Rochester Institute of Technology
RIT Digital Institutional Repository

Theses

1989

Congestion control schemes and their effect on a hypothetical network

Jeffrey S. Reifsteck

Follow this and additional works at: https://repository.rit.edu/theses

Recommended Citation
Reifsteck, Jeffrey S., "Congestion control schemes and their effect on a hypothetical network" (1989). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

ROCHESTER INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND TECHNOLOGY

Congestion Control Schemes and Their Effect on a Hypothetical Network

by
Jeffrey S. Reifsteck

A thesis, submitted to
The Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science.

Chairperson: _____ 10/30/89
Margaret Reek — Date

Committee Members:

_____ 10/30/89
Dr. A. Kitchen — Date

_____ 10/30/89
Dr. J. Heliotis — Date

Authorization Statement for the thesis:

"Congestion Control Schemes and Their Effect on a Hypothetical Network"

I, Jeffrey S. Reifsteck would prefer to be contacted each time a request for reproduction is made. It is not necessary to await for my approval before copies can be made. Any reproduction will not be for commercial use or profit. I can be reached at the following address.

6215 Rime Village Dr. #103
Huntsville, Alabama 35806

Jeffrey S. Reifsteck

Date 10/30/89

ACKNOWLEDGEMENTS

There were many people without whose help this thesis would not have been possible. I'd like to thank my Chairperson Margaret Reek without whom this paper would have never been finished; Lorna for her help and understanding; Chuck for his expertise and timely humor. And to my parents who have put up with so much for so long.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE PAGE

LIST OF TABLES

TABLE PAGE

ABSTRACT

The purpose of this thesis is to allow data communication students to experiment with various congestion control schemes to see how they will affect a simulated network. Using various parameters, the user can design a full duplex, packet switched, point-to-point virtual circuit network. Then, using this network the student can choose a congestion control scheme and run the simulation. After each run the parameter(s), the control scheme, or both, can be changed and the simulation rerun to see what the changes will do to network performance.

CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 INTRODUCTION

The dictionary defines congestion as an overfilled or overcrowded situation. An example that is often used is of trucks going into a tunnel. Just as traffic on the nation's highways can become congested at rush hour, so can computer networks. How a network will handle the problem of congestion is a very important design consideration and one that the network developers must keep in mind.

This thesis serves as an introduction to a simulation program that will give the user some "hands-on" experience in designing a network and then running it with different congestion control schemes to determine the network's optimum performance. Users will design their own network that will simulate a full-duplex, packet-switched, point-to-point, virtual circuit network. Then they will choose one of four control schemes used to regulate congestion. Various parameters of the individualized networks can be changed, as can the control schemes, to allow users to determine the best method to obtain optimum throughput and minimum delay from their network.

The user can choose the best congestion control scheme given a set of parameters or given a certain control scheme adjust the parameters to yield the greatest throughput with a minimum of delay. After each simulation run the user can change any of the input parameters and/or the congestion control scheme and rerun the simulation. At the end of each run, several output files are generated to demonstrate what has occurred during the execution of the simulation.

Briefly, in order for a packet-switched network to work, it needs to have some type of congestion control. As packets are generated and sent over a network, the performance may degrade due to lack of buffer space. Most networks cannot allow an arbitrarily high number of buffers because that would consume too much of the computer's memory and cause slowdowns or stoppages in other processes. When the number of packets reaches a certain critical point, the network will slow down and eventually come to a halt if more and more packets are introduced. To alleviate this problem, network programmers use

congestion control schemes. There are various methods that they employ to solve this problem. This simulation will show how some of these can affect a network.

1.2 PROBLEM STATEMENT

For students who are studying data communication and the concepts of network congestion control for the first time, conceptualization of the different congestion control schemes and how these schemes can affect a network may come easily. For other students it may be more difficult; but whatever the case, there is no substitute for experience. Having a hypothetical network in which to manipulate various parameters, in addition to changing congestion control schemes (to observe how they can affect the operation and effectiveness of the network), should be extremely helpful for any data communication student. This simulation presents an opportunity for hands-on experimentation leading to a fuller understanding of the subject. This program could be used as a teaching tool, as a lab, or an assignment. A self-explanatory Users Manual is available for anyone wishing to use the simulation.

1.3 BACKGROUND

1.3.1 CAUSES OF CONGESTION

Congestion is caused by having more packets in the network than it can handle efficiently. This occurs when the overhead of the intermediate nodes cause queues to build up in those nodes, even if there is excess line capacity. Another way that congestion is caused is if the incoming traffic of an intermediate node exceeds that of its output lines. Lastly, congestion tends to feed on itself. As congestion causes the network to slow down, queues become full and senders are required to retransmit packets into an already congested network because either the packets have timed out, or the packets are being discarded by an already full queue.

1.3.2 WHAT IS CONGESTION CONTROL

In a packet-switched network, packets are generated by a sending, or source, node and sent over the network to a receiving, or destination, node. Congestion occurs when there

are too many packets in the network for it to function efficiently. The delays become longer and the throughput drops off. An often used analogy compares network congestion to that of a city's streets. Packets are like the cars on a street and the nodes are the street intersections with stoplights. As rush hour approaches, which corresponds to a burst in computer network traffic, more and more cars pour out onto the street. The lights do not move cars through any faster, and it takes longer and longer for the backed up traffic to get through those lights.

Now picture the worst case. Drivers have become impatient; and as the light turns red, they try and squeeze into the intersection. The problem is that they cannot drive all the way through because other drivers at other intersections are also trying to squeeze through. This results in the blocking of traffic at all intersections, or gridlock. City streets have gridlocks; networks have deadlocks. Both result in a complete shutdown of traffic.

To better understand this situation in a network, picture a two-node network (see Figure 1). Each node has five free buffers to handle both input and output lines. Node A has all five buffers lined up trying to send to node B. The problem is that node B has its five buffers lined up trying to send to A. Each is waiting to send to the other but cannot do so successfully because there are no free buffers to accept the incoming packet. This condition is referred to as direct deadlock.

Indirect deadlock is much the same. This time picture a four-node network, each node having six free buffers (see Figure 2). Node A has all six lined up to send to B, which has its six waiting to send to C, which is waiting to send to D, which is waiting to send to A. All four nodes are deadlocked in this situation.

Congestion control schemes are used to avoid deadlock and to minimize the congestion without extracting a great cost to the network in terms of speed or reliability.

1.3.3 TECHNIQUES OF CONGESTION CONTROL

There are really two different kinds of technique used in congestion control. The first technique does not allow congestion to form at all. An extreme example of this is to consider a network allowing only an extremely low number of packets in transit at any point in time (for example, one packet for every node). Generally, the preallocation of resources falls into this category, whether it is buffers statically allocated to a virtual circuit, or dynamic buffer allocation as the circuit is set up.

Direct Deadlock

No free buffers in either node

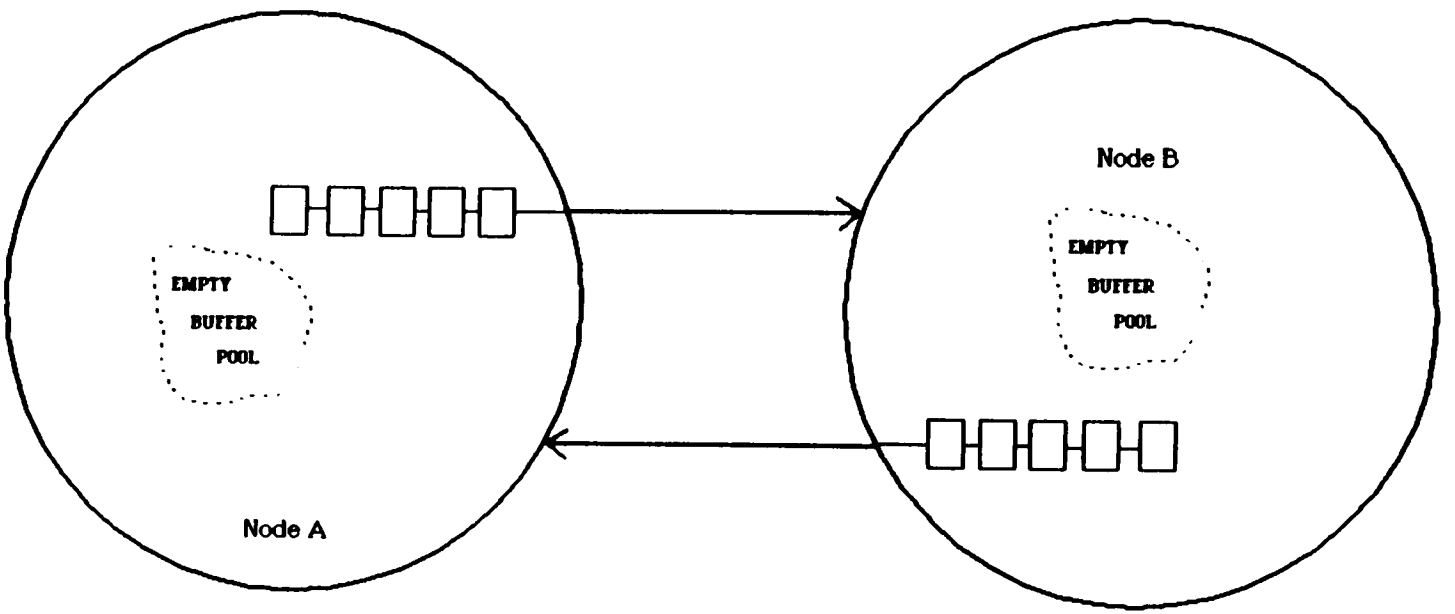
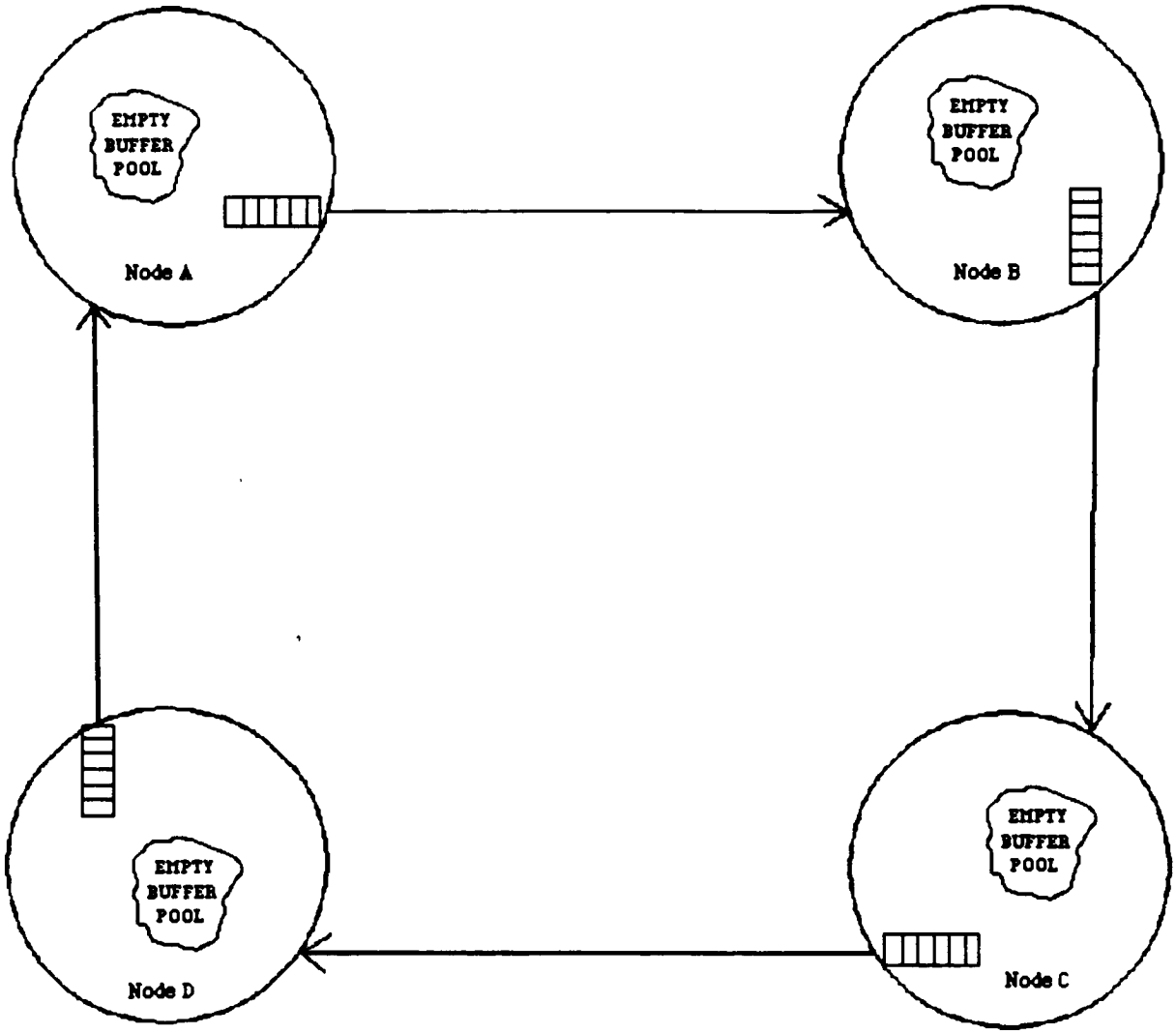


Figure 1



Example of Indirect Deadlock

Figure 2

In the other type, the congestion control scheme must perform two tasks: It must detect when congestion is occurring, and then it must slow down the input of packets into the network according to some technique. Some of the techniques used in networks are listed and explained below. There are also many variations or other techniques used to tailor congestion control to a particular network. Of the congestion control schemes mentioned below, the schemes used in this thesis simulation are ones using tokens, choke packets, and flow balancing.

PREALLOCATION OF BUFFERS - As a virtual circuit is being set up and a route established through a network, the set-up packet of each individual virtual circuit, at each intermediate node, requests a buffer space reserved just for this virtual circuit. Therefore an incoming packet will always have a buffer waiting for it. If a stop-and-wait protocol is used, only one buffer needs to be reserved but a sliding window protocol would need as many buffers as is the window size of the protocol. If there are not enough buffers for the virtual circuit, then either the intermediate node must notify the sender that this node is full, or the set-up packet must find another route around the busy node.

A variation of this technique is to have dedicated buffers for each virtual circuit that may come through a node. An illustration of this would be if each node in a five-node network could have only one outstanding message at any one time. If each node has a virtual circuit table that can hold five entries and five free buffers, then assuming a stop-and-wait protocol, they are assured of always being able to have a free buffer available to them. This ties up several resources that may not be used and is not efficient in that sense, but is effective when low delay is necessary.

PACKET DISCARDING - This scheme is fairly simple. If a packet is transmitted to a node and that node has no buffers to store it in, then the packet is discarded. Eventually the sender will time out and retransmit the packet, and hopefully there will be buffers available at that time. The major issue is how the nodes will use the buffers; whether or not there will be a maximum and/or minimum per output line, as well as how many buffers will be contained in each node. How the issue is decided will depend on the individual network and its designers.

TOKEN CONGESTION CONTROL In particular, this scheme is based on the idea of having a fixed number of packets in the network at any one time. "Permits" or tokens are initially spread throughout the network, several at each of the nodes. As a packet is generated, it must wait until a permit is available and only then is it allowed to travel

over the network to its destination. Once there, the receiving node takes the permit and tries to incorporate it into the node's own stock of permits as long as the node will not exceed its permit limit. If that permit causes the node to exceed this limit, then it must be sent along to some other node. This ensures that one node cannot dominate the network while other nodes starve. This type of control scheme is sometimes called isarithmic congestion control but will be referred throughout this thesis as "token" congestion control.

FLOW CONTROL - This method takes an end-to-end network control scheme and attempts to apply it to the intermediate nodes of a network. There are a couple of typical flow control schemes used to control the traffic rate between the sender and the receiver. The first is called the stop-and-wait protocol. In this protocol, the sender waits for explicit permission to send the next part of a message. Typically, the sender will send one packet and wait for an acknowledgement from the receiver before it sends the next packet. The second scheme, which is really an expansion on the stop-and-wait protocol, is called a sliding window protocol. With this protocol, after the virtual circuit is set up, the sender can transmit a certain number of packets; that number is derived beforehand by the network designers and is called the window size. The sender then waits for an acknowledgement from the receiver and then will send another window of packets and waits again. The flow control scheme is able to control the congestion along a particular circuit but has a hard time doing so for the entire network. Moreover, the scheme is usually used in conjunction with various control schemes at other levels.

CHOKE PACKETS Each node in the network monitors its output lines to determine how well the lines are utilized. If the utilization level reaches a certain point, then a packet leaving on that outgoing line would have a bit turned on, indicating that the output line is becoming congested. Each node checks incoming packets; and if one is discovered to be coming from a congested output line, then the node notes the sender and where the packet is destined. Next, the node turns on another bit in the packet to let other nodes know that the problem has already been noted and transmits back to the sender a packet indicating the sender must slow down its traffic to the congested area. The original packet with the bits turned on is sent along to the next node in the virtual circuit route.

To illustrate this point, suppose node A's output line to node B is becoming congested. Node A turns a bit on in the next message packet going to node B. Node B examines the incoming packet, sees that node A is becoming congested and toggles another

bit in the same packet to indicate node B is taking care of the congestion. Node B determines the sender and receiver of the packet and sends a choke packet back to the sender.

The sender has two distinct time intervals it must go through after it has received a choke packet. These are the ignore time period and the listen time period. After the sender receives a choke packet for a particular destination, it will then ignore all other choke packets arriving with the same destination for a period of time. Once the ignore time period is over, the sender will then listen for a set time for other choke packets with the same destination. If the sender receives another choke packet in the listen period, the sending rate is reduced further to the designated node. Should the sender not receive another choke packet during this listen period, the traffic rate is increased all the way back to the original level.

FLOW BALANCING CONGESTION CONTROL SCHEME - This is a variation of an end-to-end window flow control scheme that has been modified so that congestion is managed at intermediate nodes. Flow balancing is achieved by forcing the flow of packets on the incoming virtual circuits to be equal to the output channels through which the virtual circuits journey. In an initial call request, a sending node gives the window size of the message (number of packets) that will be sent. The destination node will generate a call accept packet having the window size of the call request in it. As the call accept packet enters intermediate nodes, the packet is put into a special hold queue. Each time a packet is successfully transmitted from this intermediate node, a credit is generated and a count is increased by one. When the count is equal to the window size of the call accept in the front of the queue, the call accept can now be sent along to the sending node. The control scheme has the benefit of slowing down only the virtual circuit that is causing the congestion at the node.

INPUT BUFFER LIMITS - This scheme limits the number of buffers used for incoming packets from the host. Generally, packets arriving at their destination get first priority, next, packets that are in transit between source and sink, and then those message packets being generated from the host. Actually, transit packets are allowed to take up all the free buffers if needed, while input packets from the host cannot.

1.4 PREVIOUS WORK

1.4.1 MODELING

It is an expensive mistake to set up a network, only to find that it does not work as efficiently as originally intended. To solve such a problem, specialists devise simulations to run a proposed model before the actual network is constructed. Many bugs will be discovered by running the model through a simulation before actual monies are committed to the project. Mistakes can be corrected or the networking idea can be modified or terminated.

Various papers have been published recently describing a simulation or modeling technique pertaining to networks. In 1986, a "Specification and Description Language" was used as a simulation technique to model communication processes as finite state machines. The authors were interested in call setup time and mean buffer length [Fischer 86] as they reacted to offered loads on a system. Another paper [Woodside 86] discussed a model using three different buffer-handling schemes to test an "execution workload model." This execution workload model can be used as input to a simulation and is, in essence a message generator. A third paper [Zinky 86] discusses ways of using models to monitor and compare the day-to-day operations of existing networks. In the author's words a model can be used for "detection of faults, observation of network behavior, problem resolution, and to take a global view of network interactions." All three of these papers indicate that simulation and modeling is still a valid, cheap, and proven method to analyze certain aspects of computer networks. Simulation and modeling can work with congestion control schemes as well.

Network designers use a particular congestion control scheme best suited for the network. Most control schemes are modified, enabling the network to be as efficient as possible. Parameters are adjusted; schemes are merged to form new types of control schemes; and different networks use control schemes at different levels. Also, many networks do not use the International Standards Organization (ISO) model, so the levels do not correspond to each other.

Several papers describing simulations of different congestion control techniques exist and are discussed below; however, after an extensive library search, not one was written on a simulation comparing several congestion techniques in a single model. The papers on the simulation of congestion control schemes primarily concentrate on describing just one form of control. This thesis will be a simulation comparing three different congestion control techniques, as well as having the same network demonstrate what happens without a control scheme.

1.4.2 ASSUMPTIONS MADE IN PREVIOUS NETWORK SIMULATIONS

The Input Buffer Limit (IBL) [Lam 79] and the Flow Balancing Congestion Control Scheme (FBCCS) [Kermani 82] have been simulated so these will be used as examples of what other authors have used as assumptions in their simulations. FBCCS and IBL both assume that packets arrive as a Poisson process. That is the first and last assumption that both share. FBCCS examines just one node's input and output channels. Each of the virtual circuits passing through the node has a finite number of assigned buffers; and when those buffers are full, the messages are lost. Furthermore, each message occupies a single buffer; and once a message is accepted by the node, its delivery is guaranteed.

IBL, on the other hand, examines a four-node, completely connected, homogeneous network. A major premise within IBL's theoretical framework concerns a single congested node: the assumption is that its neighbors are similarly congested. Likewise, all messages are the same length. Other network parameters, such as processor speed, channel speed, average time out, to name a few, are given as constants.

Another network simulation study [Edge 79], assumes that packets are transferred between levels in an instant and the destination process is always able to accept the packets. Unlike the FBCCS and IBL, this study demonstrates the message arrival rate as a constant. This proves that what is assumed in a network simulation can differ widely between studies and is individually tailored for the simulation or the network under study.

CHAPTER 2

PROJECT OVERVIEW

2.1 INTRODUCTION

The purpose of this simulation is to provide a tool to allow users a chance to design their own network and learn more about congestion control schemes without going through the trouble of constructing the actual mechanisms needed to perform such an operation. The design had to be such that the users would have as much flexibility as possible in designing their networks, have as few limitations as possible, and have the ease for simple change and modification. At the same time, the complexity had to be limited so that the whole simulation would not be overwhelming and too difficult and unwieldy a tool to use.

To realize the greatest benefit from the hands-on experience in working with this simulation and its congestion control schemes, students should have a basic understanding of computer networks and how these networks function. General topics that will aid students are network topology, virtual circuits, routing, and congestion control, in addition to being familiar with the ISO Reference Model. An excellent resource for obtaining the necessary background information is Andrew S. Tanenbaum's, Computer Networks [88], chapters one through five. Material contained in those five chapters clarify problems and challenges that surfaced in devising the simulation. Below is a brief synopsis on how the simulation was devised, its assumptions, simplifications, and protocols. A more extensive explanation of each follows in other sections of this thesis.

This simulation was constructed to have interlinked nodes, each having a message generator, or host, producing and receiving the actual messages. Each user can choose the topology of the network, the sending rates of the nodes, the percent of packets lost, size of buffer pool, time a host will wait before retransmission of packets, length of output queues, and the congestion control scheme. There are three control schemes and a base scheme that can be utilized with this simulation. First, there is control scheme 0 which is the network running with no congestion scheme. This is used as a base standard or "control run." Control scheme 1 is a token control scheme and scheme 2 uses choke packets. Scheme 3 is a flow balancing scheme where the input to a node is limited to equal the output.

The choke packet scheme and the token scheme were picked because they are well known and information about them is readily available. It is very important to determine a network's performance without congestion control so such a simulation was devised to be used as the "control." The flow balancing scheme was chosen because of its unique method of handling congestion control.

When someone decides to construct a simulation, often it is to test a concept or a new idea. Usually the simulation is made as simple as possible while still being able to prove its point. Assumptions are made with this in mind. Most models are based upon fundamental criteria and assumptions, all of which potential users must know before running or testing the model. While this simulation demonstrates various control schemes, it also employs other designers' assumptions. Both the designers' and thesis assumptions are described below.

2.1.1 DESIGN CONSIDERATIONS

2.1.1.1 Simplifications and Assumptions

It is important to keep in mind that this simulation is performing a relative comparison and not a quantitative one; therefore these assumptions do not adversely affect the simulation's relevance because each congestion control scheme operates on the same underlying network. It is being assumed that input buffers, output buffers, and those buffers in the free buffer pool hold only one packet at a time. The message generator for each node uses a Poisson process to determine the next clocktick when the host will want to send a message. All packets are transmitted instantaneously from host to node, node to node, and node to host. The host-node link is error free. The network clock is synchronized so that each node will send whatever is in its output buffers simultaneously. By synchronizing the clocks this way the the simulation is able to emulate parallel processes in a single serial machine. Packets are able to be time-stamped with the current clocktick for later reference by the hosts. This process will be explained in a later section.

The user specifies a percentage of packets destined to be lost during the simulation. Two special types of packets are protected from random packet loss: the packet to close the virtual circuit (the "closevc" packet) and, in the token scheme, any message packet carrying an extra token. When the process determining which packets become lost discovers a protected packet, which should be lost according to a random number generation, the discard is not performed. Other assumptions are that all queues have the same maximum length, and rejected requests are automatically resent after a period of waiting.

There are three different processes where connect requests can be generated and sent from a host. A request may be a new one, one that has not received a call accept within the allotted time and has timed out, or a request that has been rejected from some node along the virtual circuit that it was trying to establish. Requests have the highest priority in a host and are sent down to its node before a regular message packet or even a closevc packet. The closevc packet is a special message packet and is therefore given a higher priority then the regular ones.

When using the token scheme, the extra tokens are assigned a random destination and then are held in queues waiting for message packets going to that same node. They then piggyback on these packets to the mutual destination.

2.1.1.2 Sending and Receiving Protocols Used in This Simulation

The sending and receiving protocols are described below but are more fully explained in the implementation section that follows.

2.1.1.2.1 Sending Host to its Node

There is a "host" process for each node that generates the messages for this simulation. The host is in charge of determining when the next message will be sent, the message size, and the receiving node number.

Once a virtual circuit has been established, the message packets can then be sent one at a time; and once a message is started, the host will keep sending the packets from that message until it is finished. The exceptions to this are the requests and the closevc packets which have a higher priority and can preempt a message packet if the high priority packets are waiting to be sent. Once the node has finished sending a message, it will start on the next message, assuming that one is waiting to be sent. The host does not have to wait for the last acknowledgement of message 1 before sending the first packet of message 2. After it has received the last acknowledgement from the sink node, then and only then will it send a closevc packet to shutdown the virtual circuit.

Acknowledgements arriving out of order will cause a retransmission of packets starting from the last correctly received acknowledgement through to the end of the message.

2.1.1.2.2 Node to Node

Each node has an output buffer with an associated output queue. Packets are sent in the order that they arrive at the output queue. The packet next in line is loaded into the output buffer and sent to the connected input buffer of the node that is connected to that particular output buffer. That node's input buffer will be processed on the next clocktick.

2.1.1.2.3 Node to Receiving Host

The receiving node sends acknowledgements back to the sender. It does not care about packet ordering but lets the sender worry about proper message delivery and any retransmissions that need to be done. The packets are removed from the input buffer, the contents are noted and the packet disappears from the simulation. It is assumed that the packets are stored somewhere until the entire message has been received in order and can be sent on to the host.

2.2 INTRODUCTION TO THE USER'S INTERFACE

The project design enables users to create their own networks through a five-step process. Refer to the Users Manual for details on correct procedures for data input. Implementing these issues is discussed in the Project Implementation chapter. Below is a list of what needs to be considered by the user in order to construct the network and run the simulation. To build a network, the user must incorporate five design elements. They are as follows:

1. Design of the Actual Network
2. Design of the Nodes
3. Design of the Host Processes
4. Design of the Actual Simulation
5. Design of Printouts

Each of these will be explained later on in this chapter.

2.3 USER INPUTS

The first step is creating a data file that is read upon initialization of the simulation. Below are the variables the user will have to provide. Examples of the correct method of input and installation of these values is demonstrated in the Users Manual. Choosing the parameters and variables is the user's responsibility in designing the network and how the simulation will run. These variables have been broken down into four design categories.

2.4 DESIGN CATEGORIES AND EXPLANATION OF USER INPUT VARIABLES

2.4.1 DESIGNING THE NETWORK

The first issue that arises when one wants to design a network is how it should be laid out. The physical layout of a network is its topology: its size, what nodes are adjacent to one another, and the maximum number of connections per node. The best approach is to draw the desired network and then create the data file from it. Moreover, the number of packets lost or discarded will come under this heading. This simulation has been constructed to give the user as much flexibility as possible, but to do so necessitates that the user take a greater responsibility in designing the network.

2.4.1.1 Size

The number of nodes that the network can contain is limited to be between 3 and 50 nodes. The size of the network has to be directly related to which printout option the user has chosen; these are described in more detail in section 2.4.5. The user can run this simulation and obtain outputs along one of two paths. The first path will produce large printouts showing the step-by-step progression of what is happening in each node during each clocktick. Since this produces a large output file for each node and due to the fact that the Unix operating system only allows 17 files to be opened at any one time, the number of nodes is limited to 10 when using the printout scheme. Since so much time and space is used to write to the output files, users should probably keep the size to three or four nodes and the printing time to under 100 clockticks. The clockticks do not have to be the first 100 but can be 200-299, 350-449, and so forth.

The second output option is a summation of what transpired in the network during the run and not a step-by-step description of the simulation. This allows the user to

investigate complex networks of up to fifty nodes for a longer period of time. The printouts are oriented towards monitoring the throughput and delays.

2.4.1.2 Adjacency

The adjacency matrix table the user constructs informs the computer how the network is configured. It shows which nodes are connected to what other ones. The adjacency matrix table will be run through a consistency checker to ensure no mistakes were made in building the table. If node 0 is connected to nodes 1 and 2, then this checker will verify that nodes 1 and 2 are also connected to node 0. If an error is found, the simulation will write an error message out to the ERROR file and quit. The user can then make the necessary changes to the data file and rerun the simulation.

2.4.1.3 The Maximum Connectivity of the Network

The maximum connectivity of a network is the number of nodes to which one node can be physically connected. In this simulation that number can be has high as five and as low as one.

2.4.1.4 Percent of Packets Lost

In all networks some packets are lost due to line degradation or some other means besides congestion. The user should decide what the percentage of lost packets should be for the simulation. A value less than one percent is a realistic target.

2.4.1.5 Packets Discarded

In this simulation node 1 cannot send a packet from its output buffer to node 2 until node 2's input buffer is free. The simulation checks a flag to discover if that input buffer is empty or has a packet frozen in it. There are two methods provided for controlling activities when a message packet arrives at a full node which cannot process the packet at that time. This could occur if there are no free buffers available for the packet in the free buffer pool or the correct output queue is full. The first method simply discards the packet and lets the sending node worry about the retransmission. The second method freezes the packet at the input buffer, sets the "input buffer is frozen" flag, effectively cutting off any further inputs from the connecting node until the packet can be forwarded.

The packet discard variable is a flag that the user can set to on or off. If turned on, the simulation can be run with the packets being discarded when they can no longer be forwarded. If the flag is turned off, they will be held in the input buffer.

2.4.2 DESIGNING NODES OF THE NETWORK

Each physical connection that one node has to another node has an output buffer and an associated input buffer. The output buffer is connected by a representation of a physical link to the input buffer of the receiving node. Since this is a full-duplex network, if node A is connected to node B, then the reverse is also true. Each output buffer has an associated output queue. Outgoing packets are queued up on a first-come, first-serve basis.

Aside from the input buffers of other nodes, each node also has an input buffer for its own host. There is no need for an output buffer to the host since those packets are taken care of by the overseer subroutine. The overseer maintains tables and monitors the network.

2.4.2.1 Queue Sizes

Each node has an output queue for every output buffer. The user determines the maximum length of these queues, but the length is the same for every queue in every node of the network.

2.4.2.2 Free Buffer Pool

Each node has a pool of buffers held for incoming packets. When an incoming packet is received and needs to be passed on, it obtains a free buffer and is put on the appropriate output queue. If there are no free buffers available, then that packet is simply discarded. If the user chooses, the packets can be held frozen in the input buffer until a free buffer is obtained.

2.4.2.3 Static Routing Table

Each node has a static routing table created by the user and established at run time. There is no dynamic routing; and for simplification, there are no node crashes.

Suppose node A wants to send a message to node D. Whether or not A is connected to D, node A must use a table to find the best route. Each node has its own

table which contains a line for every other node in the network. Each line is composed of the destination node, the best and second best way of travelling to a destination. The way that the next node is chosen is by assigning a priority weight to some of the nodes that are adjacent to this node. The Static Routing Table is established before the network is brought on line and is then not subject to change.

User inputs will be in the form of a series of numbers. These numbers will indicate the node number and a probability of each node being assigned as the next node in the route to the destination. A random number is generated and then compared to these probabilities. Below is an example of the inputs for node 0 in the static routing table for a fully connected three-node network.

Ex. 1, 90, 1, 100, 2
2, 75, 2, 100, 1

A program scans the sending node's routing table looking in the first column for the node number of the destination node. Once found, the program generates a random number and compares it to the even numbered elements (second, fourth, etc.) until a probability greater than the random number is found; that is why the second to last element is always 100. Once the current probability is found, the next element following that contains the node number of the next node in the route. If node 0 is sending to node 1, it must go through either node 1 or node 2. In this example, node 0 will send to node 1, via node 1, 90 percent of the time and through node 2, 10 percent of the time, giving a 100 percent probability of sending through at least one of these two nodes.

2.4.2.4 Virtual Circuit Table

Each node has a virtual circuit table constructed and executed along the lines as written in the book, Computer Networks, (Tanenbaum 1988, 280-283). The user will not have to worry about creating a virtual circuit table once he chooses the size of it, since the simulation automatically performs this task. The user should know of the table's existence and functions. Basically, each table consists of an input virtual circuit number and whence it came, and an output virtual circuit number and the next node in route. Each request obtains a unique identifier based on the previous node in the route and a virtual circuit number.

When a request arrives at a node for the first time, the node verifies its origin and assigns the next available number for that incoming line starting with 0. As an example of this, (see Table 1) if nodes 1 and 2 are connected to node 0 and node 1 sends a request

TABLE 1

Virtual Circuit Table For Node 0

Incoming VC	From Node	Outgoing VC	To Node	Explanation #
0	1	0	2	#1
1	1	1	2	#2
0	0	0	1	#3
1	0	2	2	#4
2	0	1	1	#5
0	2	2	1	#6

#1 First request from node 1 is assigned incoming vc number 0. Since it is the first vc going to node 2 the outgoing vc number is also 0.

#2 Second request from node 1 is given the incoming vc number 1. Since it is the second request for node 2 it is given the outgoing vc number 1 as well.

#3 First request from the host of node 0 so it is incoming on vc 0. It is the first request going to node 1 so the outgoing vc number is a 0.

#4 This is the second request from node 0 so the vc is given the number 1. It is the third request for node 2 so the outgoing vc number is 2.

#5 First request from node 2 so the input vc number is 0 but it is the second request for node 1 so the outgoing vc is number 1.

#6 Third request from this node's host. The incoming line is set to 2 and since it is going to node 1 and this is the third request for that node it is assigned the number 2.

to node 0, that first request is assigned virtual circuit number 0. The next request from node 1 is marked 1, and the following request would arrive on virtual circuit number 2. Meanwhile, node 2's request passes through the same process, acquiring virtual circuit numbers 0, 1, and so forth.

The output lines receive the same treatment. Therefore, each virtual circuit procures both a unique incoming and outgoing identifier. When the virtual circuit is closed, the assigned numbers are freed for recycling for the next request arriving from the same node.

2.4.3 DESIGNING THE HOSTS

Each node has a host process, constructed by the simulation, where subroutines determine the time, the destination node, and the length of the message sent.

2.4.3.1 Traffic Rates

Each node's host process sending rate is determined by a Poisson distribution based on the value assigned to this variable by the user. This variable indicates the probability that the node will send in a clocktick. The input should be between 0.00 and .99.

2.4.3.2 Maximum Messages That are Outstanding

Each node in the network can have a certain number of outstanding messages. These are messages or requests that have been sent by the host but are not yet completed. A message is finished when the last acknowledgement is received and a special packet, called a closevc packet is sent to shut down the virtual circuit.

2.4.3.3 Wait or Timeout Limit

This is the amount of time that a host will wait for an acknowledgement before the host times out and tries to retransmit the packet. The larger the network, the longer the limit must be to allow packets to travel to the destination and get a response back.

2.4.3.4 Destination Selection

One of the user inputs is a two dimensional array that is used to determine the destination of a message. There is a row and a column for every node. Each row

corresponds to a sending node, with each column corresponding to the chance that the node matched to that column will be the receiver. Using a random number generator, the host will be able to determine the node to which it sends a message. The program locates the row of the node sending, then through the columns, skipping the 1's, and comparing the number found in the array element to the random number generated by the program. The column element number is the node number of the receiving node. The number 1 in the table indicates the position in the table or the array element number of the node's number. In the row for node 0, the number 1 is in the row [0] element because node 0 does not want to send to itself. In the row for node 1, the number 1 is in row [1], and for the row for node 2 the 1 is in row [2].

Ex. 1, 75, 100 Row for node 0
30, 1, 100 Row for node 1
50, 100, 1 Row for node 2

Look back at the adjacency matrix, notice node 1 is connected to nodes 0 and 2. Therefore, in this example node 1 will send 30 percent of the time to node 0 and 70 percent of the time to node 2.

2.4.4 DESIGNING THE SIMULATION

2.4.4.1 Number of Clockticks

The run time of the simulation is based on the number of "clockticks." A clocktick is just a relative unit of time. There is no maximum limit on the number of clockticks that the simulation can run. It would be surprising to find the need to set this variable beyond 20,000 to 30,000 clockticks since the user should be able to fully exercise a network without exceeding this range. On every even clocktick, each node will send whatever packet is in each output buffer to the input buffer to which it is "physically" linked. On the odd clockticks, each node polls its input buffers in a round-robin scheme and determines what to do with every packet it finds.

2.4.4.2 Control Scheme

When the user chooses certain control schemes, there are additional parameters for the user to determine as well. The words in bold indicate that the user must input the data for each variable.

For Control Scheme 1, which is the token scheme, each node has a store, called "**MAXSTORE**," for tokens that it might hold. The size of these stores must be chosen by the user. The maximum number of tokens in the network will be **MAXSTORE** times the number of nodes in the network. The size is the maximum allowable number of tokens permitted in each node's store, as well as the number of tokens residing in the store when the simulation begins running. If the number is exceeded, then the tokens will be passed on to a randomly chosen node.

For Control Scheme 2, which is the choke packet scheme, there are four additional parameters the user provides.

Warning Percent for Queue Length indicates how full the output queues become before a choke packet is sent back to the host. This variable is expressed as a fraction of 1.0 (in other words, 30 percent is .30) and should not be set too high since it takes a while for the choke packets to return to the source.

Ignore Time variable holds the number of clockticks after a host gets a choke packet that it will ignore all other choke packets with the same destination.

Listen Time pertains to the time a host node will listen after the ignore time is up for any other choke packets with the same destination.

Rate Reduction Factor is defined as the percent the host will slow down its sending when the host assimilates a choke packet. The rate reduction factor is expressed as a fraction of 1.0.

For Control Scheme 3, which is the flow balancing scheme, no additional parameters are needed to run the simulation.

2.4.5 DESIGNING PRINTOUT FORMATS

The following inputs are used to design the output files that the user can examine for analytical purposes.

More Printouts is the number that determines how often an output line is written out to a file called SUMMARY. This output file is written to every X clockticks, X being the

number that is specified here. The scheme, maximum queue length, buffer pool size, maximum number of virtual circuits, and the maximum allowable outstanding messages are printed out, as is the message count, number of packets lost, number of requests discarded, and the number of packets discarded. A running total of these variables is printed out as is the change in the variable since it was last printed out X ticks ago.

Want Printouts is a flag set to zero (0) or one (1). The flag indicates whether or not the user wants a complete, step-by-step analysis of activity in the network. A zero (0) allows for larger networks and longer run times but just prints out the monitoring statistics. There is no way to see what the network is doing or to trace packets. By setting this flag to 1 an output file is produced for each node. With this printout routine the user can see when a host decides to send and track the packet as it moves through the network. It is possible to follow what happens in the network; but to do this, the network size should be kept very small. A single clocktick can produce several hundred lines of output for each node; and as a printout file is created for each node, the output becomes expensive both in terms of disk space and "write to file" time. In order to follow packet travel over the network the user needs this detailed printout. With larger networks packet travel is difficult to trace and the necessary printouts become quite large. For a three-node, completely connected network, the output has been as long as 35,000 lines of output for 1,000 clockticks. The output is a summation of activity during the simulation run. If it is one (1), then an output file is made for each node.

Time to Start Printing determines from what time the program starts printing to the output files should the user set the WANT PRINTOUTS flag to one. The end time is set with Maxticks; the maximum time limit should be no more than 100 clockticks. If the WANT PRINTOUTS is set to zero, then the specified number has no effect on the simulation.

CHAPTER 3

PROJECT IMPLEMENTATION

3.1 SOFTWARE ARCHITECTURE

Listed below are functions and their respective definitions, followed by the structure chart of this simulation (see Figure 3). Remember, not all functions are used within one run simply because some of the functions called are based on the control scheme the simulation uses. Only the important functions will be discussed; the overhead and maintenance functions are mentioned briefly. Refer to Appendix A for a complete C-Flow listing.

Every C program must have a special function called "main" that starts and finishes a program's execution. The main function usually calls other functions to do much of the real work. In this simulation "main" is the controlling function. Main calls the functions that read in the user's data file, initialize the network, open the necessary input and output files, verify whether or not any packets timed out, control the clock, and checks to see if a node can or cannot send. Main also kicks off the function that polls the input buffers and other functions controlling different printouts to output files. Output file names as well as these functions are capitalized in the source code so that they are more easily discerned, and so they are capitalized in the explanations below. The following describes each major function.

Function Call	**Definition**
GETINPUTS	reads in the user inputs. It calls the functions "GETNUMBERS," which reads in the global variables, and "GETTABLES," which reads in the user's tables.
GETNUMBERS	simply reads in the global variables from the user's data file.

STRUCTURE CHART FOR NETWORK SIMULATION

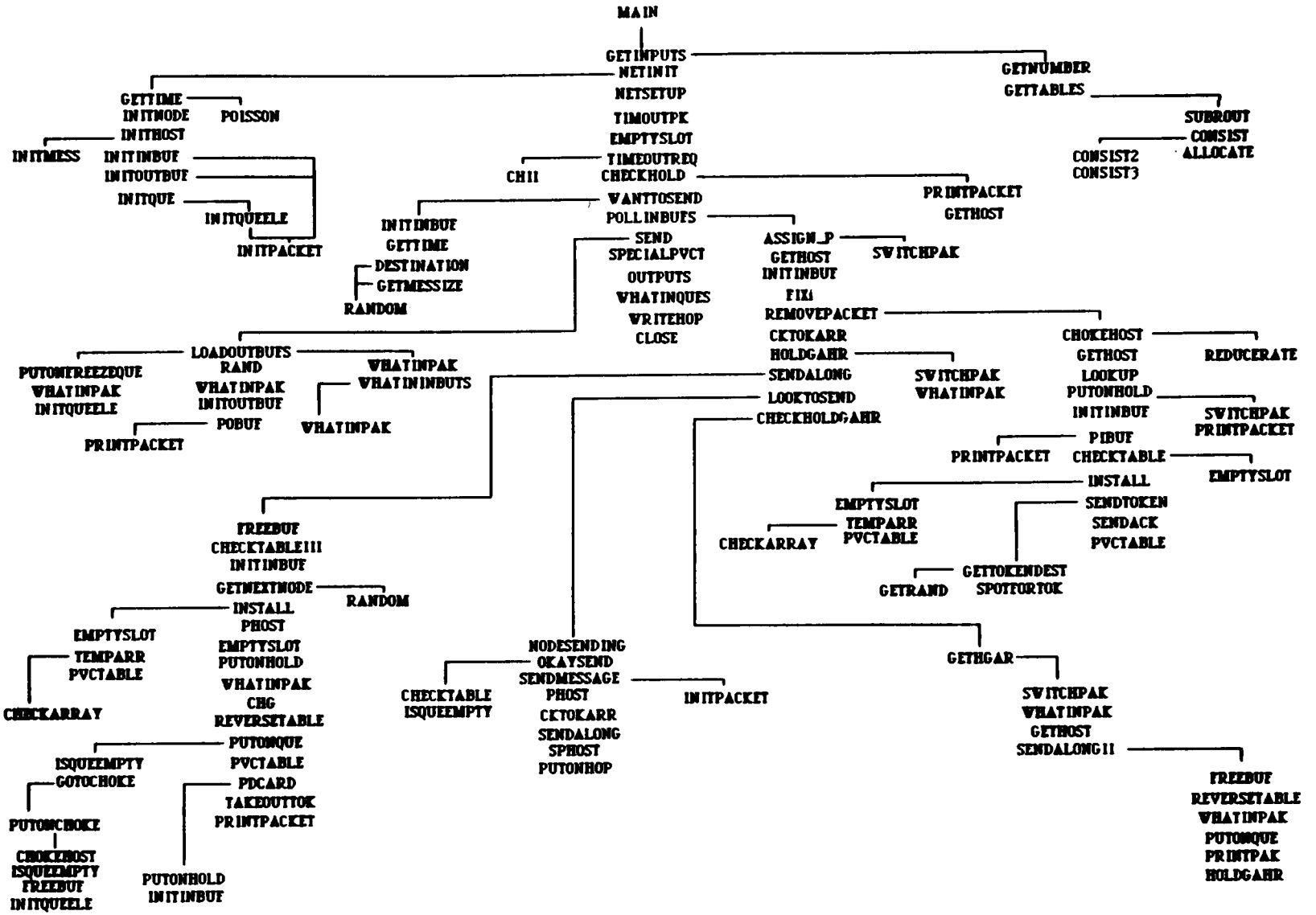


Figure 3

GETTABLES	calls functions that fill in the data arrays, checks for consistency, and allocates the dynamic memory. These functions are called SUBROUT, CONSIST, and ALLOCATE.
SUBROUT	stores tables from the user's input into data arrays that the program will read.
CONSIST	checks, where possible, the user's inputs for consistency.
ALLOCATE	performs most of the dynamic memory allocation for the simulation.
NETINIT	receives the initial sending times for the nodes and initializes the necessary arrays and variables. It calls GETTIME, INITNODE, INITHOST, INITINBUF, INITOUTBUF, and INITQUE, where upon these subroutines do the initialization of the actual arrays.
NETSETUP	takes the users inputs and fills in the network arrays so the simulation can begin the run.
TIMEOUTPK	checks to see if any packets have timed out and need retransmitting.
EMPTYSLOT	determines whether or not a node has any space in its virtual circuit table for new virtual circuits.
TIMEOUTREQ	checks to see if any requests from a node have timed out and need to be resent. This function calls CHII, which obtains the request number off of a hold queue.
CHECKHOLD	investigates whether or not any requests have been refused and are waiting to be resent. Rejected requests are on a separate hold queue from those that have timed out.
WANTTOSEND	calls GETTIME to receive a value which is the number of clockticks relative from the present clocktick. The function call will set a variable that is the number of clockticks in the future when a node will want to send. When a node reaches the specific time, it

will call WANTTOSEND. Proceeding from this point, WANTTOSEND then determines the destination and message size of the request and when the node will send next. WANTTOSEND calls INITINBUF, GETTIME, DESTINATION, GETMESSIZE.

GETTIME — uses a Poisson distribution to determine when a node shall send again. Moreover, GETTIME takes the user supplied traffic rate for that node as the lambda value to implement the selection process.

DESTINATION — summons a random number generator to determine what node will be the receiver of the next request.

GETMESSIZE — calls a random number generator to find out the size of the message the calling node now wants to send.

POLLINBUFS — polls each node's input buffers to see if there are any packets in them. The packet is removed should the node be the packet's destination. If the packet is destined for another node, then the node attempts to pass the packet along. POLLINBUFS calls four other functions used by the simulation as overhead routines (ASSIGN_P, GETHOST, INITINBUF, and FIXI) and are not discussed here. Other routines that are called are REMOVEPACKET, SENDALONG, and LOOKTOSEND. If the token congestion control is being used, then CKTOKARR is also called. If the control is the flow balancing scheme, then POLLINBUFS calls HOLDGAHR and CHECKHOLDGAHR.

REMOVEPACKET — determines what type of packet the destination node received. Possibilities include choke packet, call request, call accept, call reject, acknowledgement, regular message packet, or message packet containing a token. Once packet type has been determined, appropriate functions are called to handle each specific type.

Should REMOVEPACKET ascertain that the packet type is a:

choke packet, then CHOKEHOST is called to reduce the sending rate of the host.

call accept, then the time of the accept and the number of hops in the route are noted and written into the message queue.

call reject, then LOOKUP is called to get the virtual circuit number so that the virtual circuit can be cleared from the table followed by calling PUTONHOLD to put that request on a special hold queue.

call request, then CHECKTABLE is called to see if the node's virtual circuit table has any empty slots. If space exists, INSTALL is called to make the entry into the table. Then SENDCALLACCEPT is called to send a call accept packet back to the sender.

regular message packet, then SENDACK is called to send an acknowledgement back to the sender. If the token scheme is being used and the packet contains a token as well, a check is made to see if this token increases the node's store over the limit. If it does exceed the limit, then SENDTOKEN is called to send the token to a randomly chosen node.

CKTOKARR is called if space is lacking in a node's token store. The extra tokens are placed in a special holding queue where they wait for a packet going to the random destination assigned to that particular token. Presuming a packet heading towards that destination does not possess a token, then one is assigned to the packet. CKTOKARR checks the special array for any destination match between the waiting tokens and the packets being sent along to the destination point.

HOLDGAHR is used in the flow balancing scheme. It puts call accept packets on a hold queue for responses.

SENDALONG does most of the actual work in running the simulation. Depending on the situation, it calls FREEBUF, CHECKTABLEIII, GETNEXTNODE, INSTALL, CHG, REVERSETABLE, PUTONQUE, PDCARD, TAKEOUTTOK.

FREEBUF determines whether there are any free buffers in the buffer pool.

CHECKTABLEIII checks to see if a virtual circuit is set up already in a node's virtual circuit table.

GETNEXTNODE uses a random number generator and the static routing table to determine the next node in the route towards the destination.

INSTALL installs the virtual circuit in a node's table.

CHG is called only if the flow balancing scheme is being used. CHG ensures an incoming request does not have an accept waiting on a "hold go ahead response" queue.

REVERSETABLE returns the correct virtual circuit number when a packet is traversing the network going sink to source.

PUTONQUE is called to try and place the packet in the correct output queue. It calls ISQUEEMPTY to see if the output queue has room for the packet. If the output queue contains available space, then the packet is placed into an element of that queue. If the control scheme is the choke packet scheme, then GOTOCHOKE is invoked to see if the warning length is exceeded. Should the warning length be exceeded, GOTOCHOKE then checks to see whether or not a choke packet can be put on an output queue back towards the host.

PDCARD handles the overhead when packets are being discarded. It keeps track of the number of packets, requests, etc., that are discarded.

TAKEOUTTOK is called when using the token control scheme and a packet is not placed on an output queue, for whatever the reason. This function replaces the correct amount of tokens back into a node's token store.

LOOKTOSEND is called by the main routine. Should a node decide not to send a request, it checks for any packets to send. LOOKTOSEND calls NODESENDING, OKAYTOSEND, and if the token scheme is being used, CKTOKARR.

NODESENDING is called to see from which outstanding message the packet will be sent.

OKAYTOSEND determines if there is a free buffer in the buffer pool, and if so, whether the correct output queue has an empty slot for the packet. SENDMESSAGE then puts the packet on the correct output queue and takes care of the overhead associated with keeping track of what packet was sent and from what message.

CKTOKARR is also called if the token scheme is being used (see POLLINBUFS).

CHECKHOLDGAHR is used in the flow balancing scheme to do the calculations that check the necessary clocks, threshold limits, and credit counts. It calls GETHGAR.

GETHGAR takes a go ahead response off the hold queue. It uses a modified version of SENDALONG, called SENDALONGII, to send the accept along the correct virtual circuit.

SEND is called by the main function. SEND is the only function called on every odd clocktick. It calls LOADOUTBUFS to load the next packet in the output queue into the output buffer. LOADOUTBUFS keeps track of whether or not the queue is frozen with a call to PUTONFREEZEQUE. Output queues are frozen when the input buffer that is connected to this output buffer

becomes busy and when a packet the input buffer holds cannot be forwarded. SEND then determines if a packet is lost due to line degradation and then sends whatever was in all the output buffers in the network to their connecting input buffers.

Several functions are used to print out different structures.

WHATINQUEUES prints out some information about what was in each element of an output queue.

WHATINPAK prints out what is in a packet.

WHATININBUFS prints out what is in a node's input buffer.

MONITOR, SUMMARY, and HOPS are three output files that are printed at the end of each simulation run. The user can examine these output files to monitor network performance. These output files provide the user a break down of what has happened during the simulation run.

MONITOR analyzes activity at each node and in every queue. This file is updated based on the value the user installed under "More Outputs." For example, suppose 500 is installed under "More Outputs," then every 500 clockticks any information gathered would be written out to this file.

The function writing to MONITOR prints out how many messages each node finished up to that point and a break down of what has happened in each queue. Then it prints the totals for the network. These are the total number of messages completed, the total packet count, the total number of packets lost, and the total number of requests and discarded packets. At the end of the simulation, MONITOR prints out what was in each element in each queue of every node, as well as what was in the input buffers.

SUMMARY is written to every time MONITOR receives information. SUMMARY contains the message count, number of packets lost, and the number of requests and packets discarded. Resulting values are shown as running totals and the change in value from the last time this file had information written to it.

WRITEHOP writes to the file "HOPS" at the end of each simulation run. For each node HOPS breaks down the information according to the hop count and message size. As an example, suppose a hypothetical network has three nodes and possible messages sizes of three, six, and nine packets. Each virtual circuit can have a hopcount as great as two. For each node starting at node 0, HOPS prints out all the messages finished with a hop

count of one and a message size of three. Then it prints out all messages with a hopcount of one and a message size of six, followed by all the messages with a size of nine. After all the message sizes have been printed for each node, HOPS repeats the process for all completed messages having a hopcount of 2.

For each message that has been completed WRITEHOP prints out the message number, time the request was first sent out, the number of times the request for this message has timed out, time the host secured the last acknowledgement, the total time of the message, the time when the first packet was sent, and the message time. Total time is the period from when the first request was sent out until the last acknowledgement was received. Message time is the time from the first packet was sent until the last acknowledgement was received. HOPS then prints an average total time and average message time for each hop count/message size combination.

Other print routines used to print out various structures and tables into different files are PIBUF, POBUF, PRINTPACKET, PRINTPAK, PNODE, PVCTABLE, SPECIALPVCTABLE, PHOST, AND SPHOST.

3.2 IMPLEMENTATION ISSUES

The programming language 'C' was chosen for its portability to other machines. The simulation was successfully ported over to the AT&T 3B2 from the Pyramid. The simulation runs on the AT&T personal computers but takes approximately three times longer to compile and then run. Each user can design an individualized network by setting the values in an adjacency matrix and static routing table, as well as some other arrays previously explained in User Inputs.

Several processes, used together, can be thought of as the overseer that controls and monitors all operations in the simulation. Processes controlling the syncronization of nodal sending, timeout clocks for requests and packets, and accounting and tabulation of data are some examples of overseer type functions.

The basics of how the actual simulation works are as follows: The network's sending has been synchronized so that at each odd numbered clocktick, the overseer program will "stop time," and poll all the nodes in the network to see what transpired during that simulated clock tick. Hence, the overseer updates tables at this time keeping track of delays, throughput, and other vital statistics. All nodes read what is in their input buffers (these buffers hold one packet each) and will, if possible, transfer those packets to the appropriate output queue. An output buffer is obtained from the free buffer pool if adding that buffer to the output queue will not exceed the maximum allowable for the

queue. On the even numbered clock ticks, all nodes send packets in their output buffers to the nodes to which those output buffers are connected.

How the virtual circuits are handled is based on Computer Networks, (Tanenbaum 1988, 280-283). Each node has a table containing all active virtual circuits passing though the node. For each incoming packet a node knows where it is coming from and on what virtual circuit, creating a code based on these two items. The node examines this code ascertaining if this packet exists on a previously established circuit, or if a new circuit must be created to pass this packet along to the next node. If the circuit is already formulated, then the packet is sent along the circuit, assuming there is room in the output queue. If the virtual circuit has not yet been established through this node, then the node picks a code based on the next node in the route and the lowest available virtual circuit number for that next node.

In a real network, when a node crashes, that information is broadcast out on to the net, and backup routing tables are implemented. To save space and to keep the simulation simpler, the nodes in this implementation cannot crash; thus, backup tables are not necessary. The static routing chosen at the start of a simulation run is fixed for the duration of that run, though it certainly can be changed for subsequent runs. The network is constructed to study congestion control and not routing studies; hence, the concern of crashing nodes is not considered important to this thesis.

3.2.1 PACKET DISCARDING

The user has the option as to what to do if a packet can not be forwarded. The first option is to freeze the packet at the input buffer. If a packet can not be put on the proper output queue, then the packet is kept in the input buffer where it was received; and the "input buffer busy" flag is set for that buffer. Until the packet can be transferred out of the buffer, it will prevent the connecting output buffer from being loaded and a new packet sent, which causes the output queue for that node to begin backing up. As an example, node B passes a packet to node A. Node A has no free buffers left in the buffer pool to be assigned to the output queue and therefore cannot process the packet. The packet is left in the input buffer and a flag is set indicating that input buffer is busy. Now node B cannot send anymore packets to node A until that input buffer is clear of the packet.

However, even with congestion control schemes where the packets are not discarded, such as the one just described, the output queue fills to capacity and deadlocks the

network. One simple cause of such a situation is when a node's virtual circuit table is full, any incoming request will be stopped and held in an input buffer, thereby causing the attached output queue to start filling up. Now, if in that output queue there is a close virtual circuit packet that would free up a spot in the virtual circuit table, it will never arrive because the input buffer is occupied. The method might work if one of the congestion control schemes was a preallocation of resources scheme. This problems can be eliminated by increasing queue length or decreasing the number of allowable outstanding messages.

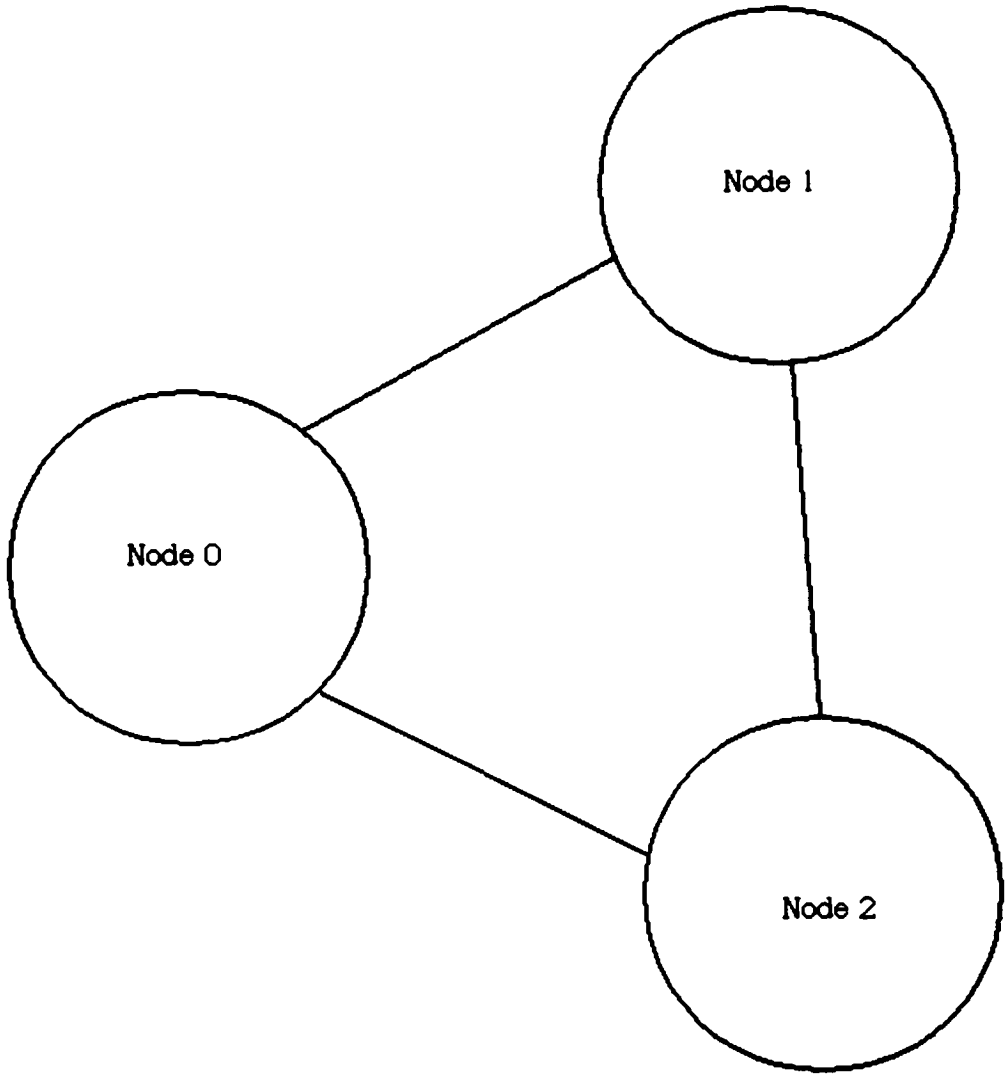
The second option is to simply discard the packet. If a packet can not be forwarded, for whatever reason, then that packet is thrown out and will have to be resent. A separate subroutine catalogs which packets are discarded and when that event occurred. The resending of these packets will cause a slow down in through traffic, but due to the congestion control schemes, it will not cause a deadlock situation.

In the ensuing pages, the topics discussed are founded on the assumption that all packets are discarded. Many of the sections explain the packet discarding in detail.

3.2.2 NODAL PROCESSES

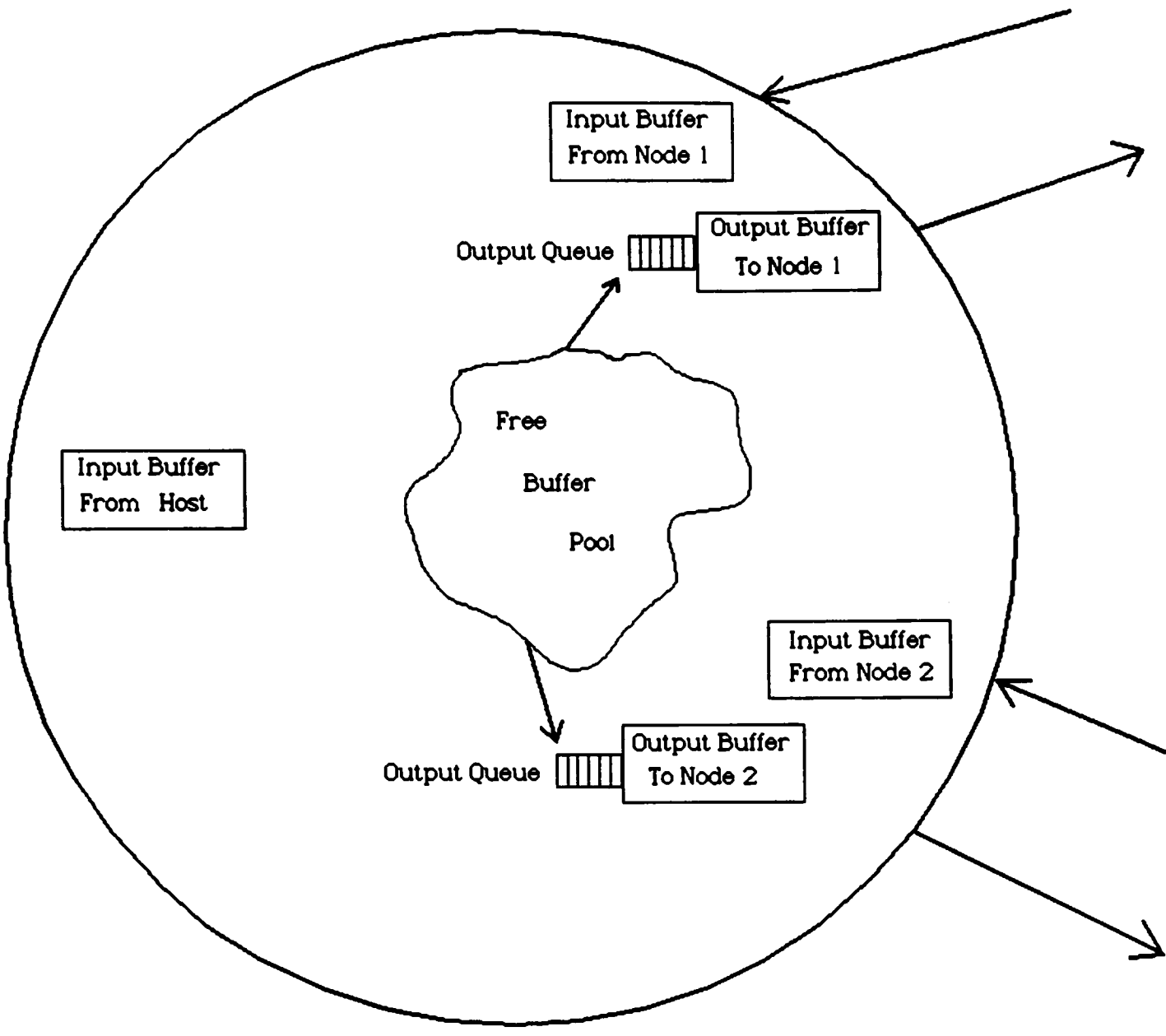
A completely connected three-node network (see Figure 4) serves as an example to discuss implementation issues. Each node has a virtual circuit table and a free buffer pool. The size of each of these parameters is user-defined by an entry in a data file. Each node also has an input and output buffer for every connection except from the host. The host-to-node connection only has an input buffer (see Figure 5). The output buffer is not needed here because the packets are never really transferred up to the host. The packets are noted by an overseer subroutine that keeps track of them and then discarded. The free buffer pool holds a number of buffers that incoming packets will need to obtain in order to be put on the output queues. Each output buffer has an associated output queue. Packets arrive at an input buffer; and if they need to be sent along to another node, they obtain a buffer from the free buffer pool, and then are put on the correct output queue. The user is able to make changes to most of these variables. All output queues are set to the same user-defined length.

On every even clocktick each node will poll its input buffers, including the one from its host, to find out if a packet has arrived. What happens when a packet is found depends on the packet type and the packet's destination.



Example of a completely connected
full duplex 3-node network

Figure 4



An example of a node in the simulation

Figure 5

3.2.2.1 Polling Input Buffers

The polling of input buffers is accomplished by a round-robin polling scheme with the starting input buffer changing every even clocktick. As an example, if there are three input buffers (0, 1, and 2), the cycle would be 2, 1, 0; then on the next even clocktick it would be 1, 0, 2 and then 0, 2, 1. On the next tick the cycle would repeat itself 2, 1, 0.

First, the program determines if the input buffer is a viable one. The number of possible input buffers is determined by the degree of connectivity of the network. This degree of connectivity is defined as the maximum number of connections any single node of the network has attached to it. However, it does not mean that all the input buffers will be connected to other nodes. In other words, if the degree of connectivity is three, but this node is only connected to two other nodes, an unconnected input buffer exists that the polling scheme must skip over.

Next, the program verifies if the input buffer contains a message packet or a "request to send" packet. If the input buffer is connected to some other node and does not contain a packet, then the polling scheme checks the next input buffer. If the input buffer is connected to the host and there is a request from the host, the request will already be in the input buffer by the time the polling scheme is activated. Otherwise, the input buffer will be empty. It is at this point that a message packet may then be put into the input buffer and it can then be sent along. This is due to the fact that there are priorities in the request process. One can look at the process in the sense that a request is sent to the input buffer while a message packet is drawn to it. If there are packets to be sent, they are put on the correct output queue. Otherwise, nothing is done and the next input buffer is checked.

When the input buffer contains a packet destined for some other node, and this node is not the source node, then the node must be intermediary. The packet is passed along to the correct output queue towards its destination. If the queue is full or the free buffers are all taken, then the packet is discarded, if the user chose that option. If the packet is not for this node, but this node is the source node, then the packet is a request and is passed along to the correct output queue. Again, if that queue is full or the free buffers are all taken, the packet is discarded.

When the sink node is the destination of a packet, the packet is either a call request or a message packet. Assuming the packet is a call request, it means that some other node wants to send to this node. If there is room in the virtual circuit table, a virtual

circuit is established in the receiver node, a call accept is generated, and sent back to the source.

If the packet is a message packet generated by some other host, an acknowledgement is generated and sent back along the same virtual circuit. A special case is when the sink obtains a closevc packet letting the sink know that it is time to close the virtual circuit. The circuit is closed immediately. There is no transfer to a host process at the sink, nor is a packet sent back to the source telling it that the sink has closed the virtual circuit.

When a source node is the destination of a packet the packet could be a call accept, a call reject, or an acknowledgement of a message packet. If the packet is a call request, then a request from this node has been accepted by the destination. The virtual circuit was set up from source to sink, and a flag is set in the message queue indicating the message waiting for the circuit can now be sent.

If the packet is a call reject, then a request from this node has been rejected due to lack of virtual circuit table space, either by the sink node or some other node along the route. The sending node does not care which node dismissed the request. The call request is put on a special queue for rejected call requests and the virtual circuit is wiped from the table. After 20 clockticks, the request will be resent to the same destination but on a different virtual circuit and possibly through a different route.

When the process finds an acknowledgement in the input buffer, the sink sent an acknowledgement for one packet of one message that came from this node's host. When the last acknowledgement of a message returns, a flag is set indicating the message is now finished and the virtual circuit can be closed down. However, it will not shut down until a closevc packet is put onto an output queue to the sink. A host's request process has priority, but the next tick that the host is not transmitting a request, the closevc packet will be sent, and the virtual circuit closed down. If the output queue is full, the request process is disabled so that the close packet will stay in the input buffer until it can be put on the output queue.

3.2.2.2 Host Process

The host process uses a mean packet arrival rate, referred to as a lambda value, supplied by the user, to set up a Poisson distribution: The size is determined from a random number generator and compared to the users input of message sizes. The primary

purpose is to determine when the host will want to send its next request. If the host does want to send, it uses a random number generator to determine the destination of the message and its size.

The host generates a request packet, which contains the message number, destination, and size of message. These packets are transferred to the connecting input buffer of the node. This information, along with other variables such as the time the original request was sent, the time of this request, and the number of times this request has been refused, are put into a structure on the message queue. The structures in the message queue are there to keep track of such variables as the time the request was first sent, the time the first packet was sent, and the time that the last acknowledgement was received.

The assumption is that there will be no errors in the packet transfer from the host to its node. Also, when a host wants to send either a request or a message packet, that packet is immediately put into its node's input buffer. The message packet is not sent during the "SEND" tick. The host process is used just to make up requests, generate the message packets, and keep track of other overhead.

3.2.2.3 Request Processes

Requests may enter into the network by one of three different methods. These request methods are listed in the order that the simulation will process them. First, a request might time out and need retransmitting. For later reference this request process will be called Request Process 1. Secondly, should a call request be rejected, the request passes into a request hold queue designed to hold it for a specified time before retransmission; this is Request Process 2. And thirdly, the original request process described above under the host process will be Request Process 3. To sum up the simulation's algorithm for choosing which request it will send, it verifies if any requests have timed out, then it checks to see if any rejected requests are ready for retransmission; and if both of these are negative, only then will the simulation look to see if a node wants to send a new request.

Request Process 1 operates in the following manner: The simulation first looks to see if any requests have timed out; if one has, then the request is resent along the original virtual circuit. The request is transmitted along the original virtual circuit because it is not known why the request initially timed out, the virtual circuit may or may not be currently functional.

Within the second request process particular criteria warrant examination. Supposing that no requests have timed out, the hold queue for call requests is examined to locate any requests that need resending. Usually rejection is caused by a virtual circuit table

being full at the destination or one of the intermediate nodes. When any node rejects a request, the node sends a call reject back to the host; and as the rejected request is passed back through the virtual circuit, the rejected request shuts the virtual circuit down as it propagates back to the source. When a call is rejected, the message number, message size, and the destination are put on the message queue. After 20 clockticks, this message is allowed to be retransmitted, possibly with a new route chosen, certainly with a new virtual circuit.

If the request hold queue is empty or time has not expired on any requests that are being held, then the simulation scrutinizes the regular request process. Request Process 3 uses a Poisson distribution (based on the users input of the traffic rate for this node) in determining whether or not the host wants to send a message. The simulation then uses a random number generator to obtain the destination and message size.

A request is then generated. The message number, destination, size of message, and the virtual circuit number from the host is put on to the message queue. Each host can have a certain number of outstanding messages that are not yet finished. This number is specified by the user.

These request processes are frozen or disabled if the following conditions are met within a node.

A. Request Processes 2 and 3: The virtual circuit table for the sending node is full. Remember for Process 1 the virtual circuit is still set up in at least the source node.

B. Request Processes 1, 2 and 3: The input buffer from the host is still full (the packet in there was not able to be transferred to an output queue, either because the queue was full or there were no more free buffers in the buffer pool). This is true only if the packet discard flag is not set.

C. Request Process 3: The number of outstanding messages has reached the allowable maximum. Therefore, no new requests will be permitted; however, requests in Process 2 and 3 are already part of the outstanding messages.

With a high host traffic rate, it might seem as if the system would flood with requests, and the network does when the simulation first starts up. However, a finite number of outstanding messages per node is allowed by the system. After the limit is

reached, no more requests can be sent out and so the packets can catch up to the requests. When a low host traffic rate is used, overloading is not as prevalent.

3.2.2.4 Sending and Receiving Protocols

3.2.2.4.1 Sending Protocol

Once a sender has received an accept packet from the sink, it can now start sending the message packets. When the host can send a message packet, it will continue to do so until all the packets of a particular message have been sent. This process can be interrupted if the host needs to send a request or a closevc packet. The sender is using a variation of the stop-and-wait or a sliding window protocol. As each packet is sent, it contains the message number, packet number of the message, and is time-stamped along with other fields.

The sender keeps track of the acknowledgements as they arrive from the receiver. If an acknowledgement arrives out of order, the sender must retransmit all the packets starting with the packet number one above the last correct acknowledgement. To illustrate this, assume node A is sending a message of nine packets to node E. Node A has sent all nine but has only received acknowledgements one through four. It then receives the sixth acknowledgement. Node A now has to retransmit packets numbered five through nine so that it can be sure that node E is receiving the packets in order as is necessary for a virtual circuit. When the fifth packet is sent out it is "time-stamped" and the time noted by the host. Any acknowledgements arriving with a message number time-stamped before these packets are sent out are now ignored since they are from the original packets and not the retransmitted ones. Why the acknowledgement arrived out of order is not important to the host. It could be that the packet was lost or the acknowledgement was lost, but either way the sender must retransmit.

3.2.2.4.2 Receiving Protocol

When the receiving node obtains a message packet, the packet number becomes the acknowledgement number. The message number and time stamp are transferred over to the acknowledgement packet as is the source and sink node numbers. The originator becomes the sink node and the destination is now the source node. If there is room in the correct output queue, the acknowledgement is immediately put in it. Otherwise the acknowledgement is discarded.

The receiver does not care if the packets are in order, the sink node just acknowledges the packets as they are received. It is assumed that the message would be stored some place until the entire message is in and can be transferred up to the receiving host. This process is not implemented in this simulation.

3.2.2.5 Sending Message Packets

The process that sends message packets operates under a variety of conditions. For example, assuming there are no requests to be sent, the host will send a packet from one of the outstanding messages, if in fact there are any. But, if any message has finished sending its message packets and is ready to close down the virtual circuit, the special closevc packet takes top priority. If there are two messages ready to close down, then the one that received its last acknowledgement first is sent. In contrast, if no messages are ready to close, then a packet is sent from the message with the lowest call accept time. The call accept time is set to be the clocktick when the accept packet arrived at the source node. Now that a call accept has been received, the host can initiate sending message packets, one packet for each even clocktick, unless there is a request or a closevc packet being sent out instead, until all the packets of the message are sent. The host then waits for the last acknowledgement from the receiving node before closing the virtual circuit in the sending node.

If the source node does not receive the acknowledgement within a user-specified time limit, the host needs to retransmit the message packet. Since this is a virtual circuit network, the sink must receive the packets in order. Therefore, the host will have to resend all the packets that follow the original message packet.

3.2.3 THE NODES

A node of the network is either a source (sending), a sink (receiving), or an intermediate node. The classification is dependent on the packet that the node is handling at the time. As illustrated below, a node can be all three during a single clocktick.

3.2.3.1 Sending Node

The request is put into the input buffer from the host. If the request can be put on the proper output queue, it is. If the request packet cannot be placed on the queues, one

of two methods will be invoked. If the discard flag has been set the request is placed on a special request hold queue. This hold queue contains those requests that can not be placed onto an output queue and requests that have been rejected from some node on the virtual circuit that the request was trying to establish. A clock is started for each request placed on this queue. Once the time out clock counts to 20, the request will be transmitted over the same virtual circuit. The reason the request is sent over the same virtual circuit is because the state of the virtual circuit is unknown. The host does not know if the request is lost, delayed, or discarded; nor does it know how much of the virtual circuit has been set up. Therefore, to avoid a double entry in a node's virtual circuit table for the same message, the request packet is sent out on the same virtual circuit as the original request.

If a call request returns rejected rather than accepted, the message number, size, and destination are saved in a special hold queue; the virtual circuit is closed. Acknowledgements are not sent on to the host; they are merely noted and then discarded.

3.2.3.2 Intermediate Node

Upon receiving a request, an intermediate node checks the virtual circuit table for free space. After locating the space, the node installs a new virtual circuit for the request and tries to pass it on to the next node. The next node is decided by a random number and the static routing table based on the destination of the message. If the correct output queue is full, the request packet is frozen at the input buffer or discarded if the packet discard flag is turned on. The request packet does not try and seek another path to the sink.

If the virtual circuit table is full when the request arrived, the node sends a call reject packet back to the source node. The virtual circuit is not created in the table of the node which generated the call reject. As the call reject packet is sent back through the network, it will clear the virtual circuit in each node's table.

When an intermediate node receives a message packet, the node tries to send it along. Once again if the correct output queue is full or all the free buffers are taken, the packet will either be frozen at the input buffer or discarded, depending on how the packet discard flag is set. When the intermediate node obtains a closevc packet, it places the packet in the correct output queue. The node then clears that virtual circuit from the virtual circuit table. Closevc packets are never discarded; the input buffers are always frozen instead, thus ensuring none of the closevc packets are lost.

3.2.3.3 Receiving Node

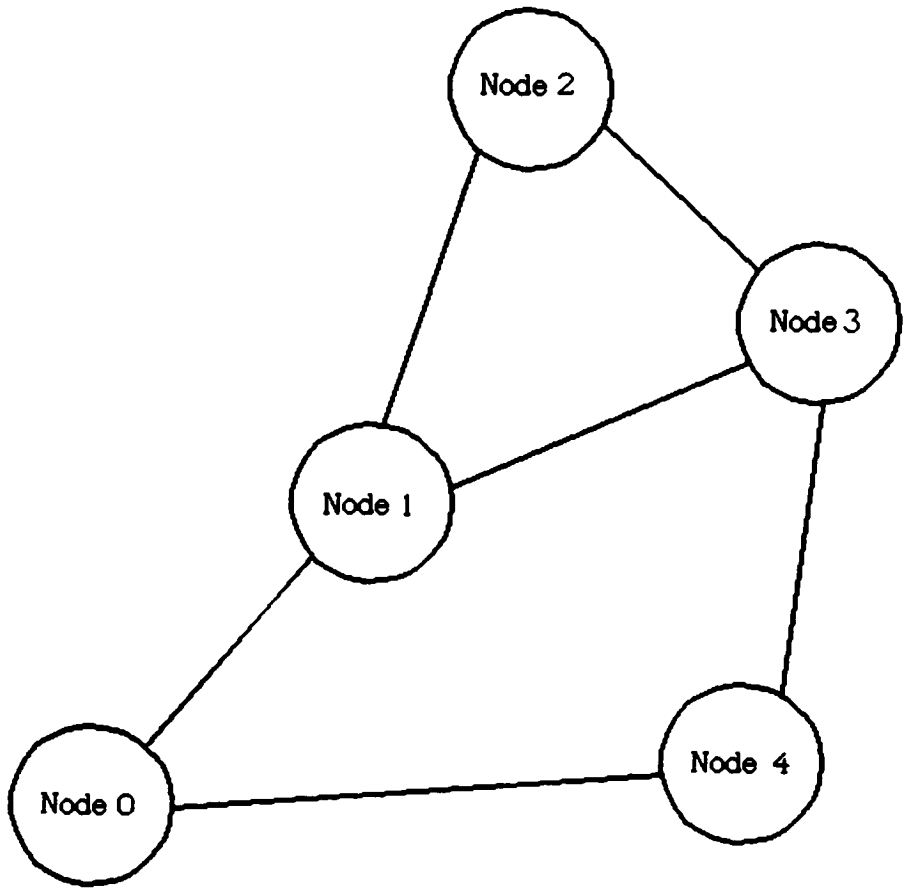
Presuming the receiving node picks up a call request and space is accessible within the virtual circuit table, the virtual circuit is installed. A call accept is generated and sent back to the sending node. On the other hand, suppose there is no room in the table; the receiving node does not install the virtual circuit, but instead sends a call reject back to the sending node. The resulting action forces the virtual circuit's closing through the intermediate nodes, as well as sending on the rejection packet.

When a receiving node accepts a message packet, it will send an acknowledgement back to the source. Each message packet has a message number, a packet number, and sender identification. The receiving node uses these to format the acknowledgement packet. For each message packet a sink receives, it sends an acknowledgement back to the source node informing it what packet number it just received and from what message. If the output queues are full, that acknowledgement is thrown out, causing the source node to time out eventually and resend the packet, plus those succeeding it since the message must arrive at the destination with its packets in order. The receiver does not care if the packets arrive out of order, but lets the sender worry about retransmission of the lost packet along with all of those that come after it in that message.

3.2.4 ROUTING ALGORITHM

As previously explained, routing is done using a static routing table formulated for each node. To prevent a request from doubling back on itself and becoming installed in a node's virtual circuit table twice during the simulation run, a block is established preventing such situations. This method works well for a ring type network topology. As soon as other nodes are added inside or outside of the ring, problems with a request doubling back on itself can re-occur.

The routing method used in the simulation disallows the next node to be the same as the previous node. However notice the five-node network in Figure 6, specifically nodes 1, 2, and 3. If a request travels from node 0 to node 4 by way of node 1, node 1 could send the request to node 2, which would send the request to node 3, which might then send the request back to node 1. This dilemma is solved by forcing a node receiving the same request from a different node to send a call reject packet back to the source node, which clears the virtual circuit tables of intermediate nodes as the packet returns to the source. When this request is resent from the host, hopefully the randomness of the routing selection will prevent the same path from being chosen again.



Ring of nodes 0-1-3-4
with node 2 added to outside

Figure 6

3.2.5 THE CONTROL SCHEMES

3.2.5.1 Scheme 1 - Tokens

This control scheme's main premise is that each node starts with a certain number of tokens in a "token store." The request processes work the same as described above. The difference comes in the sending of the packets. In order to send a message packet, the source node must have a token. Without a token no message packets can be sent. Consequently, a message packet automatically contains a token. If a packet is discarded, it will eventually time out and be resent; since the original packet had a token, so will this one because the token was lost as well as the packet.

Each message packet traveling through the network can pick up one extra token to carry on to another node. When a node's token count exceeds the maximum that the token store can obtain, then the extra tokens are put into a queue. These extra tokens are then picked up by message packets passing through to other destinations.

When the sink secures a message packet, the token is extracted and placed into the sink's store. Should the maximum be exceeded, then a random number generator determines a destination for this token and puts that destination into an array. The array length will be the same as the number of nodes in the network and all the elements are initialized to zero. As an example, assume that the random destination is node 2. The program will look at array [2] and will increment that element by one. Whenever a message packet arrives at a node, two items are checked: first, whether or not the extra token bit is already turned on in the packet and secondly, what is this packet's destination. If the extra token bit is not on (meaning that the packet is not yet carrying an extra token to a random destination), then the program looks at the array element of the destination of the packet seeking any waiting token. If the number in the array element is greater than 0, a bit is turned on in the packet indicating that this packet now carries an extra token; and the element of the array is decremented by one. These packets are then protected like the closevc packets because if one of these is lost, the token is lost forever. There are other means of assuring against lost tokens, but they would take up too much processing time and were deemed unnecessary for the few times this might happen.

3.2.5.2 Scheme 2 - Choke Packets

The choke packet scheme is activated solely by message packets or requests, not acknowledgements or call rejects. Whenever one of the first two is put on an output queue, the number of queue elements is compared to a warning length established by the user; this should be 50 percent or less of its maximum length. If the queue is now above that value, then a choke packet is sent back to the host telling it to slow the requests to the destination of that message packet or request.

Each node has an array with a rate reduction factor initialized to zero for each of the other nodes in the network. When the host acquires the choke packet, it sets the array element for the sink to the user-defined rate reduction factor (RRF), for example 30 percent. To illustrate what happens, look at a three-node network; the destination table of node 0 is 1, 70, 100. Node 0 will want to send 70 percent of the time to node 1 and the other 30 percent to node 2. Assume the random number generator produced a 68. Normally that would mean the host would send to node 1 since it is under the 70 percent mark; but the question as to whether or not the request will be allowed is based on the choke factor. Take the rate reduction factor, multiply it by the 70 and obtain a new comparison. Now 68 is greater than 49 (70 - [70*.3] = 49); so in this case node 0 will not send a request. No other destinations are considered or new routes examined -- the request process is terminated at this point. If the random number had been 45, node 0 would still send since 45 is less than 49. Restated another way, the sending process determines what node it would normally send to and then applies the choke factor. If the selection is still valid, a request is sent; otherwise, the requesting process is terminated.

The next step after the host receives a choke packet is to ignore all other choke packets with the same destination for a user-defined length of time. After that time is up, the host listens for additional chokes with the same destination. If, during that listen time, the host does not receive any more chokes, it will set the RRF for the destination back to zero. However, if within the specified listen time (again, user defined) the host receives another choke packet, it will increase the rate reduction factor by the formula (RRF=RRF*1.5). In our example, the RRF for destination node 1 starts at zero. A choke packet comes in and the RRF is set to .30. If, after the ignore period is over but the listening period is still on, another choke comes in the RRF would be raised to .45. Again, if after the ignore time is up and still another choke packet arrives, the RRF would be raised to .675 and so on until the RRF reaches 1.00.

3.2.5.3 Scheme 3 - Flow Balancing

In this scheme each output queue has a special hold queue associated with it. As each call accept is received, it is put on this queue. In the case where the node is the source node, the accept is transmitted to the host process and this step is bypassed.

Every time a message packet is successfully sent from an output buffer, a variable called credit count [Kermani 82] is incremented by one. There is a threshold limit attached to the output queue. If the number of packets in the output queue rises above the threshold [Kermani 82], no accepts are allowed to proceed to the source node and the credit count is set back to zero. Presuming the count of packets is below the threshold, then this credit count is compared to the message size of the accept packet on the front of the hold queue. If the credit count is greater than the size, then the accept is passed along and the credit count is decremented by the message size of the accept packet.

The threshold limit must be one greater than the largest message size but one less than the output queue length. To work as a congestion control mechanism, the threshold limit must be lower than the queue length. The threshold has to be one larger than the largest message size to ensure that a call accept on the hold response queue whose message is of the maximum size will be able to be released. Consider what would happen if the threshold was not greater than the message size. Say threshold was set to 10 and the message size to 11. If the output queue length reached the threshold count the credits would be set to 0. Now, if the call accept at the front of the hold queue contained an accept for an 11 count message the credit count would never get that high thus freezing up the hold response queue.

However, should the output queue be empty and accepts are waiting on the hold queue, the accept packet in the front of the hold queue is released and a clock started. If the clock counts to ten and still accepts are waiting and the output queue is still empty, another accept is sent and the clock reset. Such a process continues until there are either no more accepts or the output queue begins filling.

If the accept packet that has just been released cannot be put on the correct output queue, then it is reassigned to the back of the hold queue where the accept packet must start its wait all over again. The credit count is decremented only after the accept packet has been successfully placed in an output queue.

CHAPTER 4

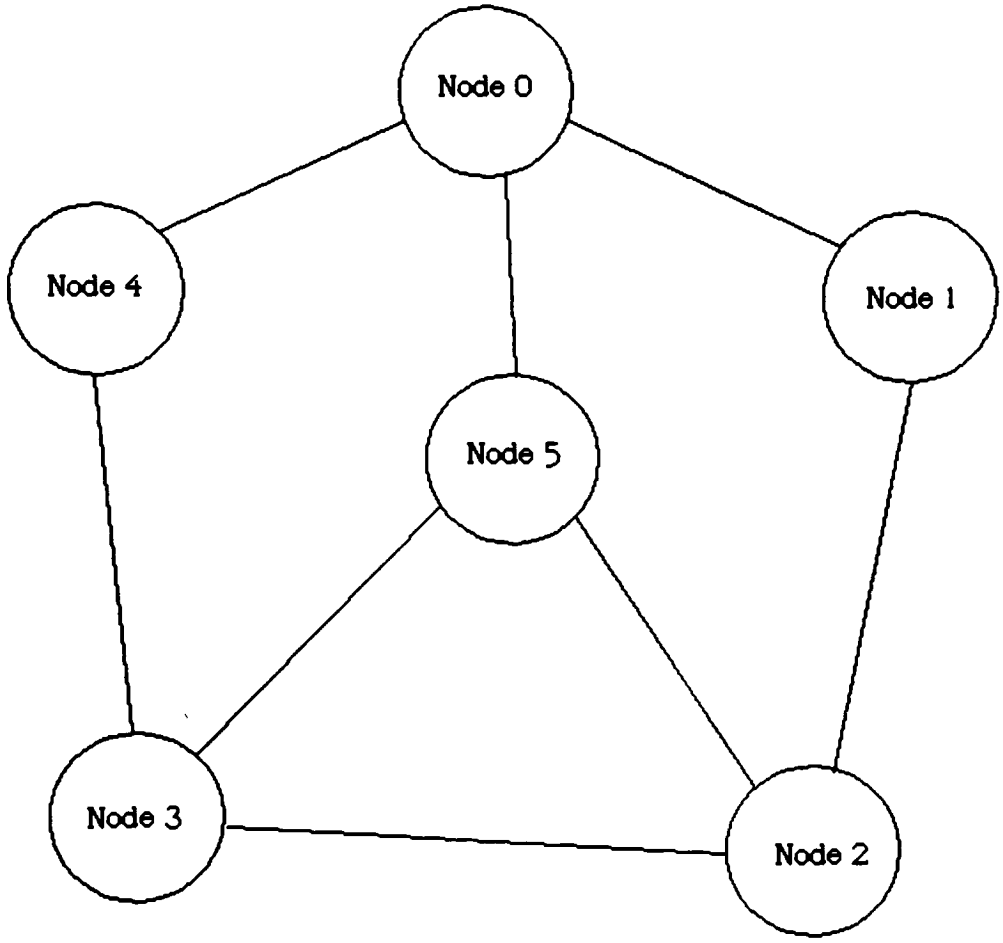
ANALYSIS OF RESULTS

To analyze the results of this simulation process, three example networks were constructed: the first was a six-node network as shown in Figure 7; the second network is the same six-node network with the line between node 0 and node 1 disconnected as shown in Figure 8; the third example is a ten-node network connected as shown in Figure 9. For each of these networks, four simulation runs were performed, one for every congestion control scheme and the fourth to show what happens without a control scheme. Each run lasted for 20,000 clockticks. While the results discussed in this chapter are primarily descriptive, listings of the actual output files outlining specific results are located in Appendix C. Additionally, Appendix B contains the input data files used in constructing each simulation run. The SUMMARY and MONITOR output files that show the condition of the network during the run are found in Appendix C.

The output file, SUMMARY, holds a short recapitulation of the finished message count, number of packets lost, and number of requests and packets discarded. The summary has been written to this file every 500 clockticks, as specified by the user in the variable "More Outputs," and has been broken down further to running totals and totals that have accumulated since the last write. Examining the last row under the RUNNING TOTALS columns shows the final totals at the end of the simulation run.

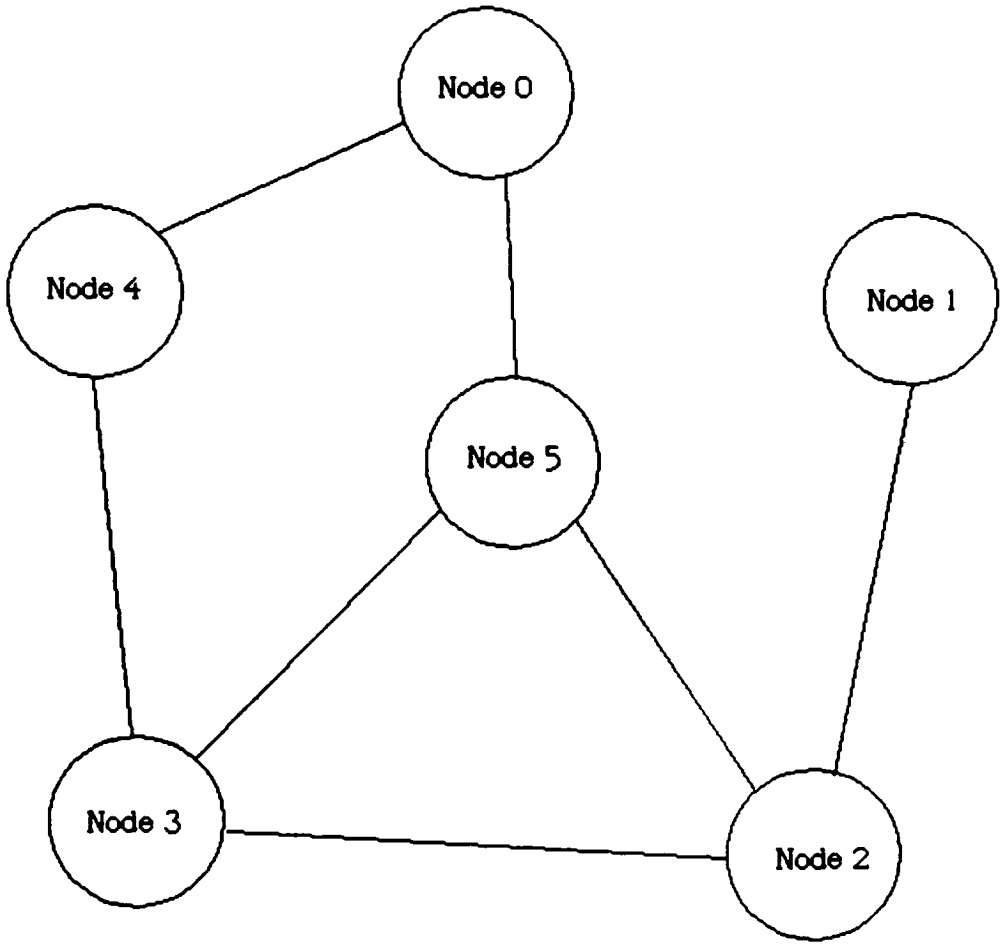
The output file, MONITOR, holds a listing of the state of the network. This output file has also been written to every 500 clockticks; however, to save space in this document, only the last write to this file is shown in Appendix C. For each node MONITOR lists the running totals of packets and messages. The file also describes the status of each queue at the present clocktick.

With the virtual circuit table capable of holding 30 entries, the maximum outstanding message count at 10, the buffer pool size set at 20, maximum queue length specified as 15, packet discarding set to 1 (packets are discarded if unable to be forwarded) and moderate sending rates (50, 30, 20, 40, 25, 35 for a six-node network), the results were relatively predicable. Based on the throughputs, as written out to the file SUMMARY, the poorest performer was the non-control scheme simulation run. It generally discarded a



Six-node network

Figure 7



Six-node network with one line down

Figure 8

Ten-node network

Figure 9

substantial number of packets and also, except for one instance, discarded the most requests. In that one instance the flow balancing scheme had 1,232 requests discarded, but only threw away 738 packets as compared to the non-control scheme, which discarded 979 requests and 5,133 packets. Due to this great loss, the non-control scheme has a very high frequency of retransmissions, adding to the general congestion problem.

Of the three control schemes, the worst control scheme is the choke packet scheme. The control scheme is effective in preventing deadlock, but at the expense of additional packets that needed to be produced to send the choke message back to the host. The choke packet scheme and the non-control scheme both lose a copious number of packets. The difference in the throughput is that in the choke packet scheme some of those packets are choke packets that are not retransmitted. The second worst control scheme is the token scheme. It lost the fewest packets and requests, but is held back in reaching its full potential by the method used in this simulation for forwarding the spare tokens to a random node. To get better results, the tokens could have been forwarded in packets by themselves after a certain time limit had been reached. This method was not implemented in the simulation, however, due to the overhead it would generate. This implementation proved to be a serious flaw that goes unnoticed when certain parameters are used or with certain network topology.

The best congestion control scheme is the flow balancing control scheme since it is the most custom tailored and lets the intermediate nodes control the flow that passes through them. Each of these intermediate nodes can slow the traffic through them as much as needed without affecting the other nodes to any great degree.

The ten-node network was then exercised to see if the results were constant or would vary when parameters where changed. The results were fascinating and at times unexpected. With a larger buffer pool, longer queue length, and shorter virtual circuit table, the choke scheme had the best throughput. The non-control scheme came in second; the flow balancing scheme came in third; the token scheme placed last. The token scheme froze due to an unexpected design flaw.

When node A cannot use an arriving token, a random number generator determines another node to which the token can be sent. Node A then places it upon a special queue to wait for a message packet headed towards the same destination. The token would then piggy-back its way with the message packet to the chosen destination. If no packets arrive scheduled for that destination, then the token just sits in the queue. The network essentially loses the token. As tokens build up in the queues, fewer message packets can be sent. Since there are no tokens, no message packets can be sent, just call requests. Soon either the virtual circuit tables fill up or all nodes have reached the

maximum outstanding messages they can have open, thereby effectively freezing the network. This error was noticed too late in the final phases of delivery to be corrected. One method to correct this situation would be to install a time-out clock whereby if a token sat in the queue for a certain length of time, it would then either piggy-back its way on a message packet to any destination or set up its own virtual circuit to the token's proper destination node. One potential problem with this correction might be that the processing time needed for this overhead procedure (refer to Chapter 5) would send the simulation into a lower priority situation creating excessive run times on multi-user systems. By careful design of the parameters while using the token scheme, the problems of lost tokens can be circumvented. Another possibility would be to see if the current node could use the token and if it could not, then to send it to any connecting node to determine if the token could be used there.

Aside from the token scheme's flaw, the rest of the results depend on which parameter was changed. As expected the Packet Discard variable had the greatest effect on the outcome. When packets are discarded, the network is really incapable of deadlock since the packets can always be sent from the output buffer. Remember with the Packet Discard flag set, if a packet cannot be forwarded, it is simply discarded. When the packets are not discarded, greater care is necessary in selecting the parameters, especially the queue length, free buffer pool, and maximum number of outstanding messages owned by a node. It becomes imperative to have enough buffers and long enough queues to allow the control mechanisms a chance to return to the senders. With packet discarding these control packets would eventually be sent along the virtual circuit. When the packets are not discarded, the queue can be frozen before these packets reach the front of the queues. Generally speaking however, the difference in performance when packets are being discarded, is that while the non-control scheme will not freeze, the control schemes can yield better results.

When packets are not discarded, then each control scheme must have its parameters modified to optimize the control scheme or sometimes to even have it work at all. An example of this, as mentioned above, is the token scheme.

To see what happens with different parameters, the choke scheme was run with a warning queue length of 50 percent and then 80 percent. With the warning length set at 50 percent, the number of messages completed reached 1,663. However, with the warning length set up to 80 percent, the completed messages dropped to 1,539.

Traffic rates also make a difference. When running the token scheme and packets are not being discarded, and the token store is set at eight, the traffic rates of the network was set at 20 percent, 50 percent, and then 80 percent for all nodes. At 20 percent

1,411 messages were completed, at 50 percent 1,377 were finished, and at 80 percent only 1,281 were processed.

The difference in performance between these control schemes and the non-control method is that control schemes can have certain control parameters changed to give optimal results. By changing specific parameters, the control schemes can be "tweaked" into giving a better performance. The interesting aspect to all this is the interrelationship between all the parameters and the number of permutations that can be devised. This will be left to the experimentation of future users.

In the final analysis this simulation proved to be a useful tool for data communication students wishing to learn more about congestion control schemes and how they affect a network. Data communication students will learn why having a congestion control scheme built into the network is important when they design a network and use the simulation to run it. This teaching tool will give the user the hands-on experience of designing a network and then modifying it to get the best results. Moreover, the hands-on experience they receive by creating and solving problems associated with networks will aid their understanding of challenges in the data communication field. The user will hopefully come away with a better understanding of these three congestion control schemes, how they work, and how they can affect a network to make it more productive with greater throughput and a minimal delay time.

CHAPTER 5

PROBLEMS ENCOUNTERED IN DEVELOPMENT OF THE SIMULATION

When developing almost any type of model or simulation, challenges occur as a matter of course. Basically, four main problems were encountered in developing this thesis. First, the simulation took an inordinate amount of time to run to completion. Secondly, unexpected details emerged and had to be contended with efficiently. Thirdly, there was an ever present temptation to keep adding major revisions throughout the project's developmental stages. Lastly, as a rather unique project, not a great deal of data is readily available for information and insights.

The greatest problem encountered in the development of this simulation was the great length of time that it took to run the simulation. Depending on the number of users on the system, compiling the program ranged anywhere from 13 to 20 minutes and, at one time, over three hours to run. This made development extremely slow as any minor change would not appear until the end of the next run. Apparently, a timing mechanism in the compiler examines the amount of time a process is using and lowers a particular process's priority should it use too much cpu time. Removing a single Unix memory allocation call lowered the run time to a usable level.

Many unexpected details appeared and had to be dealt with before the simulation could function properly. Each control scheme had its own unique problems. For example, one problem with the token control scheme was that extra tokens accumulated at the receiving end of a virtual circuit. Also unique with the token scheme was what happened when a packet was lost that also contained a token. The choke packet scheme had several problems that appeared as the simulation was developed. First, the control scheme needed a much lower warning limit for the queues; and secondly, what would happen should the choke packet not be placed on a queue back towards the source? Common with all schemes, including the non-control scheme, was a situation created when a request is routed back onto itself before it reaches the receiver. These problems were not foreseen during the planning stages of this project and were dealt with by either making assumptions or rewriting the source code to solve the problems.

It was easy adding additional functionality to this project. As in any programming project, the more options given the user, the more involved the code becomes, as well as the instructions for running the program. It was difficult to strike a medium where the user would be able to have a meaningful and flexible program but would not be overwhelmed by all the choices and options.

The final problem in the development of this thesis was how little information was available on modeling of congestion control schemes. The library search provided nothing regarding simulations with multiple congestion control schemes. Most networks are tailored for one control scheme at conception, which makes the networks easier to implement, as well as simulate.

The problems stated here were overcome one at a time as they materialized. The solutions to these problems have been incorporated into other chapters of this thesis, either under the implementation section or the project overview section. It has been a real challenge; and hopefully, the thesis will provide a base for other uses.

CHAPTER 6

FUTURE POSSIBILITIES FOR OTHER THESIS WORKS

When discussing future possibilities, the first item that comes to mind is the addition of other congestion control schemes. The preallocation of resources would be a good choice. However, simply adding control schemes is not enough to make up a thesis, although it might be considered as a project along with other possible modifications.

Other modifications that could be added, in addition to other congestion control schemes, could be a sliding window protocol or different methods to handle tokens, choke packets, or request processes. Another possibility could be speed enhancements to decrease the simulation's run time. Increased versatility can be incorporated into this simulation; however, it would make the user face that many more choices and increase run time.

The addition of dynamic routing to this network monitoring simulation might make a good thesis project for a graduate student in the future. The dynamic routing algorithm could take care of node crashes and perhaps use a 'best path' method. This method allows a node to send messages along the best path until the load became too high and then switches to alternatives channels. The developer could use this thesis program as a starting point and just concentrate on constructing the routing algorithms letting this simulation handle the user interface, packet handling, and the network monitoring.

APPENDIX A

The Cflow Diagram of the Simulation

This is the Cflow diagram of the simulation. It contains all the simulation function calls. The regular C functions like printf, fprintf, have been excluded. The indentation gives the reader an idea of parent/child relationships. Those indented farther over are the children of those above them. For further clarification of what function calls what others, refer to Table 1.

```
main: int(), <main.c 13>
    CATCHER: void(), <catcher.c 6>
    GETINPUTS: int(), <input.c 6>
     GETNUMBERS: int(), <input.c 138>
     GETTABLES: int(), <input.c 15>
       SUBROUT: int(), <input.c 77>
       ALLOCATE: int(), <input.c 311>
    NETINIT: int(), <init.c 8>
     GETTIME: int(), <func1.c 105>
       POISSON: int(), <func1.c 300>
     INITNODE: int(), <init.c 137>
     INITHOST: int(), <init.c 187>
       INITMESS: int(), <init.c 202>
     INITINBUF: int(), <init.c 78>
       INITPACKET: int(), <init.c 102>
     INITOUTBUF: int(), <init.c 92>
       INITPACKET: 57
     INITQUE: int(), <init.c 234>
       INITQUEELE: int(), <init.c 256>
            INITPACKET: 57
    NETSETUP: int(), <init.c 41>
    TIMEOUTPK: int(), <func5.c 7>
    EMPTYSLOT: int(), <func2.c 236>
    TIMEOUTREQ: int(), <func5.c 87>
```

```
 CHII: int(), <func5.c 159>
 INITINBUF: 56
CHECKHOLD: int(), <func4.c 51>
 PRINTPACKET: int(), <print.c 56>
 GETHOST: int(), <func7.c 234>
WANTTOSEND: int(), <func1.c 127>
 INITINBUF: 56
 GETTIME: 46
 DESTINATION: int(), <func1.c 255>
   RANDOM: int(), <func1.c 310>
 GETMESSIZE: int(), <func1.c 238>
   RANDOM: 89
POLLINBUFS: int(), <main.c 357>
 ASSIGN_P: int(), <func6.c 7>
   SWITCHPAK: int(), <func2.c 254>
 GETHOST: 81
 INITINBUF: 56
 FIXI: int(), <main.c 565>
 REMOVEPACKET: int(), <main.c 824>
   CHOKEHOST: int(), <scheme2.c 97>
        REDUCERATE: int(), <scheme2.c 116>
   GETHOST: 81
   LOOKUP: int(), <func2.c 36>
   PUTONHOLD: int(), <func4.c 7>
        SWITCHPAK: 107
        PRINTPACKET: 79
   INITINBUF: 56
   PIBUF: int(), <print.c 27>
        PRINTPACKET: 79
   CHECKTABLEIII: int(), <func2.c 7>
        EMPTYSLOT: 67
   INSTALL: int(), <func2.c 86>
        EMPTYSLOT: 67
        TEMPARR: int(), <func2.c 157>
                CHECKARRAY: int(), <func2.c 199>
```

PVCTABLE: int(), <print.c 157>
SENDCALLACEPT: int(), <func7.c 129>
SENDTOKEN: int(), <scheme1.c 8>
GETTOKENDEST: int(), <scheme1.c 77>
AAA: int(), <scheme1.c 148>
SPOTFORTOK: int(), <scheme1.c 169>
GOTSPOT: int(), <scheme1.c 238>
SENDACK: int(), <func7.c 107>
PVCTABLE: 146
CKTOKARR: int(), <scheme1.c 287>
HOLDGAHR: int(), <scheme3.c 7>
SWITCHPAK: 107
WHATINPAK: int(), <func6.c 93>
SENDALONG: int(), <func3.c 10>
FREEBUF: int(), <func3.c 573>
CHECKTABLEIII: 132
INITINBUF: 56
GETNEXTNODE: int(), <func1.c 12>
RANDOM: 89
INSTALL: 135
PHOST: int(), <print.c 217>
EMPTYSLOT: 67
PUTONHOLD: 121
WHATINPAK: 173
CHG: int(), <scheme3.c 308>
REVERSETABLE: int(), <func2.c 54>
PUTONQUE: int(), <func3.c 595>
ISQUEEMPTY: int(), <func3.c 664>
GOTOCHOKE: int(), <scheme2.c 9>
PUTONCHOKE: int(), <scheme2.c 36>
CHOKEHOST: 113
ISQUEEMPTY: 204
FREEBUF: 178
INITQUEELE: 61

PVCTABLE: 146
PDCARD: int(), <func3.c 497>
PUTONHOLD: 121
WHATINPAK: 173
INITINBUF: 56
TAKEOUTTOK: int(), <func3.c 691>
PRINTPACKET: 79
LOOKTOSEND: int(), <func7.c 8>
NODESENDING: int(), <func7.c 197>
OKAYSEND: int(), <func7.c 258>
CHECKTABLEIII: 132
ISQUEEMPTY: 204
SENDMESSAGE: int(), <func7.c 147>
INITPACKET: 57
PHOST: 191
CKTOKARR: 166
SENDALONG: 176
SPHOST: int(), <print.c 254>
PUTONHOP: int(), <func4.c 198>
INITMESS: 55
CHECKHOLDGAHR: int(), <scheme3.c 51>
GETHGAR: int(), <scheme3.c 130>
SWITCHPAK: 107
WHATINPAK: 173
PRINTPAK: int(), <print.c 93>
GETHOST: 81
SENDALONGII: int(), <scheme3.c 222>
FREEBUF: 178
REVERSETABLE: 198
WHATINPAK: 173
PUTONQUE: 200
PRINTPAK: 254
HOLDGAHR: 168
SEND: int(), <main.c 579>

```
      LOADOUTBUFS: int(), <main.c 684>
        PUTONFREEZEQUE: int(), <func4.c 144>
        WHATINPAK: 173
        INITQUEELE: 61
      WHATINPAK: 173
      INITOUTBUF: 58
      POBUF: int(), <print.c 44>
        PRINTPACKET: 79
    SPECIALPVCT: int(), <print.c 184>
    OUTPUTS: int(), <func1.c 321>
    WHATINQUES: int(), <func6.c 25>
      WHATINPAK: 173
      WHATININBUFS: int(), <func6.c 60>
        WHATINPAK: 173
    WRITEHOP: int(), <func1.c 497>
    CLOSE: int(), <func1.c 557>
PNODE: int(), <print.c 129>
```

APPENDIX B

Examples of User Input Data Files

Data files for the input to a six node network, all the connections between the nodes are up and working. Refer to Figure 7 to see a diagram of this network. These files are labeled by the congestion control scheme number.

SCHEME 0 - no control scheme

MAXNODES
6
Maxconnect
3
Maxticks
20000
Maxvc
30
Maxoutstanding Mess
10
Buffer pool size
20
Maxque length
15
Waitlimit
130
Percent of packets lost
.01
PAKDISCARD
1
Congestion scheme
0
MAXSTORE
5

WarnPercent for quelen
0
IgnoreTime
0
Listen Time
0
Rate Reduction Factor
0
More Outputs
500
Want Printouts
0
Time to Start Printing
19998
Static Routing Table
Node 0
1,90,1,100,5
2,70,1,100,5
3,50,4,100,5
4,85,4,100,5
5,60,5,100,1
Node 1
0,90,0,100,2
2,80,2,100,0
3,85,2,100,0
4,50,2,100,0
5,40,0,100,2
Node 2
0,95,1,100,5
1,85,1,100,5
3,75,3,100,5
4,80,3,100,1
5,90,5,100,1

Node 3
0,50,4,100,5
1,10,4,100,2
2,80,2,100,5
4,90,4,100,5
5,75,5,100,2
Node 4
0,95,0,100,3
1,85,0,100,3
2,60,3,100,0
3,80,3,100,0
5,50,3,100,0
Node 5
0,90,0,100,2
1,50,0,100,2
2,80,2,100,3
3,90,3,100,2
4,60,0,100,3
Destination Selection
1,20,40,60,80,100
40,1,60,70,90,100
30,40,1,65,75,100
25,50,60,1,75,100
10,20,45,75,1,100
50,55,65,85,100,1
Traffic Rates
50,30,20,40,25,35
Adjacency Table
0,1,4,5
1,0,2,-1
2,1,3,5
3,2,4,5
4,0,3,-1
5,0,2,3
Number of packets
3,30,6,60,9,100

```
/*********************************************************************/
SCHEME 1    -    token scheme

MAXNODES
6
Maxconnect
3
Maxticks
20000
Maxvc
30
Maxoutstanding Mess
10
Buffer pool size
20
Maxque length
15
Waitlimit
130
Percent of packets lost
.01
PAKDISCARD
1
Congestion scheme
1
MAXSTORE
5
WarnPercent for quelen
0
IgnoreTime
0
Listen Time
0
Rate Reduction Factor
0
```

More Outputs

500

Want Printouts

0

Time to Start Printing

20

Since the static routing table, traffic rates, destination selection, adjaceny table, and the number of packets are the same as in data6.old they have been deleted here.

```
/*************************************************************************/
SCHEME 2        choke packet scheme

MAXNODES
6
Maxconnect
3
Maxticks
20000
Maxvc
30
Maxoutstanding Mess
10
Buffer pool size
20
Maxque length
15
Waitlimit
130
Percent of packets lost
.01
PAKDISCARD
1
Congestion scheme
2
MAXSTORE
5
WarnPercent for quelen
0.3
IgnoreTime
40
Listen Time
120
Rate Reduction Factor
0.6
```

More Outputs

500

Want Printouts

0

Time to Start Printing

0

Since the static routing table, traffic rates, destination selection, adjaceny table, and the number of packets are the same as in data6.old they have been deleted here.

```
/*********************************************************************/
```

SCHEME 3 - flow balancing scheme

```
MAXNODES
6
Maxconnect
3
Maxticks
20000
Maxvc
30
Maxoutstanding Mess
10
Buffer pool size
20
Maxque length
15
Waitlimit
130
Percent of packets lost
.01
PAKDISCARD
1
Congestion scheme
3
MAXSTORE
5
WarnPercent for quelen
0
IgnoreTime
0
Listen Time
0
Rate Reduction Factor
0
```

More Outputs

500

Want Printouts

0

Time to Start Printing

0

Since the static routing table, traffic rates, destination selection, adjaceny table, and the number of packets are the same as in data6.old they have been deleted here.

Data files for a six node network with one line down between node 0 and node 1
See figure X for reference

SCHEME 0 no control scheme

MAXNODES
6
Maxconnect
3
Maxticks
20000
Maxvc
30
Maxoutstanding Mess
10
Buffer pool size
20
Maxque length
15
Waitlimit
130
Percent of packets lost
.01
PAKDISCARD
1
Congestion scheme
0
MAXSTORE
5
WarnPercent for quelen
0
IgnoreTime
0

Listen Time
0
Rate Reduction Factor
0
More Outputs
500
Want Printouts
0
Time to Start Printing
10
Static Routing Table
Node 0
1,90,5,100,4
2,70,4,100,5
3,50,4,100,5
4,85,4,100,5
5,60,5,100,4
Node 1
0,0,0,100,2
2,0,0,100,2
3,0,0,100,2
4,0,0,100,2
5,0,0,100,2
Node 2
0,95,5,100,3
1,85,1,100,5
3,75,3,100,5
4,80,3,100,5
5,90,5,100,3
Node 3
0,50,4,100,5
1,10,4,100,2
2,80,2,100,5
4,90,4,100,5
5,75,5,100,2

Node 4

0,95,0,100,3

1,85,0,100,3

2,60,3,100,0

3,80,3,100,0

5,50,3,100,0

Node 5

0,90,0,100,2

1,10,3,100,2

2,80,2,100,3

3,90,3,100,2

4,60,0,100,3

Destination Selection

1,20,40,60,80,100

40,1,60,70,90,100

30,40,1,65,75,100

25,50,60,1,75,100

10,20,45,75,1,100

50,55,65,85,100,1

Traffic Rates

50,30,20,40,25,35

Adjacency Table

0,4,5,-1

1,2,-1,-1

2,1,3,5

3,2,4,5

4,0,3,-1

5,0,2,3

Number of packets

3,30,6,60,9,100

```
/*********************************************************/
SCHEME 1        token scheme

MAXNODES
6
Maxconnect
3
Maxticks
20000
Maxvc
30
Maxoutstanding Mess
10
Buffer pool size
20
Maxque length
15
Waitlimit
130
Percent of packets lost
.01
PAKDISCARD
1
Congestion scheme
1
MAXSTORE
5
WarnPercent for quelen
0
IgnoreTime
0
```

Listen Time

0

Rate Reduction Factor

0

More Outputs

500

Want Printouts

0

Time to Start Printing

20

Since the static routing table, traffic rates, destination selection, adjaceny table, and the number of packets are the same as in data6.d they have been deleted here.

```
/*******************************************************/
SCHEME 2    -   choke packet scheme

MAXNODES
6
Maxconnect
3
Maxticks
20000
Maxvc
30
Maxoutstanding Mess
10
Buffer pool size
20
Maxque length
15
Waitlimit
130
```

Percent of packets lost

.01

PAKDISCARD

1

Congestion scheme

2

MAXSTORE

5

WarnPercent for quelen

.3

IgnoreTime

40

Listen Time

120

Rate Reduction Factor

.60

More Outputs

500

Want Printouts

0

Time to Start Printing

2875

Since the static routing table, traffic rates, destination selection, adjaceny table, and the number of packets are the same as in data6.d they have been deleted here.

```
/*********************************************************/
```

SCHEME 3 - flow balancing scheme

```
MAXNODES
6
Maxconnect
3
Maxticks
20000
Maxvc
30
Maxoutstanding Mess
10
Buffer pool size
20
Maxque length
15
Waitlimit
130
Percent of packets lost
.01
PAKDISCARD
1
Congestion scheme
3
MAXSTORE
5
WarnPercent for quelen
0
IgnoreTime
0
Listen Time
0
```

Rate Reduction Factor

0

More Outputs

500

Want Printouts

0

Time to Start Printing

2875

Since the static routing table, traffic rates, destination selection, adjaceny table, and the number of packets are the same as in data6.d they have been deleted here.

Data input files for a 10 node network all connections up and working. See figure X for reference.

SCHEME 0 - no control scheme

MAXNODES
10
Maxconnect
4
Maxticks
20000
Maxvc
30
Maxoutstanding Mess
10
Buffer pool size
20
Maxque length
15
Waitlimit
300
Percent of packets lost
.01
PAKDISCARD
1
Congestion scheme
0
MAXSTORE
5
WarnPercent for quelen
0
IgnoreTime
0

Listen Time
0
Rate Reduction Factor
0
More Outputs
500
Want Printouts
0
Time to Start Printing
2875
Static Routing Table
Node 0
1,95,1,100,9
2,90,1,100,9
3,92,1,100,9
4,85,1,100,9
5,85,9,100,1
6,92,9,100,1
7,90,9,100,1
8,95,9,100,1
9,95,9,100,1
Node 1
0,95,0,100,2
2,60,2,100,3
3,75,3,100,2
4,50,2,100,3
5,60,2,100,3
6,65,3,100,2
7,50,3,100,2
8,75,0,100,2
9,85,0,100,3

Node 2
0,85,1,100,3
1,50,1,100,3
3,60,3,100,4
4,75,4,100,3
5,80,4,100,3
6,85,4,100,3
7,95,4,100,3
8,50,4,100,1
9,90,1,100,3
Node 3
0,50,1,100,2
1,60,1,100,2
2,85,2,100,4
4,90,4,100,1
5,93,4,100,2
6,80,4,100,2
7,80,4,100,1
8,75,1,100,2
9,95,1,100,2
Node 4
0,50,3,100,2
1,50,3,100,2
2,90,2,100,3
3,95,3,100,2
5,95,5,100,2
6,90,5,100,2
7,90,5,100,2
8,75,2,100,5
9,85,2,100,5

Node 5
0,70,4,100,7
1,80,4,100,7
2,90,4,100,7
3,90,4,100,7
4,95,4,100,7
6,95,6,100,7
7,90,7,100,6
8,90,8,100,6
9,80,8,100,7
Node 6
0,50,5,100,7
1,55,5,100,7
2,60,5,100,7
3,65,5,100,7
4,70,5,100,7
5,75,5,100,7
7,80,5,100,7
8,85,5,100,7
9,90,5,100,7
Node 7
0,25,5,100,8
1,30,5,100,8
2,35,5,100,8
3,40,5,100,6
4,50,5,100,6
5,90,5,100,6
6,80,6,100,5
8,96,8,100,6
9,50,8,100,5

Node 8

0,76,9,100,5

1,45,9,100,5

2,50,7,100,9

3,50,9,100,5

4,85,5,100,7

5,60,5,100,7

6,50,5,100,7

7,90,7,100,5

9,92,9,100,5

Node 9

0,95,0,100,8

1,87,0,100,8

2,75,0,100,8

3,65,0,100,8

4,50,0,100,8

5,40,0,100,8

6,30,0,100,8

7,10,0,100,8

8,5,0,100,8

Destination Selection

1,10,20,30,40,50,60,70,80,100

20,1,25,40,50,65,70,79,95,100

50,70,1,80,85,90,93,95,97,100

24,34,45,1,65,74,80,87,95,100

12,23,35,44,1,49,55,70,90,100

19,33,41,64,77,1,86,93,97,100

9,16,28,40,56,77,1,87,95,100

5,15,35,40,56,69,77,1,82,100

30,45,55,60,64,70,75,80,1,100

32,44,58,62,70,76,83,94,100,1

Traffic Rates

50,30,20,40,25,35,15,20,28,38

Adjacency Table

0,1,9,-1,-1

1,0,2,3,-1

2,1,3,4,-1

3,1,2,4,-1

4,2,3,5,-1

5,4,6,7,8

6,5,7,-1,-1

7,5,6,8,-1

8,5,7,9,-1

9,0,8,-1,-1

Number of packets

3,30,6,60,9,100

****

SCHEME 1 token scheme

MAXNODES
10
Maxconnect
4
Maxticks
20000
Maxvc
30
Maxoutstanding Mess
10
Buffer pool size
20
Maxque length
15
Waitlimit
300
Percent of packets lost
.01
PAKDISCARD
1
Congestion scheme
1
MAXSTORE
5
WarnPercent for quelen
0
IgnoreTime
0
Listen Time
0
Rate Reduction Factor
0

More Outputs

500

Want Printouts

0

Time to Start Printing

2875

Since the static routing table, destination selection, traffic rates, adjacency table and the nnmber of packets are the same as for data10.d they are not included here.

Static Routing Table

Destination Selection

Traffic Rates

Adjacency Table

Number of packets

****

SCHEME 2 - choke packet scheme

MAXNODES
10
Maxconnect
4
Maxticks
20000
Maxvc
30
Maxoutstanding Mess
10
Buffer pool size
20
Maxque length
15
Waitlimit
300
Percent of packets lost
.01
PAKDISCARD
1
Congestion scheme
2
MAXSTORE
5
WarnPercent for quelen
0.30
IgnoreTime
40

Listen Time

120

Rate Reduction Factor

0.60

More Outputs

500

Want Printouts

0

Time to Start Printing

2875

Since the static routing table, destination selection, traffic rates, adjacency table and the number of packets are the same as for data10.d they are not included here.

Static Routing Table

Destination Selection

Traffic Rates

Adjacency Table

Number of packets

****

SCHEME 3 flow balancing scheme

MAXNODES
10
Maxconnect
4
Maxticks
20000
Maxvc
30
Maxoutstanding Mess
10
Buffer pool size
20
Maxque length
15
Waitlimit
300
Percent of packets lost
.01
PAKDISCARD
1
Congestion scheme
3
MAXSTORE
5
WarnPercent for quelen
0
IgnoreTime
0
Listen Time
0
Rate Reduction Factor
0

More Outputs
500
Want Printouts
0
Time to Start Printing
17235
Since the static routing table, destination selection, traffic rates, adjacency table and the number of packets are the same as for data10.d they are not included here.
Static Routing Table
Destination Selection
Traffic Rates
Adjacency Table
Number of packets

APPENDIX C

1. EXAMPLES OF SUMMARY FILES
2. EXAMPLE OF A MONITOR FILE
3. EXAMPLE OF A HOPS FILE
4. EXAMPLE OF AN ERROR LOG FILE
5. EXAMPLE OF A LOST PACKETS FILE
6. EXAMPLE OF A DISCARDED PACKETS FILE
7. EXAMPLE OF A NODE FILE USED IN PRINTOUT SCHEME 1

EXAMPLES OF SUMMARY OUTPUT FILES

6 NODE NETWORK SCHEME IS 0

maxquelen is 15, buffer pool is 20, maxvc is 30 and messoutstanding is 10

CLOCKTICK	RUNNING TOTALS				SINCE LAST WRITE			
	Message Count	Paks Lost	Reqs Discard	Paks Discard	Message Count	Paks Lost	Reqs Discard	Paks Discard
500	64	36	9	231	64	36	9	231
1000	150	71	25	448	86	35	16	217
1500	211	103	37	674	61	32	12	226
2000	293	142	48	813	82	39	11	139
2500	370	174	70	1059	77	32	22	246
3000	432	215	82	1302	62	41	12	243
3500	501	239	97	1518	69	24	15	216
4000	579	269	112	1719	78	30	15	201
4500	660	299	131	1905	81	30	19	186
5000	730	326	141	2083	70	27	10	178
5500	816	354	157	2280	86	28	16	197
6000	886	387	175	2456	70	33	18	176
6500	955	416	190	2674	69	29	15	218
7000	1031	442	206	2862	76	26	16	188
7500	1091	466	224	3031	60	24	18	169
8000	1157	501	249	3209	66	35	25	178
8500	1223	523	257	3440	66	22	8	231
9000	1282	557	266	3683	59	34	9	243
9500	1370	594	280	3820	88	37	14	137
10000	1448	622	297	4012	78	28	17	192
10500	1527	645	302	4228	79	23	5	216
11000	1588	683	324	4479	61	38	22	251
11500	1673	719	333	4624	85	36	9	145
12000	1764	750	346	4906	91	31	13	282
12500	1819	786	356	5189	55	36	10	283
13000	1888	815	370	5409	69	29	14	220
13500	1965	851	384	5600	77	36	14	191
14000	2045	878	401	5783	80	27	17	183
14500	2118	907	414	6001	73	29	13	218
15000	2182	940	431	6281	64	33	17	280
15500	2247	964	441	6526	65	24	10	245
16000	2325	986	455	6730	78	22	14	204
16500	2406	1014	469	6897	81	28	14	167
17000	2477	1047	485	7133	71	33	16	236
17500	2538	1070	508	7409	61	23	23	276
18000	2606	1098	527	7569	68	28	19	160
18500	2685	1122	550	7784	79	24	23	215
19000	2747	1145	568	8006	62	23	18	222
19500	2817	1175	581	8244	70	30	13	238
20000	2888	1205	592	8460	71	30	11	216

6 NODE NETWORK SCHEME IS 1

maxquelen is 15, buffer pool is 20, maxvc is 30 and messoutstanding is 10

CLOCKTICK	RUNNING TOTALS				SINCE LAST WRITE			
	Message Count	Paks Lost	Reqs Discard	Paks Discard	Message Count	Paks Lost	Reqs Discard	Paks Discard
500	70	26	5	17	70	26	5	17
1000	129	40	10	87	59	14	5	70
1500	202	65	15	97	73	25	5	10
2000	268	84	17	115	66	19	2	18
2500	344	104	18	119	76	20	1	4
3000	433	129	22	130	89	25	4	11
3500	513	153	25	146	80	24	3	16
4000	610	190	26	155	97	37	1	9
4500	700	212	33	171	90	22	7	16
5000	787	242	35	172	87	30	2	1
5500	864	263	43	201	77	21	8	29
6000	932	280	50	248	68	17	7	47
6500	1023	299	53	283	91	19	3	35
7000	1106	333	56	307	83	34	3	24
7500	1193	364	60	327	87	31	4	20
8000	1286	389	63	329	93	25	3	2
8500	1382	404	64	341	96	15	1	12
9000	1480	428	65	341	98	24	1	0
9500	1553	457	70	375	73	29	5	34
10000	1643	478	74	397	90	21	4	22
10500	1716	504	78	435	73	26	4	38
11000	1797	529	81	459	81	25	3	24
11500	1895	560	83	474	98	31	2	15
12000	1960	580	88	486	65	20	5	12
12500	2045	604	91	527	85	24	3	41
13000	2123	630	98	578	78	26	7	51
13500	2201	655	104	636	78	25	6	58
14000	2289	679	107	654	88	24	3	18
14500	2398	708	111	672	109	29	4	18
15000	2500	734	116	688	102	26	5	16
15500	2573	765	117	707	73	31	1	19
16000	2661	786	120	730	88	21	3	23
16500	2752	817	121	743	91	31	1	13
17000	2824	838	123	764	72	21	2	21
17500	2896	862	126	811	72	24	3	47
18000	2989	887	128	823	93	25	2	12
18500	3083	906	129	844	94	19	1	21
19000	3156	933	132	853	73	27	3	9
19500	3248	962	135	857	92	29	3	4
20000	3334	987	142	860	86	25	7	3

6 NODE NETWORK SCHEME IS 2

maxquelen is 15, buffer pool is 20, maxvc is 30 and messoutstanding is 10

CLOCKTICK	RUNNING TOTALS				SINCE LAST WRITE			
	Message Count	Paks Lost	Reqs Discard	Paks Discard	Message Count	Paks Lost	Reqs Discard	Paks Discard
500	76	32	1	121	76	32	1	121
1000	144	73	4	215	68	41	3	94
1500	210	108	8	342	66	35	4	127
2000	273	137	11	465	63	29	3	123
2500	352	171	14	531	79	34	3	66
3000	414	207	22	742	62	36	8	211
3500	506	236	23	778	92	29	1	36
4000	596	260	23	808	90	24	0	30
4500	679	290	30	881	83	30	7	73
5000	769	313	30	907	90	23	0	26
5500	851	339	35	1017	82	26	5	110
6000	943	367	36	1069	92	28	1	52
6500	1033	387	37	1185	90	20	1	116
7000	1122	421	39	1269	89	34	2	84
7500	1192	440	41	1387	70	19	2	118
8000	1263	476	43	1479	71	36	2	92
8500	1321	511	48	1681	58	35	5	202
9000	1384	536	49	1807	63	25	1	126
9500	1461	555	50	1849	77	19	1	42
10000	1546	591	52	1975	85	36	2	126
10500	1602	620	59	2125	56	29	7	150
11000	1671	645	61	2253	69	25	2	128
11500	1755	675	61	2340	84	30	0	87
12000	1831	708	62	2447	76	33	1	107
12500	1918	739	64	2514	87	31	2	67
13000	2008	770	66	2573	90	31	2	59
13500	2078	798	69	2695	70	28	3	122
14000	2167	817	71	2735	89	19	2	40
14500	2244	844	74	2827	77	27	3	92
15000	2306	880	78	3022	62	36	4	195
15500	2376	912	83	3171	70	32	5	149
16000	2427	942	88	3388	51	30	5	217
16500	2503	964	91	3532	76	22	3	144
17000	2589	982	95	3600	86	18	4	68
17500	2648	1011	95	3731	59	29	0	131
18000	2713	1034	98	3865	65	23	3	134
18500	2762	1065	102	4049	49	31	4	184
19000	2826	1093	106	4165	64	28	4	116
19500	2905	1114	109	4311	79	21	3	146
20000	2991	1126	109	4443	86	12	0	132

6 NODE NETWORK SCHEME IS 3

maxquelen is 15, buffer pool is 20, maxvc is 30 and messoutstanding is 10

CLOCKTICK	RUNNING TOTALS				SINCE LAST WRITE			
	Message Count	Paks Lost	Reqs Discard	Paks Discard	Message Count	Paks Lost	Reqs Discard	Paks Discard
500	81	38	9	83	81	38	9	83
1000	162	71	16	199	81	33	7	116
1500	244	106	25	288	82	35	9	89
2000	318	135	37	408	74	29	12	120
2500	427	165	46	442	109	30	9	34
3000	510	202	62	549	83	37	16	107
3500	597	233	75	640	87	31	13	91
4000	665	263	86	765	68	30	11	125
4500	755	291	95	869	90	28	9	104
5000	847	316	100	942	92	25	5	73
5500	955	348	108	970	108	32	8	28
6000	1039	383	117	1063	84	35	9	93
6500	1138	414	125	1127	99	31	8	64
7000	1245	431	131	1133	107	17	6	6
7500	1355	465	136	1167	110	34	5	34
8000	1436	495	143	1201	81	30	7	34
8500	1532	527	149	1273	96	32	6	72
9000	1615	557	163	1367	83	30	14	94
9500	1712	590	166	1386	97	33	3	19
10000	1809	617	177	1462	97	27	11	76
10500	1906	659	189	1520	97	42	12	58
11000	2007	689	202	1617	101	30	13	97
11500	2103	719	220	1655	96	30	18	38
12000	2207	754	226	1689	104	35	6	34
12500	2273	785	240	1791	66	31	14	102
13000	2338	823	249	1925	65	38	9	134
13500	2407	853	258	2031	69	30	9	106
14000	2499	882	265	2046	92	29	7	15
14500	2588	912	272	2072	89	30	7	26
15000	2669	936	279	2103	81	24	7	31
15500	2750	961	288	2149	81	25	9	46
16000	2830	990	297	2181	80	29	9	32
16500	2919	1029	306	2200	89	39	9	19
17000	3008	1053	313	2244	89	24	7	44
17500	3083	1086	319	2306	75	33	6	62
18000	3143	1106	332	2489	60	20	13	183
18500	3221	1130	348	2570	78	24	16	81
19000	3322	1154	356	2602	101	24	8	32
19500	3410	1180	369	2693	88	26	13	91
20000	3482	1207	379	2765	72	27	10	72

6 NODE NETWORK ONE LINE DOWN SCHEME IS 0

maxquelen is 15, buffer pool is 20, maxvc is 30 and messoutstanding is 10

CLOCKTICK	RUNNING TOTALS				SINCE LAST WRITE			
	Message Count	Paks Lost	Reqs Discard	Paks Discard	Message Count	Paks Lost	Reqs Discard	Paks Discard
500	42	34	11	252	42	34	11	252
1000	98	69	26	537	56	35	15	285
1500	129	102	35	727	31	33	9	190
2000	187	130	52	920	58	28	17	193
2500	235	157	72	1089	48	27	20	169
3000	274	186	103	1337	39	29	31	248
3500	304	215	127	1599	30	29	24	262
4000	329	239	154	1860	25	24	27	261
4500	358	260	175	2075	29	21	21	215
5000	408	292	194	2303	50	32	19	228
5500	448	318	214	2524	40	26	20	221
6000	496	342	247	2729	48	24	33	205
6500	539	376	273	2970	43	34	26	241
7000	582	400	297	3170	43	24	24	200
7500	633	422	323	3359	51	22	26	189
8000	689	446	343	3518	56	24	20	159
8500	747	468	359	3698	58	22	16	180
9000	809	504	381	3846	62	36	22	148
9500	853	528	401	4112	44	24	20	266
10000	908	549	424	4271	55	21	23	159
10500	965	578	445	4447	57	29	21	176
11000	1003	609	473	4717	38	31	28	270
11500	1033	630	502	5022	30	21	29	305
12000	1069	648	527	5248	36	18	25	226
12500	1107	670	541	5484	38	22	14	236
13000	1154	698	556	5690	47	28	15	206
13500	1208	726	581	5868	54	28	25	178
14000	1258	753	603	6032	50	27	22	164
14500	1292	779	622	6225	34	26	19	193
15000	1356	807	639	6352	64	28	17	127
15500	1412	836	654	6508	56	29	15	156
16000	1467	864	673	6685	55	28	19	177
16500	1511	897	692	6906	44	33	19	221
17000	1572	920	704	7118	61	23	12	212
17500	1615	950	722	7428	43	30	18	310
18000	1667	974	740	7597	52	24	18	169
18500	1727	995	762	7740	60	21	22	143
19000	1784	1027	779	7889	57	32	17	149
19500	1825	1056	796	8056	41	29	17	167
20000	1862	1079	819	8242	37	23	23	186

6 NODE NETWORK ONE LINE DOWN SCHEME IS 1

maxquelen is 15, buffer pool is 20, maxvc is 30 and messoutstanding is 10

CLOCKTICK	RUNNING TOTALS				SINCE LAST WRITE			
	Message Count	Paks Lost	Reqs Discard	Paks Discard	Message Count	Paks Lost	Reqs Discard	Paks Discar
500	52	18	7	33	52	18	7	33
1000	104	42	15	86	52	24	8	53
1500	166	71	20	119	62	29	5	33
2000	236	96	24	157	70	25	4	38
2500	291	115	31	186	55	19	7	29
3000	358	135	38	266	67	20	7	80
3500	411	159	41	293	53	24	3	27
4000	464	185	45	350	53	26	4	57
4500	529	209	54	371	65	24	9	21
5000	589	232	63	397	60	23	9	26
5500	633	255	70	467	44	23	7	70
6000	683	270	74	491	50	15	4	24
6500	748	296	79	529	65	26	5	38
7000	814	309	87	563	66	13	8	34
7500	851	339	94	633	37	30	7	70
8000	921	371	108	655	70	32	14	22
8500	978	395	120	713	57	24	12	58
9000	1052	410	127	738	74	15	7	25
9500	1095	436	137	804	43	26	10	66
10000	1121	454	143	893	26	18	6	89
10500	1181	474	150	927	60	20	7	34
11000	1234	491	161	951	53	17	11	24
11500	1296	517	171	1011	62	26	10	60
12000	1358	548	175	1053	62	31	4	42
12500	1425	573	180	1080	67	25	5	27
13000	1494	590	190	1116	69	17	10	36
13500	1551	610	196	1153	57	20	6	37
14000	1610	640	206	1207	59	30	10	54
14500	1670	661	210	1211	60	21	4	4
15000	1743	683	219	1224	73	22	9	13
15500	1803	712	224	1268	60	29	5	44
16000	1871	740	229	1308	68	28	5	40
16500	1935	771	240	1338	64	31	11	30
17000	1999	793	245	1377	64	22	5	39
17500	2068	819	254	1422	69	26	9	45
18000	2117	848	259	1437	49	29	5	15
18500	2163	876	269	1476	46	28	10	39
19000	2232	895	278	1520	69	19	9	44
19500	2278	909	283	1541	46	14	5	21
20000	2340	925	288	1541	62	16	5	0

6 NODE NETWORK ONE LINE DOWN SCHEME IS 2

maxquelen is 15, buffer pool is 20, maxvc is 30 and messoutstanding is 10

CLOCKTICK	RUNNING TOTALS				SINCE LAST WRITE			
	Message Count	Paks Lost	Reqs Discard	Paks Discard	Message Count	Paks Lost	Reqs Discard	Paks Discard
500	55	29	5	191	55	29	5	191
1000	110	55	12	318	55	26	7	127
1500	150	88	17	620	40	33	5	302
2000	181	113	25	943	31	25	8	323
2500	224	141	28	1160	43	28	3	217
3000	267	179	31	1359	43	38	3	199
3500	312	207	35	1553	45	28	4	194
4000	388	235	36	1642	76	28	1	89
4500	465	259	38	1735	77	24	2	93
5000	516	284	42	1975	51	25	4	240
5500	555	316	46	2216	39	32	4	241
6000	599	341	51	2455	44	25	5	239
6500	631	364	54	2731	32	23	3	276
7000	671	392	56	2980	40	28	2	249
7500	703	419	62	3245	32	27	6	265
8000	732	437	71	3550	29	18	9	305
8500	781	475	73	3795	49	38	2	245
9000	834	503	75	3968	53	28	2	173
9500	875	527	80	4220	41	24	5	252
10000	920	544	83	4434	45	17	3	214
10500	981	582	84	4584	61	38	1	150
11000	1029	613	86	4811	48	31	2	227
11500	1078	640	90	5031	49	27	4	220
12000	1127	667	91	5198	49	27	1	167
12500	1195	695	92	5296	68	28	1	98
13000	1230	728	94	5538	35	33	2	242
13500	1283	754	101	5754	53	26	7	216
14000	1324	780	112	6036	41	26	11	282
14500	1385	803	114	6160	61	23	2	124
15000	1443	825	115	6259	58	22	1	99
15500	1472	848	122	6615	29	23	7	356
16000	1498	881	130	6989	26	33	8	374
16500	1547	915	139	7204	49	34	9	215
17000	1590	936	144	7427	43	21	5	223
17500	1628	969	148	7649	38	33	4	222
18000	1680	991	154	7899	52	22	6	250
18500	1734	1013	157	8093	54	22	3	194
19000	1795	1045	162	8204	61	32	5	111
19500	1845	1068	164	8361	50	23	2	157
20000	1899	1091	165	8464	54	23	1	103

6 NODE NETWORK ONE LINE DOWN SCHEME IS 3

marquelen is 15, buffer pool is 20, marvc is 30 and messoutstanding is 10

CLOCKTICK	RUNNING TOTALS				SINCE LAST WRITE			
	Message Count	Paks Lost	Reqs Discard	Paks Discard	Message Count	Paks Lost	Reqs Discard	Paks Discard
500	68	32	5	81	68	32	5	81
1000	146	67	13	100	78	35	8	19
1500	210	95	20	118	64	28	7	18
2000	277	122	35	199	67	27	15	81
2500	349	148	50	237	72	26	15	38
3000	424	173	57	242	75	25	7	5
3500	499	210	68	287	75	37	11	45
4000	581	233	75	291	82	23	7	4
4500	641	263	86	314	60	30	11	23
5000	719	287	91	320	78	24	5	6
5500	785	311	105	323	66	24	14	3
6000	853	338	117	346	68	27	12	23
6500	922	363	120	397	69	25	3	51
7000	1001	388	127	416	79	25	7	19
7500	1083	412	141	453	82	24	14	37
8000	1159	430	153	498	76	18	12	45
8500	1241	464	160	517	82	34	7	19
9000	1308	488	166	534	67	24	6	17
9500	1371	515	175	547	63	27	9	13
10000	1435	544	187	600	64	29	12	53
10500	1512	570	197	610	77	26	10	10
11000	1587	592	208	629	75	22	11	19
11500	1656	614	214	647	69	22	6	18
12000	1723	645	223	648	67	31	9	1
12500	1778	671	235	694	55	26	12	46
13000	1841	699	245	715	63	28	10	21
13500	1915	727	252	736	74	28	7	21
14000	1989	750	259	742	74	23	7	6
14500	2044	780	270	798	55	30	11	56
15000	2108	813	282	837	64	33	12	39
15500	2179	841	289	871	71	28	7	34
16000	2239	868	301	895	60	27	12	24
16500	2298	894	312	957	59	26	11	62
17000	2374	920	322	998	76	26	10	41
17500	2443	939	339	1012	69	19	17	14
18000	2488	960	346	1030	45	21	7	18
18500	2556	988	353	1056	68	28	7	26
19000	2629	1008	364	1076	73	20	11	20
19500	2688	1032	370	1078	59	24	6	2
20000	2739	1059	378	1094	51	27	8	16

10 NODE NETWORK SCHEME IS 0

maxquelen is 15, buffer pool is 20, maxvc is 30 and messoutstanding is 10

CLOCKTICK	RUNNING TOTALS				SINCE LAST WRITE			
	Message Count	Paks Lost	Reqs Discard	Paks Discard	Message Count	Paks Lost	Reqs Discard	Paks Discard
500	83	50	17	172	83	50	17	172
1000	136	100	49	287	53	50	32	115
1500	190	142	80	409	54	42	31	122
2000	248	182	106	522	58	40	26	113
2500	308	226	136	666	60	44	30	144
3000	368	273	158	832	60	47	22	166
3500	429	316	182	939	61	43	24	107
4000	487	360	207	1059	58	44	25	120
4500	564	402	244	1158	77	42	37	99
5000	639	445	265	1273	75	43	21	115
5500	696	489	287	1365	57	44	22	92
6000	757	535	302	1498	61	46	15	133
6500	817	577	323	1612	60	42	21	114
7000	870	608	348	1736	53	31	25	124
7500	912	655	367	1886	42	47	19	150
8000	979	698	386	1992	67	43	19	106
8500	1027	753	405	2134	48	55	19	142
9000	1099	793	435	2279	72	40	30	145
9500	1161	840	457	2425	62	47	22	146
10000	1209	879	476	2548	48	39	19	123
10500	1273	920	492	2685	64	41	16	137
11000	1328	955	522	2804	55	35	30	119
11500	1381	1006	550	2927	53	51	28	123
12000	1436	1047	573	3036	55	41	23	109
12500	1492	1089	599	3182	56	42	26	146
13000	1546	1122	631	3372	54	33	32	190
13500	1602	1163	660	3533	56	41	29	161
14000	1675	1198	677	3664	73	35	17	131
14500	1727	1234	693	3804	52	36	16	140
15000	1797	1270	732	3962	70	36	39	158
15500	1857	1303	758	4043	60	33	26	81
16000	1924	1343	781	4164	67	40	23	121
16500	1977	1393	809	4284	53	50	28	120
17000	2057	1434	837	4395	80	41	28	111
17500	2119	1483	862	4491	62	49	25	96
18000	2178	1515	882	4671	59	32	20	180
18500	2229	1550	898	4793	51	35	16	122
19000	2291	1581	923	4898	62	31	25	105
19500	2369	1632	953	5036	78	51	30	138
20000	2439	1681	979	5133	70	49	26	97

10 NODE NETWORK SCHEME IS 1

maxquelen is 15, buffer pool is 20, maxvc is 30 and messoutstanding is 10

CLOCKTICK	RUNNING TOTALS				SINCE LAST WRITE			
	Message Count	Paks Lost	Reqs Discard	Paks Discard	Message Count	Paks Lost	Reqs Discard	Paks Discard
500	53	36	9	82	53	36	9	82
1000	124	83	30	106	71	47	21	24
1500	188	116	54	111	64	33	24	5
2000	231	145	73	116	43	29	19	5
2500	275	173	95	116	44	28	22	0
3000	335	210	106	128	60	37	11	12
3500	401	251	135	137	66	41	29	9
4000	473	283	153	164	72	32	18	27
4500	539	321	173	189	66	38	20	25
5000	616	366	193	211	77	45	20	22
5500	684	398	221	236	68	32	28	25
6000	744	421	237	257	60	23	16	21
6500	816	460	254	290	72	39	17	33
7000	902	489	273	315	86	29	19	25
7500	981	521	288	335	79	32	15	20
8000	1049	564	316	363	68	43	28	28
8500	1118	607	333	371	69	43	17	8
9000	1185	649	351	394	67	42	18	23
9500	1267	687	367	406	82	38	16	12
10000	1339	727	387	423	72	40	20	17
10500	1388	770	419	466	49	43	32	43
11000	1454	803	436	475	66	33	17	9
11500	1525	838	451	476	71	35	15	1
12000	1585	872	477	509	60	34	26	33
12500	1669	908	494	528	84	36	17	19
13000	1724	946	517	546	55	38	23	18
13500	1770	982	542	547	46	36	25	1
14000	1827	1012	570	552	57	30	28	5
14500	1875	1037	589	563	48	25	19	11
15000	1945	1064	622	573	70	27	33	10
15500	2005	1101	642	603	60	37	20	30
16000	2082	1134	657	607	77	33	15	4
16500	2142	1164	686	642	60	30	29	35
17000	2199	1197	708	654	57	33	22	12
17500	2262	1221	736	659	63	24	28	5
18000	2337	1254	756	664	75	33	20	5
18500	2408	1292	777	697	71	38	21	33
19000	2474	1336	802	717	66	44	25	20
19500	2537	1376	826	764	63	40	24	47
20000	2595	1405	854	776	58	29	28	12

10 NODE NETWORK SCHEME IS 2

maxquelen is 15, buffer pool is 20, maxvc is 30 and messoutstanding is 10

CLOCKTICK	RUNNING TOTALS				SINCE LAST WRITE			
	Message Count	Paks Lost	Reqs Discard	Paks Discard	Message Count	Paks Lost	Reqs Discard	Paks Discard
500	63	59	16	257	63	59	16	257
1000	129	122	31	478	66	63	15	221
1500	201	175	53	659	72	53	22	181
2000	266	228	70	870	65	53	17	211
2500	323	274	96	1053	57	46	26	183
3000	384	321	118	1226	61	47	22	173
3500	435	374	140	1450	51	53	22	224
4000	516	418	163	1566	81	44	23	116
4500	580	462	178	1786	64	44	15	220
5000	644	515	192	1934	64	53	14	148
5500	702	560	218	2098	58	45	26	164
6000	776	600	234	2241	74	40	16	143
6500	830	640	261	2470	54	40	27	229
7000	896	697	278	2665	66	57	17	195
7500	949	746	295	2883	53	49	17	218
8000	1022	793	325	3056	73	47	30	173
8500	1072	845	348	3255	50	52	23	199
9000	1147	887	368	3354	75	42	20	99
9500	1209	938	388	3547	62	51	20	193
10000	1285	972	410	3697	76	34	22	150
10500	1341	1023	431	3866	56	51	21	169
11000	1415	1070	452	4065	74	47	21	199
11500	1479	1106	471	4222	64	36	19	157
12000	1525	1142	494	4401	46	36	23	179
12500	1585	1186	518	4553	60	44	24	152
13000	1644	1231	556	4698	59	45	38	145
13500	1704	1278	575	4899	60	47	19	201
14000	1776	1311	597	5049	72	33	22	150
14500	1830	1349	617	5241	54	38	20	192
15000	1901	1401	640	5433	71	52	23	192
15500	1950	1450	664	5675	49	49	24	242
16000	2001	1495	703	5879	51	45	39	204
16500	2056	1540	723	6122	55	45	20	243
17000	2121	1569	742	6278	65	29	19	156
17500	2198	1616	768	6432	77	47	26	154
18000	2270	1665	785	6614	72	49	17	182
18500	2339	1724	797	6770	69	59	12	156
19000	2395	1767	812	7021	56	43	15	251
19500	2452	1812	832	7175	57	45	20	154
20000	2510	1856	858	7338	58	44	26	163

10 NODE NETWORK SCHEME IS 3

maxquelen is 15, buffer pool is 20, maxvc is 30 and messoutstanding is 10

CLOCKTICK	RUNNING TOTALS				SINCE LAST WRITE			
	Message Count	Paks Lost	Reqs Discard	Paks Discard	Message Count	Paks Lost	Reqs Discard	Paks Discard
500	98	49	24	39	98	49	24	39
1000	173	91	55	58	75	42	31	19
1500	250	128	85	60	77	37	30	2
2000	323	173	107	75	73	45	22	15
2500	396	220	133	82	73	47	26	7
3000	463	263	159	94	67	43	26	12
3500	524	297	197	113	61	34	38	19
4000	584	336	227	130	60	39	30	17
4500	646	370	255	152	62	34	28	22
5000	706	400	285	168	60	30	30	16
5500	769	446	316	182	63	46	31	14
6000	834	485	357	197	65	39	41	15
6500	898	530	394	214	64	45	37	17
7000	966	563	426	234	68	33	32	20
7500	1018	609	461	255	52	46	35	21
8000	1069	650	492	272	51	41	31	17
8500	1127	702	537	301	58	52	45	29
9000	1194	740	577	303	67	38	40	2
9500	1245	779	600	358	51	39	23	55
10000	1309	814	624	395	64	35	24	37
10500	1375	852	661	410	66	38	37	15
11000	1440	882	705	421	65	30	44	11
11500	1502	922	737	434	62	40	32	13
12000	1572	972	761	447	70	50	24	13
12500	1644	1011	804	470	72	39	43	23
13000	1703	1035	832	484	59	24	28	14
13500	1769	1076	860	510	66	41	28	26
14000	1834	1118	894	529	65	42	34	19
14500	1886	1158	917	560	52	40	23	31
15000	1952	1201	938	568	66	43	21	8
15500	2003	1224	970	589	51	23	32	21
16000	2079	1258	1005	606	76	34	35	17
16500	2150	1304	1044	622	71	46	39	16
17000	2221	1352	1064	639	71	48	20	17
17500	2288	1394	1094	658	67	42	30	19
18000	2332	1428	1126	695	44	34	32	37
18500	2392	1455	1156	708	60	27	30	13
19000	2458	1490	1191	711	66	35	35	3
19500	2529	1526	1211	722	71	36	20	11
20000	2602	1578	1232	738	73	52	21	16

EXAMPLE OF A MONITOR FILE

THE MONITOR FILE

Below is a sampling of the monitor file; to include all of the material would expand the thesis to an impractical extent.

6 NODE NETWORK WITH ALL LINES CONNECTED SCHEME IS 0

****
****

CLOCKTICK IS 20000

The number of nodes in this network is 6 with a max connect of 3
TICKS IS 20000, SCHEME IS 0 and a WAITLIMIT OF 130
MAXQUELEN IS 15, and the PERCENT OF PACKETS LOST IS 0.010000

FOR NODE 0

This node has finished sending 457 messages.

Queue for output buffer to Node 1
9899 packets have passed through this queue
the average time was 23.18
with the maximum time of 31
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
1465	1467
5329	5331
5905	5907
7785	7787
7961	7963
11377	11387
11489	11491
12853	12855

13043	13045
13753	13755
14273	14275
14629	14631
19705	19709
19809	19811

For the que to 1 the QUECOUNT is 12
The running total of packets passing through this node is 9899

Queue for output buffer to Node 4
5934 packets have passed through this queue
the average time was 4.41
with the maximum time of 29
and the minimum time of 1

This queue was never frozen
For the que to 4 the QUECOUNT is 0
The running total of packets passing through this node is 15833

Queue for output buffer to Node 5
8150 packets have passed through this queue
the average time was 7.61
with the maximum time of 31
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
1170	11707

For the que to 5 the QUECOUNT is 2
The running total of packets passing through this node is 23983
The number of Packets lost at this node is 235
The total number of requests that were discarded was 189
The total number of packets that were discarded was 2721

FOR NODE 1

This node has finished sending 320 messages.

Queue for output buffer to Node 0
9616 packets have passed through this queue
the average time was 17.75
with the maximum time of 31
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
1883	1885
19211	19213

For the que to 0 the QUECOUNT is 0
The running total of packets passing through this node is 9616

Queue for output buffer to Node 2
9516 packets have passed through this queue
the average time was 16.76
with the maximum time of 29
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
105	107
7369	7371
12521	12523

For the que to 2 the QUECOUNT is 12
The running total of packets passing through this node is 19132

THIS QUEUE WAS NOT CONNECTED
The number of Packets lost at this node is 186
The total number of requests that were discarded was 183
The total number of packets that were discarded was 2124

FOR NODE 2

This node has finished sending 352 messages.

Queue for output buffer to Node 1
9943 packets have passed through this queue
the average time was 24.78
with the maximum time of 33
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
1223	1225
1777	1779
6057	6059
6671	6673
9119	9121
9255	9259
10983	10985
11415	11419
11447	11451
12455	12457
14359	14361
14383	14385
19031	19033

For the que to 1 the QUECOUNT is 14
The running total of packets passing through this node is 9943

Queue for output buffer to Node 3
6357 packets have passed through this queue
the average time was 4.08
with the maximum time of 29
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
841	879

For the que to 3 the QUECOUNT is 0
The running total of packets passing through this node is 16300

Queue for output buffer to Node 5
5498 packets have passed through this queue
the average time was 3.89
with the maximum time of 29
and the minimum time of 1

This queue was never frozen
For the que to 5 the QUECOUNT is 1
The running total of packets passing through this node is 21798
The number of Packets lost at this node is 222
The total number of requests that were discarded was 162
The total number of packets that were discarded was 2741

FOR NODE 3

This node has finished sending 494 messages.

Queue for output buffer to Node 2
7595 packets have passed through this queue
the average time was 8.49
with the maximum time of 31
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
223	229
2299	2301
3137	3139
3681	3685
5937	5939
6607	6609
6719	6723
8255	8259
9167	9171
11231	11235
11673	11675
13107	13109
13543	13547
13663	13669
15563	15565
16337	16345
16929	16931
17025	17031
18047	18051

For the que to 2 the QUECOUNT is 1
The running total of packets passing through this node is 7595

Queue for output buffer to Node 4
7695 packets have passed through this queue
the average time was 8.92
with the maximum time of 29
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
7467	7469

For the que to 4 the QUECOUNT is 0
The running total of packets passing through this node is 15290

Queue for output buffer to Node 5
5847 packets have passed through this queue
the average time was 4.41
with the maximum time of 29
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
2045	2047
8369	8381
18543	18547
18713	18715

For the que to 5 the QUECOUNT is 3
The running total of packets passing through this node is 21137
The number of Packets lost at this node is 207
The total number of requests that were discarded was 18
The total number of packets that were discarded was 314

FOR NODE 4

This node has finished sending 581 messages.

Queue for output buffer to Node 0
7354 packets have passed through this queue
the average time was 7.99
with the maximum time of 33
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
2335	2339
2473	2475
3177	3181
3641	3649
3887	3891
3969	3971
5153	5155
5169	5171
5481	5483
6815	6819
6847	6853
7465	7467
7881	7883
9327	9331
12703	12715
12825	12827
14171	14173

14247	14251
14871	14873
15855	15863
16033	16035
17695	17699
18399	18403

For the que to 0 the QUECOUNT is 0
The running total of packets passing through this node is 7354

Queue for output buffer to Node 3
7892 packets have passed through this queue
the average time was 9.58
with the maximum time of 29
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
3031	3035

For the que to 3 the QUECOUNT is 8
The running total of packets passing through this node is 15246

THIS QUEUE WAS NOT CONNECTED
The number of Packets lost at this node is 163
The total number of requests that were discarded was 17
The total number of packets that were discarded was 228

FOR NODE 5

This node has finished sending 684 messages.

Queue for output buffer to Node 0
8792 packets have passed through this queue
the average time was 12.27
with the maximum time of 33
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
4061	4067
5183	5187
8973	8975
9557	9561
12135	12137
13845	13847
14805	14809

For the que to 0 the QUECOUNT is 7
The running total of packets passing through this node is 8792

Queue for output buffer to Node 2
6150 packets have passed through this queue
the average time was 5.80
with the maximum time of 29
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
685	689
1751	1755
2437	2443
2703	2707
7977	7979
9445	9449
11677	11683
13413	13417
14455	14459
15599	15603
17031	17037
17093	17099
18765	18771
19261	19267

For the que to 2 the QUECOUNT is 0
The running total of packets passing through this node is 14942

Queue for output buffer to Node 3
5643 packets have passed through this queue
the average time was 4.47
with the maximum time of 29
and the minimum time of 1

The times that this output queue got frozen and then freed again are as follows:

TIME FROZEN	TIME FREED
3765	3777
10225	10227

For the que to 3 the QUECOUNT is 3
The running total of packets passing through this node is 20585
The number of Packets lost at this node is 192
The total number of requests that were discarded was 23
The total number of packets that were discarded was 332

TOTAL MESSAGES SENT ON THIS NETWORK IS 2888
TOTAL PACKET COUNT WAS 237445
The TOTAL NUMBER OF PACKETS LOST IS 1205
TOTAL NUMBER OF REQUESTS DISCARDED IS 592
TOTAL NUMBER OF PACKETS DISCARDED IS 8460

****
FOR NODE 0

THE QUE TO NODE 1 the counter is at 5
The contents of que_ele number 0 is:

acknum 5 for message number 655
source 5, sink 0
The contents of que_ele number 1 is:

call request 466 for message number 466
source 0, sink 1
NOTHING IN ELEMENT 2
NOTHING IN ELEMENT 3
NOTHING IN ELEMENT 4
The contents of que_ele number 5 is:

acknum 4 for message number 325
source 1, sink 3
The contents of que_ele number 6 is:

acknum 5 for message number 339
source 2, sink 0
The contents of que_ele number 7 is:

call request 585 for message number 585
source 4, sink 2
The contents of que_ele number 8 is:

acknum 6 for message number 339
source 2, sink 0
The contents of que_ele number 9 is:

acknum 7 for message number 339
source 2, sink 0
The contents of que_ele number 10 is:

acknum 8 for message number 339
source 2, sink 0
The contents of que_ele number 11 is:

acknum 6 for message number 325
source 1, sink 3
The contents of que_ele number 12 is:

acknum 9 for message number 339
source 2, sink 0
The contents of que_ele number 13 is:

acknum 3 for message number 655
source 5, sink 0
The contents of que_ele number 14 is:

acknum 4 for message number 655
source 5, sink 0

THE QUE TO NODE 4 the counter is at 0
NOTHING IN ELEMENT 0
NOTHING IN ELEMENT 1
NOTHING IN ELEMENT 2
NOTHING IN ELEMENT 3
NOTHING IN ELEMENT 4
NOTHING IN ELEMENT 5
NOTHING IN ELEMENT 6
NOTHING IN ELEMENT 7
NOTHING IN ELEMENT 8
NOTHING IN ELEMENT 9
NOTHING IN ELEMENT 10
NOTHING IN ELEMENT 11
NOTHING IN ELEMENT 12
NOTHING IN ELEMENT 13
NOTHING IN ELEMENT 14

THE QUE TO NODE 5 the counter is at 3
NOTHING IN ELEMENT 0
NOTHING IN ELEMENT 1
NOTHING IN ELEMENT 2
The contents of que_ele number 3 is:

packet number 2 for message number 463
source 0, sink 5
The contents of que_ele number 4 is:

acknum 2 for message number 689
source 5, sink 0
NOTHING IN ELEMENT 5
NOTHING IN ELEMENT 6
NOTHING IN ELEMENT 7
NOTHING IN ELEMENT 8

NOTHING IN ELEMENT 9
NOTHING IN ELEMENT 10
NOTHING IN ELEMENT 11
NOTHING IN ELEMENT 12
NOTHING IN ELEMENT 13
NOTHING IN ELEMENT 14

****

The INPUT BUFFER from Node 1 contains:

for message number -1
source -1, sink -1
The last time something was put into this inbuf was 19998

The INPUT BUFFER from Node 4 contains:

acknum 8 for message number 456
source 0, sink 3
The last time something was put into this inbuf was 20000

The INPUT BUFFER from Node 5 contains:

packet number 3 for message number 689
source 5, sink 0
The last time something was put into this inbuf was 20000

The INPUT BUFFER FROM THE HOST CONTAINS:

for message number -1
source -1, sink -1
The last time something was put into this inbuf was 19998

THE NUMBER OF BUFFERS TAKEN FROM POOL IS AT 14

EXAMPLE OF A HOPS FILE

This is the output file HOPS for a six node network with all lines up. The simulation has run for 1,000 clockticks, using control scheme 3.

****

FOR NODE 0

HOPCOUNT OF 1 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
25	574	0	628	54	618	10
22	526	1	724	198	700	24
32	800	0	882	82	836	46

AVERAGE TOTAL TIME IS 111.333 and AVERAGE MESSAGE TIME IS 26.667

HOPCOUNT OF 1 WITH A MESSIZE OF 6

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
5	24	0	114	9	92	22
7	30	0	230	200	112	118
9	46	1	326	280	300	26
6	270	0	412	142	312	100
8	384	0	430	46	404	26
23	550	0	582	32	566	16

AVERAGE TOTAL TIME IS 131.667 and AVERAGE MESSAGE TIME IS 51.333

HOPCOUNT OF 1 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
2	8	0	92	84	32	60
4	20	0	102	82	68	34
8	44	0	184	140	140	44
1	516	0	554	38	522	32
29	632	1	988	356	914	74

AVERAGE TOTAL TIME IS 140.000 and AVERAGE MESSAGE TIME IS 48.800

HOPCOUNT OF 2 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
3	16	0	268	252	86	182
17	328	0	412	84	394	18

AVERAGE TOTAL TIME IS 168.000 and AVERAGE MESSAGE TIME IS 100.000

HOPCOUNT OF 2 WITH A MESSIZE OF 6

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
0	2	0	46	44	10	36
13	116	0	376	260	286	90
6	26	0	402	376	128	274

AVERAGE TOTAL TIME IS 226.667 and AVERAGE MESSAGE TIME IS 133.333

HOPCOUNT OF 2 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
19	406	0	524	118	430	94
12	106	0	526	420	204	322
11	98	0	580	482	340	240

AVERAGE TOTAL TIME IS 340.000 and AVERAGE MESSAGE TIME IS 218.667

HOPCOUNT OF 3 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
14	190	1	472	282	424	48

AVERAGE TOTAL TIME IS 282.000 and AVERAGE MESSAGE TIME IS 48.000

HOPCOUNT OF 3 WITH A MESSIZE OF 6

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
20	414	0	682	268	548	134
24	560	0	792	232	710	82

AVERAGE TOTAL TIME IS 250.000 and AVERAGE MESSAGE TIME IS 108.000

****

FOR NODE 1

HOPCOUNT OF 1 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
3	32	0	84	52	56	28
8	64	0	152	88	108	44
17	440	0	564	124	512	52
20	550	0	650	100	632	18

AVERAGE TOTAL TIME IS 91.000 and AVERAGE MESSAGE TIME IS 35.500

HOPCOUNT OF 1 WITH A MESSIZE OF 6

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
7	62	0	134	72	96	38
16	340	0	514	174	462	52
25	652	0	878	226	820	58

AVERAGE TOTAL TIME IS 157.333 and AVERAGE MESSAGE TIME IS 49.333

HOPCOUNT OF 1 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
2	26	0	66	40	34	32
11	92	0	188	96	136	52
12	142	1	844	702	578	266
24	642	0	954	312	840	114

AVERAGE TOTAL TIME IS 287.500 and AVERAGE MESSAGE TIME IS 116.000

HOPCOUNT OF 2 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
23	680	0	986	306	920	66

AVERAGE TOTAL TIME IS 306.000 and AVERAGE MESSAGE TIME IS 66.000

HOPCOUNT OF 2 WITH A MESSIZE OF 6

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
9	82	0	334	252	226	108
0	10	1	520	510	244	276
15	216	1	562	346	500	62
18	478	0	844	366	520	324

AVERAGE TOTAL TIME IS 368.500 and AVERAGE MESSAGE TIME IS 192.500

HOPCOUNT OF 2 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
4	44	0	208	164	70	138
1	16	1	434	418	278	156
10	88	0	612	524	166	446

AVERAGE TOTAL TIME IS 368.667 and AVERAGE MESSAGE TIME IS 246.667

HOPCOUNT OF 3 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
5	54	0	456	402	194	262
6	58	2	638	580	416	222

AVERAGE TOTAL TIME IS 491.000 and AVERAGE MESSAGE TIME IS 242.000

****

FOR NODE 2

HOPCOUNT OF 1 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
0	10	0	32	22	18	14
22	632	0	684	52	670	14
24	788	0	930	142	922	8

AVERAGE TOTAL TIME IS 72.000 and AVERAGE MESSAGE TIME IS 12.000

HOPCOUNT OF 1 WITH A MESSIZE OF 6

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
2	30	0	64	34	38	26
4	54	0	152	98	90	62
6	74	0	166	92	122	44
7	78	0	186	108	142	44
12	154	0	218	64	192	26
15	176	0	240	64	206	34

AVERAGE TOTAL TIME IS 76.667 and AVERAGE MESSAGE TIME IS 39.333

HOPCOUNT OF 1 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
5	70	0	138	68	102	36
11	152	0	236	84	164	72
17	198	0	876	678	414	462
25	878	0	988	110	954	34

AVERAGE TOTAL TIME IS 235.000 and AVERAGE MESSAGE TIME IS 151.000

HOPCOUNT OF 2 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
1	24	0	132	108	84	48

AVERAGE TOTAL TIME IS 108.000 and AVERAGE MESSAGE TIME IS 48.000

HOPCOUNT OF 2 WITH A MESSIZE OF 6

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
9	244	0	554	310	332	222
10	146	1	570	424	486	84

AVERAGE TOTAL TIME IS 367.000 and AVERAGE MESSAGE TIME IS 153.000

HOPCOUNT OF 2 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
3	36	0	124	88	58	66

AVERAGE TOTAL TIME IS 88.000 and AVERAGE MESSAGE TIME IS 66.000

HOPCOUNT OF 4 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
13	158	2	626	468	508	118

AVERAGE TOTAL TIME IS 468.000 and AVERAGE MESSAGE TIME IS 118.000

****

FOR NODE 3

HOPCOUNT OF 1 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
26	570	0	666	96	620	46
25	556	1	780	224	760	20
33	956	0	982	26	970	12

AVERAGE TOTAL TIME IS 115.333 and AVERAGE MESSAGE TIME IS 26.000

HOPCOUNT OF 1 WITH A MESSIZE OF 6

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
7	48	0	92	44	62	30
11	98	0	220	122	168	52
15	200	0	314	114	274	40
24	532	0	720	188	546	174
27	670	0	746	76	706	40

AVERAGE TOTAL TIME IS 108.800 and AVERAGE MESSAGE TIME IS 67.200

HOPCOUNT OF 1 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
9	78	0	160	82	106	54
10	86	0	164	78	126	38
12	108	0	258	150	206	52
16	226	0	296	70	242	54
21	300	0	530	230	366	164
29	756	0	846	90	778	68
31	854	0	926	72	866	60

AVERAGE TOTAL TIME IS 110.286 and AVERAGE MESSAGE TIME IS 70.000

HOPCOUNT OF 2 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
3	14	0	252	238	182	70
6	36	0	284	248	234	50
4	22	0	286	264	192	94
18	266	1	562	296	504	58

AVERAGE TOTAL TIME IS 261.500 and AVERAGE MESSAGE TIME IS 68.000

HOPCOUNT OF 2 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
1	6	0	76	70	18	58
0	2	0	82	80	42	40
2	12	0	196	184	80	116
5	34	1	420	386	296	124
13	164	1	552	388	430	122
23	422	0	952	530	648	304

AVERAGE TOTAL TIME IS 273.000 and AVERAGE MESSAGE TIME IS 127.333

****

FOR NODE 4

HOPCOUNT OF 1 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
28	628	0	746	118	714	32
29	756	0	844	88	778	66

AVERAGE TOTAL TIME IS 103.000 and AVERAGE MESSAGE TIME IS 49.000

HOPCOUNT OF 1 WITH A MESSIZE OF 6

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
10	78	0	168	90	116	52
17	272	0	340	68	306	34
12	154	0	372	218	220	152
34	850	0	936	86	908	28

AVERAGE TOTAL TIME IS 115.500 and AVERAGE MESSAGE TIME IS 66.500

HOPCOUNT OF 1 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
3	12	0	76	64	26	50
9	74	0	310	236	132	178
23	444	0	588	144	464	124
32	774	0	904	130	848	56

AVERAGE TOTAL TIME IS 143.500 and AVERAGE MESSAGE TIME IS 102.000

HOPCOUNT OF 2 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
5	28	0	156	128	100	56
11	88	0	268	180	214	54
14	190	1	422	232	402	20

AVERAGE TOTAL TIME IS 180.000 and AVERAGE MESSAGE TIME IS 43.333

HOPCOUNT OF 2 WITH A MESSIZE OF 6

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
0	0	0	58	58	10	48
6	32	0	178	146	98	80
7	58	0	228	170	176	52
13	172	0	500	328	446	54
18	322	0	562	240	492	70
30	758	0	954	196	892	62
31	760	0	976	216	922	54

AVERAGE TOTAL TIME IS 193.429 and AVERAGE MESSAGE TIME IS 60.000

HOPCOUNT OF 2 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
4	16	0	150	134	72	78
8	64	0	326	262	192	134
21	386	0	716	330	648	68
24	502	0	728	226	608	120
22	432	0	850	418	554	296

AVERAGE TOTAL TIME IS 274.000 and AVERAGE MESSAGE TIME IS 139.200

HOPCOUNT OF 3 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
16	232	1	586	354	522	64
25	514	0	744	230	686	58

AVERAGE TOTAL TIME IS 292.000 and AVERAGE MESSAGE TIME IS 61.000

HOPCOUNT OF 3 WITH A MESSIZE OF 6

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
1	4	0	736	732	48	688

AVERAGE TOTAL TIME IS 732.000 and AVERAGE MESSAGE TIME IS 688.000

HOPCOUNT OF 3 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
15	198	0	440	242	250	190

AVERAGE TOTAL TIME IS 242.000 and AVERAGE MESSAGE TIME IS 190.000

HOPCOUNT OF 4 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
2	8	1	468	460	292	176

AVERAGE TOTAL TIME IS 460.000 and AVERAGE MESSAGE TIME IS 176.000

****

FOR NODE 5

HOPCOUNT OF 1 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
0	10	0	36	26	18	18
5	60	0	138	78	126	12
7	72	0	276	204	116	160
12	148	0	280	132	222	58
10	120	0	368	248	190	178
20	336	0	370	34	362	8
22	392	0	492	100	462	30
29	544	0	678	134	650	28
24	460	0	686	226	504	182
33	676	0	762	86	710	52
34	694	0	782	88	740	42
36	716	0	800	84	766	34
31	614	0	816	202	656	160
40	784	0	828	44	796	32
44	822	0	880	58	862	18
38	750	0	930	180	894	36

AVERAGE TOTAL TIME IS 120.250 and AVERAGE MESSAGE TIME IS 65.500

HOPCOUNT OF 1 WITH A MESSIZE OF 6

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
15	184	0	312	128	260	52
21	382	0	476	94	430	46
26	508	0	606	98	562	44
32	644	0	752	108	692	60
41	788	0	906	118	846	60
43	820	0	964	144	874	90

AVERAGE TOTAL TIME IS 115.000 and AVERAGE MESSAGE TIME IS 58.667

HOPCOUNT OF 1 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
2	22	0	90	68	44	46
3	36	0	108	72	68	40
4	58	0	134	76	92	42
8	94	0	204	110	132	72
23	404	0	458	54	412	46
17	288	0	526	238	474	52
27	518	0	574	56	530	44
30	584	0	670	86	626	44
19	320	2	730	410	662	68
42	790	0	862	72	806	56
47	888	0	946	58	900	46

AVERAGE TOTAL TIME IS 118.182 and AVERAGE MESSAGE TIME IS 50.545

HOPCOUNT OF 2 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
14	158	0	334	176	272	62
18	292	0	402	110	354	48
11	124	0	524	400	196	328
16	206	2	640	434	618	22

AVERAGE TOTAL TIME IS 280.000 and AVERAGE MESSAGE TIME IS 115.000

HOPCOUNT OF 2 WITH A MESSIZE OF 6

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
13	150	0	744	594	246	498
25	478	0	768	290	600	168

AVERAGE TOTAL TIME IS 442.000 and AVERAGE MESSAGE TIME IS 333.000

HOPCOUNT OF 2 WITH A MESSIZE OF 9

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
6	66	0	508	442	160	348
9	114	0	670	556	228	442

AVERAGE TOTAL TIME IS 499.000 and AVERAGE MESSAGE TIME IS 395.000

HOPCOUNT OF 3 WITH A MESSIZE OF 3

MESSAGE NUMBER	REQUEST SENT	TIMED OUT	LAST ACK	TOTAL TIME	FIRSTPK SENT	MESSAGE TIME
1	16	0	8	70	34	52

AVERAGE TOTAL TIME IS 70.000 and AVERAGE MESSAGE TIME IS 52.000

EXAMPLE OF AN ERROR LOG FILE

These are **ERROR** files for a six node network with all lines up. The simulation has run for 1,000 clockticks.

SCHEME 0

data6.old

```
FUNC6:  TICKS 1000; Node 0; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 0; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 1; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 2; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 2; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 3; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 3; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 4; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 5; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 5; in Loadoutbufs nothing in packet
```

/**********/

SCHEME 1

data61.old

```
Discarding a packet so increasing node2's store by 1
Store is now 1
Discarding a packet so increasing node2's store by 1
Store is now 1
Discarding a packet so increasing node2's store by 1
Store is now 5
Discarding a packet so increasing node2's store by 1
Store is now 4
Discarding a packet so increasing node2's store by 1
Store is now 3
FUNC6:  TICKS 1000; Node 0; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 1; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 2; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 2; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 2; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 3; in Loadoutbufs nothing in packet
FUNC6:  TICKS 1000; Node 3; in Loadoutbufs nothing in packet
```

FUNC6: TICKS 1000; Node 4; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 4; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 5; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 5; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 5; in Loadoutbufs nothing in packet

/***/

SCHEME 2

data62.old

FUNC6: TICKS 1000; Node 0; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 1; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 2; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 3; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 4; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 5; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 5; in Loadoutbufs nothing in packet

/***/

SCHEME 3

data63.old

MAIN: TICKS 492; Node 0; ERROR: in GETHOST maxout exceeded
Have passed in the node 0, and number 14
This did not mean that we stopped the run however
FUNC3: TICKS 524; Node 3; ERROR: ct is -1 in ReverseTable can't find the virtual circuit
So we are discarding the packet
FUNC6: TICKS 1000; Node 0; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 0; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 1; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 2; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 3; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 4; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 4; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 5; in Loadoutbufs nothing in packet
FUNC6: TICKS 1000; Node 5; in Loadoutbufs nothing in packet

EXAMPLE OF A LOST PACKETS FILE

LOST FOR A THREE NODE NETWORK, 500 CLOCKTICKS

ticks is 131,n is 2, just lost a packet
packet number 8 for message number 4
source 2, sink 0

ticks is 225,n is 0, just lost a packet
acknum 9 for message number 8
source 2, sink 0

ticks is 309,n is 2, just lost a packet
acknum 1 for message number 14
source 0, sink 2

ticks is 329,n is 2, just lost a packet
acknum 6 for message number 13
source 1, sink 2
ticks is 393,n is 1, would have lost this pak but is a closevc so protected

ticks is 439,n is 0, just lost a packet
packet number 8 for message number 20
source 0, sink 2
ticks is 467,n is 0, would have lost this pak but is a closevc so protected

ticks is 495,n is 1, just lost a packet
acknum 5 for message number 23
source 0, sink 1

EXAMPLE OF A DISCARDED PACKETS FILE

DISCARDS FOR A THREE NODE NETWORK, 1000 CLOCKTICKS

TICKS IS 186,Node is 2: acknum 8 for message number 5
source 1, sink 2

TICKS IS 450,Node is 2: acknum 3 for message number 21
source 0, sink 2

TICKS IS 456,Node is 2: acknum 2 for message number 22
source 0, sink 2

TICKS IS 754,Node is 0: call accept 34 for message number 34
source 2, sink 0

TICKS IS 760,Node is 0: acknum 4 for message number 32
source 2, sink 0

TICKS IS 816,Node is 2: acknum 2 for message number 34
source 0, sink 2

EXAMPLE OF A NODE FILE USED IN PRINTOUT SCHEME 1

This is an example of the output that is produced for each individual node when the user sets the Want Printout flag to 1. This is for node 0, scheme 0, with a three node network

```
************************************************************
************************************************************
     CLOCKTICKS  IS  495

Node 0 is sending from que_ele 5, acknum 6 for message number 14
source 1, sink 0
to Node 1
Buffercount is 14

Node 0 is sending from que_ele 5, packet number 6 for message number 15
source 0, sink 2
to Node 2
Buffercount is 13
```

PVCTable for Node 0

INCOMING Node	INCOMING VC	OUTGOING Node	OUTGOING VC	SOURCE	SINK	MESSAGE NUMBER	PACKET SIZE
0	0	2	0	0	2	21	6
0	5	1	4	0	1	22	6
2	0	1	5	2	1	17	3
1	0	2	1	1	2	1	9
-1	-1	-1	-1	-1	-1	-1	-1
0	3	2	2	0	2	14	9
2	1	0	1	2	0	18	3
-1	-1	-1	-1	-1	-1	-1	-1
1	3	0	0	1	0	18	3
0	8	2	3	0	2	15	9
0	6	2	4	0	1	19	9
0	1	2	6	0	2	23	6
0	7	2	7	0	2	8	9
2	3	0	2	2	0	21	3
2	8	0	14	1	0	13	6
0	9	1	0	0	1	24	9
1	2	0	3	1	0	14	6
2	2	1	3	2	1	5	9
-1	-1	-1	-1	-1	-1	-1	-1
1	6	0	7	1	0	22	3
1	5	0	4	1	0	20	6
2	9	0	6	2	0	20	9
2	4	0	10	2	0	10	9
2	5	0	11	2	0	11	3
2	6	0	12	2	0	12	9
-1	-1	-1	-1	-1	-1	-1	-1
1	1	2	8	1	2	12	6
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

****
****
CLOCKTICKS IS 496

```
************************************************************
************************************************************
     CLOCKTICKS  IS  496


UNFREEZINGVCFULL  cause vc table is no longer full to this node
For this node disable is -1, freezevcfull is -1, and freezemess is -1

TIMEOUTREQ:
CHECKHOLD:  No messages on que, returning -1

this next call request is number 25
WANTTOSEND:  node 0 wants to send a message of 6 packets to node 2


Checking input buffer from node 1
source 2, sink 1:  1's packet is being PASSED ALONG to Node 2
SENDALONG:  from is 1 and vcnum is 3 and messnum is 5
source is 2 and priginator is 1
REVERSTABLE: returning 136
In middle nodes going sink->source ct is 136
In middle node going sink->source,
And nextnode is 2 and vcnum is 2
PUTONQUE:  for input buffer from 1
PUTONQUE:  next node is 2
this packet got put in the 4 element for the output queue for node 2

Node 0 inbuf.from 1 is now free
```

Checking input buffer from node 0
SENDALONG: from is 0 and vcnum is -1 and messnum is 25
source is 0 and priginator is 0
GETNEXTNODE: nextnode is 2
SETTING t = b
The new installed line is 0 2 2 5 0 2 25 6
PUTONQUE: for input buffer from 0
PUTONQUE: next node is 2
this packet got put in the 5 element for the output queue for node 2

Node 0 inbuf.from 0 is now free

DISABLE is now set to -1 for this node

Checking input buffer from node 2
source 1, sink 2: 2's packet is being PASSED ALONG to Node 1
SENDALONG: from is 2 and vcnum is 1 and messnum is 1
source is 1 and priginator is 2
REVERSTABLE: returning 24
In middle nodes going sink->source ct is 24
In middle node going sink->source,
And nextnode is 1 and vcnum is 0
PUTONQUE: for input buffer from 2
PUTONQUE: next node is 1
this packet got put in the 0 element for the output queue for node 1

Node 0 inbuf.from 2 is now free

```
************************************************************
************************************************************
     CLOCKTICKS  IS  497

Node 0 is sending from que_ele 0, acknum 7 for message number 1
source 1, sink 2
to Node 1
Buffercount is 15

Node 0 is sending from que_ele 6, packet number 7 for message number 15
source 0, sink 2
to Node 2
Buffercount is 14



************************************************************
************************************************************
     CLOCKTICKS  IS  498


UNFREEZINGVCFULL  cause vc table is no longer full to this node
For this node disable is -1, freezevcfull is -1, and freezemess is -1

TIMEOUTREQ:
CHECKHOLD:  No messages on que, returning -1

WANTTOSEND:  Node 0 does not want to send now
It will send in 12 ticks


Checking input buffer from node 0:  No requests here now
So look to see if this node has anything to send
```

In LOOKTOSEND
disable is -1, requesting is -1, and busy is -1
SETTING the paktime of message number 21 packet 3 of size 6
Now the packet times of this message is:
492 494 498 -1 -1 -1 -1 -1 -1

Node 0 is putting packet 3, message 21, on que
SENDALONG: from is 0 and vcnum is 0 and messnum is 21
source is 0 and priginator is 0
PUTONQUE: for input buffer from 0
PUTONQUE: next node is 2
this packet got put in the 6 element for the output queue for node 2

Node 0 inbuf.from 0 is now free

DISABLE is now set to -1 for this node

Checking input buffer from node 2
Got a packet
Sending ack 9,mess 12 to Node 2
SENDALONG: from is 2 and vcnum is 6 and messnum is 12
source is 2 and priginator is 0
This is the sink so nextnode is 2 and vcnum is 6
PUTONQUE: for input buffer from 2
PUTONQUE: next node is 2
SORRY, THE QUEUE FOR OUTPUT BUFFER 2 IS FULL
DISCARDING THIS PACKET

Checking input buffer from node 1
Node 0 just got a call accept for messnum 24 from node 1

```
**************************************************************
**************************************************************
     CLOCKTICKS  IS  499

Nothing to send from this queue to Node 1

Node 0 is sending from que_ele 7, acknum 6 for message number 13
source 1, sink 0
to Node 2
Buffercount is 14

NOTHING IN PACKET
```

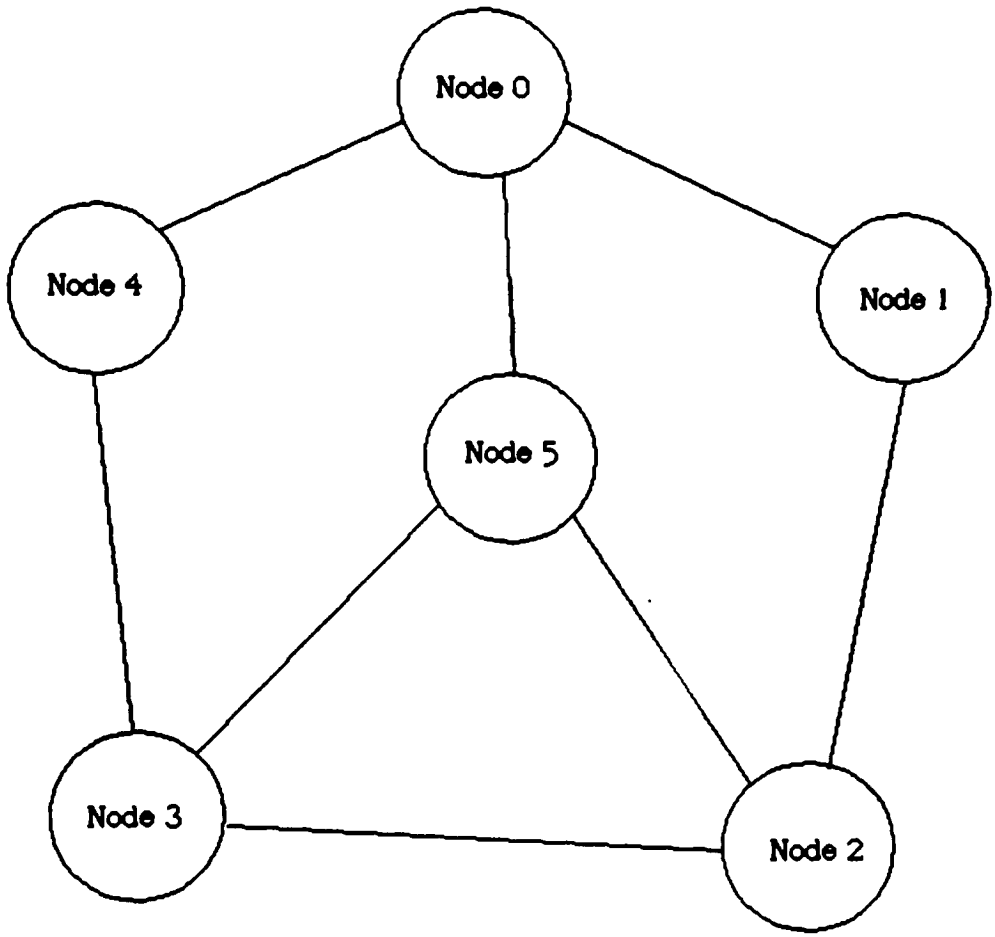
APPENDIX D

USERS MANUAL TO CONSTRUCT DATA FILES FOR THE CONGESTION CONTROL SCHEME SIMULATION

This manual is intended as a guide for those users who wish to use this simulation. It is intended as a quick reference guide while constructing the necessary data files to run the simulation. For a more complete understanding of this simulation, refer to the thesis, "Congestion Control Schemes and Their Effect on a Hypothetical Network." A user should have a basic knowledge about networks and how they work because designing an individualized network requires defining variables such as size, topology, sending rates, and so forth.

SETUP

The first step is to determine what your network will look like. You need to decide on the number of nodes and how you want them connected to one another. This physical representation is referred to as the network's topology. Draw this topological information on a piece of paper so you can refer to it later. Figure 1 is an example of a six-node network with the nodes having a maximum of three degrees of connectivity (in other words a node can be connected to, at most, three other nodes). Notice that the node numbers start at 0 and increase to 5, making for a total of six nodes. This is very important to follow since the entire network is based on that naming assumption. So, for this six-node network you would have nodes 0 through 5. A test run of the simulation for this six-node network that lasted for 10,000 clockticks took over four minutes to run on the Pyramid computer at the Rochester Institute of Technology. The number of clockticks you want to run and the number of nodes in your network will have a direct correlation with the length of time it takes for the simulation to finish.



Six-node network

Figure 1

INPUT

There will be a file called "DATA" containing the categories listed below. Each category will be explained individually in later sections. The user can make one or several copies of the data file and add the necessary numeral input. It is suggested that the user make a data file for each of the four control schemes. In this example these schemes could be called data60.d, data61.d, data62.d and data63.d. The 6 is for the number of nodes and the 0, 1, 2, and 3 represent the congestion control scheme. There is an example of the data file data62.d at the end of this manual if reference is needed.

Maxnodes
Maxconnect
Maxticks
Maxvc
Maxoutstanding Mess
Buffer Pool Size
MaxQueLen
Waitlimit
Percent of Packets Lost
PakDiscard
Congestion Scheme
MaxStore
WarnPercent for QueLen
Ignore Time
Listen Time
Rate Reduction Factor
More Outputs
Want Printouts
Time to Start Printing
Static Routing Table
Traffic Rates
Adjacency Matrix
Number of Packets

The reason for starting with the "DATA" file is to make the input of the numbers easier on the user. Now that the variables are listed, the numbers can be filled in.

The names can be changed to whatever the user likes, but the order is very important. The words are there only for the benefit of the user. Any line that begins with a letter will not be read; you can think of them as comment lines. Any line that starts with a number or period (.) will be read in as data. The simulation is expecting to see a certain number of inputs in a certain order. If the simulation does not receive the inputs or if they are out of order, the simulation will not function properly. There is a system of checks and balances, but it is only a rough checker. In other words, the system can produce erroneous output if the user makes a mistake that still makes sense to the simulation or it can be done intentionally. For example, it would be possible to set up the adjacency matrix and routing table so a call request is routed back to the host without ever reaching the destination.

Maxnodes - defines the number of nodes in your network. The number can be between 3 and 50.

Maxconnect - is the maximum number of nodes to which any one node of your network may be connected. This number must be between 2 and 5. Look at the drawing of your network. Find the node that has the most connections to other nodes, count them and the total is the number to install under Maxconnect. In our example, node 1 has two connections. It is connected to node 0 and node 2. However, nodes 0, 2, 3 and 5 are connected to three other nodes, so the number to install here is three.

Maxticks - is the number of time cycles you would like the simulation to run. In test runs, 10,000 cycles took four minutes or so to run on the Pyramid for a six-node network, so keep that in mind when you are planning your run. The PC's take even longer to compile and run.

Maxvc - Each node has a virtual circuit table. This variable will contain the size of the virtual circuit table. The size indicates the number of virtual circuits that may pass through a node at any one time.

Maxoutstanding Mess - is the maximum number of messages each host can have open or unfinished at any one time. You should keep the number of virtual circuits and the number of outstanding messages in some kind of proportion. Obviously, if you have six nodes and each can have 10 outstanding messages, it does not make sense to set the virtual circuit table to 5. By doing so that would allow each node to carry only 5 virtual circuits at any one time. You will have a multitude of requests being rejected all

due to full virtual circuit tables. Try starting out with a 3:1 ratio -- Maxvc:Maxoutstanding.

Buffer Pool Size - is the number of free buffers in node's buffer pool. When a node receives a packet, it needs a buffer in which to put the packet (unless that packet is for this node) in order for it to be put on an output queue. The size of your buffer pool should be smaller than the length of your combined output queues.

Maxque Length - is the length of your output queues. All output queues for every node will be this length.

Waitlimit - is the length of time the node will wait before timing out and then trying to resend a packet. This is a function of the size of your network and the length of your output queues. The bigger the network and the longer the output queue length, the more time it will take a packet to travel to its destination. A rule of thumb would be:

(maximum output queue length x 2) x (average number of hops x 2)

Percent of Packets Lost is the percentage of packets you want the simulation to lose, expressed as a decimal (.01 would be one percent of all packets would be lost, .001 would be one out of a thousand).

Pakdiscard - is used as a flag to indicate whether the user wants to have packets either discarded or frozen in their input buffers. By way of an explanation, when a packet arrives at a node, there may be several reasons why it can not be passed on. There may not be any free buffers left or the proper output queue may be full, and so forth. If the packets are frozen, no other packet can be loaded into the input buffer, causing the output buffer connected to that input buffer to start backing up. A 0 here means you don't want packets discarded. A 1 means you do want the packets discarded.

Congestion Scheme - is the number of the congestion control scheme.

0 is the control of having no congestion control scheme.
1 is for token control scheme.
2 is for a choke packet control scheme.
3 is for a flow balance control scheme.

Maxstore - is used only with the token scheme. Each node has a token store holding any tokens the node may have. Maxstore holds the number of tokens that start off in each node. It is also the maximum each node may have. If the number is exceeded, then the tokens will be passed on to a node chosen at random.

Note: The next four variables are only used with the choke packet scheme and are not even read by the other schemes.

Warnpercent for Quelen is how full you want the output queues to become before a choke packet is sent back to the host. The variable is expressed as a decimal -- .30 is 30 percent. This should not be set too high since it takes a while for the choke packets to return to the source.

Ignore Time - is the number of ticks after a host takes in a choke packet that it will ignore all other choke packets with the same destination.

Listen Time - is the allowable time period a host node will listen after the ignore time has expired for other choke packets with the same destination.

Rate Reduction Factor - is the percentage the host slows down its sending after receiving a choke packet. Percents are in decimal form -- expressed as .60 for 60 percent.

More Outputs - causes a write to several output files every X ticks. If this variable is set to 100, then every 100 clockticks the simulation will print out to two different output files. SUMMARY is a table of the number of messages completed, packets lost, and requests and packets discarded. A running list is produced in addition to an accounting of all activity since the last write to this file. Also, some of these facts are written to the regular output file MONITOR. A further explanation of these output files is conducted under the OUTPUT FILES section below.

Note: Caution should be used with the next two inputs.

Want Printouts - A 1 here will cause a printout to the files "NODE0" through "NODE(maxnodes)", a 2 will write to standard I/O with a more detailed printout (but it takes longer), and a 3 here will print out to both. Output to these files take up quite a bit of disk space. If a 1 is typed here and the user has a three-node network

running for 1,000 ticks, the file for a single node will take up 35,000 lines. Placing a 2 here will use even more space. If at all possible, solve problems by looking at the files produced by installing a 1 for this variable. Ideally, the 2 and 3 are just used for error checking if one cannot figure out problems. Since writing to these output files takes up so much space, the user can limit the time that the writes occur by setting the start time below and have the writes end when Maxticks is reached. If you don't want to see what is happening but just want the outcome of your run, the user can put a 0 in for this variable. It is recommended that if the user uses a 1 for this variable then the network should have only 3 nodes and the run should be limited to 100 clockticks. The greater the number of nodes then the shorter the run should be. If the long listing is necessary try and keep the time as short as possible.

Time to Start Printing - indicates the time you want to start receiving a printout of the exercise. This variable is read only if a 1, 2, or 3 is typed in the Want Printouts variable.

Static Routing Table - is a table that each node has. It contains two different routes for each possible destination in the network. It is suggested that you separate this section into several smaller ones by writing in "Node #" for each node's section. In our example it would look like this for node 0.

Static Routing Table
Node0
1,90,1,100,5
2,70,1,100,5 This line is used as an example below
3,50,4,100,5
4,85,4,100,5
5,60,5,100,1
Node1
etc.

When a node receives a call request and the request is not destined for this node, the node will look in its routing table to see where to send the request next. You can think of this as how a node discovers which node will be next. However, the node will not send the request back on the same line whence it came. Suppose node 0 gains a request from node 1, which tries to communicate with node 2. The program goes to node 0's routing table and searches the first column to find the destination. In this

case that line would be 2, 70, 1, 100, 5. The line can be read as follows: for destination 2, 70 percent of the time the request will be sent through node 1 and 30 percent (100-70) of the time it will be sent through node 5. Since the request arrived from node 1, it cannot be sent back along the same path so the program perseveres until a new path is selected to node 5. Even if a node has three or more connections, only two choices are available within a routing table.

The third and fifth numbers must be nodes to which node 0 is directly connected and is reflected in the Adjacency Matrix. Notice how the fourth number in all the lines is 100. The reason for this is that you want a 100 percent probability that the packet will be sent out either on one line or the other. Therefore, two out of the five numbers are a given: The first number in the example (line 2) is the destination, and the fourth number (100) assures the request is sent out on at least one line. The second number (70) is the probability that the next node along the path will be 1 (the third number). The fifth number (5) is the other choice for routing.

Destination Selection - is a Maxnode by Maxnode table. For our example based upon Figure 1, the row and column reference numbers range from 0 to 5. So, to address the elements in this table, you have to refer to Table_A [0][0] through Table_A [5][5]. The row and column number corresponds to the node numbers. Remember, this is one of the reasons why the node numbers start at 0. The first row would be referenced by Table_A [0][0.5], therefore, Table_A [0][0] contains a 1, Table_A [0][3] is 60. Notice when the row and column reference number is the same, Table_A [0][0], [1][1], [2][2], etc., all contain a 1.

Table_A

```
1,20,40,60,80,100 (*for node 0)
40,1,60,70,90,100 (*for node 1)
30,40,1,65,75,100 (*for node 2)
25,50,60,1,75,100 (*for node 3)
10,20,45,75,1,100 (*for node 4)
50,55,65,85,100,1 (*for node 5)
```

Note: Do not insert the information within parenthesis into the data file.

Assuming the host for node 1 has decided to send, it consults this table to see where it will send the message. A random number is generated. As an arbitrary

example, we will use 47. The searching algorithm traverses to the second row Table [1][0..5] (40, 1, 60, 70, 90, 100). The algorithm then proceeds to work its way through the row looking for a number that is more than the random number, skipping over the 1 since the host does not want to send a message to itself. Walking through the procedure, 47 is more than 40, the second number is a 1 and therefore disregarded; then it compares 47 with the third number, which is 60. Since 47 is less than 60, it is this column number that will be the destination node. Sixty is in the number 2 column (40 being in the 0 column, and 1 in the 1 column). Therefore, node 1 will send to node 2.

When inputting the numbers into the table, the position where the row and column elements are equal [0][0],[1][1],[2][2], etc., must be a 1. The last number must be 100 except for in the last row where the 100 must come before the 1 (see table above). In the row for node 1, there is a 40 percent chance to send to node 0, 20 percent (60-40) to send to node 2, 10 percent (70-60) to send to node 3, 20 percent (90-70) to send to node 4 and finally 10 percent (100-90) to send to node 5.

Traffic Rates - is a one dimensional array with the elements again corresponding to the nodal numbers. As an example say Array [0..5] is 50, 30, 20, 45, 25, 35. So the traffic rate for node 0 is 50, for node 1 is 30, and so on. This goes into a Poisson number generator where it determines when the node will want to send its next call request.

Adjacency Table - is Maxnodes by (Maxconnect + 1). The reason for adding 1 is that the first column is the node number that we are examining.

0,1,4,5
1,0,2,-1
3,2,4,5
4,0,3,-1
5,0,2,3

Refer to the table above. The first number in each row will be the node that we are examining. The other numbers are the other nodes that are connected to it. If only two instead of three connections exist, then a -1 must be inserted to fill the table. Node 0 is connected to nodes 1, 4, and 5. Node 1 is connected to just nodes 0 and 2.

Number of Packets - provides three choices for the message sizes to be used in the simulation. First, the size is stated and then its percentage, for example 3, 30, 6, 60, 9, 100. Thirty percent of the time the message will be of size 3; another 30 percent (60-30) it will be of six packets; and the rest of the time (40 percent = 100-60) it will be 9 packets long. The maximum number of packets, which is the maximum message size, must be one less than the maximum queue length. This is to ensure that the flow balancing control scheme works. The threshold value must be set one greater than the maximum message size but cannot be larger than the maximum queue length.

Maximum Message Size < Threshold <= Maximum Queue Length

The threshold is automatically set for the user as 1 greater than the maximum number of packets and a check is done to be sure that this does not exceed the maximum queue length. This check is performed only when the flow balancing scheme has been specified.

OUTPUT FILES describes six files the simulation writes to, which provides documentation to the user on the simulation's activities.

SUMMARY - file contains a listing generated every X ticks (X being the number the user installed under More Outputs) including the last clocktick. The file displays the number of messages completed, number of packets lost and discarded, and the number of requests discarded. It presents results as a running total and the difference from the last write.

MONITOR file is written to every time SUMMARY has information written to it. This file tells the user what each node has done in terms of packets passing through, completed messages, number of packets lost and discarded, and number of requests discarded. On the final clocktick the MONITOR file shows when any of the output queues were frozen and then freed, as well as what was in each output queue element.

HOPS - file is written to at the end of the run and displays the message number, time the request was sent out, how many times that request timed out, time the last acknowledgement was received, the total time (time request sent out to time the last acknowledgement was received), time first packet was sent, and the message time (first packet to last acknowledgement) and then average total time and average message time. The messages are broken down by the node and then reduced further to the hop count and message size.

ERROR - file incorporates error messages as well as some other notes that might be important to the user.

LOST - file is written to whenever a packet is lost due to line noise. It writes out the time, node and message number, and the source and sink nodes.

DISCARDS - file is like the LOST file except that it is written to whenever a request or packet is discarded.

The LOST and DISCARDS files are available to the user as a trouble shooting guide, or just for the curious. These files provide information on how to find an important packet if it is missing and the user wishes to determine if the packet is lost or discarded.

Now that the data file has been constructed, you are ready to run the simulation. Type in the executable name for this simulation, "Net_Simulator," a space, and then the name of your data file. The input line with a data file with the name of data62.d would resemble the following example:

```
Net_Simulator data62.d
```

HOW TO ANALYZE THE RESULTS

Now that you ran the simulation and have the results, what do they mean? First, look at the output file called SUMMARY. The last row in the table will include the final message count, which is the throughput in messages for this simulation run. SUMMARY also provides a running total of the number of packets lost and discarded and the number of requests discarded. There are also columns for the same variable that contain the numerical increase in these variables since this file was last written to. The output file, HOPS, provides the user with a listing of the average message time as well as average total time. HOPS is broken down by node, hop count, and then message size. The total time is the time from "REQUEST SENT" to "LAST ACK" and message time is the time from "FIRSTPK" to "LAST ACK."

The file MONITOR can be examined to discover what happened in each node. This file is written to every X clockticks where X is the input for the variable "More Outputs." The listing is broken down by node and then by each output queue of that node. It lists the number of packets that passed through the queue, average time on queue, the maximum and minimum times that a packet spent on the queue, and whether

the output queue ever had to be frozen. On the final clocktick, this file will contain the totals for message count, packet count, number of packets lost, and the number of discarded requests and packets. Then for each node, the file lists what each element of their output queues and input buffers contained.

Usually, the user wants to maximize throughput; therefore, SUMMARY will be the most important file. As an exercise, try changing some of the parameters to obtain a better message count.

A normal run can be determined by verifying that the last row of the file SUMMARY was written to on the final clocktick, which you have specified in the variable Maxticks. If the clocktick of the last row is anything else or if that file is empty, the run was terminated early for some reason.

TROUBLE SHOOTING

The first step taken is always look at the ERROR file. The last line within the file should appear similar to the following example:

FUNC6: TICKS 20000; Node 0; in Loadoutbufs nothing in packet

The number after TICKS should be the number for Maxticks. The number after node should be one of the nodes in the network. The number of lines that look like the above sample vary, but there should be at least one of them. If there is anything else as a last line, then there is indeed a problem. With a terminated simulation run, the last line of ERROR will have some kind of error message as to what node the program was working with and on which clocktick the program terminated.

If the network is less then ten nodes, you can set the Want Printouts flag to one, set the Time to Start Printing variable to 100 clockticks less than the error message time and rerun the simulation. This run will now produce files for each individual node in the network. Open the file of the node where the error occurred, and proceed to the end of the file. Problems that arise in this simulation are very similar to Unix pointer errors, the location of where the program terminates as no correlation as to where the real problem lies. The next step would be to try and trace the packet that is causing the problem back through the network and locate the actual problem source. These problems will hopefully be few and far between.

Example of User Input Data File

Data files for the input to a six node network, all connections between nodes are working.

```
 /*************************************************************************/
SCHEME 2

NAME:  data62.old

MAXNODES
6
Maxconnect
3
Maxticks
20000
Maxvc
30
Maxoutstanding Mess
10
Buffer pool size
20
Maxque length
15
Waitlimit
130
Percent of packets lost
.01
PAKDISCARD
1
Congestion scheme
2
MAXSTORE
5
WarnPercent for quelen
0.3
IgnoreTime
40
```

Listen Time
120
Rate Reduction Factor
0.6
More Outputs
500
Want Printouts
0
Time to Start Printing
0
Static Routing Table
Node 0
1,90,1,100,5
2,70,1,100,5
3,50,4,100,5
4,85,4,100,5
5,60,5,100,1
Node 1
0,90,0,100,2
2,80,2,100,0
3,85,2,100,0
4,50,2,100,0
5,40,0,100,2
Node 2
0,95,1,100,5
1,85,1,100,5
3,75,3,100,5
4,80,3,100,1
5,90,5,100,1
Node 3
0,50,4,100,5
1,10,4,100,2
2,80,2,100,5
4,90,4,100,5
5,75,5,100,2

Node 4
0,95,0,100,3
1,85,0,100,3
2,60,3,100,0
3,80,3,100,0
5,50,3,100,0
Node 5
0,90,0,100,2
1,50,0,100,2
2,80,2,100,3
3,90,3,100,2
4,60,0,100,3
Destination Selection
1,20,40,60,80,100
40,1,60,70,90,100
30,40,1,65,75,100
25,50,60,1,75,100
10,20,45,75,1,100
50,55,65,85,100,1
Traffic Rates
50,30,20,40,25,35
Adjacency Table
0,1,4,5
1,0,2,-1
2,1,3,5
3,2,4,5
4,0,3,-1
5,0,2,3
Number of packets
3,30,6,60,9,100

BIBLIOGRAPHY AND CITATIONS

Auja, V. "Routing and Flow Control In System Network Architecture." IBM System Journal, 18, no. 2 (1979).

Edge, S. "Comparison of the Hop-by-Hop and Endpoint Approaches to Network Interconnection." International Symposium Held in France February 1979. Edited by Jean-Louis Grange and M. Gein. Amsterdam, New York and Oxford: North-Holland Publishing Company, 359-377 (1979).

Fischer, W., K. P. Sauer and W. Denzel. "A Simulation Technique For Communications Protocols Based on a Formal Specification by SDL." In Protocol Specification, Testing and Verficiation, V. Proceedings of the IFIPWG.1 Fifth International Workshop on Protocol Specification, Testing, and Verification Organized by the LAAS du CNRS, Toulouse-Moissac, France 10-13 June 1985. Edited by Michel Diaz. Amsterdam, New York and Oxford: North-Holland and Elsevier Science Publishing Co. 333-347 (1986).

Gerla, M. and L. Kleinrock. "Flow Control: A Comparative Survey." IEEE Trans. Comm. (Institute of Electrical and Electronics Engineers), COM-28, no. 4 (April 1980).

Jones, W. Programming Concepts, A Second Course (With Pascal). Englewood Cliffs, NJ: Prentice-Hall Inc., 295-296 (1982).

Kermani, P. and K. Bharath-Kumar. "A Congestion Control Scheme for Window Flow Controlled Computer Networks." Computer and Electrical Engineering, 10, no. 3, 229-243 (1983).

Lam, S. and M. Reiser. "Congestion Control of Store and Forward Networks by Input Buffer Limits - Analysis." IEEE Trans. Comm. (Institute of Electrical and Electronics Engineers), COM-27, no. 1 (January 1979).

Pouzin, L. "Methods, Tools, and Observations On Flow Control in Packet Switched Data Networks." IEEE Trans. Comm. (Institute of Electrical and Electronics Engineers), (April 1981).

Price, W. L. "Simulation of Routing Doctrines, Flow Control And Congestion Avoidance." Computer Networks and Simulation. Edited by S. Schoemaker. Amsterdam, New York and Oxford: North-Holland Publishing Company, 141-153 (1978).

Remes, A. "Simulation Techniques in Network Design." Computer Networks and Simulation. Edited by S. Schoemaker. Amsterdam, New York and Oxford: North-Holland Publishing Company, 69-84 (1978).

Rinde, J. and A. Caisse. "Passive Flow Control Techniques for Distributed Networks." In Flow Control in Computer Networks: Proceedings of International Symposium Held in France, February 1979. Amsterdam, New York and Oxford: North-Holland Publishing Company, 155-160 (1979).

Schoemaker, S. "On Simulation." Computer Networks and Simulation. Edited by S. Schoemaker. Amsterdam, New York and Oxford: North-Holland Publishing Company, 69-84 (1978).

Tanenbaum. A. Computer Networks. 2d ed. Englewood Cliffs, NJ: Prentice-Hall, Inc. (1988).

Woodside, C. M. and J. R. Montealegre, "On Packet Buffering and Protocol Performance." In Protocol Specification, Testing and Verficiation, V. Proceedings of the IFIPWG.1 Fifth International Workshop on Protocol Specification, Testing, and Verification Organized by the LAAS du CNRS, Toulouse-Moissac, France 10-13 June 1985. Edited by Michel Diaz. Amsterdam, New York and Oxford: North-Holland and Elsevier Science Publishing Co. 375-386 (1986).

Zhang, L. "Some Thoughts on the Packet Network Architecture." Computer Communication Review, 3-17 (January/April 1987).

Zinky, J. A. and J. Etkin. "Integrating Executable Models Into the Network Development Life Cycle." IEEE Symposium on Computer Networks. IEEE Computer Society Press, 87-93 (1986).