throughput via the BDP is about 695 kilowords per second for both the read and write operations, but realistic throughput rates of only 1.5 megabits per second are actually expected with this figure degrading as other SBI activity increases [WILK82, MCNU82]. BDP transfers are restricted to block transfers with blocks of length greater than one byte. All transfers within the block must be to consecutive and increasing addresses and all transfers must be of the same function, either read or write.

The MASSBUS subsystem and the DR780 high performance, 32-bit parallel interface will not be described in this summary, since an understanding of their functional characteristics is not needed to determine their relative impacts on the HYPERchannel network. The influence of both may be felt indirectly, however, since activity on
the MASSBUS or DE780 will translate to SBI activity which will affect DDP and BDP transfers rates as previously described.

The VAX CPU will also not be described in detail but several comments may be made about the CPU and its effects on system throughput. The CPU represents the most intensive traffic load on the memory subsystem and hence on the SBI. Obviously, if the processor is engaged in intensive computing, it will request data much more often than it will write data, and this memory access represents substantial SBI activity. Fortunately, the large cache memory of eight kilobytes available to the CPU is able to reduce the SBI traffic load considerably. In addition to the effects of the CPU of SBI activity, the SBI traffic from other sources may also affect the CPU efficiency. Published figures indicate that in a system with two memory controllers the processor will be slowed by about four percent per averaged megabyte per second of I/O traffic, while the impact of a single memory controller is to slow the processor by a factor varying from two to four. Table 2.2 summarizes the I/O characteristics of the DEC VAX 11/780 processing system.
### TABLE 2.2 SUMMARY OF DEC VAX 11/780 I/O CHARACTERISTICS

<table>
<thead>
<tr>
<th>PROCESSOR</th>
<th>32-bit word architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN MEMORY:</td>
<td></td>
</tr>
<tr>
<td>Virtual Address Space:</td>
<td>4.3 billion bytes.</td>
</tr>
<tr>
<td>Cycle Times:</td>
<td>800 nanoseconds per 64 bit read.</td>
</tr>
<tr>
<td></td>
<td>1400 nanoseconds per 64 bit write.</td>
</tr>
<tr>
<td>I/O UNIBUS ADAPTER</td>
<td></td>
</tr>
<tr>
<td>Maximum aggregate rate:</td>
<td>1.5 Mbyte/sec through BP</td>
</tr>
<tr>
<td>Buffered Data Paths (BDP):</td>
<td>15 total, 8 byte buffer in each.</td>
</tr>
<tr>
<td></td>
<td>695 kilowords/sec read and write.</td>
</tr>
<tr>
<td></td>
<td>Used for fast data transfers.</td>
</tr>
<tr>
<td>Direct Data Path (DDP):</td>
<td>425 kilowords/sec for write</td>
</tr>
<tr>
<td></td>
<td>316 kilowords/sec for read</td>
</tr>
<tr>
<td></td>
<td>Used for transfers to nonconsecutive memory locations.</td>
</tr>
</tbody>
</table>
The Perkin Elmer 3240 series of computers is also a high throughput machine with a 32-bit architecture. The HOSC currently uses two PE3244 machines with primary responsibilities as front end processors receiving real-time data streams from the KSC firing room.

The 3244 memory subsystems is organized into banks, each capable of handling four megabytes of addressable memory. Total system memory ranges from 256 kilobytes in one bank to a full system complement of four 4-megabyte banks, for a maximum of 16 megabytes of addressable memory. All memory is connected to a common memory bus which consists of two unidirectional, asynchronous, 32-bit busses. One bus is dedicated to memory write functions and the other is dedicated to memory read functions (Figure 2.4).

Input/Output is accomplished using five external communication busses: one multiplexer bus for medium speed devices and a maximum of four high-speed Direct Memory Access (DMA) busses that each support eight high speed bidirectional ports. Each DMA port is controlled by a selector channel tied to the multiplexer bus that controls and terminates transfers through the CPU. Once the channel is activated, the processor is released and is free to continue processing on an unrelated task. Published I/O transfer rates for the PE3244 DMA bus indicate that transfer rates of up to 10 megabytes per second in the burst mode are possible for each DMA bus [PEC81]. The I/O characteristics of the PE3240 series is summarized in Table 2.3.
Figure 2.4 Block Diagram of PE 3244 Computer
TABLE 2.3. SUMMARY OF PERKIN ELMER 3240 I/O CHARACTERISTICS

**PROCESSOR:** 32 bit/word.

**MAIN MEMORY**

Virtual Address Space: 16 Megabytes.

Cycle Time: 500 nanoseconds.

**DMA BUS DATA TRANSFER RATE**

Burst Mode: 10 megabytes per second with a maximum of 4 DMA busses per system.
The Network Systems Corporation (NSC) HYPERchannel high speed local area computer network is a 50 Megabit per second (Mbps) baseband communication system employing a Carrier Sense Multiple Access (CSMA) contention scheme with prioritized, staggered delays. Network messages called frame sequences are broadcast over a 75-ohm CATV cable (called a trunk) at a fixed 50 Mbps rate, using a phase modulation encoded Manchester code and received by network components at various nondirectional taps or ports along the trunk (Figure 2.5). Access to a trunk can be gained only at a port and is controlled by a network adapter present at that port. NSC supplies three types of network adapters: processor, device, and link. The processor adapter is designed to allow trunk access under a strict set of network protocol rules for intelligent devices such as minicomputers and mainframe processors. NSC designates these adapters as A400 HYPERchannel Adapters which will hereinafter be referred to as A400 adapters, or more simply as adapters. The device adapter provides trunk access for devices such as printers, dumb terminals, and mass storage devices. Network to network communications are made possible through the link adapters. Due to the nature of the communication environment at HOSC, the only adapter included in this model is the processor adapter [THOR83].
When an adapter receives a request from a host processor for trunk usage, the adapter assembles the host data along with the appropriate control and destination information into a package and attempts to transmit the first frame packet of the package on the trunk. At the trunk level, adapters are capable of carrier sensing, which means that before a trunk transmission is attempted the adapter first determines whether or not a carrier is present on the trunk. The existence of a carrier indicates an ongoing transmission and the ready adapter defers its request in favor of the ongoing communication. Carrier sensing, however, in itself is not sufficient to overcome the problems that arise when two or more adapters attempt simultaneous or near simultaneous trunk transmissions. This condition, which usually results in unintelligible communications and subsequent loss of transmitted frames, is known as a collision. To overcome this shortcoming and increase the 'hospitality' of the network, the adapters provide several other services that are adequately summarized in the following list from Donnelly and Wey.

... [The] ports supplied by NSC are packaged in an adapter that includes some high-speed buffer memory and a microprocessor that:

- Decouples the 50-Mbit/sec trunk speed from the possibly slower speeds of the interfaces to the attached systems.
- Handles duplicate detection on the trunk.
- Supplies a flow of control mechanism.
- Recovers gracefully from various error conditions.
- Allows flexibility in communicating with attached systems [DONN79].
As mentioned previously, carrier sensing in itself provides inadequate contingency processing actions for adapters sensing a busy trunk. By using timers in each adapter, HYPERchannel is able to supplement the carrier sensing by providing linear priority ordering of adapters on each trunk. Upon detection of a busy trunk, an adapter goes into a waiting state by setting a series of delay clocks that will determine the time of the next possible transmission attempt. This action occurs in every adapter any time a carrier is sensed on the trunk regardless of whether the adapter is actually ready to transmit. As soon as the carrier drops, indicating that a current frame transmission has passed and that the trunk is free, the clock countdown toward the next transmission attempt is initiated. If at anytime during the delay state, the trunk is again sensed busy, the delay times are reset and reinitiated as soon as the carrier drops.

As the traffic of transmitted packets increases toward a level where the trunk is in a state of constant demand, this series of delays will effectively slot the transmission medium, thereby allowing prioritized time slots in which each adapter may gain trunk access. The series of delays is called the total delay and may be composed of three separate delay events: fixed delay, priority delay, and end delay. The fixed delay represents the maximum amount of time required to transmit a frame between the two adapters separated by the longest physical length of cable. In addition to being a component of the total delay event, the fixed delay also represents the maximum amount of time an adapter must wait to receive
a response frame from any other adapter on the trunk and, as such, prevents additional adapters from transmitting until the receiving adapter has had time to send a response frame to the transmitting adapter. This delay in microseconds (μsec) is determined for each trunk in the network by the following equation [NSC79]:

\[
\text{Fixed Delay} = \frac{\text{Total Trunk Length (ft)}}{40} + 13 \\
(\mu\text{sec})
\]

This value is set into switches in the adapter and is recommended by NSC to be the same for each adapter on a particular trunk.

The priority delay (DLy) allows the linear priority ordering of adapters on the trunk. This delay will permit adapters with higher trunk accession priorities proportionally more trunk time than adapters with lower priorities. This parameter is an installation-established value that can be different for each adapter. Computation of the priority delays begin by choosing the adapter(s) to have zero priority delay and then establishing the priority classes for the remaining adapters according to the following equation [NSC79]:

\[
\text{Priority Dly} = 10 + \frac{\text{priority delay}}{\mu\text{sec}} + \frac{\text{cable length (ft) to next highest adptr}}{40}
\]

Once the priority delay has experienced, the adapter may capture the trunk. At this level, the trunk is in a pure priority scheme with each adapter allowed a time slot for transmission (Figure 2.6). At
Figure 2.6 Delay Time Events
the conclusion of the entire priority delay period, all adapters are free to contend for the trunk.

If an adapter completes a transmission during its priority delay time, it may conceivably attempt to regain the trunk sometime during the remainder of the priority delay period. To prevent this occurrence, each adapter is provided with a wait flip-flop, which is enabled by a wire jumper on the printed circuit board assembly in the adapter. If the user desires to allow each adapter only one actual access to the trunk during any one priority delay period, this jumper may be removed to enable the wait flip-flop. This action forces the adapter to cycle through an end delay before again contending for the trunk thus allowing the adapters in the other priority classes an opportunity to gain access to the trunk. Therefore, the end delay is the time following the fixed delay and the priority delay in which all adapters on the trunk have free access to the trunk. This delay is calculated for each adapter as shown in the following equation [NSC79]:

\[
\text{End Dly} = 10 + \frac{\text{cable to length (ft) from (priority dly adptr to farthest adptr)}}{40} + \text{of lowest adptr}
\]

The timing control just described is implemented and augmented by a resident microprocessor in each adapter. In addition to protocol management, these microprocessors perform several other critical functions. The adapter microprocessor receives parallel data from the host processor requesting trunk access, buffers the
data, forms it into network frame sequences, and then transmits the packets serially over the trunk under the previously described protocol. Acting as receiver, the adapter collects the serial data from the trunk and assembles the data into parallel packets for transfer to the appropriate host processor. The functional characteristics of the adapter are shown in Figure 2.7.

The effect of the described priority protocol is to lessen considerably the possibility that an adapter will attempt to begin a communication session when the trunk is engaged in an ongoing transmission. Although the time window for such a collision has been closed to a mere crack, a small time slot still exists in which a collision may occur. This occasion most often arises when two or more adapters become ready to transmit almost simultaneously, that is to say they seek to access the trunk within a time interval that is less than the trunk end-to-end transmission time. This resultant collision and loss of data is thus a very real possibility and contingent actions are needed.

HYPERchannel provides a retry mechanism to recover from such collisions. Under this scheme, an adapter is able to detect when a collision has occurred, and the adapter reschedules subsequent transmission attempts. In order to understand more completely this retry mechanism, a more thorough understanding of the packet transmission apparatus is needed.

The first step in communicating over the trunk is the establishment of a virtual circuit between the two or more communicating adapters. When an adapter receives a request from a
NOTES:

1. The A400 Adapter supports the maximum host processor data transfer rates. Typical Values:
   - HE 3244 - 10 Mbytes/sec N/Spiral Interface
   - 2.2 Mbytes/sec normal.
   - P/E 8/32c - 6 Mbytes/sec in Burst Mode
   - 4 Mbytes/sec in non-burst.
   - DEC Unibus - 1.5 Mbytes/sec through Buffered Data Path

2. Microprocessor (logical) interface ("ROMS) to direct and control device transmissions, trunk selections, trunk transmissions and assemble/disassemble of frame sequences.

3. Trunk side consists of I/O buffering, serial/parallel conversion, checkword generator and checker, and trunk transceiver.

4. Device Interface side consists of I/O Extension Registers and DMA Controller

Figure 2.1. HYPEChannel Adapter Block Diagram.
host for trunk access, the adapter first reserves itself, thereby marking itself as unable to become part of another communication session. After reserving itself, the adapter attempts to reserve the prospective receiving adapter(s) by sending a Set Reserve frame on the trunk. If the receiving adapter is idle, it will respond by setting its reserved flag and transmitting a ready-to-receive response frame to the initiating adapter. (Figure 2.8 shows the contents of the reservation frame sequences.) After the reservation sequence, the initiating adapter releases the trunk (both adapters remaining reserved but other adapters could now use the trunk) while it forms the message frame sequences containing the data. These data sequences are of two classes, message only sequences and message with data sequences. Message only sequences allow the transmission of a message proper, which includes the frame sequence data, access codes, destination codes, and a small amount of data up to a total frame sequence length of 64 bytes (Figure 2.9). For larger data transfers, a message with data sequences is utilized in which data are transmitted in blocks as large as two kilobytes (kbytes) in length (Figure 2.10). (Four kbytes are an option when the adapters have increased buffer capacity). After the communicating adapters have been reserved with a message proper sequence and trunk access has been regained for the data transfer, a Copy Register frame is sent to the receiving adapter and, if successfully responded to, the sending adapter captures the trunk and sends the two kbyte data block. Subsequent data blocks are sent until the entire message has been sent. At this time, an End of Data message is sent by the
### Transmission Frame Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leading syncs</td>
<td>12 - 18</td>
</tr>
<tr>
<td>Frame type</td>
<td>1</td>
</tr>
<tr>
<td>Access code</td>
<td>2</td>
</tr>
<tr>
<td>To address</td>
<td>2</td>
</tr>
<tr>
<td>From address</td>
<td>2</td>
</tr>
<tr>
<td>Function code</td>
<td>2</td>
</tr>
<tr>
<td>Message length</td>
<td>2</td>
</tr>
<tr>
<td>Check word</td>
<td>2</td>
</tr>
<tr>
<td>Trailing syncs</td>
<td>8</td>
</tr>
</tbody>
</table>

### Response Frame Format (No Data)

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leading syncs</td>
<td>3</td>
</tr>
<tr>
<td>Frame type</td>
<td>1</td>
</tr>
<tr>
<td>Access code</td>
<td>2</td>
</tr>
<tr>
<td>To address</td>
<td>1</td>
</tr>
<tr>
<td>From address</td>
<td>1</td>
</tr>
<tr>
<td>Function code</td>
<td>2</td>
</tr>
<tr>
<td>Message length</td>
<td>2</td>
</tr>
<tr>
<td>Check word</td>
<td>2</td>
</tr>
<tr>
<td>Trailing syncs</td>
<td>8</td>
</tr>
</tbody>
</table>

### Message Proper and Associated Data Frame Format

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leading syncs</td>
<td>3</td>
</tr>
<tr>
<td>Frame type</td>
<td>1</td>
</tr>
<tr>
<td>Access code</td>
<td>2</td>
</tr>
<tr>
<td>To address</td>
<td>1</td>
</tr>
<tr>
<td>From address</td>
<td>1</td>
</tr>
<tr>
<td>Function code</td>
<td>2</td>
</tr>
<tr>
<td>Message length</td>
<td>2</td>
</tr>
<tr>
<td>Check word</td>
<td>2</td>
</tr>
<tr>
<td>Message/data</td>
<td>8-64/0-2K</td>
</tr>
<tr>
<td>MSG/ID</td>
<td>16</td>
</tr>
<tr>
<td>Checkword</td>
<td>2</td>
</tr>
<tr>
<td>Trailing syncs</td>
<td>8</td>
</tr>
</tbody>
</table>

### Figure 2.8. Frame Formats.
Figure 2.9. Message-only Frame Sequence
<table>
<thead>
<tr>
<th>SENDER</th>
<th>RECEIVER</th>
<th>CHANNEL</th>
<th>CHANNEL</th>
<th>INTERVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delay</td>
<td>(Preparation to Transmit)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Set Reserve</td>
<td>Response</td>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Delay</td>
<td></td>
<td></td>
<td>23</td>
</tr>
<tr>
<td>Copy Registers</td>
<td>Response (Reg. Info.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>Response</td>
<td></td>
<td></td>
<td>74.5</td>
</tr>
<tr>
<td>(64 Bytes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clear Flag 8</td>
<td>Response</td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>Delay</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.10. Timings for Message-only Sequences.

(Transmission times do not include 1.32 microsecond per foot propagation delay.)
*Frames enclosed within brackets are transmitted without interference by beginning their transmission in the fixed delay period.

Figure 2.10 (Cont.). Message with Data Frame Sequence.
Figure 2.10 (Cont.). Timings for Message-with-data sequences.
(Transmission times do not include 2.32 microsecond per foot propagation delay.)
sending adapter, and both adapters release themselves, returning to
an unreserved state. A brief summary of the HYPER channel protocol
frames is given in Table 2.4.

During the communication session just described, the trunk is
released after each data frame (two-kbyte packet) sequence.
Consequently, the trunk must be regained for the transmission of each
two-kbyte data block. This allows other adapters a chance to contend
for the trunk during the time a message sequence is being formed.
Adapters also provide an alternative to this transmission mode known
as burst mode. Under this scheme, the trunk is not relinquished
between data block transfers but is retained until the entire message
has been transmitted. This mode will not be described in further
detail since it is not used in the BOSC network.

Each adapter is capable of interfacing with and controlling the
trunk transmission requests of from one to four minicomputer hosts of
the same or of differing manufacturers. This interface is effected
by an NSC Peripheral Interface board resident in each minicomputer
host, a Device Interface for each computer, located in the A400
adapter and the associated cabling and connectors between the adapter
and the host processor. Whenever an adapter is not actively engaged
in some function, the adapter microprocessor scans the device
interface ports in a round robin polling sequence for a request to
provide some network service. When a request is detected, the
scanning is suspended and the service is performed. After completing
the service task, the polling process is resumed.
### TABLE 2.4
HYPERchannel Protocol Figures

#### Transmission Frames

<table>
<thead>
<tr>
<th>Frame Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Reserve</td>
<td>Reserve receiving adapter</td>
</tr>
<tr>
<td>Copy Registers</td>
<td>Request for Register information</td>
</tr>
<tr>
<td>Clear Flag 8</td>
<td>When set, Flag 8 indicates that the adapter is ready to receive a frame. When clear, it indicates that the frame has been received.</td>
</tr>
<tr>
<td>Set Flag A</td>
<td>Flag A when set indicates that the receiver will receive associated data. It is cleared when the last block of associated data is received.</td>
</tr>
<tr>
<td>Clear Flag A</td>
<td></td>
</tr>
<tr>
<td>Clear Flag 9</td>
<td>When set in the receiver, Flag 9 indicates that the receiver is ready to receive data. When set in the transmitter, it indicates that the transmitter is ready to send data. When cleared in the receiver, it indicates that the data block has been received. When cleared in the transmitter, it indicates that data can now be transmitted.</td>
</tr>
</tbody>
</table>

#### Data and Message Frames

<table>
<thead>
<tr>
<th>Frame Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Proper</td>
<td>Used to transmit host messages of length less than or equal to 65 bytes.</td>
</tr>
<tr>
<td>Associated Data</td>
<td>Used to transmit data blocks of length up to 2K bytes.</td>
</tr>
</tbody>
</table>

#### Response Frames

<table>
<thead>
<tr>
<th>Frame Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response</td>
<td>A receiving adapter sends a response frame for each non-response frame received. Its purpose is to acknowledge received frames or, as in the case of the response to a copy registers frame, to send requested data.</td>
</tr>
</tbody>
</table>
In addition to the four device ports, the adapter may access from one to four different trunks through separate trunk interface boards resident in the adapter (Refer to Figure 2.6). When a network service requiring a trunk activity is required, the host processor sends a request to the adapter containing a list of trunks to try. The adapter microprocessor then polls viable trunks in search of an open transmission path. This selection is also made from a round robin polling of trunks connecting the transmitting adapter and the prospective receiving adapter, with each poll taking about 5.5 μsec [FRAN82].

If an adapter attempts the reservation of a disabled or previously reserved adapter, a Reject Reservation frame is sent by the reserved adapter. This rejection will cause the requesting adapter to put itself into a reservation retry loop in which entries are scheduled after a binary exponential delay. That is, after the first rejection, the sender delays 1 μsec before attempting another reservation. If the requested adapter is still not free, the requesting adapter will wait 2 μsec more before attempting the third reservation request. This scheme is repeated, with each subsequent retry delayed an amount of time equal to twice the previous delay until the adapter is reserved or until the retry delay reaches 128 μsec. At this point, the retry delay clock is reset to 1 μsec and the process repeated for a retry count number of times. The retry count is an adapter specific value based on the six low order bits of the manually set adapter unit number and is in the range of 55 to 250. Thus, 64 unique retry counts are possible. If after retry-count
delay sequences the adapter is still not able to secure the desired reservation, the adapter will return to the idle state, informing the requesting host of the inability to transmit.

This retry mechanism is enhanced by the ability of the requesting adapter to determine the identity of the receiving adapter's reservation partner. If the requesting adapter discovers that the reservation lock up is caused by both adapters attempting to simultaneously reserve each other, the requesting adapter aborts its reservation attempt and releases itself for reservation by the other adapter.

Although the retry scheme provides some advantages, it also leaves unresolved two important problems. First, due to the slowness of the back-off mechanism (about 50 μsec are needed for an adapter to realize the simultaneous reservation problem and to react), both adapters may back off. Consequently, neither host is permitted to transmit on the trunk.

The second and more serious problem presents itself when a reservation request loop is formed. This occurs when three or more adapters attempt to reserve each other simultaneously. For example, Adapter A attempts to reserve Adapter B, who attempts to reserve Adapter C, who in turn attempts to reserve Adapter A. The exponential delay mechanism is initiated in each adapter and proceeds to completion in each adapter, terminated only by the retry count. Obviously, serious degradation of the trunk throughout may be predicted during these reservation request loop periods [DONN79].
Finally we arrive at the mechanism employed when a true collision of data occurs on the trunk. This is detected when an adapter fails to receive a response frame from a frame transmission within time interval equal to the adapter's fixed delay. If this situation occurs, the adapter simply retries the transmission. Reservation request frames are repeated 16 times on each trunk and all other frames are retransmitted 256 times.

A concluding note to the HYPERchannel transmission protocol is the messages can be transmitted in one of two ways: intra-adapter or inter-adapter. A message transmission between devices linked to the same adapter is called an intra-adapter transmission and takes place entirely locally with no trunk transfer. A transmission between devices on different adapters is termed inter-adapter communication. Although intra-adapter communications occur off-trunk, these transfers also directly affect the network throughput and trunk usage figures. Since adapters may be engaged in one and only one communication session at any time, if an adapter requests the services of an adapter engaged in intra-adapter business, the requested adapter will appear busy, as if it were engaged in another trunk transfer activity. If, however, resources do not conflict, intra-adapter activities may occur concurrently with inter-adapter activities.

Obviously, as the message generation and reception needs of the network components change, or the network components themselves change, the network throughput and trunk usage figures will change. What is needed is a method for examining the effects of changes in the system configuration and this is the justification for this modelling and simulation effort.
SECTION 3

HOSC MODEL DESCRIPTION

3.1 The Real System

The HOSC is, as has been previously described, a distributed computer network. This network is used to analyze and disseminate information that is of interest to scientists and engineers at the Marshall Space Flight Center as they monitor the progress of NASA Space Shuttle missions. The primary components of the HOSC system are the computers that perform the data manipulation tasks and the HYPERchannel high speed local network that provides the dissemination services. As the Space Shuttle mission priorities change from flight to flight, so do the operating parameters of the Huntsville Operation Support Center. In an effort to predict the effects of these changes on the HOSC network, a model and a simulation of the HOSC were developed.

3.2 Experimental Framework for the Base Model

The simulation task was accomplished by developing a general model of a HYPERchannel network and by imposing parameters on this model which describe the operations of HOSC. These constraints form an experimental frame through which the real system is observed and exercised. The intent was to produce a relatively simple model that would produce replicatively and predictively valid data corresponding
with the actual HOSC network. The term replicatively valid indicates that the model data should match the data obtained from the real system operating under conditions that match the parameters of the simulation. Predictively valid implies that the results of the model could be used to accurately predict the performance of the real system in a hypothetical configuration. To achieve this model validity, an attempt was to also make the model loosely structurally valid [ZIEG76]. The modifier 'loosely' is applied to indicate that although the model was not developed to correspond directly to the hardware functions at the electrical and mechanical level of the real system, the model was designed to correspond as nearly as possible to the functional characteristics of the real system at the protocol level. Relating this in some way to the proposed standard, seven level, ISO-OSI model for distributed computer processing (Appendix E), the model could be thought of as having been developed between levels three and four, the Network and Transport layers, as opposed to having been developed at the Physical layer, level 1.

The performance of the network model will be reported, using the following measures:

- **D**: The delay that occurs between the time a frame sequence becomes ready to transmit and the actual completion of the transfer.

- **S**: The throughput of the network reported as the total rate of data transmitted over the network.

- **U**: The utilization of the network normalized to one. This figure represents the fraction of the total trunk capacity used.

- **G**: The offered trunk load. This is the total rate of data presented to the network including control frames and retransmission frames.
0 I: The input load. This is the rate of data that is generated by an adapter attached to the trunk. This represents only the user-generated data that is transmitted over the trunk and thus excludes the control and retransmission frames.

Where possible, these figures will be reported for the network as a whole, the individual trunks in the network, the individual adapters, and the individual devices.

3.3 Informal Description of the Base Model

Components

TRUNK: Coaxial Cable interconnection HYPERchannel users.
ADAPTER: Port through which a user gains access to a trunk.
DEVICE: HYPERchannel user. The device is the fundamental entity of the network.
OFFNET-SOURCE: Non-network supplier of data to a device. Supplies that raw data that will be reformed by the devices and routed over the network.
FRAME-SEQUENCE: Basic quantity of information that can be transported on the network.

Descriptive Variables

TRUNK(i) (i=1,4)
SUCCESSFUL-TRANSFERS: Range of non-negative numbers indicating number of bits successfully transferred on the trunk.
OFFERED-TRUNK-LOAD: Range of non-negative channel for transfer. Includes control and retransmission sequences.
INPUT-LOAD: Range of non-negative numbers. Total number of user generated data bits offered to the trunk for transfer.
TOTAL-DELAY: Propagation delay of the trunk in feet per microsecond (ft/μsec).
LENGTH-OF-TRANSMISSION-FRAME: Number of bits in a trunk transmission sequence. (Range from 64 bytes to 4 kilobytes.)

ADAPTER(i) (i=1,N)
SUCCESSFUL-TRANSFERS: Range of non-negative numbers indicating number of bits successfully transferred on the trunk.

OFFERED-TRUNK-LOAD: Range of non-negative numbers indicating number of bits successfully transferred on the trunk.

INPUT-LOAD: Range of non-negative numbers. Total number of user generated data bits offered to the trunk for transfer.

DISTANCE-FROM-REFERENCE-ADAPTER: Number of feet of trunk cable of the adapter from a trunk reference adapter. Used for computing adapter delay times.

PRIORITY-DELAY: Time (in μsec) indicating the relative transmission priority of the adapter.

PORT-STATUS: Indication of whether an adapter port is supporting an active hose.

DEVICE(i) (i=1,4)

TRANSMISSION-FREQUENCY: Probability distribution indicating the relative frequency that the device will request the trunk (range from 0 to 1.0).

INTERNAL-BUFFER-SIZE: Non-negative number indicating the amount of data buffered by a host before requesting the trunk.

RETRY-COUNT: Integer in the range of 1 to 64 indicating the number of times an adapter will repeat a reservation request.

I/O-BUS-TRANSFER-RATE: Real number in units of kilobytes per second giving the speed with which data may be transferred from the host processor to the adapter.

SUCCESSFUL-TRANSFERS: Range of non-negative numbers indicating number of bits successfully transferred on the trunk.

OFFERED-TRUNK-LOAD: Range of non-negative numbers. Total number of bits offered to the channel for transfer. Includes control and retransmission sequences.

INPUT-LOAD: Range of non-negative numbers. Total number of user-generated data bits offered to the trunk for transfer.
SOURCE\((i)\) (\(i=1,N\))
DATA-STREAM-RATE: Real number in units of kilobytes per second indicating the continuous rate at which data arrives at a device from the offnet source.

TRANSMITTING-DEVICE: The initiating device engaged in a transmission session.

RECEIVING-DEVICE: A non-initiating device engaged in a transmission session.

PARAMETERS
BANDWIDTH: Bandwidth of network trunk (50 Mbits per second).

TIME: Non-negative real number representing the interval of time during which the model will be valid.

Component Interaction
FRAME-SEQUENCES are transmitted over a TRUNK from DEVICE to DEVICE through the network ADAPTERS. The number of FRAME-SEQUENCES in a transmission session is determined from the INTERNAL-BUFFER-SIZE of the DEVICE. The frequency of transmission sessions is determined from the SOURCE DATA-STREAM-RATE and the DEVICE TRANSMISSION FREQUENCY.

SUCCESSFUL-TRANSFERS occur if the FRAME-SEQUENCE is transferred from the TRANSMITTING-DEVICE to the RECEIVING-DEVICE without collision. If the TRUNK is engaged in a transmission when a DEVICE becomes ready to transmit, the ADAPTER will be delayed until the PRIORITY-DELAY expires for the ADAPTER and the TRUNK is free.
SECTION 4

HOSC SYSTEM SIMULATION DESIGN

4.1 Scope

This section describes a computer simulation developed from the system model for the Huntsville Operation Support Center. The goal of the simulation is to provide a software tool which may be used to study the operating characteristics of the HOSC. The tool should have the capability of generating data that replicates the performance of the existing HOSC system, as well as the capability of predicting the performance of the HOSC system to hypothetical configurations.

4.1.1 Major Software Functions

The primary functions of the software should be to:

1. Provide a replicatively and predictively valid simulation of the real system model described in Section 3,
2. Provide an easy method for modifying the simulation environment so that it may be used to simulate environments different from the existing system,
3. Provide meaningful output that may be converted to the performance indicators described in Section 3 and 5,
4. Provide a user friendly interface at all levels of user interaction.
The software should be designed:

1. To be as portable as possible so that the simulation may be transferred easily from the development computer to other computer environments.
2. To allow enhancements of the fundamental algorithm as time and further study permit.

4.1.2 System Files

The HOSC software simulation requires the maintenance of three files for proper execution; the program file contains the object code for the simulation program, and two auxiliary data files store information that is used by, or created by, the simulation program. The auxiliary data file, DESCRIP-FILE, is used for long term storage of data used to describe the system to be simulated. At the beginning of each simulation, the user is given the option of simulating the system described in DESCRIP-FILE or of creating a new system description while on-line with the simulation program. The auxiliary file, AUX-OUT, is maintained totally by the simulation software and provides long term storage for the detailed summary of the simulation. This file is print-formatted and may be directed to a system printer at the user's request.

Modification of the system description file, DESCRIP-FILE, allows the user to change to the simulation scenario and can be changed during the characterization stage of each run. During this stage of the run, the user is given the opportunity to provide
additional input for the simulation run through interactive queries. These queries establish such run dependent parameters as the seed for the random number generator, descriptive titles for the detailed output summaries, and the interval of simulated time for which the simulation run will execute.

Both auxiliary files are maintained as standard ASCII text files. The program file may be maintained as either source or object code since code modification should be unnecessary except during algorithm enhancements.

4.1.3 Major Design Constraints

The primary design constraint imposed on this simulation effort was the simulation be transportable between computer systems of unspecified size and manufacturer. To accommodate this constraint, the use of a large, general purpose simulation language, such as SLAM or GPSS, was considered impractical. Therefore, the language Pascal was chosen because of its widespread availability and its richness of data structures.

4.2 Design Specifications

4.2.1 Derived Software Structure

The fundamental system model (FSM) for the simulation program, shown in Figure 4.1, illustrates the interfaces between the simulation program, the user, and the two auxiliary files.

The FSM may be decomposed into three separate activities: characterization, simulation, and summary. This composition leads to
Figure 4.1 Full Scale Model

Figure 4.2 First Level Refinement
the possible refinement of the FSM into the data flow diagram of Figure 4.2.

The Characterization module has a dual task; initialization of the run parameters, and specification of the experimental framework of the simulation. The Summary module may also be divided into two distinct activities: a brief summarization of the network statistics and a more detailed reporting of the simulation activities. This breakdown suggests the refinement shown in Figure 4.3.

The Simulation activity can be described by the following algorithm which increments the simulation clock each time a significant event occurs in the simulation.

Set Clocks at Beginning Time
Find Transmitter
Determine a Receiver
Attempt to Transmit
If No Collision then
Clocks may be Updated to reflect successful transmission
Find next Transmitter
else
Clocks must be updated to reflect collision
Find next Transmitter
Repeat until simulation complete

With this algorithm, the data flow diagram shown in Figure 4.4 may be used to describe the simulation activities.
Figure 4.4 Third Level Refinement
### Interfaces within the structure

<table>
<thead>
<tr>
<th>MODULE</th>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VARIABLES</td>
<td>VARIABLES</td>
</tr>
</tbody>
</table>

| Initialize  | none             | Heading           |
|            |                  | MaxTime           |
|            |                  | MaxTx             |
|            |                  | U, RNG seed       |
|            |                  | PrintAll          |
|            |                  | InputNode         |

| Characterize| None             | Adapter array    |
| Network     |                  | Pr, Transmis. Prob.|
|            |                  | NumofTrunks      |
|            |                  | TrunkLength array|

| PrintNetDescription | Adapter array     | none |
|                     | Pr                |

| FindNext          | Adapter array    | Twitter |
|                  | Pr                |

| PicA             | Transmitter      | Receiver |
| Pr               | Adapter array    |

| ACollision       | Adapter          | ACollision |
| Tmitter          | Tmitter          |

| UpdateClocks     | Tmitter          | Time       |
|                  |                  | BitsTx     |
|                  |                  | TtxTally   |
|                  |                  | NextTx     |
|                  |                  | TxCt       |
|                  |                  | TxCtTime   |
|                  |                  | RxCt       |
|                  |                  | RxTxTime   |
|                  |                  | TimeActive |

<p>| PrintNetworkStats| TxTally          | CollisionTally |
|                 | CollisionTally   | WaitTally     |
|                 | TtotalAttempts   | TotalLost     |
|                 | TotalLost        | TotalLost     |
|                 | TimeActive       | TimeActive    |</p>
<table>
<thead>
<tr>
<th>MODULE</th>
<th>INPUT VARIABLES</th>
<th>OUTPUT VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivitySummary</td>
<td>Adapter, TtTally, CollisionTally, WaitTally, TotalAttempts, TotalLost, TimeActive</td>
<td>none</td>
</tr>
</tbody>
</table>
4.3 Module Descriptions

4.3.1 Mod3c

4.3.1.1 Processing Narrative

Mod3c is the top level module of the HOSC simulation program. It defines the simulation environment and provides the flow of control necessary for the simulation. Mod3c calls procedures Initialize and CharacterizeNetwork to create and define the simulation environment. Using this characterization of the environment, Mod3c controls the actual system simulation as was illustrated in the data flow diagram, Figure 4.4.

At the completion of the simulation activity, Mod3c initiates procedures PrintNetworkStats and ActivitySummary. PrintNetworkStats provides a brief summary of the simulation and prints this summary in the auxiliary output file and on the system default output device. ActivitySummary compiles a detailed recount of the simulation at the individual device level and records this summary in the auxiliary output file.
4.3.1.2 Interface Description

Procedure call: program Mod3c (Input, Output, DescripFile, AuxOut)

Parameters:
Input: system defined default input file
Output: system defined default output file
DescripFile: Text (* File of Char *)
AuxOut: (Text (* File of Char *)

Called by: none — Top Level

Calls:
Initialize
CharacterizeNetwork
PrintNetDescription
FindNext (Tmittr)
PickA (Rcvr)
ACollision
UpdateClocks (Condition[])
PrintNetworkStats
ActivitySummary

Global variables referenced:

- Current Time (* Simulation time *)
- MaxTime (* Simulation parameter, When CurrentTime > MaxTime, simulation terminates *)
- MaxTx (* Simulation parameter, When TxTally > MaxTx, simulation terminates *)
- TxTally (* Sum of successful transmitted *)
- CollisionTally (* Sum of collisions *)
- TotalAttempts (* Sum of transmissions and collisions *)
- Tmittr (* Active Transmitter *)
- Rcvr (* Active Receiver, object of transfers from Tmittr *)
- Condition (* Indicates whether or not collision occurred in transmission *)
4.3.1.3 Design Description

Data Structure

type
AdapterRecord = record
  Device: array[1..4] of Device Records;
  PriorityDelay: real;
  EndDelay: real;
  DistFromAl: real;
  ATxCT: integer; (* Adapter Transmission Tally *)
  AWaitCt: integer; (* Adapter tally of Waits *)
  ACollCt: integer; (* Adapter tally of Collision *)
  ATxTime: real; (* Adapter time spent in Transm.*)
  AWaitTime: real; (*Adapter time spent waiting *)
  ACollTime: real; (*Adapter time spent in Colls. *)
end;

DeviceRecord = record
  Open: boolean (* Flag signalling port status *)
  if NOT Open then
    ID: string of char; (* Device name *)
    RetryMax: integer; (* Retry Count *)
    TferRate: real; (* I/O port transfer rate *)
    NumofSources: integer; (*Num of_offset sources*)
  Source: array[1..MaxNumSources] of
    SourceRecords;
  LoadTime: real (* Inverse of Tfer Rate *)
end;

SourceRecord = record
  GenRate: real; (* Data generation rate Bytes/s *)
  BufferSize: integer; (* Size of Device buffer, Amount of data accum. before tx *)
  ID: string of char;
  OnTrunk: array[1..4] of boolean; (*List of trunks to try *)
  NextTx: real; (*Time of next transm.*)
  LastTx: real; (*Time of last transm.*)
  TxIntvl: real; (*Time between transm.*)
  TxCt: integer; (*Tally of transm.*)
  WaitCt: integer; (*Tally of Waits *)
  CollCt: integer; (*Tally of Collisions *)
  RxCt: integer; (*Tally of times as receiver *)
  TxTime: real; (*Time in transm.*)
  WaitTime: real; (*Time in Waiting *)
  CollTime: real; (*Time in Collisions*)
  RxTime: real; (*Time as Receiver *)
end;
Variables

Adapter: array[1..MaxNumAdapters] of AdapterRecord;
Pr: array[1..NumOfDevices,1..NumOfDevices] of real;
(* Pr is probability of Transmission matrix
stored as cumulative probabilities *)

Tmittr, Rcvr: SourceDescriptions;

Algorithm
begin
  Initialize variables

  Characterize network

  Print description in auxiliary output file

  Set clock at beginning time

  Find Transmitter

  Determine Receiver

  Attempt to transmit
  if No Collision then
    Clocks may be updated to
    reflect successful transmission

    Find next transmitter
  else
    Clocks must be updated to
    reflect collision

    Find next transmitter

  Repeat until simulation complete

  Print Summary of Network Statistics

  Print Activity Summary to auxiliary output file
end.
4.3.1.4 Modules Used

Initialize
CharacterizeNetwork
PrintNetDescription
FindNext
PickA
ACollision
UpdateClocks
PrintNetworkStats
ActivitySummary

Descriptions of these modules will be contained in the following sections.

4.3.2 Initialize

4.3.2.1 Processing Narrative

Procedure Initialize is invoked by the top level module, Mod3c, at the outset of each simulation run with the purposes of initializing the global program variables, and of establishing other parameters for the run. Variable initializations are performed without user-intervention with the purpose of giving each variable a defined and constrained value at the beginning of the run. The global parameters that can not be adequately defined through initialization are assigned values in response to interactive queries posed to the user. (A list of the interactive queries is shown in Table 4.1. Also included in the table are some of the constraints imposed on the responses to these questions.)
<table>
<thead>
<tr>
<th>Query:</th>
<th>Identification Heading for run:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response:</td>
<td>Valid responses are alphanumeric strings with length less than 50 characters. This heading appears as a heading for the detailed output summaries.</td>
</tr>
<tr>
<td>Example:</td>
<td>Simulation Configuration la -- 4/24/84</td>
</tr>
<tr>
<td>Variable:</td>
<td>Heading</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query:</th>
<th>Maximum run time in secs?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response:</td>
<td>Valid responses are real numbers specifying the interval of simulated time for which the simulation will run.</td>
</tr>
<tr>
<td>Example:</td>
<td>60.0</td>
</tr>
<tr>
<td>Variable:</td>
<td>MaxTime</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query:</th>
<th>Maximum successful transmissions?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response:</td>
<td>Valid responses are integer numbers setting the simulation interval in terms of the number of successful transmissions that occur of the network. Simulation is terminated when transmission count exceeds this value or when maximum run time has been exceeded.</td>
</tr>
<tr>
<td>Example:</td>
<td>500</td>
</tr>
<tr>
<td>Variable:</td>
<td>MaxTx</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query:</th>
<th>Seed for random number generator?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response:</td>
<td>Integer number. For best results, the number specified should be an odd, positive number with at least 5 digits, but not divisible by 5 or 7.</td>
</tr>
<tr>
<td>Example:</td>
<td>316227</td>
</tr>
<tr>
<td>Variable:</td>
<td>U</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query:</th>
<th>PRINT-ALL condition on?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response:</td>
<td>Y or N. If Y, then a record of each network, activity will be printed on the system's default output device. Due to the magnitude of output generated, this query should be answered Y for short simulation runs only.</td>
</tr>
<tr>
<td>Example:</td>
<td>N</td>
</tr>
<tr>
<td>Variable:</td>
<td>PrintAll and Resp (See Comments)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query:</th>
<th>Input mode: INTERACTIVE or FILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response:</td>
<td>I or F. If File is specified, network description will come from DecripFile, otherwise, the interactive network specification will be initiated.</td>
</tr>
<tr>
<td>Example:</td>
<td>F</td>
</tr>
<tr>
<td>Variable:</td>
<td>Input Mode and Resp (See Comments)</td>
</tr>
</tbody>
</table>
4.3.2.2 Interface Description

Procedure call: Initialize

Parameters: none

Called by: Mod3c

Calls: none

Global variables referenced:

- WaitTally (*Tally of devices put into waiting states during trans attempts*)
- CollisionTally (*Tally of collisions occurring on network*)
- TotalAttempts (*Sum of waits, colls., and transmissions*)
- Pct (*Tally of pages in output file*)
- TimeActive (*Total time during which trunk in active*)
- BitsTx (*Total number of bits transmitted on network*)
- Heading (*Identification heading for the run*)
- MaxTime (*Interval of simulated time for which sim. will continue*)
- MaxTx (*Number of transm. for which simul. will run*)
- U (*Seed for random number generator*)
- PrintAll (*Flag for detailed trace of simulation*)
- InputMode (*Variable indicating source of network descr. File or Interactive*)

4.3.2.3 Design Description

This procedure is simple straight line code containing the statements necessary to make the desired initializations and assignments.

4.3.2.4 Modules Used

None
4.3.3 CharacterizeNetwork

4.3.3.1 Processing Narrative

Procedure CharacterizeNetwork is invoked by the top level module, Mod3c, with the purpose of establishing the experimental framework of the simulation. In this procedure, the primary data structures representing the abstraction of HOSC network are assigned values. These values are defined either interactively by the user at the time of simulation, or they may be called from the auxiliary data file, DescripFile. The choice of origin of the system description is controlled by the variable InputMode, which is assigned a value in procedure Initialize. If InputMode has the value Interactive, the system description is controlled by the variable InputMode, which is assigned a value in procedure Initialize. If InputMode has the value Interactive, the system description is performed interactively and stored in DescripFile. An InputMode value of FileInput specifies that the network description will be read from DescripFile, implying a previous run in which DescripFile was created interactively.

Network characterization is accomplished in two stages: network description and probability-of-transmission establishment. A detailed listing of the interactive queries posed during the characterization process is presented in Table 4.2.

Following the characterization of the network, the probability-of-transmission matrix is enumerated. This enumeration is prompted by the query, 'CUMMULATIVE PROBABILITY MATRIX', whereupon, the probabilities will then be converted to cumulative values and stored in the matrix, PR.
<table>
<thead>
<tr>
<th>Query</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total number of trunks in network:</strong></td>
<td>(Maximum number of n)</td>
</tr>
<tr>
<td>Response</td>
<td>Integer number in range 1 to MaxNumTrunks.</td>
</tr>
<tr>
<td>Variable</td>
<td>NumOfTrunks</td>
</tr>
<tr>
<td><strong>Length of trunks in feed (used for END DELAY)</strong></td>
<td></td>
</tr>
<tr>
<td>Trunk 1:</td>
<td></td>
</tr>
<tr>
<td>Trunk 2:</td>
<td></td>
</tr>
<tr>
<td>Trunk n:</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>Positive real numbers</td>
</tr>
<tr>
<td>Variable</td>
<td>TrunkLength[i]</td>
</tr>
<tr>
<td><strong>Number of Adapters in Network:</strong></td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>Integer number in range 1 to MaxNumAdapters</td>
</tr>
<tr>
<td>Variable</td>
<td>NumOfAdapters</td>
</tr>
<tr>
<td><strong>Adapter #i:</strong></td>
<td></td>
</tr>
<tr>
<td>Distance in ft from common trunk Adptr 1:</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>Positive real number. Establishes a reference adapter on each trunk.</td>
</tr>
<tr>
<td>Variable</td>
<td>Adapter[i].DistFromA1</td>
</tr>
<tr>
<td><strong>Adapter Priority delay:</strong></td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>Positive real number representing seconds of delay</td>
</tr>
<tr>
<td>Variable</td>
<td>Adapter[i].PriorityDelay</td>
</tr>
<tr>
<td><strong>Adapter[i].Device[j].description</strong></td>
<td></td>
</tr>
<tr>
<td>Device status (OPEN or CLOSED):</td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>0 or C. 0 if adapter[i], device port[j] has an active, network device connected to it. C otherwise.</td>
</tr>
<tr>
<td>Variable</td>
<td>Adapter[i].Device[j].Open</td>
</tr>
<tr>
<td><strong>Device ID (≤24 char)</strong></td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>Alphanumeric string</td>
</tr>
<tr>
<td>Variable</td>
<td>Adapter[i].Device[j].Id</td>
</tr>
<tr>
<td><strong>Device to Adapter I/O rate (Bytes/s):</strong></td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>Positive real number representing the speed of the I/O port of the device.</td>
</tr>
<tr>
<td>Variable</td>
<td>Adapter[i].Device[j].TransferRate</td>
</tr>
<tr>
<td><strong>Number of data sources in Device</strong></td>
<td></td>
</tr>
<tr>
<td>Response</td>
<td>Integer number representing the number of off-net sources supplying device with data.</td>
</tr>
<tr>
<td>Variable</td>
<td>Adapter[i].Device[j].Source[k].Id</td>
</tr>
</tbody>
</table>
Query: **Buffer size in Bytes:**
Response: Integer number. Size of buffer filled in device before network transmission is requested.
Variable: Adapter[i].Device[j].Source[k].BufferSize

Query: **Trunks accessed by Source[k]**
   *(Enter Y if connected, N if not)*
   Trunk 1:
   Trunk 2:
   .
   .
   .
   Trunk 1:
Response: Y or N. Indicates the trunks over which data from each source will be transmitted.
Variable: Adapter[i].Device[j].Source[k].OnTrunk[l]
4.3.3.2 Interface Description

Procedure call: CharacterizeNetwork

Parameters: none

Called by: Mod3c

Calls: RewriteDescripFile
       InitializeProbMatrix

Global variables references:

Adapter (*Records containing description of each adapter in network *)
PR (*Probability of Transmission Matrix *)
NumofAdapters (*Network adapter count *)
NumofTrunks (*Network trunk count *)
Dnum (*Network device count *)

Files accessed:

Input (*Default Input File*)
Output (*Default Output File*)
DescripFile (*Auxiliary Output File*)

4.3.3.3 Design Description

This procedure is designed using simple code containing the statements necessary to make the desired queries and variable initializations.

4.3.3.4 Modules Used

RewriteDescripFile -- Subordinate procedure employed to rewrite the auxiliary description file, DescripFile. Requires access to the global variables. Adapter, PR, NumofAdapters, NumofTrunks, and Dnum.

InitializeProbMatrix-- Subordinate procedure employed to establish the values of the probability of-transmission matrix. PR. Requires access to global variables Adapter, PR, NumofAdapters, and Dnum.
4.3.4 PrinNetDescription

4.3.4.1 Processing Narrative

Procedure PrintNetDescription is invoked by the top levels module, Mod3c, with the purpose of printing a copy of the network description in the auxiliary output file, AuxOut.

4.3.4.2 Interface Description

Procedure call: PrintNetDescription
Parameters: none
Called by: Mod3c
Global variables referenced:
Adapter (*Array of records containing the description of each adapter in the network.*)
PR (*Probability of transmission matrix*)
NumofAdapters (*Number of adapters in network*)
NumofTrunks (*Number of trunks in network*)
Dnum (*Number of devices in network*)
Lct (*Number of lines in AuxOut*)
Pct (*Number of pages in AuxOut*)

Files accessed:
AuxOut (*Auxiliary output files*)

4.3.4.3 Design Description

This procedure is designed using simple code containing the statements to copy the Adapter records into the auxiliary output file, AuxOut. Adapter records are copied with an appropriated heading of numbered pages in the file.

4.3.4.4 Modules Used

NewPage Procedure used to copy blank lines into the auxiliary output file at places appropriate to give page breaks. Procedure NewPage requires access to the file, AuxOut and to the global variables Lct, Pct, and Heading.

4.3.5 FindNext

4.3.5.1 Processing Narrative
Procedure FindNext is invoked by the top level module Mod3c, with the purpose of choosing an appropriate transmitter in the simulated network. The simulation process is event-driven with a device's request for network service acting as the primary event. The simulation clock is advanced each time a significant event occurs in the simulation. Although the clock may be advanced in cases of simulated collisions and wait conditions, the primary simulation event is the completion of a successful transmission. Procedure FindNext examines the characteristics of each device, and chooses a ready transmitter based on the amount of data that a device may receive from an off-network source and the buffer size of that source. This determination is based on the current value of the system clock and the time of the device's next transmission request.

4.3.5.2 Interface Description

Procedure call: FindNext(Tmittr)

Parameters: Tmittr (*Variable of type SourceDescription representing the simulated network's currently active transmitter*)

Called by: Mod3c

Calls: none

Global variables referenced:

Adapter , (*Array of records containing description of each adapter in network *)
NumofAdapters (*Number of adapters in the network *)
4.3.5.3 Design Description

Algorithm

begin (* procedure FindNext *)

for I := 1 to Number of Adapters

for J := 1 to 4 (* scan each of the 4 possible devices on the adapter *)

for K := 1 to Number of Sources

choose adapter with earliest time of next transmission

end

end

end

return the attributes of the next transmitter through the procedure parameter
end (* procedure *)

4.3.5.4 Modules used

none

4.3.6 Picka

4.3.6.1 Processing narrative

Procedure Picka examines all possible recipients of data generated by transmitter that is currently active, and designates one of these devices to be the active receiver. This choice is made by examining the probability-of-transmission matrix, PR, and a random number generated by the pseudo-random number generating procedure, Uniform. The values stored in PR represent cumulative probabilities of transmission from one device to another, so, the determination of the receiver is made by searching the row of the matrix representing
the possible receiving devices of the transmitter until the table entry is larger than the random number.

### 4.3.6.2 Interface Description

**Procedure call:** `Picka(Rcvr, Tmitter)`

**Parameters:**
- `Rcvr` (*Variable of type SourceDescription assigned the attributes of the currently active receiving device.*)
- `Tmitter` (*Variable of type SourceDescription assigned the attributes of the currently active transmitting device*)

**Called by:** `Mode3c`

**Calls:**
- `Uniform` (*Uniform random num generator*)

**Global variables referenced:**
- `Adapter` (*Array of records containing the descriptions of each adapter in the network*)
- `NumofAdapters` (*Number of adapters in the network*)
- `U` (*Real variable assigned the value of the random number chosen by Uniform*)

### 4.3.6.3 Design Description

**Algorithm**

```plaintext
begin (* procedure Picka *)
  choose Uniform random number
  determine receiver based on random number and probability matrix
  assign attributes of receiver to parameter `Receiver`
end (* procedure *)
```

### 4.3.6.4 Modules used

**Uniform**

Uniform is a routine that calculates a uniform pseudo-random number in the range from 0 to 1. The global variable, `U`, is accessed in the procedure and used as the seed for the random number calculation. After the first call to the routine, the seed becomes
the last random number calculated. The initial value of the seed should be an odd, positive number of at least 5 digits, but not divisible by 5 or 7.

4.3.7 ACollision

4.3.7.1 Processing narrative

The purpose of the boolean function, ACollision, is to determine whether or not the currently transmitting device has free and unrestricted use of the simulated transmission medium. If another device attempts to transmit within a length of time equal to, or less than, the EndDelay of the trunk, a collision is likely to occur. The purpose of ACollision is to compare the transmission time of the currently active device with the NextTx time of each of the other devices on the network. If another device has a NextTx time scheduled to fall within EndDelay number of seconds from the current time, then a collision will be declared and the value of ACollision will be set to true. The function compares the transmission times to a constant called CollEps, which should be assigned a value equal to the EndDelay of the trunk.

4.3.7.2 Interface description

Function call: ACollision

Parameters: none

Called by: Mod3c

Calls: none

Global variables referenced:

NumofAdapters (*Network adapter count *)
Adapter (*Array of adapter records in which are stored the attributes of each record in the network *)
CollisionTally (*Tally of collisions that occur on the network*)
PrintAll (*Program flag that, when set to True, specifies the printing of each collision for a detailed trace of the simulation*)

4.3.7.3 Design description

Algorithm

begin (* function ACollision *)
for I := 1 to NumofAdapters do
for J := 1 to 4 do (* Examine each port *)
for K := 1 to NumofSources do
if abs(Device.NextTx - Transmitter.NextTx) <= CollEps then
set function value to True
end
end
end
end (* function ACollision *)

4.3.7.4 Modules used
none

4.3.8 UpdateClocks

4.3.8.1 Processing narrative

Procedure UpdateClocks is invoked by the top-level module, Mod3c. UpdateClocks is the most crucial link in the simulation program since this is the module in which the bulk of the
The HYPERchannel protocol is modelled. As pointed out in previous sections, the simulation program is event-driven, indicating that the simulation clock is advanced each time a significant event occurs. This method of driving the simulation is often referred to as activity scanning, or event scanning.

The event that initiates the procedure is a simulated transmission request. Each device (or entity) is assumed to possess certain transmission attributes determined from observations of the real system. For example, if a device receives data for trunk transfer from an off-network source at an average rate of 15 kilobytes per second, and stores that data in a 2096-byte buffer, it will request a trunk transmission about once every 0.14 seconds. (At the present level of detail, the model does not account for reservation requests made by adapters wishing to transmit on the trunk, therefore, the simulation assumes that a trunk transfer is a transfer of actual data.) This transmission request becomes the event that drives the simulation clock.

The clock will be updated in one of several ways, depending on the circumstances surrounding the requested transmission. If another device wishes to transmit at the same time, or within EndDelay seconds of the current time, then a collision will occur and the clock will be updated to simulate the response of the real HYPERchannel system to a collision. The real system response to this situation is to repeat the transmission attempt up to a total of 256 retries. In the real system, this is an adequate method for
separating the conflicting adapters since a certain amount of time is required for each adapter to respond to the detected collision. This response time will probably be different for each adapter based on the random differences in hardware. Even in the worst case, where two adapters attempt to reserve each other at exactly the same time, the adapters would probably be able to separate themselves in the 256 retries. The simulation program must respond to a collision in a slightly different, and more artificial, way. In the event of a collision, the NextTx time of each of the conflicting devices is advance by a random percentage of the trunk EndDelay. With this algorithm, the conflicting adapters may be separated.

If a device becomes ready to transmit during the time that another adapter has control of the transmission medium, the ready adapter's NextTx time is advanced by an amount equal to the adapter's DelayTime. This is, therefore, the method used by UpdateClocks to simulate the wait state of the real system. The third method for updating the clock is the simulation of a successful trunk transmission. In this case, the simulated system clock is advanced by the amount of time required for the block of data to be transferred over the trunk. Any device that becomes ready to transmit within this interval will, of course, be forced into a wait state.

As can be inferred from the preceding discussion, parts of the model are rather naive abstractions of the real system. This is especially evident from the fact that the simulation does not attempt
an abstraction of the reservation rejection problems described in the
detailed description of the HYPERchannel protocol of Section 2.2.

4.3.8.2 Interface description

Procedure call: UpdateClocks(Condition[])

Parameters: Condition (*Var. of Normal, Collision,
or Other. Normal indicates
no collision *)

Called by: Mod3c

Calls: TransferTime(Tmittr, Rcvr) (*Calculates time required
to transmit entire data
block between adapters *)

PrintWaitStats (*Utility routine that
prints detailed record
of wait state *)

PrintLostStats (*Utility routine that
prints detailed record
of lost messages*)

Global variables referenced:

Tmittr (*Currently active
transmitting device *)

Rcvr (*Currently active receiving
device *)

CurrentTime (*Simulation time *)

BitsTx (*Total bits transm. on
trunk*)

TxTally (*Transmission Count *)

PrintAll (*Flag for detailed trace of
simulation *)

NumofAdapters (*Adapter count *)

WaitTally (*Wait state count *)

TimeActive (*Time trunk involved in
transmissions *)
4.3.8.3 Design description

Variables

DelayTime: real (*Amount of time device to be delayed in event of collision*)

Time: real (*Auxiliary storage for time to transdata between devices*)

Algorithm

begin
  case COND of
    Normal (* No collision *): begin
      Calculate Time by invoking TransferTime
      Increment BitsTx (*Total bits transferred*)
      Advance clock
      Tally a successful transmission
      Check to see if another adapter becomes ready during transfer interval
      if true then (* initiate wait state *) begin
        Calculate Total Adapter Delay
        Advance adapter's NextTx by Total Delay
        Tally wait conditions
      end
      Advance transmitter's NextTx
    end (* Successful Transmission case *)
    Collision: begin
      Tally collision statistics
      Compute time to delay for collision
      Increment Retry tally
      Repeat until RetryTally > 256
      If RetryTally > 256 then tally lost message
    end (* Collision case *)
    Other; end
  end (* case *)
end (* Procedure UpdateClocks *)
4.3.8.4 Modules used

TransferTime — Calculates the time requires to transmit a message from a sender to a receiver. Passes parameters for transmitter and receiver.

Uniform — Calculates a uniform pseudorandom number in the range 0 to 1. Parameter is seed for the RNG.

4.3.9 PrintNetworkStats

4.3.9.1 Processing narrative

Procedure PrintNetworkStats prints a brief summary of the simulation activity at the end of the simulation run. The abbreviated summary is directed to the system's default output device, and to the auxiliary data file, AuxOut. The format of the brief summary is as follows:

***END OF RUN***

Current time: xxxxxxx.xxxx secs

Successful transmissions: nnnnn
Collisions: nnnnn
Waits: nnnnn
Total Attempts: nnnnn
Total messages lost: nnnnn

Total time active: xxxxxxx.xxxx
% total time active: xxx.xx

Total Bytes transmitted: nnnnnnnnnnnn

Seed for RNG: nnnnnnn
4.3.9.2 Interface description

Procedure call: PrintNetworkStats
Parameters: none
Called by: Mod3c
Calls: none

Global variables referenced

- CurrentTime (* Simulation clock *)
- TxtTally (* Successful transm. *)
- CollisionTally
- WaitTally
- TotalAttempts (* Sum of transmissions, collisions, and waits *)
- TotalLost (* Tally of lost transm. *)
- TimeActive (* Time trunk spend servicing devices *)
- PerCentActive (* TimeActive/CurrentTime *)
- BitsTx (* Total bits transm. *)
- Lu (* Storage for random number seed *)
- Lct (* Line count in AuxOut *)

Files accessed:

- Output (* Default output file *)
- AuxOut (* Auxiliary output file *)

4.3.9.3 Design description

This procedure is designed using simple code containing the statements necessary to write the desired variable values in the auxiliary output file in the default output file.
4.3.9.4 Modules used
none

4.3.10 ActivitySummary

4.3.10.1 Processing narrative

Procedure ActivitySummary produces a detailed summary of the simulation activities and prints this summary in the auxiliary output file, AuxOut. The summary is compiled at the device label, with a tally of the activities of each device included in the summary of tables. In addition to the simple tallies of transmissions, collisions and waits, a separate table is compiled containing the percentage times that the devices spend engaged in each activity.

4.3.10.2 Interface description

Procedure call: ActivitySummary
Parameters: none
Called by: Mod3c
Calls: NewPage

Global variables referenced:

- NumofAdapters (*Number of adapters in the network *)
- Adapter (*Array of records in which attributes of each adapter of each adapter stored *)
- TimeActive (*Total time trunk spend in servicing adapter requests *)
- Let (*Line count of AuxOut *)

Files Accessed:

- Output (* Default Output file *)
- AuxOut (* Auxiliary Output file *)
4.3.10.3 Design description

Procedure ActivitySummary is primarily a formatting routine and, as such, little processing is done. Percentage-active times are calculated in two ways: percentage of the time the device is active on the trunk, and percentage of the total time the trunk is active.

4.3.10.4 Modules used


4.5 Validation Criteria

The validation of this model will depend primarily on its structural validity. Although, the modelled system exists physically, due to the nature of ROSC's mission, access to the system for the purpose of creating a validation test-bed was not possible or feasible. Consequently, at the time of this writing, all validation of the model has been done empirically, based on the understanding of the system hardware presented in the earlier sections of this document.
SECTION 5

HOSC System Analysis and Conclusion

This section examines some of the interrelationships that affect the performance of the HOSC HYPERchannel High Speed Local Network (HSLN) and draws some conclusions about the operating environment of HOSC. HOSC interrelationships will be examined by comparing figures derived experimentally from the model described in the preceding sections with certain analytical upper bounds which will be described in the following paragraphs.

5.1 Performance Bounds

In Section 3, several metrics were briefly described as being valid indicators for describing the performance of a network. These measures were labelled D, S, and U, representing the delay, D, that occurs between the time a frame is ready for transmission and the actual time of its successful transmission, the channel throughput, S, and the channel utilization, U. In order for the experimental figures to provide a meaningful picture of the network performance, some standard is needed against which they may be compared. The development of any such set of standard metrics must be predicated on the knowledge of the network factors that affect the performance of
the HSLN. In particular, the factors that independent of the attached devices are of prime importance. The chief factors are:

- Bandwidth
- Propagation delay
- Number of bits per typical frame
- Local network protocol
- Offered load
- Number of stations [STALL84].

The first three of these factors can be used to calculate a local network parameter known as $a$. This parameter may be used to deduce the maximum channel utilization, $U$, and the maximum channel throughput, $S$.

Another important parameter of the channel is defined to be the bandwidth, $B$, of the medium multiplied by the distance, $d$, of the communication path. By dividing this quantity by the propagation velocity, $V$, a component known as the bit length of the channel can be calculated.

$$\text{Bit Length} = \frac{Bd}{V}$$

Network System Corporation suggests that the propagation velocity of 40 feet per microsecond be used when considering the channel performance. Using this figure and assuming a trunk length of 1000 feet, the bit length of a theoretical HYPERchannel trunk with a 50 MHz bandwidth may be calculated to be 1900 bits.
The ratio of the bit length, L, to the length of a typical frame forms the dimensionless component called a.

Thus,

\[ a = \frac{\text{bit length of trunk}}{\text{frame length (bits)}} \]

or

\[ a = \frac{Bd}{VL} \]

Noticing that d/V is the propagation time on the medium, this component may be replaced by the HYPERchannel Fixed Delay. Recalling from Section 2 that the Fixed Delay may be calculated as

\[ \text{Fixed Delay} = \frac{\text{length of trunk (ft)}}{40} + \mu\text{sec} \]

(\(\mu\text{sec}\))

the parameter a may be calculated as

\[ a = \frac{B}{L} \cdot (\text{Fixed Delay}) \]

Thus, for a HYPERchannel trunk of length 1000 feet, a is determined to be .48.

An intuitive discussion from William Stalling's book Local Networks, indicates that a may be used to determine an upper bound on the utilization of a local network. Stalling also indicates that
this hypothesis appears to be validated by experimental evidence, therefore, a part of this discussion is presented here.

...Consider a perfectly efficient access mechanism that allows only one transmission at a time. As soon as one transmission is over, another node begins transmitting. Furthermore, the transmission is pure data—no overhead bits. . .

What is the maximum possible utilization of the network? It can be expressed as the ratio of total throughput of the system to the capacity or bandwidth:

\[
U = \frac{\text{throughput}}{B} = \frac{L/(\text{propagation} \ast \text{transmission time})}{B} = \frac{L/(D/V + L/B)}{B} = \frac{1}{1 + a}
\]

So, utilization varies inversely with \( a \) [STALL84].

An example of the use of \( a \) to predict the theoretical bounds of a HYPERchannel trunk is summarized in Table 5.1.
TABLE 5.1. THEORETICAL BOUNDS ON A HYPERCHANNEL TRUNK LENGTH 1000 FEET

- **Fixed Delay** = 38 $\mu$sec
- **Bit Length** = 190 bits
- **$a = .48$**

\[
U = \frac{1}{1 + a} = .68
\]

\[
S = U \times B = 33.8 \text{ Mbits/sec}
\]
5.2 Correlation with Simulation Results

The discussion on performance bounds in the preceding section culminates with the calculation of a throughput rate for a HYPERchannel trunk. This prediction does not, however, take into account the protocol under which the trunk transmissions are conducted, and therefore, represents only a theoretical throughput rate. Nevertheless, based on the empirical worth of this figure given by Stalling, [Stal84], several useful conclusions may be drawn by relating these figures to the figures obtained from the simulation program.

In Table 5.3, the configuration parameters for a multiple of simulation runs. The intent of these runs is to demonstrate the performance of the HOSC system over a variety of data source generation rates, including some conditions of very high Trunk Activity times. Figure 5.1 and Table 5-2 illustrate the HOSC system model and identifies the various adapters and devices.

Table 5.4 tabulates the statistical results for the various simulation runs. It may be observed that the trunk activity is very high for run 5. In fact the trunk is very close to saturation for this case. The data source generation rates are very large for run 5 and are not expected to occur for an appreciable length of time in practice.

The results clearly demonstrate the robust nature of HYPERchannel under heavy load conditions. As has been pointed out in earlier sections, the HYPERchannel modifies its protocol from CSMA
Figure 5.1 Multiple Simulation Run Configuration
<table>
<thead>
<tr>
<th></th>
<th>110</th>
<th>211</th>
<th>221</th>
<th>311</th>
<th>321</th>
<th>331</th>
<th>411</th>
<th>421</th>
<th>431</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.70</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>211</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>221</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.00</td>
</tr>
<tr>
<td>311</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0.02</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>321</td>
<td>0</td>
<td>0.01</td>
<td>0.02</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>331</td>
<td>0</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>411</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>421</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>431</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
TABLE 5-3
CONFIGURATION PARAMETERS FOR THE MULTIPLE SIMULATION RUNS
(Refer to Figure 5.1)

<table>
<thead>
<tr>
<th>DEVICE NUMBER</th>
<th>RUN1</th>
<th>RUN2</th>
<th>RUN3</th>
<th>RUN4</th>
<th>RUN5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>2211</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>3221</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>4311</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>5321</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>6331</td>
<td>3300</td>
<td>3300</td>
<td>3300</td>
<td>3300</td>
<td>3300</td>
</tr>
<tr>
<td>7441</td>
<td>3300</td>
<td>3300</td>
<td>3300</td>
<td>3300</td>
<td>3300</td>
</tr>
<tr>
<td>8421</td>
<td>1200</td>
<td>1200</td>
<td>1200</td>
<td>1200</td>
<td>1200</td>
</tr>
<tr>
<td>9431</td>
<td>1200</td>
<td>1200</td>
<td>1200</td>
<td>1200</td>
<td>1200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOURCE DATA GENERATION RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN1</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>6.4</td>
</tr>
<tr>
<td>2.0</td>
</tr>
<tr>
<td>1.0</td>
</tr>
<tr>
<td>.625</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TRUNK TRANSMISSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST INTERVAL (SEC)</td>
</tr>
<tr>
<td>RUN1</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>1111</td>
</tr>
<tr>
<td>2211</td>
</tr>
<tr>
<td>3221</td>
</tr>
<tr>
<td>4311</td>
</tr>
<tr>
<td>5321</td>
</tr>
<tr>
<td>6331</td>
</tr>
<tr>
<td>7441</td>
</tr>
<tr>
<td>8421</td>
</tr>
<tr>
<td>9431</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEVICE BUFFER SIZE (BYTES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN1</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>2096</td>
</tr>
<tr>
<td>4000</td>
</tr>
<tr>
<td>6400</td>
</tr>
<tr>
<td>512</td>
</tr>
<tr>
<td>512</td>
</tr>
<tr>
<td>625</td>
</tr>
<tr>
<td>RUN</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
contention protocol to a slotted priority protocol as the trunk activity grows.

To demonstrate that the theoretical bounds from textbook theory are not easily applied to practical systems the data in Table 5-5 is presented. As may be observed the theoretical bounds are unrealistic and are never approached by the simulation model. It should be noted that the simulation model has been correlated against a few statistical measurements and does seem to be quite a valid model.
<table>
<thead>
<tr>
<th>SITUATION</th>
<th>S(AVERAGE)</th>
<th>U(AVERAGE)</th>
<th>% TIME TRUNK ACTIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>THEORETICAL MAXIMUMS</td>
<td>33.8 Mbps</td>
<td>.68%</td>
<td></td>
</tr>
<tr>
<td>RUN 1</td>
<td>.2978 Mbps</td>
<td>.59%</td>
<td>25.05%</td>
</tr>
<tr>
<td>RUN 2</td>
<td>.420 Mbps</td>
<td>.85%</td>
<td>27.84%</td>
</tr>
<tr>
<td>RUN 3</td>
<td>.529 Mbps</td>
<td>1.06%</td>
<td>51.12%</td>
</tr>
<tr>
<td>RUN 4</td>
<td>.761 Mbps</td>
<td>1.53%</td>
<td>67.16%</td>
</tr>
<tr>
<td>RUN 5</td>
<td>1.293 Mbps</td>
<td>2.58%</td>
<td>93.18%</td>
</tr>
</tbody>
</table>
6.0 REFERENCES AND BIBLIOGRAPHY


CHU77 Chu, W. C. 'A Hierarchical Routing and Flow Control Policy (HRFC) for Packet Switched Networks.' University of California, Los Angeles, Computer Science Department, August, 1977.


THOR83  Thornton, J. E., and Christensen, G. S. 'HYPERChannel Network Links.' Computer, September, 1983.

**APPENDIX I**

Summary of ISO-OSI Model of Distributed Networks

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Physical</td>
<td>Concerned with transmission of unstructured bit stream over physical link; involves such parameters as signal voltage swing and bit duration; deals with the mechanical, electrical, and procedural characteristics to establish, maintain, and deactivate the physical link. (RS-232-C, RS-499, X.21)</td>
</tr>
<tr>
<td>2. Data Link</td>
<td>Converts an unreliable transmission channel into a reliable one; sends blocks of data (frames) with checksum; uses error detection and frame acknowledgement (HDLC, SDLC, BiSync)</td>
</tr>
<tr>
<td>3. Network</td>
<td>Transmits packets of data through a network; packets may be independent (datagram) or traverse a preestablished network connection (virtual circuit); responsible for routing and congestion control. (X.25, layer 3)</td>
</tr>
<tr>
<td>4. Transport</td>
<td>Provides reliable, transparent transfer of data between end points; provides end-to-end error recovery and flow control.</td>
</tr>
<tr>
<td>5. Session</td>
<td>Provides means of establishing, managing, and terminating connection (session) between two on data to provide a standardized application two processes; may provide checkpoint and restart service, quarantine service.</td>
</tr>
<tr>
<td>6. Presentation</td>
<td>Performs generally useful transformations on data to provide a standardized application interface and to provide common communications services; examples: encryption, text compression, reformatting.</td>
</tr>
<tr>
<td>7. Application</td>
<td>Provides services to the users of the OSI environment; examples: transaction server, file transfer protocol, network management.</td>
</tr>
</tbody>
</table>

This summary taken from the book, Local Networks, An Introduction, by William Stallings [Stall84].
APPENDIX II

Simulation Code of HOSC HYPERchannel Network
Simulation of a HYPERchannel Network:

Model of the Huntsville Operations Support Center

John S. Paulin
Mississippi State University

A general software model was developed to simulate the performance characteristics of a steady-state system. This model was designed to allow the user to easily specify different physical configurations of the network resources at Marshall Space Flight Center's Huntsville Operation Support Center.

Simulation results in figures representing estimated throughput versus the offered channel load for the configured system. These results also include collision statistics and transmission attempts of each network resource. These parameters are used to indicate system performance and assist in overload conditions in the simulation configurations.

```
program "of", (Input, Output, Descriptrile, AuxOut);

const
numSources = 7;
maxTransfers = 5;
Tm = 6; (Times per page)
trunk = 11;
trunkSettup = 0.002. (Trunk set-up and release time)
trunkRate = 4; (Trunk in rate in bits/second)
```

```
numTrans = 4; (Max number trunks in network)
```
```plaintext
void monitor()
{
    int i;
    for (i = 0; i < numSources; i++)
    {
        if (sources[i].type == DEVICE)
        {
            deviceRecords[i].totalTrans = 0;
            deviceRecords[i].totalDelay = 0;
            deviceRecords[i].totalWait = 0;
        }
    }
}
```

procedure Initialize;
  / Perform initialization of run variables, initializations and 
  / sets of simulation run parameters. 
  / Invoked by: PROC_INIT
  / Calls to : No other routines 
  / Local variables affected: RUN, TALLY, COLLISION_TALLY, 
  / ARTIFICIAL, INITIAL, ACTIVE_INPUT, 
  / ART, MARK, ALL, INPUT, GUE.
begin (procedure INITIALIZE )
  / New run initializations: 
  for Cond1 := Normal to Other do Condition[Cond1] := Cond1;
  "run.all := true;
  u := 8745; ( u is seed for RNG )
  initiality := 0;
  collision.tally := 0;
  total.totals := 0;
  hit := 0;
  time.ctive := 0.0;
  "axial := 0; 
  writeln;
  writeln("Identification heading for run:");
  readln; writeln;
  writeln("maximum run time in secs?" );
  readln "axial" ; writeln;
  writeln("maximum successful transmissions?"");
  "axial := 0.0; writeln;
  writeln("need for random number generator?" );
  readln; "axial := 3; writeln;
  writeln("string/flag condition on" );
  readln "esp[i]" until [esp[i] < ]; readln;
  if (esp[i] = 0) or (esp[i] = 'Y') then 
  "run.all := true 
  else 
  "run.all := false; 
  writeln;
  writeln("input node: INTERACTIVE or (FILE)?");
  readln; if (esp[i] = 1) or (esp[i] = 'Y') then 
  "input_node := interactive 
  else 
  "input_node := file input;
  "output := false; writeln;
  processing := true; 
end (procedure INITIALIZE );
procedure CharacterizeNetwork;
  // process for network descriptive info.
  // Invoked by: PROGRESS
  // Calls to: $PROCEDURE DESCRIPT FILE
  // INITIALIZE MATRIX
  // Global variables affected: ADAPT[ ]

var
  i, j, k, l: Integer; // Loop counters
  Temp: Real;

procedure WriteDescriptive;
  // WRITES THE AUXILIARY INPUT FILE

var
  i, j, k, l: Integer; // Loop variables

begin // procedure WRITE
  // (OpenDescriptiveFile, "Descript", old);
  Rewrite(DescriptiveFile);
  Writeln(DescriptiveFile, NumOfTrunks);
  for i := 1 to NumOfTrunks do Writeln(DescriptiveFile, TrunkLength[i]);
  Writeln(DescriptiveFile, NumOfAdapters);
  for i := 1 to NumOfAdapters do
    begin
      with Adapter[i] do
        begin
          Writeln(DescriptiveFile, DistFromA1);
          Writeln(DescriptiveFile, PriorityDelay);
          Writeln(DescriptiveFile, "Begin individual device descriptions");
          for j := 1 to 4 do
            begin
              with Device[j] do
                begin
                  Writeln(DescriptiveFile, Open);
                  if not (Open) then
                    begin
                      Writeln(DescriptiveFile, 1);
                      Writeln(DescriptiveFile, 1);
                      Writeln(DescriptiveFile, Source);
                      for k := 1 to NumOfSources do
                        with Source[k] do
                          begin
                            Writeln(DescriptiveFile, Device);
                            Writeln(DescriptiveFile, 1);
                            Writeln(DescriptiveFile, Generate);
                            Writeln(DescriptiveFile, BufferSize);
                            for l := 1 to NumOfTrunks do
                              Writeln(DescriptiveFile, Trunk); end;
                          end;
                      end;
                    end;
                end;
            end;
          end;
        end;
    end;
  end;
end; // procedure WRITE

// write the probability matrix

end.
I I I I I * |
[Image 0x0 to 610x792]

```
software
begin
for k := 1 to num_trunks do
  read(descriptive_file, or(How, K));
  read(descriptive_file); + (for k := 1 to num_trunks do)
end; (file case)
end;

end; (procedure to initialize cumulative prob matrix)
begin (procedure CUMULATIVE PROB)
end; (in description of network by describing adapters)
case input mode of
interactive:
begin
  writeln("*************************************************************");
  writeln;
  writeln("DESCRIPTION OF NETWORK");
  writeln("DEVICE -- ADAPTER -- ADAPTER");
  writeln;
  writeln("Total number of trunks in network");
  writeln("Maximum number is", "numTrunks: 2, ");
  writeln("numTrunks");
  writeln;
  writeln("Length of trunks in ft (used for FNP delay)");
  writeln("for i := 1 to numTrunks do begin";
  writeln("Trunk ": i, " "");
  writeln("trunkLength(i)"));
  writeln;
  writeln("Number of Adapters in Network");
  writeln("numAdapters")
  writeln;
  writeln("for i := 1 to numAdapters do begin";
  writeln("with Adapter(i), Device(j) do begin";
  writeln("open := True;");
  writeln("open := False;");
  writeln("device status (OPEN or CLOSED)");
  write (or(Yes, No), " "");
  readln;
  if (open) = (or(Yes, No)) then open := false;
end;
```

```
for K := 1 to NumOfSources do
  with (Adapter[1], Device[j], Source[K]) do
  begin
    DistFromA := Adapter[1].DistFrom1;
    writeln;
    writeln( "data source ", K: 1, " 10" );
    writeln( 10 );
    Num := Dnum + 1;
    DevNum := Dnum * 1000 + 1 * 100 + J * 10 + K;
    writeln( "source ", K: 1 );
    writeln( "data gen rate (bytes/s)" );
    GenRate := GenRate + 3;
    writeln("buffer size in bytes:" );
    BufferSize := BufferSize * 8;
    TierRate := Adapter[1].Device[j].TierRate;
    if TierRate > C then
      Taintrvl := BufferSize / GenRate
    else
      Taintrvl := 1000;
    NextRx := Taintrvl;
    if NumOfTrunks < 1 then
      OnTrunk[1] := true
    else
      begin
        Trunk[1] := true;
        writeln("Trunks accessed by Source", K: 1, ": 1");
        writeln("Trunk", L: 1, ": 1");
        begin
          for L := 1 to NumOfTrunks do
            with (Adapter[1], Device[j], Source[L]) do
              begin
                writeln("Trunk", L: 1, ": 1");
                if Resp[1] = "Y" or (Resp[1] = "y") then
                  OnTrunk[L] := true
                else
                  OnTrunk[L] := false;
              end;
          end; (for L)
        end; (for K)
      end; (if not(On[2]))
    end; (for j := 1 to 4)
    end; (for i := 1 to num of adapters)
  end;
  end; (for i := 1 to num of adapters)
```

with Adapter[i], Device[j] do
  begin
    if Adapter[i].Device[j].Priority(i) then
      begin
        if Adapter[i].Device[j].Priority(i) = 'T' then
          if Adapter[i].Device[j].Priority(i) = 'F' then
            begin
              Adapter[i].Device[j].Open := false;
              Adapter[i].Device[j].Open := true;
            end
          end
        end
      end
  end
end
for k := 1 to NumOfSources do
  begin
    with Adapter[i], Device[j], Source[k] do
      begin
        if Adapter[i].Device[j].Priority(i) then
          begin
            if Adapter[i].Device[j].Priority(i) = 'T' then
              if Adapter[i].Device[j].Priority(i) = 'F' then
                begin
                  Adapter[i].Device[j].Open := false;
                  Adapter[i].Device[j].Open := true;
                end
            end
          end
      end
  end
for L := 1 to NumOfTrunks do
  begin
    repeat
      Adapter[i].Device[j].Priority(i) := read describeline;
      if Adapter[i].Device[j].Priority(i) = 'F' then
        begin
          Adapter[i].Device[j].Open := false;
          Adapter[i].Device[j].Open := true;
        end
    until Adapter[i].Device[j].Priority(i) = 'T'
  end
for L := 1 to NumOfTrunks do
  for k := 1 to NumOfSources do
    begin
      Adapter[i].Device[j].Priority(i) := read describeline;
      if Adapter[i].Device[j].Priority(i) = 'F' then
        begin
          Adapter[i].Device[j].Open := false;
          Adapter[i].Device[j].Open := true;
        end
    end

Local Area Network is a 50 MHz baseband communication system operating under a prioritized, CSMA-like protocol. A Pascal software model was developed to simulate the operating characteristics of a steady state system. The model was designed to allow the user to easily specify different configurations of the network resources. A "Small Source Flight Center"'s Huntsville Operation Supercenter was adapted for use in the simulation. The simulation results are presented in figures representing estimated throughput versus the offered channel load for the configured system. These results also include collision statistics and transmission attempts of each network resource. These parameters are used to indicate system performance and possible operational conditions in the simulation configurations. Let := Let + 1; { procedure 'PrintBanner' }

    var I, J, K, L, M, N: Integer;
    result: real;

begin ( procedure 'Print DESCRIPTION' )

    uw := 0;
    writeln('Auxout', 'Auxout', 'old', dw); WRITE('Auxout');
    'PrintBanner';
    for I := 1 to 'numAdapters' do
        begin
        writeln('Auxout');
```plaintext
procedure PrintCollisonStats('Attr, Dev: SourceDescriptions);
    "Prints stats on event of collision.

var
    source, rsource, adapt, 'adapt, 'dev, 'rdev: Integer;

begin
    source := 'attr.devium mod 10;
    dev := ('attr.devium div 10) div 10;
    'adapt := ('attr.devium mod 100) div 100;
    rsource := 'devr.devium mod 10;
    rdev := ('devr.devium div 10) div 10;
    'radapt := ('devr.devium mod 100) div 100;
    writeln(output);
    writeln(output, '** SUCCESSFUL TRANSMISSION -- COLLISION OCCURRED **
    writeln(output, 'line:, 'currentTime := 12, 77);
    writeln(output, 'transmitter: 'adapter, 'adapt := 1);
    writeln(output, 'device, 'dev := 17);
    writeln(output, 'source, 'source := 'source);
    writeln(output, 'receiver, 'receiver := 'adapter, 'radapt := 1);
    writeln(output, 'device, 'rdev := 17);
    writeln(output, 'source, 'rsource := 'rsource);
end; (PRINT COLLISION STATS)

procedure PrintLostStats('attr, Dev: SourceDescriptions);
    "Prints stats on event of collision.

var
    source, rsource, 'adapter, 'adapt, 'dev, 'rdev: Integer;

begin
    source := 'attr.devium mod 10;
    dev := ('attr.devium div 10) div 10;
    'adapt := ('attr.devium mod 100) div 100;
    rsource := 'devr.devium mod 10;
    rdev := ('devr.devium div 10) div 10;
    'radapt := ('devr.devium mod 100) div 100;
    writeln(output);
    writeln(output, '** SUCCESSFUL TRANSMISSION -- MESSAGE LOSS OCCURRED **
    writeln(output, 'line:, 'currentTime := 12, 77);
    writeln(output, 'transmitter: 'adapter, 'adapt := 1);
    writeln(output, 'device, 'dev := 17);
    writeln(output, 'source, 'source := 'source);
    writeln(output, 'receiver, 'receiver := 'adapter, 'radapt := 1);
    writeln(output, 'device, 'rdev := 17);
    writeln(output, 'source, 'rsource := 'rsource);
end; (PRINT LOST STATS)

procedure FindNext('attr: 'SourceDescriptions);
    "Detects the next transmitter by examining each device's time until next transmission.
    "if device is device with shortest time until next transmission.

var
    J, C: Integer;
```
function Uniform(var Seed: Integer): Real;
{ Calculates a pseudorandom number in range
  [0, 1] }
begin
const
  L = 79;
  M = 196;
  N = 1026;
begin
  Seed := ((Seed + 1) mod N);
  Uniform := Seed;
end;
end;

procedure Pick?(var Receiver: SourceDescriptions;
  Sender: SourceDescriptions);
{ Picks an eligible receiver for the current
  transmitter based on the normalized transmission
  probability matrix. }
begin
  var
    J, K, L, S, F, C: Integer;
    Found: Boolean;
    Pro: "real;
  begin
    procedure to pick a receiver
    Begin
      Pro := Uniform[ M ];
      J := Sender.CumSum div 1026;
      while [ J < Pro ] and ( J < B 'num ) do
        begin
          J := J + 1
        end;
        while [ J > num ] and ( not Found ) do
          begin
            Adapter := Device[J].Source;
            if [ Found ] then
              begin
                Next := Adapter[ J ];
              end;
        end;
  end;
end;
function Collision: Boolean;
  (determines the chance of a collision occurring)
  for a given transmission attempt. Collision
  occurs if two transmitters attempt to transmit
  within y time seconds of each other.

var
  t, j, k: Integer;

begin
  Collision := False;
  for i := 1 to numOfAdapters do
    for j := 1 to 6 do
      with Adapter[j].Device[i] do
        if not Open then
          for k := 1 to numOfSources do
            with Adapter[i].Device[j].Source[k] do
              if Transmitter.DevNum < DevNum then
                if Addr.Transmitter.Next * Addr.Transmitter.Next then
                  Collision := True;
                  Collisions := Collisions + 1;
                  if Print then PrintCollisionStats(Transmitter, Receiver);
                  end;
  end;  \Collision DETECTION\)

procedure UpdateClocks(ford: Clockfnd);  
  (updates both the system clocks and device clocks whenever a system action has occurred.

var
  a, b, c, d, e, f, g, h, i, j, k: Integer;
  DelayTime, EntryTime, Time, t: Real;

{\ Action TransferTime (Sender, Receiver, SourceDescriptions): \ Real;
  \ calculates time required to transmit a message \ from a sender to a receiver.
  \ This overloads the includes the fixed delay for each \ Transmitter which is length of time required by a sending \ device to receive a message from
BEGIN
  IF (Sender.TierRate < SlowRate) THEN SlowRate := Receiver.TierRate;
  Separation := Abs(Sender.DistFromA1 - Receiver.DistFromA1); 
  PktSize := Round(Sender.BufferSize / 2048);
  IF PktSize = 0 THEN
    Trunk_Overhead := 1;
  ELSE
    PktSize := 1
    TrunkRate := TrunkRate + Trunk_Overhead + Separation / 4096
  END;
  TransferTime := T;
END;

BEGIN
  / Is transmitter adapter no., d is device no., s is source number
  T = (Limiter.Devium mod 1920) div 1920;
  r = (Limiter.Devium mod 1920) div 1920;
  T0 := (Limiter.Devium mod 1920) div 1920;
  T1 := (Limiter.Devium mod 1920) div 1920;
  T2 := (Limiter.Devium mod 1920) div 1920;
  T3 := (Limiter.Devium mod 1920) div 1920;
  case Cont of
  normal:
    BEGIN
      Time := TransferTime(Limiter, Receiver);
      PktSize := PktSize * Limiter.BufferSize;
      / Update system clock
      CurrentTime := CurrentTime + Time;
      Tally := Tally + 1;
      IF PrintStats THEN PrintStats(Limiter, Receiver);
      / Update each device clock
      (and determine if other devices forced to wait)
      FOR j = 1 TO NumAdapters DO
        FOR k = 1 TO 6 DO
          WITH Adapter[i].Device[j] TO
          IF (Tally = 1) AND (Tally = 1) AND (Tally = 1) THEN
            FOR K := 1 TO NumSources DO
              WITH Adapter[i].Device[j].Source[k] DO
                IF Tally = K THEN
                BEGIN
                  PktSize := CurrentTime + Time;
                  DelayTime := Adapter[i].PriorityDelay + 
                    Adapter[i].NoDelay + Adapter[i].FixedDelay;
                  Variable := CurrentTime + DelayTime;
                  WaitCount := WaitCount + 1;
                Uptime := Uptime + 1;
                WaitTime := WaitTime + DelayTime + Time;
                IF PrintStats THEN PrintStats(Adapter[i].Device[j].Source[k]);
                END;
            END;
          END;
        END;
      END;
    END;
  Cases: 
    BEGIN
      Time := TransferTime(Limiter, Receiver);
      PktSize := PktSize * Limiter.BufferSize;
      / Update system clock
      CurrentTime := CurrentTime + Time;
      Tally := Tally + 1;
      IF PrintStats THEN PrintStats(Limiter, Receiver);
      / Update each device clock
      (and determine if other devices forced to wait)
      FOR j = 1 TO NumAdapters DO
        FOR k = 1 TO 6 DO
          WITH Adapter[i].Device[j] TO
          IF (Tally = 1) AND (Tally = 1) AND (Tally = 1) THEN
            FOR K := 1 TO NumSources DO
              WITH Adapter[i].Device[j].Source[k] DO
                IF Tally = K THEN
                BEGIN
                  PktSize := CurrentTime + Time;
                  DelayTime := Adapter[i].PriorityDelay + 
                    Adapter[i].NoDelay + Adapter[i].FixedDelay;
                  Variable := CurrentTime + DelayTime;
                  WaitCount := WaitCount + 1;
                Uptime := Uptime + 1;
                WaitTime := WaitTime + DelayTime + Time;
                IF PrintStats THEN PrintStats(Adapter[i].Device[j].Source[k]);
                END;
            END;
          END;
        END;
      END;
    END;
begin
  with Adapter[T1].Device[T1].Source[T1] do
  begin
    if RetryTally > 25$ then ( message will be lost )
    begin
      retryTally := retryTally + 1;
      TotalLost := TotalLost + 1;
      if PrintAll then PrintLostStats(Tmitr, Device); NextTx := CurrentTime + TInterval;
    retryTally := 0;
    end; ( message lost )
    retryDelay := UniformU() * Adapter[T1].FixedDelay
    retryDelay := retryDelay + adapter[T1].fixedDelay;
    NextTx := NextTx + retryDelay;
    CollDet := CollDet + 1;
    CollTime := CollTime + retryDelay;
    retryTally := retryTally + 1;
  end; ( with Adapter[T1].Device[T1] )
end; ( collision case )
end; ( case )
end; ( update clocks )

procedure PrintNetworkStats;
( print cumulative statistics of network at end )
( of simulation )
begin
  var PercentActive: Real;
  : Integer;
begin
  percentActive := 0.1 * 100;
  Default := TimeActive / CurrentTime * 100.0;
  writeln(AuxOut, " ");
  writeln(AuxOut, "TimeActive / CurrentTime * 100.0:
  writeln(AuxOut, "Current time: ");
  writeln(AuxOut, "Current time: ");
  writeln(AuxOut, "Successful transmissions : ");
  writeln(AuxOut, "Total attempts : ");
  writeln(AuxOut, "Total attempts : ");
  writeln(AuxOut, "Total time active : ");
  writeln(AuxOut, "Total time active : ");
end; ( update clocks )

procedure PrintNetworkStats;
( print cumulative statistics of network at end )
( of simulation )
begin
  var PercentActive: Real; 1
  : Integer;
begin
  percentActive := 0.1 * 100;
  Default := TimeActive / CurrentTime * 100.0;
  writeln(AuxOut, " ");
  writeln(AuxOut, "TimeActive / CurrentTime * 100.0:
  writeln(AuxOut, "Current time: ");
  writeln(AuxOut, "Current time: ");
  writeln(AuxOut, "Successful transmissions : ");
  writeln(AuxOut, "Total attempts : ");
  writeln(AuxOut, "Total attempts : ");
  writeln(AuxOut, "Total time active : ");
  writeln(AuxOut, "Total time active : ");
end; ( update clocks )
/* Calculate and print percent summaries */

NewAddr;

Write(AuxOut); //PERCENT
Write(AuxOut); //DEVICE ACTIVITY SUMMARIES
Write(AuxOut); //COLLISION
Write(AuxOut); //TRUNK
Write(AuxOut); //TRUNK
Write(AuxOut); //DEVICES

for i := 1 to NumAdapters do
  for j := 1 to 4 do
    with Adapter[i], Device[j] do
      if not Open then
        for k := 1 to NumSources do
          with Adapter[i], Device[j], Source[k] do
            begin
            DevActiveTime := TxAkt + WaitTime + CottAct + XaTime;
            if DevActiveTime > 0.0 then
              begin
                DevActTr := DevActiveTime / TimeActive * 100.0;
                TCollTr := TxAkt / TimeActive * 100.0;
                TWaitTr := WaitTime / TimeActive * 100.0;
                TCollDev := CottAct / TimeActive * 100.0;
                if (DevActTr = 0.0) or (TCollTr = 0.0) then
                  TextDev := 0;
                else
                  TextDev := TWaitTr / TxAkt * Txintrvl * 100.0;
                if (CollAct = 4) or (TCollDev = 0.0) then
                  TextDev := 0;
                else
                  TextDev := TCollDev / TCollAct / Txintrvl * 100.0;
                end; // "Dev_Act_Tm
            end;
          end;
        end;
      end;
    end;
  end;
end;
end.

115

/* LE */
begin ( main program )

( Network cold start conditions )
Initialize;
CharacterizeNetwork;
PrintNetDescription;
FindNextInitr);
CurrentTime := Tinitr,NextTx;
( Network steady state operation )
repeat ( until time exceeds max time )
  nick ( cvr, initr );
  ( Calculate a collision probability )
  if not ACollision then
    begin
      UpdateClocks ( ConditionNormal );
      FindNext( initr );
      CurrentTime := Tinitr,NextTx;
    end;
  else
    begin
      UpdateClocks ( ConditionCollision );
      FindNext ( initr );
      CurrentTime := Tinitr,NextTx;
    end;
  until ( CurrentTime >= maxTime ) or ( TTotal >= MaxTx );
  TtotalAttempts := TTotal + CollisionAttempts;
  PrintNetworkStats;
  ActivitySummary;
  Write ( Output legitimate stuff as input );
end. ( main program )

compilation complete - no errors found.