University of Canterbury

Department of Computer Science

Honours Project Report

Packet Flow Analysis in X.25 Public Data Networks

A Simulation Tool

Paul Reynolds

Supervisor R. Hunt

October 1984.

Abstract

This report describes a Simula program designed to model the behaviour of X.25 Public Data Networks and produce statistics on various performance indices of the network. These include such indices as:

- packet transfer times for each pair of nodes
- time spent in each node
- line utilisations
- wait times for each line
- buffer utilisations

The user descibes the network by suppling the model with information on each node and the lines that connect the nodes. This allows a large variety of X.25 network configurations to be studied.

Contents

1 Introduction

The aim of this project was to develop a tool for the study of the effect of different network configurations and routing algorithms on packet switched network efficiency. The tool is a Simula program which models the behaviour of Packet Switching Networks (PSN). The model takes as input a configuration file describing the nodes of the network and the lines between them. It produces statistics on:

- packet transfer times for each pair of nodes
- the time packets spend in each node
- the number of packets with errors that were transmitted between each pair of nodes
- the number of retransmissions that occured at each node
- the number of acknowledgements between each pair of nodes
- the line utilisation
- the wait times for each line
- the buffer utilisation at each node
- the number of packets rejected at each node due to a shortage of buffers.

These statistics are useful for the intial design or in planning extensions to an existing network. It was the interest expressed by the New Zealand Post Office in obtaining this type of information, for possible expansions to the existing PSN, that prompted the initation of this project.

A Packet Switching Network is a data communication service in which the information transmitted on the lines is grouped into variable length frames which are separated by variable length pauses depending on the volume of data being transmitted. Each data frame consists of service data (to enable the packet to be routed through the network and allow the detection of transmission errors) and user data. (see figure 1) These frames are transmitted independently across the network and are regrouped by the network at the destination for transmission to the receiving data terminal equipment (DTE). As a result, packets from different terminals can travel via the same communications channel thereby providing increased channel utilisation.

The user/network interface for a PSN is described in CCITT standard X.25 [1]. This standard covers the physical, data link, and network layers of the International Standards Organisation Open System Interchange (ISO OSI) model [2] and uses a sliding window protocol to handle transmission errors [3]. There is no standard covering the internal protocols of PSN. However most use some form of sliding window protocol at layer 2 (Data Link) to handle errors and at layer 3 (Network) to impose flow control to ensure that one terminal does not monopolises a line.

This report assumes that the reader is reasonably familiar with the concepts of both the CCITT standard X.25 and the ISO OSI model [4,5].

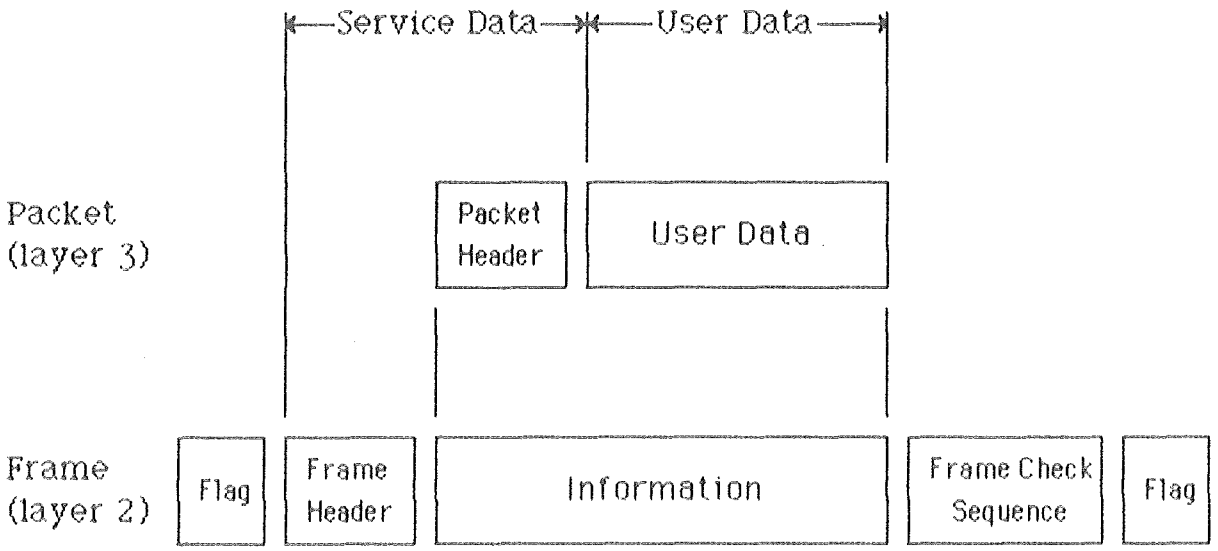


Figure 1 Packet Format

2 Model design

The decision to use a simulation rather than an analytical model was taken after an examination of the literature on routing in PSN [6,7,8,9], which consists mainly of proposed analytical models supported by limited simulation. It was considered that the assumptions required to produce tractible analytical models were too broad and in some cases were not reasonable. The main example of this is the assumption of node independence, which means that the number of packets transmitted from any node has no effect on the number of packets received by any other node. This assumption holds only in very limited conditions that are not often experienced in actual networks. It was also felt that a simulation model would be more flexible.

2.1 Design decisions

In this project the Simula programing language was used on a Data General Eclipse S/130. Simula has distinct advantages in flexiblity and power over GPSS and similar simulation languages particularly for more complex models such as the one under study. There are other less well know simulation programming languages available which may have been just as suitable, but Simula was available on a suitable machine.

The model uses a sliding window internal protocol without piggybacking of acknowledgements. Piggybacking was not implemented because of the coding difficulties that this would cause.The major difficulty is ensuring that the correct layer 2 receive state variable V(R) was placed in the N(R) of each packet. It is not known until the transmission stage which node the packet will go to and so it is at this point that the N(R) must be assigned the correct value. While this is possible the code to carry out this operation and to extract the N(R) at the other end is complex and so it was decided that this should be avoided. The lack of piggybacking should be minimal as there are only a small number of small (48 bits) packets being transmitted. This will cause slightly higher line and buffer utilisations and marginally larger wait times for lines.

This model does not consider DCE-DTE communication in that packets arrive at a data switching exchange (DSE) from a DTE without errors so long as there is a free buffer for that packet. Packets are removed from the DSE upon arrival at their destination immediately freeing the buffer occupied. A virtual circuit through a PSN passes from a DTE to a DCE and then through one or more DSEs to another DCE and finally another DTE. A DSE being a switching exchange that switches a packet between DCEs or other DSEs. These simplifications can be considered as divorcing the role of the DCE from that of the DSE and providing both with seperate buffer pools. When a packet arrives at its destination the DSE passes the packet to the DCE freeing the buffer the packet occupied in the DSE as it does so. The DCE is then left with the task of transmitting the packet to the required DTE. The decision to enforce this seperation was made on the grounds that

the wide range of performance characteristics of the many DTEs attached to a DCE would make it impossible to model the rate at which the buffers would clear successfully. (A packet mode DTE connected with a 9600bps line could clear a packet in approximately 117 milliseconds whereas a character mode DTE connected with a 300bps line would take over 3.5 seconds to clear the same buffer.) While layer 2 retransmissions take care of transmission errors layer 3 retransmissions are responsible for recovering from sequence errors at layer 3 caused by variable routing algorithms. These can occur because packets switched, by the routing algorithm, to less congested routes may arrive at their final destination ahead of the last few packets to use the old route.

A PSN can be viewed as a collection of nodes connected by lines (see figure 2). Each node consists of:

- a source of packets. This generates packets and handles layer 3 retransmissions
- a sink. This accepts packets which are at their destination and handles layer 3 acknowledgements
- a receiver. This receives packets and generates layer 2 acknowledgements
- a route procedure that sends a packet to the sink or an output line depending on its destination
- a transmit procedure which sends a packet to another node and sets up a layer 2 retransmission. This procedure also introduces transmisson errors.

2.2 Simulation Program

The model consist of three main classes; a link class "Packet", and two process classes "ServerClass" and "NodeClass". The link class **Packet** acts as the packets of the network and is passed between the instances of "NodeClass". Each instance of "NodeClass" represents a node of the network. There is one instance of the "ServerClass" which represents the lines of the network. It's procedures "Acquire" and "Release" ensure mutual exclusion on the lines of the network. As well as these classes to model the operation of the network the class "StatsCollecter" contains all the statistic gathering and reporting procedures used by the program. The procedure "Initialise" as its name suggests intialises the network by reading from the configuration file. The process class **RunningMessage** displays a message on the terminal every 5 simulated seconds indicating that the model is still running and how many simulated seconds have elapsed.

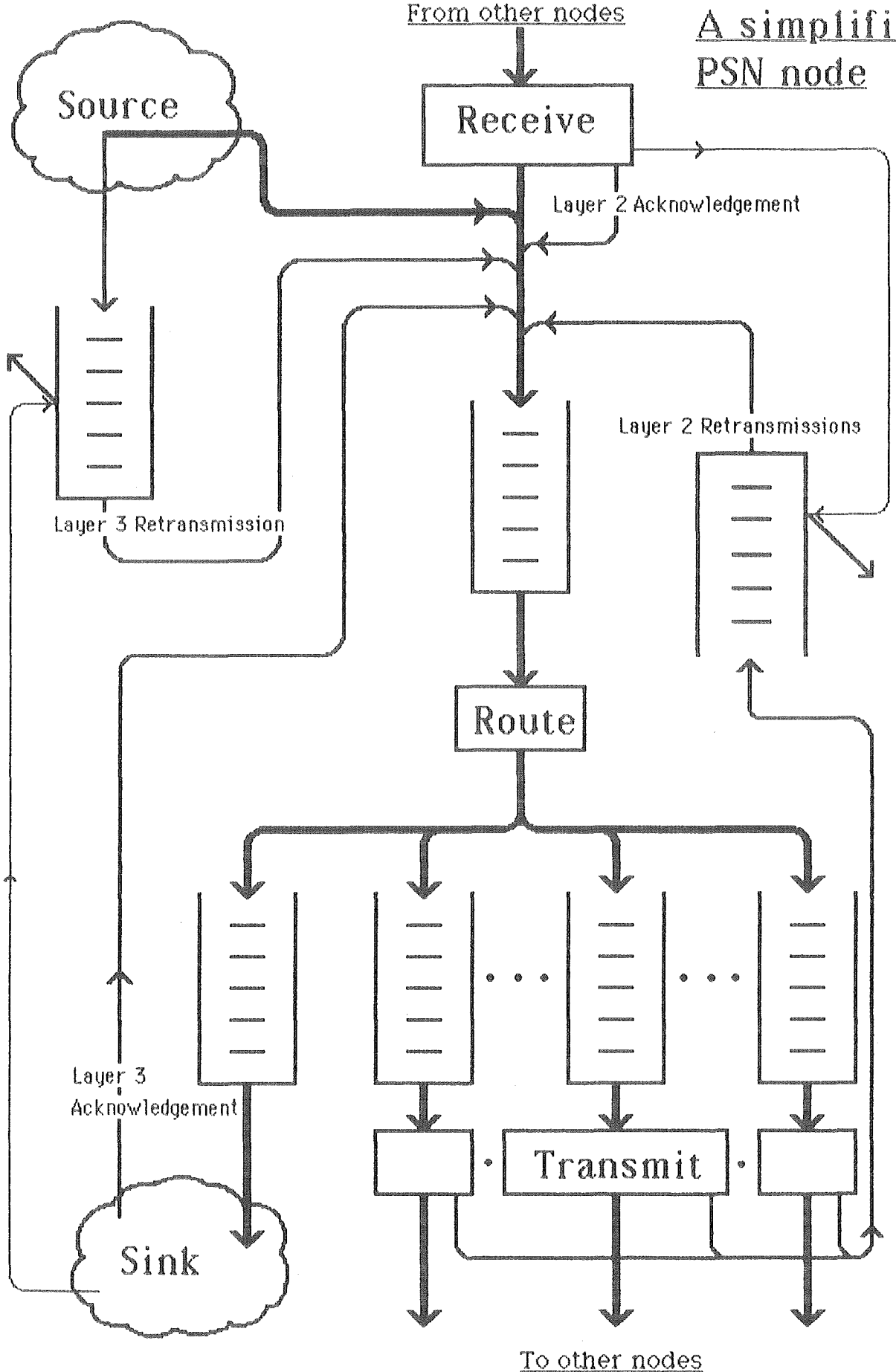
The logical structure of a node of the model is displayed in figure 3 and a more detailed description of the function of each procedure makes up the remainder of this section.

2.2.1 ServerClass

This process class ensures mutual exclusion on the lines between the nodes. The boolean array "channelfree" indicates the state of each line and is only altered by the procedures

Figure 2

A simplified PSN node



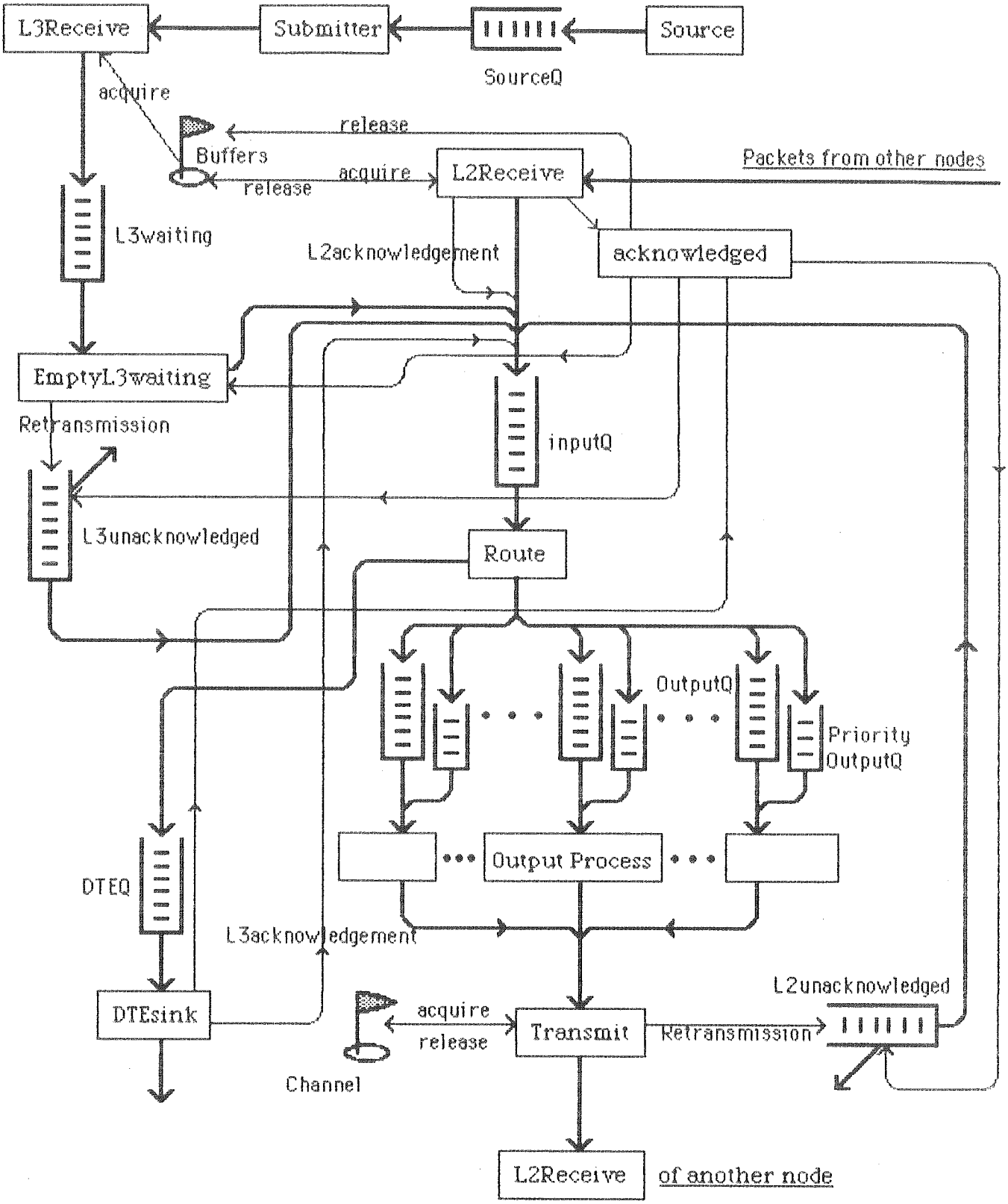


Figure 3

A Node of a PSN

"Acquire" and "Release".

The procedure **Acquire** places the current process into a queue for the required line and then waits until that line is free. Once it is free the start of the use of that line is monitored then the array "channelfree" is altered to indicate that the line is no longer free. Finally the current process is removed from the waiting queue.

The procedure **Release** monitiors the end of the use of the line and changes the state of the line to free. It then wakes up the first process waiting for that line if there are any.

2.2.2 NodeClass

This class and its related process classes (described in section 2.2.3) model the activities that take place within each node of the network. The procedure "L2Receive" handles the reception of packets from other nodes in the network. The procedure "Route" is called by "L2Receive" or "EmptyL3waiting" to decide where the packet should next be sent unless the packet is an acknowledgement in which case the procedure "Acknowledged" is called by "L2Receive" to process the acknowledgement (an RR frame).

The procedure "Route" activates the Sink process if the packet has reached its destination otherwise the "Outputline" process for the next line on the packets path is activated. This process calls the procedure "Transmit" to schedule the retransmission and carry out the actual mechanics of the transmission.

The procedure **L2Receive** is responsible for receiving packets transmited from other nodes of the network. It ensures that there is a spare buffer available for the arriving packet, that there have been no transmission errors and the packet is in sequence at layer 2 before the packet is accepted for further processing. There are four possible ways in which an incoming packet can be processed. If the packet has been correctly received and is an information packet the local layer 2 receive state variable "L2VR" is incremented and the layer 2 acknowledgement timer is restarted. If the layer 2 receive window has been reached a layer 2 acknowledgement (an RR frame) is created and added to the head of the input queue and the receive window is advanced. Finally the information packet is added to the end of the input queue and the procedure "Route" is called. If the received packet is a layer 2 acknowledgement the procedure "Acknowledged" is called to process the acknowledgement and the packet is not added to the input queue and route is not called. If the packet is a layer 3 acknowledgement that has arrived at its destination then the procedure "Acknowledged" is called to process the acknowledgement and the packet is not forwarded. Otherwise it is added to the head of the input queue and the procedure "Route" is called.

The procedure **L3Receive** is called by the process class "Submitter" (see section 2.2.3.4) to introduce packets to the network via the queue "L3waiting" once the source has created the packets.

The procedure **EmptyL3waiting** is called by "L3Receive" or "Acknowledged" to take packets from the queue "L3waiting", set up a layer 3 retransmission, add the packet to the input queue and call the procedure "Route". This procedure also increments the layer 3 send state variable "L3VS" each time it adds a packet to the input queue and will not add a packet to the input queue if the layer 3 send window has been reached for that packets destination.

The procedure **Acknowledged** is called when a layer 2 or 3 acknowledgement has been received, it then searches the appropirate queue ("L2unacknowledged" or "L3unacknowledged") for all the pending retransmissions that are acknowledged by the received packet. The send window of the correct layer is then advanced if any retransmissions were acknowledged and the buffers occupied by the pending retransmissions are freed.

The procedure **Route** takes the first packet from the input queue and determines if it has reached its destination; if so it is added to the DTE queue and the sink is activated. If not the procedure "FindLine" is called to determine the "Outputline" the packet should be transmitted on. The packet is then placed in either, the priority output queue if it is an acknowledgement or in the output queue, and then the output process for that line is activated.

The procedure **FindLine** is local to the procedure "Route" and called to determine which of the available output lines should be used for this packet. The way in which this line is selected depends on the routing method being used by the network. If static routing is being used the line is selected by a simple table look up. If one of the biased shortest queue methods is used the selection amongst the possible options is made by comparing the lengths of the priority and normal output queue lengths of each line.

The procedure **Transmit** handles the mechanics of the actual transmission and is called by the output process to actually transmit the packet. This procedure sets up a layer 2 retransmission, determines if a transmission error has occured, acquires the line, holds for the time required to transmit the packet, releases the line, and finally calls "L2Receive" of the destination node to pass the packet on.

The procedure **Debug** is called by "L2Receive" and "Transmit" if the global flag "debugon" is true indicating that debugging output is required. It prints the debugging output for the packet received or sent to the debug file.

2.2.3 Related Processes

Conceptually these processes are part of the node and should therefore be declared within the class "NodeClass" however in Simula process classes must be declared at the outer most level. This will explain the sometimes convoluted use made of the Simula dot notation.

2.2.3.1 TimerProcess

This process creates a layer 2 or 3 acknowledgement, advances the appropriate receive window, adds the acknowledgement to the head of the input queue, and calls "Route" to send the acknowledgement on its way. This process is reactivated each time a packet is received at the corresponding layer so that it will only become active if the gap between two incoming packets at either layer 2 or 3 is larger than the delay of the reactivate. That is if the gap between the packets is larger than the acknowledgement timer.

2.2.3.2 OutputProcess

This process takes packets from either the priority output queue if it is not empty or the output queue, and calls the procedure "Transmit" to transmit the packet to the node to which this output line is linked. At the same time the layer 2 send state variable "L2VS" is incremented. This process continues sending packets until the layer 2 send window is reached or both output queues are empty.

The procedure **PriorityPackets** takes packets out of the priority output queue and transmits them to the node to which this output line is linked. Note that packets in this queue are sent in preference to packets in the output queue.

2.2.3.3 Source

This process creates the packets that are to be passed around the network. As the packets are created they are placed in the source queue and the process "Submitter" is activated to submit the packet to the network. The distributions used in the source are discussed in section 3.1 and a means of experimenting with the source parameters is discussed in section 5 of this report.

2.2.3.4 Submitter

This process is activated by a "Source" and calls "L3Receive" so long as there is a buffer available and the source queue is not empty. If there is not a buffer available the process holds for a short time and then continues attempting to submit packets.

2.2.3.5 Sink

This process takes packets from the DTE queue and discards

them after first checking if they are acknowledgements or information packets. If the packet is a layer 3 acknowledgement (no other acknowledgements reach the sink) the procedure "Acknowledged" is called to cancel the pending layer 3 retransmissions that the acknowledgement covers. If the packet is an information packet and is in sequence at layer 3 the fact that the packet has arrived is recorded by the monitor, the layer 3 receive state variable is incremented, and a layer 3 acknowledgement is created if the layer 3 receive window is reached. If the packet is out of sequence it is discarded.

2.2.3.6 Retransmitter

This process is activated by the procedure "Transmit" or "EmptyL3waiting". It holds for a specified time and then adds the packet it was set up with to the head of the input queue and calls "Route" to send it on its way. It will continue to do this until the boolean needed is set to false. This is done in the procedure "Acknowledged" when the approiate acknowledgement has been received. Each instance of this process class is placed in either the "L2unacknowledged" or "L3unacknowledged" queue depending on which layer the retransmission is set up by.

2.2.4 StatsCollecter

There is one instance of this class; "monitor". As it name suggests this class contains all the statistic gathering and reporting procedures.

2.2.4.1 The collection procedures

These procedures collect the statistics produced by the model. The procedures "UtilStart" and "UtilEnd" monitor the utilisation of lines. The procedure "Acknowledgements" monitors the number of times the procedure "Acknowledged" is called, that is how many layer 2 or 3 acknowledgements are received. The procedure "Waiting" monitors the time a packet spends waiting for a line. The procedure "Creation" monitors the creation of packets at each source. The procedures "Entry" and "Departure" measure the amount of time a packet sends in each node. The procedure "PacketTransfer" notes the successful arrival of a packet at its destination and collects information from the packet on the time it spent in each node. The procedure "Error" counts the number of packets with transmisson errors that are transmitted between each pair of nodes. The procedure "BufferUtilisation" monitors the utilisation of buffers in each node. The procedure "Bufferlack" notes the number of occasions on which a packet can not be received because there is no free buffer in the node. For a detailed description of the statistics gathered by these procedures see section 3.3.

The procedure "Reset" initialises all the statistic collecting variables and is called twice, in the main program. Once before the start of the presimulation period and again before the start of the simulation period.

2.2.4.2 The procedure Report

This procedure calls the eleven reporting procedures to print out the gathered statistics in suitable formats. The report is written to the file outf. The external name of this file is specified by the user when the program is run. A message is displayed on the screen as soon as this procedure is called to indicate that the simulation is complete and that the report is being written.

The procedure **Mean** is called by several of the reporting procedures to calculate the mean of the passed values. Note that this procedure returns zero instead of attempting to divide by zero.

The procedure **StDev** is called by several of the reporting procedures to calculate the standard deviation of the passed values. Note that this procedure returns zero instead of attempting to divide by zero or find the square root of a negative number.

2.2.5 Initialise

This procedure reads from the configuration file the network constants and calls "EstablishNode" to read the data required for each node and to create an instance of "NodeClass" and its related processes.

The procedure **EstablishNode** is called to read each node description from the configuration file and to create an instance of "NodeClass" and one instant of each of its related processes.

2.2.6 Main Program

The main program calls the procedure "Initialise" to establish the network, it then activates each source and the "RunningMessage" process. The "monitor" is reset, the program holds for the presimulation time, the "monitor" is reset again, and the program holds for the simulation time. Finally the "monitor" reports the statistics and all open files are closed.

3 Use of the model

3.1 The configuration file

The configuration file describes the network to be modelled. It consists of a network title (not more than 80 characters), the number of nodes, and the routing method followed by a number of node descriptions and finally some network constants.

Each node description consists of the name of the node (not more than 6 characters), the number of buffers in the node, the number of output lines from that node, a number of line descriptions, a description of the source associated with that

node, and finally an initial routing table.

For static routing this table consists of a pair of node numbers, the desired destination and the packets immediate destination. For the biased shortest queue methods instead of one immediate destination up to five alternatives may be specified (if less than five are required the rest must be 0). Each line description consists of the destination of the line, its transmission rate (in bits per second), its length (in kilometres), its propagation time (in microseconds per kilometre), and the bit error probability for the line.

The description of the source gives the average intermessage time, the average interpacket time, the mean length of each message in packets, and the resubmission back off time. The source generates messages consisting of one or more packets with a negative exponential delay between the completion of one message and the start of the next, the mean of this delay is the intermessage time. There is a negative exponential delay between the packets that make up a message, the mean of this delay is the interpacket time. The number of packets in each message is negative exponentially distributed. The destination of each message is uniformly selected from all the nodes in the network.

This source was designed to give an interesting packet creation pattern based on what seem to be reasonable assumptions. No claim as to the representative nature of this source is made. The limited amount of published data on traffic generation for Packet Switched Networks makes statisical analysis leading to the formulation of a general source impractical. To assist in the selection of source parameters a small program called the source emulator has been written and is described in section 5 of this report.

Note that words in a configuration file, except the network and node names, are ignored by the program; they are there to make the file more readable for the user. The relative postions of the numbers are important so the blank lines and the lines containing words must be retained to ensure the correct interpretation of the numbers by the program.

3.1.1 Routing algorithms

There are at present three routing methods [3,10] available:

1. Static Routing - this entails a simple table look up from a static routing table
2. Biased Shortest queue (ignoring line speed) - this entails selection of the output line from a range based on queue length for each line
3. Biased Shortest queue - this entails selection of the output line from a range

based on queue length for
each line and the speed of
that line

An example configuration file

Figure 4 shows the configuration of this network, all the lines are run at 64 kilobits per second.

```
Configuration Example
4 nodes
2 Biased shortest Queue (ignoring line speed)

Node 1
CHCH
50 buffers
3 output lines
line dest, speed, length (km), propagation, error prob.
      2    65536      390           4          0.00005
      3    65536     1065           4          0.00005
      4    65536      360           4          0.00005
source (intermessage, interpacket times; packets/message; backoff)
 1.5 0.2 1.25 0.015
routing table (to, via)
 1 1 0 0 0 0
 2 2 3 4 0 0
 3 3 4 2 0 0
 4 4 2 3 0 0

Node 2
WELL
50 buffers
3 output lines
line dest, speed, length (km), propagation, error prob.
      1    65536      390           4          0.00005
      3    65536      675           4          0.00005
      4    65536      750           4          0.00005
source (intermessage, interpacket times; packets/message; backoff)
 1.5 0.2 1.25 0.015
routing table (to, via)
 1 1 3 4 0 0
 2 2 0 0 0 0
 3 3 4 1 0 0
 4 4 1 2 0 0

Node 3
AUCK
50 buffers
3 output lines
line dest, speed, length (km), propagation, error prob.
      1    65536     1065           4          0.00005
      2    65536      675           4          0.00005
      4    65536     1425           4          0.00005
source (intermessage, interpacket times; packets/message; backoff)
 1.5 0.2 1.25 0.015
```

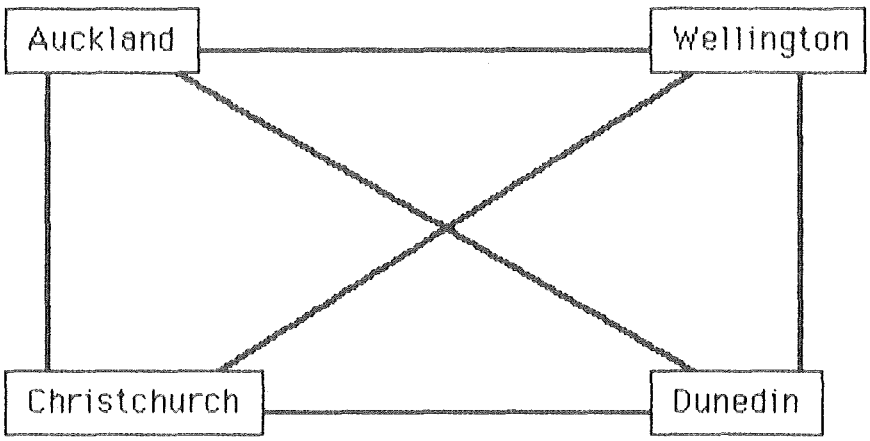


Figure 4 A configuration example

```
routing table (to, via)
 1 1 2 4 0 0
 2 2 4 1 0 0
 3 3 0 0 0 0
 4 4 1 2 0 0

Node 4
DUND
50 buffers
3 output lines
line dest, speed, length (km), propagation, error prob.
      1    65536      360           4          0.00005
      2    65536      750           4          0.00005
      3    65536     1425           4          0.00005
source (intermessage, interpacket times; packets/message; backoff)
 1.5 0.2 1.25 0.015
routing table (to, via)
 1 1 2 3 0 0
 2 2 3 1 0 0
 3 3 1 2 0 0
 4 4 0 0 0 0

routing alg time
0.0001
frame check seq time
0.0
time to process an acknowledgement
0.00001 0.005
Send window size
7 3
Receive window size
7 2
retransmission timer (in bit times)
28000 70000
Acknowledgement timer (in bit times)
4000 4000
```

3.2 Simulation times and seeds

The presimulation time allows time for a number of packets to be sent through the network before the statistics collection commences. This means that there may be several packets in transit when the actual simulation commences and thus it is possible for a few more packes to arrive than were generated during the simulation time. This also means that some of the buffers and lines of the network will be in use at the time the actual simulation occurs. A presimulation of less than 20 seconds will be adaquate in most cases as a longer time than this will simply allow large numbers of packets to pass through without being measured.

The simulation time should be of a reasonable length. It should not be less than 120 seconds for realistic simulation because of the small number of packets that are generated and the large effect one or two retransmissions would have on apparent

performance. The model when run on a Data General Eclipse S/130 takes approximately 2.5 minutes of actual time for each minute of the simulation. However the Eclipse used did not have a floating point unit and so real numbers have to be simulated by software reducing the running speed of programs.

The five random number seeds should be odd and have no obvious pattern. They are included in the statistics report so that they may be used again to repeat an experiment under the same workload patterns but with different network configuration. The first seed is used to determine if a transmission error has occured, the second determines the intermessage gap, the third determines the number of packets in a message, the fourth determines the destination of a message, and the fifth determines the interpacket gap.

3.3 The collected statistics

3.3.1 Title

The title gives the network name, the presimuation time, simulation time, and random number seeds. This information allows the user to recreate an experiment using different network configurations to examine the effects of various changes.

3.3.2 Number of packets created

This table gives the number of information packets created at each node destined for each other node during the simulation time.

3.3.3 Packet transfer times

This table gives the number of information packets sucessfully transmitted between each pair of nodes in the network, the minimum, maximum, and mean times taken and the standard deviation of these times. The maximum may be several times the minimum if any retransmissions have occured. Note that if the presimulation time is non-zero there may be slightly more packets transmitted than created as some packets created during the presimulation time may still be in transit. This table gives an indication of the overall network performance.

3.3.4 Time spent in each node

This table gives an indication as to where in the network information packets are spending their time. The time spent in a node is the time between the arrival of the last bit of the packet at the node and the sucessful departure of the last bit of the packet from the node. If the packet suffers a transmission error it is not considered to have succesfully departed the node. The distinction between local and nonlocal is made to stop the very short times spent in the destination node artifically lowering the mean time spent in a node. A packet is in a local node when that node is its destination and the packet is

therefore simply handed to the DCE.

3.3.5 Number of packets with errors received

This table gives the number of packets suffering transmission errors transmitted between each pair of nodes. This does not include errors suffered by acknowledgement packets.

3.3.6 Number of Retransmissions

This table gives the number of information packets that have been retransmittred between each pair of nodes at layer 2 and layer 3. This table gives a good indication as to how well the network is recovering from errors that do occur. If large numbers of retransmissions occur while there are little or no errors it is an indication that either the retransmission timer is too short or that the acknowledgement timer is too long.

3.3.7 Number of acknowledgements

This table gives the number of acknowledgement frames (at layer 2) and packets (at layer 3) that have been successfully transmitted between each pair of nodes. If the number of acknowledgements is excessive this is an indication that the acknowledgement timer is too short or that the receive window is too small.

3.3.8 Line Utilisation

This table gives the percentage utilisation of each line in the network. If any of these utilisations are excessively high this will be reflected in high wait times for lines and high times spent in the affected nodes.

3.3.9 Wait times for lines

This table gives the time that information packets spend waiting for the lines of the network. A packet starts waiting for transmission once it has been routed to a particular output line and stops when its first bit is transmitted.

3.3.10 Buffer Utilisation

This table gives the percentage utilisation of buffers in each node by both information and acknowledgement packets. A buffer is used by an information packet from the time it is received until an acknowledgement for it arrives. At the source of the packet the acknowledgement must be at layer 3, at intermediate nodes on its path a layer 2 acknowledgement frees the buffer. The buffer of an acknowledgement packet is freed as soon as its contents has been transmitted.

3.3.11 Packets rejected due to lack of buffers

This table gives the number of times a packet was not

accepted by a node because there was no spare buffer in the node. If the packet was submitted by the source of the node this is considered a local rejection. If the packet should have been received from another node this is considered a nonlocal rejection. These should be zero, if not the configuration has major problems in transmitting the offered workload.

3.4 The Dump and Debug files

The Dump and Debug files are provided to assist in the modification and validation of the model. The dump files provide a means of ensuring that packets are indeed arriving at their destination in sequence and without duplicates. One dump file is produced for each node in the network and is numbered with the number of the node that produced it (for example dump1 is the dump file for node 1). A **dump** file has the following form:

```
Packets receive by Node 1. From Nodes:
   1   2
--------
   1
       1
   2
   3
       2
```

This indicates that node 1 has received packets 1,2 and 3 from node 1 and packets 1 and 2 from node 2. Note that the packets are indeed in order and there are no duplicates.

The **debug** file shows the packets that are transmited between a given pair of nodes and the values of various state variables in each node. Figure 5 attempts to clarify the format of the debug file output, it should be read in conjunction with the following example debug file. It should also be noted that layer 2 retransmissions are indicated by an "*" to the right of the NS and layer 3 retransmissions by a "+" (neither occur in this example).

4 An example of use

The following is a simple example of the use of a model on a network. The configuration file, the terminal session, and the output from the model are presented.

An example of a configuration file

This configuration file is called NZPO.

```
NZPO Packet Switching Network
3 nodes
1 Static routing

Node 1
CHCH
```

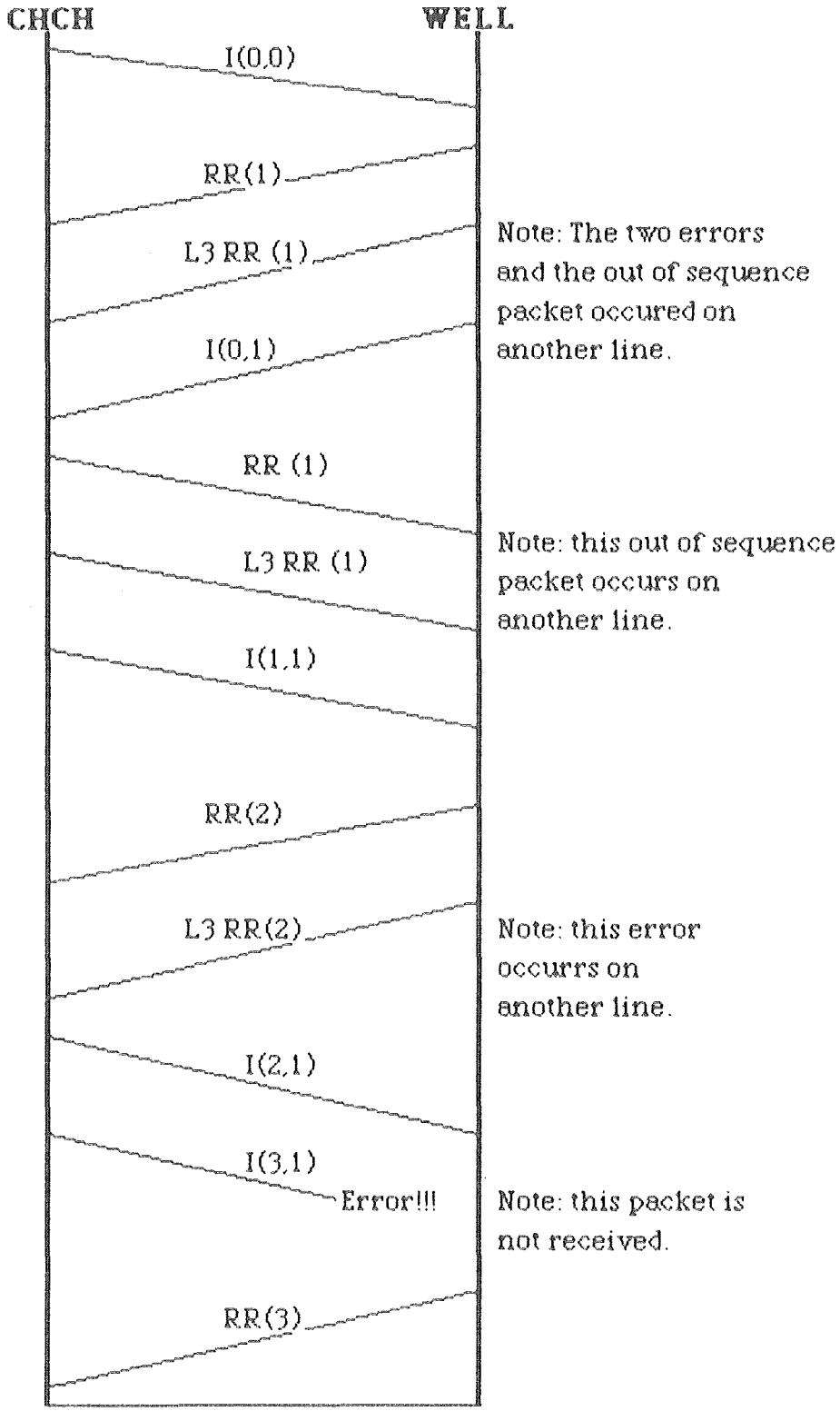


Figure 5 An example of a Debug file.

DEBUG

	CHCH								WELL						
	L2VS	L2VR	Send window	L3VS	L3VR	Send window		NS number NR	L2VS	L2VR	Send window	L3VS	L3VR	Send window	
send	1	0	7	1	0	7	i 1	0 1 2							
							i 1	0 1 2	0	1	7	0	0	7	receive
							I 1	RR 1 2	0	1	7	0	1	7	send
receive	1	0	0	1	0	7	I 1	RR 1 2							
							i 1	L3 RR 1 2	0	1	7	0	1	7	send
receive	1	0	0	1	0	0	I 1	L3 RR 1 2							
								Error!!!							
								Out of Seq Error!!!							
							I 1	0 1 2	1	1	7	1	1	7	send
receive	1	1	0	1	0	0	I 1	0 1 2							
send	1	1	0	1	1	0	I 1	RR 1 2							
							I 1	RR 1 2	1	1	0	1	1	7	receive
send	1	1	0	1	1	0	I 1	L3 RR 1 2							
								Out of Seq							
							I 1	L3 RR 1 2	1	1	0	1	1	0	receive
send	2	1	0	2	1	0	I 1	1 2 2							
							i 1	1 2 2	1	2	0	1	1	0	receive
							I 1	RR 2 2	1	2	0	1	2	0	send
receive	2	1	1	2	1	0	I 1	RR 2 2							
							i 1	L3 RR 2 2	1	2	0	1	2	0	send
receive	2	1	1	2	1	1	I 1	L3 RR 2 2							
								Error!!!							
send	3	1	1	3	1	1	I 1	2 3 2							
							i 1	2 3 2	1	3	0	1	2	0	receive
send	4	1	1	4	1	1	I 1	3 4 2							
								Error!!!							
							I 1	RR 3 2	1	4	0	1	4	0	send
receive	4	1	3	4	1	1	I 1	RR 3 2							

```
50 buffers
2 output lines
line dest, speed, length (km), propagation, error prob.
      2    65536      390           4         0.00005
      3    65536     1065           4         0.00005
source (intermessage, interpacket times; packets/message; backoff)
 1.5 0.2 1.25 0.015
routing table (to, via)
 1 1
 2 2
 3 3

Node 2
WELL
50 buffers
2 output lines
line dest, speed, length (km), propagation, error prob.
      1    65536      390           4         0.00005
      3    65536      675           4         0.00005
source (intermessage, interpacket times; packets/message; backoff)
 1.5 0.2 1.25 0.015
routing table (to, via)
 1 1
 2 2
 3 3

Node 3
AUCK
50 buffers
2 output lines
line dest, speed, length (km), propagation, error prob.
      1    65536     1065           4         0.00005
      2    65536      675           4         0.00005
source (intermessage, interpacket times; packets/message; backoff)
 1.5 0.2 1.25 0.015
routing table (to, via)
 1 1
 2 2
 3 3

routing alg time
0.0001
frame check seq time
0.0
time to process an acknowledgement
0.00001 0.005
Send window size
7 7
Receive window size
7 7
retransmission timer (in bit times)
28000 70000
Acknowledgement timer (in bit times)
4000 4000
```

Figure 6 shows the configuration of this network, all lines are run at 64 kilobits per second.

An example screen dialouge

```
psn
Enter name of configuration file
>nzpo
Enter name of the statistics file
>report
Enter y for dumps                    {enter nothing if dumps
>y                                    not required}
Enter y for debugging
>y
Enter numbers of left and right nodes   {asked only if answered
>1                                       y to debugging question}
>2
Enter 5 random number seeds
>12345                                  {bad random number seeds
>22345                                   because of pattern}
>32345
>42345
>52345
Enter Presimulation and Simulation time (in seconds)
>5                                      {both times are too short}
>60
Running!     5 seconds elapsed
Running!    10 seconds elapsed
Running!    15 seconds elapsed
Running!    20 seconds elapsed
Running!    25 seconds elapsed
Running!    30 seconds elapsed
Running!    35 seconds elapsed
Running!    40 seconds elapsed
Running!    45 seconds elapsed
Running!    50 seconds elapsed
Running!    55 seconds elapsed
Running!    60 seconds elapsed
Simulation complete, writing report.
```

This run of the model will produce the files; report, dump1, dump2, and debug. In most cases only the statistics report will be required.

An example report

This output is in the file report.

```
Report on the
=============
NZPO Packet Switching Network

  Presimulation Time, Simulation Time (seconds)
        5.00                60.00
               Random Number Seeds
```

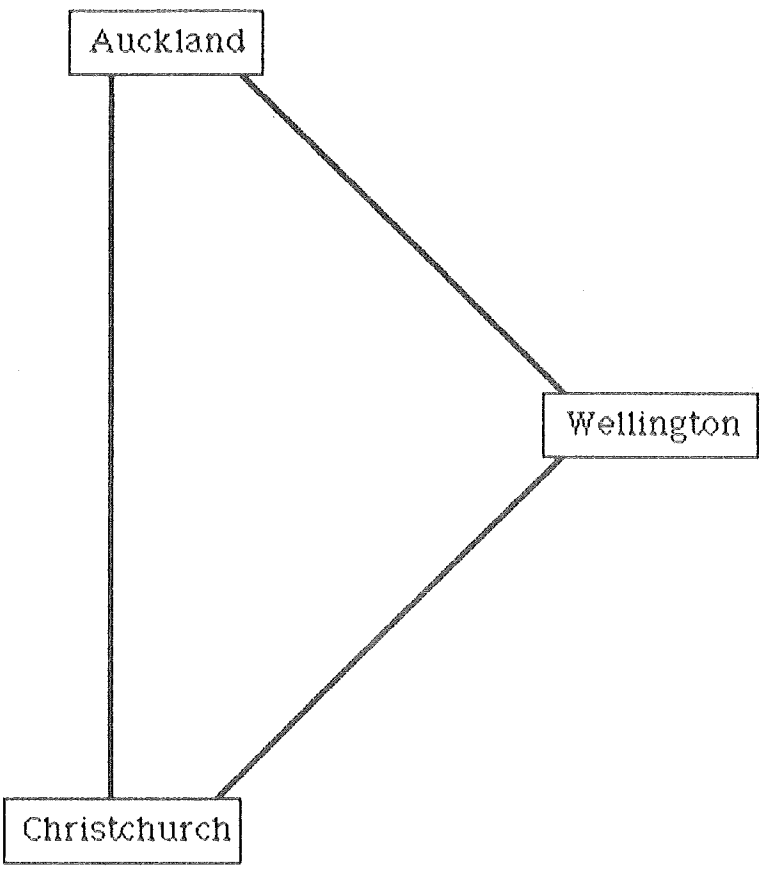


Figure 6 The New Zealand Post Office
Packet Switching Network

12345 22345 32345 42345 52345

Number of packets created

Source	Destination CHCH	WELL	AUCK	Total
CHCH	14	26	11	51
WELL	19	13	11	43
AUCK	13	19	11	43
Total	46	58	33	137

{note variation in the number of packets between each pair of nodes due to the short simulation time}

Packet transfer times (milliseconds)

From	To	Number	Mean	StDev	Minimum	Maximum
CHCH	CHCH	14	0.04854	0.05191	0.00124	0.11501
CHCH	WELL	2	75.17070	162.98880	17.91768	637.61499
CHCH	AUCK	11	21.34470	1.93610	20.61766	27.17564
WELL	CHCH	19	18.02789	0.10054	17.91768	18.13098
WELL	WELL	13	0.05842	0.05247	0.00016	0.11231
WELL	AUCK	11	135.69739	199.49614	19.05920	446.31519
AUCK	CHCH	13	20.76190	0.09664	20.61835	20.83246
AUCK	WELL	19	41.88597	98.95015	19.05740	450.49927
AUCK	AUCK	11	0.06987	0.05143	0.00017	0.11429
Total		137				

Time spent in each node (milliseconds)

Local

Node	Number	Mean	StDev	Minimum	Maximum
CHCH	46	0.06014	0.05062	0.00005	0.11501
WELL	64	0.05397	0.05094	0.00000	0.11519
AUCK	37	0.05854	0.05148	0.00017	0.11513

Nonlocal

Node	Number	Mean	StDev	Minimum	Maximum
CHCH	37	40.39417	138.34636	0.00024	619.69751
WELL	30	42.78459	130.34767	0.00026	427.25757
AUCK	32	13.55060	76.25642	0.00000	431.44189

Number of packets with errors received

From	At	Number
CHCH	CHCH	0
CHCH	WELL	3
CHCH	AUCK	0
WELL	CHCH	0
WELL	WELL	0
WELL	AUCK	1
AUCK	CHCH	0
AUCK	WELL	1
AUCK	AUCK	0
Total		5

Number of Retransmissions

From	Layer 2	Layer 3

CHCH	6	0
WELL	3	0
AUCK	1	0
Total	10	0

Number of acknowledgements

From	To	Layer 2	Layer 3
CHCH	CHCH	0	13
CHCH	WELL	15	15
CHCH	AUCK	12	12
WELL	CHCH	20	20
WELL	WELL	0	12
WELL	AUCK	17	17
AUCK	CHCH	8	8
AUCK	WELL	10	10
AUCK	AUCK	0	8
Total		82	115

Line Utilisation

From	To	Utilisation (%)
CHCH	WELL	1.04101
CHCH	AUCK	0.61235
WELL	CHCH	0.72072
WELL	AUCK	0.63982
AUCK	CHCH	0.58011
AUCK	WELL	0.75008

Wait times for lines (milliseconds)

From	To	Number	Mean	StDev	Maximum
CHCH	WELL	31	20.36418	111.24162	619.69751
CHCH	AUCK	12	0.54976	1.89218	6.55823
WELL	CHCH	19	0.00642	0.00528	0.01424
WELL	AUCK	14	0.00682	0.00455	0.01440
AUCK	CHCH	13	0.00654	0.00492	0.01331
AUCK	WELL	20	0.21535	0.93697	4.19605

Buffer Utilisation

Node	Utilisation (%)	Minimum	Maximum	Available
CHCH	2.31098	0.00000	4.00000	50
WELL	2.17135	0.00000	4.00000	50
AUCK	2.16057	0.00000	4.00000	50

Packets rejected due to lack of buffers

Node	Local	Nonlocal
CHCH	0	0
WELL	0	0
AUCK	0	0

The following are some of the observations that can be made from the above report. The very low line and buffer utilisations

together with the high number of acknowledgements indicate that there are large gaps between packets and therefore the network as configured is capable of tramsmitting a lot more traffic. If the traffic offered to the network is as described the network could most probably could be run using 9600 bps lines. This could be confirmed by another run of the model. The large number of retransmissions relative to the number of errors indicates that the retransmission timer may be too short. The high number of acknowledgements may be reduced by increasing the acknowledge timer, this must however remain less than the retransmission timer. Further runs of the model could also be made with higher and higher workloads to give an indication of how the network will react to increased load.

An example of a dump file: dump1

These files are produced only if you responded "y" to the prompt "Enter y for dumps".

```
Packets received by Node   1. From Nodes:
   1   2   3
-------------
               1
               2
               3
               4
               5
               6
   1
       1
               7
   2
               8
               9
       1           {note end of presimulation period}
               1
       2
       3
       4
   1
   2
   3
       5
               2

{and so on}
```

An example of the debug file

This file is produced only if you responded "y" to the "Enter y for debugging", it is 130 characters wide. See the following page for an example and section 3.4 for an explanation of this type of output.

	L2VS	L2VR	Send window	L3VS	CHCH L3VR
send	1	0	7	1	0
receive	1	0	0	1	0
receive	1	0	0	1	0
receive	1	1	0	1	0
send	1	1	0	1	1
send	1	1	0	1	1
send	2	1	0	2	1
send	2	1	2	2	1
receive	2	1	1	2	1
receive	2	1	1	2	1
send	3	1	1	3	1
send	4	1	1	4	1
receive	4	1	3	4	1
receive	4	1	3	4	1
send	5	1	3	5	1
receive	5	1	4	5	1
receive	5	1	4	5	1
send	6	1	4	6	1
send	6	1	6	6	1
receive	6	1	5	7	1
receive	6	1	5	7	1
receive	6	2	5	7	1
send	6	2	5	7	2
send	6	2	5	7	2

(and so on)

5 The Source emulator

The source emulator is designed to assist in the selection of the three main parameters of a source: intermessage time, interpacket time, and the average number of packets in each message. Given the three parameters and a simulation time it produces a histogram of the number of packets generated against time and the average number of packets generated per second and per hour during the simulation period. The fourth parameter for a source in the configuration file is the time that the source waits before attempting to resubmit a packet that has been rejected by a node because there was no free buffer for it. This can not be simulated by the source emulator as there is no way of simulating buffer congestion except by running the main model which does not collect these detailed statistics on packet creation. This timer is not used unless packets are rejected due to a lack of buffers.

An example run of source emulator:

Input:

```
enter name of outputfile (nothing for screen)
>
enter width of histogram classes
>5
enter 3 random number seeds
>12345
>22345
>32345
enter intermessage and interpacket times
>1.5
>0.2
enter average number of packets per message
>1.25
enter simulation time
>30
```

Output:

```
Random number seeds
   12345   22345   32345
Intermessage and Interpacket time
   1.50000   0.20000
Average number of packets per message
  1.25
Simulation time
 30.00
```

```
 Time
      |****                                4
 5.00 |
      |******                              6
10.00 |
      |*********                           9
15.00 |
      |*******                             7
20.00 |
      |******                              6
25.00 |
      |*****                               5
30.00 |
Average number of packets generated per second  1.23
Average number of packets generated per hour  4440.00
```

The program has three procedures Initialise, Report and Plot, and one Process Class Source. The procedure **Initialise** asks the user for an output file name, the width of the histogram classes, 3 random number seeds, the three source parameters, and the simulation time. If there are more than 60 classes in the simulation time it is shortened and the user is informed. The prodecure **Report** as its name suggests writes the report for the program, it calls the procedure **Plot** to plot the histogram. The process class **Source** is the same as that in the model except that instead of introducing packets to the network it calls the standard procedure histo to update the histogram.

6 Conclusion

This model appears to succesfully model the behaviour of a packet switching network with similar internal protocols to that described in section 2.1 of this report. The limited amount of testing carried out supports this assertion in comparison with the New Zealand Post Office Packet Switching Network. It should be noted that the Post Office network implements piggybacking. The model however requires validation, one possible means of doing this would be to compare the results obtain from the model with those of an actual network. There is extensive room for further work on this project particularly in providing a better workload description and source of packets. Further work could also be undertaken in allowing the user to specify some aspects of the internal protocol to enable the model to be applied to a wider range of networks. The inclusion of further routing algorithms is another area with large scope. Finally the actual use of the model to predict the performance of an actual network or to assist in the design of a real network should be the ultimate aim.

7 Acknowledgements

Acknowledgement is made of the assistance rendered by the New Zealand Post Office on some of the more technical questions that where raised during the course of this project. The assistance of Wolfgang Kreutzer in developing this Simula model,

and of Ray Hunt and Bruce McAuley in various discussions on the communication issues raised in this project, and the resources of the Department of Computer Science are also acknowledged.

8 References

1. CCITT Recommendation X.25, Fascicle VIII.2 Data Communication Networks - Yellow Book, Geneva, July 1981.

2. CCITT Recommendiation X.200 COMM VII R22(C)-E, Geneva, 1981.

3. "Network Protocols", Andrew S. Tannenbaum, Computing Surveys, Vol. 13, No. 4, December 1981.

4. Proceedings of the IEEE, Vol. 71, No. 12, December 1983.

5. Sloman M. S., "X.25 explained", Computer Communication, Vol. 1, No. 6, December 1978.

6. Pouzin L., "Methods, Tools, and Observations on Flow Control in Packet-Switched Data Networks", IEEE Transactions on Communication, Vol. COM-29, No. 4, April 1981.

7. Boorstyn R. R. and Livne A., "A Technique for Adaptive Routing in Networks", IEEE Transactions on Communication, Vol. COM-29, No. 4, April 1981.

8. Chou W., Bragg A. W., and Nilsson A. A., "The Need for Adaptive Routing in the Chaotic and Unbalanced Traffic Environment", IEEE Transactions on Communication, Vol. COM-29, No. 4, April 1981.

9. Alton B., Patel A., Purser M., and Sheehan J., "The Performance of a Packet Switched Network - A Study of EURONET", Proceedings of the International Conference on Performance of Data Communication Systems and their Applications, Paris, September 1981.

10. Tannenbaum A. S., "Computer Networks", Prentice Hall, 1981.

Appendix 1 Code of PSN, the model itself

```
BEGIN
SIMULATION BEGIN
comment ************************************************
        *                                              *
        *    Packet Switched Network Model             *
        *    =============================             *
        *                                              *
        *       Written by  Paul Reynolds,             *
        *                                              *
        *          Date  20/9/1984.                    *
        *                                              *
        ************************************************
 This model is designed to give information on packet flow
 in X.25 Public Data Networks. It reads a description of the
 network from the given configuration file and writes the
 results to the given statistics file;


    REF (NodeClass) array DSE (1:20);
    REF (ServerClass) channel;
    REF (StatsCollecter) monitor;
    REF (RunningMessage) running;
    REF (INFILE) inf;
    REF (OUTFILE) outf, debugfile;
    REF (OUTFILE) array dump(1:20);
    TEXT networkname, dumpfile;
    real array acknowledgetimer(1:2);
    real routingalgtime, FCStime,
         presimtime, simtime;
    integer array seed(1:5), initseed(1:5),
                  L2VS(1:20,1:20), L2VR(1:20,1:20),
                  L3VS(1:20,1:20), L3VR(1:20,1:20),
                  L2sendwindow(1:20,1:20),
                  L3sendwindow(1:20,1:20),
                  L2receivewindow(1:20,1:20),
                  L3receivewindow(1:20,1:20);
    integer minnode, maxnode, i, routingmethod,
            L2timerdelay, L3timerdelay,
            L2sendwindowsize, L2receivewindowsize,
            L3sendwindowsize, L3receivewindowsize,
            L2windowtimer, L3windowtimer,
            maxsequencenum,
            leftnode, rightnode;
    boolean dumpon, debugon;

comment ************************************************;
LINK CLASS Packet;
comment This class is passed around the network acting as
        the packets in the network;
Begin
    boolean L2acknowledgement, L3acknowledgement,
            error, retransmission, layer3, callrequest,
            network;
    integer source, destination, size, NS, NR, PS, PR;
    real transtime, waitingtime;
    real array entrytime(1:maxnode),
                totalinternaltime(1:maxnode, 1:2);
    integer number;
End of LINK CLASS packet;
```

```
comment ************************************************;
CLASS ServerClass;
comment This is a collection  of procedures relating to the
        operation of the lines connecting the nodes of the
        network;
Begin
    REF (HEAD) array waiting(1:maxnode, 1:maxnode);
    boolean array channelfree(1:maxnode, 1:maxnode);
    integer i, j;

    comment ------------------------------------------------;
    Procedure Acquire(source, destination);
        integer source, destination;
    comment This procedure ensures mutual exclusion on each of
            the channels between the nodes. The queue 'waiting'
            contains those processes waiting for the channel to
            become free;
    Begin
    CURRENT.INTO (waiting(source, destination));
    While not channelfree(source, destination) do
        PASSIVATE;
    monitor.utilstart(source, destination);
    channelfree(source, destination) := false;
    CURRENT.OUT;
    End of procedure Acquire;

    comment ------------------------------------------------;
    Procedure Release (source, destination);
        integer source, destination;
    comment This procedure releases the channel and wakes up
            the first process waiting for tnat channel;
    Begin
        REF (PROCESS) transmitter;
    channelfree(source, destination) := true;
    monitor.utilend(source, destination);
    if not waiting(source, destination).EMPTY then
        begin
        transmitter :- waiting(source, destination).FIRST;
        transmitter.OUT;
        ACTIVATE transmitter;
        end;
    End of procedure Release;

comment ------------------------------------------------;
comment CLASS ServerClass initialisation;

for i := minnode step 1 until maxnode do
    for j := minnode step 1 until maxnode do
        begin
        waiting(i, j) :- NEW HEAD;
        channelfree(i, j) := true;
        end;
End of CLASS ServerClass;
```

```
comment ************************************************;
CLASS NodeClass (node);
    integer node;

comment This class is a collection of procedures dealing with
        the processes that take place within each node on the
        network.;
Begin
    REF (OutputProcess) array outputline(1:maxnode);
    REF (Source) DTEsource;
    REF (Submitter) Submit;
    REF (Sink) DTEsink;
    REF (TimerProcess) array L2timer(1:maxnode),
                             L3timer(1:maxnode);
    REF (HEAD) array outputQ(1:maxnode),
                     priorityoutputQ(1:maxnode);
    REF (HEAD) inputQ, DTEQ, L2unacknowledged,
               L3unacknowledged,
               sourceQ, L3waiting;
    TEXT nodename;
    integer array routingtable(1:maxnode,1:5),
                  linechoice(1:maxnode);
    integer buffersinuse, totalbuffers;

    comment ----------------------------------------------------;
    Procedure L2Receive (message, from);
        REF (packet) message;
        integer from;
    comment This procedure is called by the transmit procedure
            of another node. This procedure adds the incomming
            message to the input queue of this node if there
            is a spare buffer and there were no transmission
            errors. The procedure Route is called to snift
            the message onto the correct outgoing queue. An
            L2acknowledgement is created and sent either by
            the timer process, if the gap between the packets
            is large or when the receive window size number
            of 'I packets' have been received from that node;
    Begin
        REF (packet) ack, temp;

    if buffersinuse < totalbuffers then
        begin
        comment room for packet;
        buffersinuse := buffersinuse + 1;
        monitor.bufferutilisation(node, buffersinuse);
        if not (message.L2acknowledgement or
                message.L3acknowledgement) then
            monitor.entry(message, node);
        HOLD (FCStime);
        if message.error or (message.NS () L2VR(from, node) and
                    not (message.L2acknowledgement or
                            message.L3acknowledgement)) then
            begin
            comment packets with errors or out of sequence are
                    simply discarded;
            buffersinuse := buffersinuse - 1;
            monitor.bufferutilisation(node, buffersinuse);
            if not (message.L2acknowledgement or
```

```
            message.L3acknowledgement) then
        begin
        monitor.departure(message, node);
        if message.error then
           monitor.error(from, node);
        end;
    if debugon then
        begin
        debugfile.OUTTEXT(blanks(45));
        if message.error then
           debugfile.OUTTEXT("Error!!!")
        else
           debugfile.OUTTEXT("Out of Seq");
        debugfile.OUTIMAGE;
        end;
    end
else

    begin
    comment process correctly received packet;
    if not message.L2acknowledgement then
        begin
        if not message.L3acknowledgement then
           begin
           comment process I frame;
           if not message.layer3 then
              message.retransmission := false;
           REACTIVATE L2timer(from) DELAY (L2windowtimer /
                                    outputline(1).linespeed);

           comment increment receive state variable;
           L2VR(from, node) := L2VR(from, node) + 1;
           if L2VR(from, node) > maxsequencenum then
              L2VR(from, node) := 0;
           if L2VR(from, node) =
                          L2receivewindow(from, node) then
              begin
              comment create acknowledement (RR frame) if
                       receive window reached;
              ack :- NEW packet;
              INSPECT ack do
                 begin
                 L2acknowledgement := true;
                 L3acknowledgement := false;
                 error := false;
                 retransmission := false;
                 callrequest := false;
                 network := false;
                 source := node;
                 destination := from;
                 NR := L2VR(from, node);
                 size := 48;
                 end of INSPECT ack;
              comment insert acknowledgement before I
                       packets in input@;
              temp :- input@.FIRST;
              while (temp =/= NONE) andthen
                    (temp.L2acknowledgement or
                     temp.L3acknowledgement) do
```

```
                temp := temp.SUC;
            if temp =/= NONE then
                ack.PRECEDE(temp)
            else
                ack.INTO(input@);
            ack := NONE;
            comment advance receive window;
            L2receivewindow(from, node) :=
                                L2receivewindow(from,node) +
                                    L2receivewindowsize;
            if L2receivewindow(from, node) >
                                    maxsequencenum then
                L2receivewindow(from, node) :=
                                L2receivewindow(from, node) -
                                    maxsequencenum - 1;
            end of L2acknowledgement creation;
        end of processing 'I packet'

    else
        begin
        if node = message.destination then
            begin
            comment note arrival of layer 3 acknowledgement;
            acknowledged(message, from, 3);
            if debugon then
                debug("  receive", node, from, message);
            message := NONE;
            end
        else
            begin
            comment forward L3 acknowledgement;
            temp := input@.FIRST;
            while (temp =/= NONE) andthen
                    (temp.L2acknowledgement or
                     temp.L3acknowledgement) do
                temp := temp.SUC;
            if temp =/= NONE then
                message.PRECEDE(temp)
            else
                message.INTO(input@);
            if debugon then
                debug("  receive", node, from, message);
            message := NONE;
            Route;
            end;
        buffersinuse := buffersinuse - 1;
        monitor.bufferutilisation(node, buffersinuse);
        end;
    end

else
    begin
    comment note arrival of layer 2 acknowledgement;
    acknowledged (message, from, 2);
    buffersinuse := buffersinuse - 1;
    monitor.bufferutilisation(node, buffersinuse);
    if debugon then
        debug("  receive", node, from, message);
    message := NONE;
```

```
            end;

        if message =/= NONE then
            begin
            if debugon then
                debug("  receive", node, from, message);
            message.INTO(inputQ);
            message :- NONE;
            Route;
            end;
        end of processing correctly received packet;
    end
else
    begin
    comment note lack of buffer for incomming packet;
    monitor.bufferlack(node, from);
    end;
End of Procedure L2Receive;

comment ------------------------------------------------------------------;
Procedure L3Receive(message);
    REF (packet) message;
comment This procedure introduces packets to their
        source node;
Begin
buffersinuse := buffersinuse + 1;
monitor.bufferutilisation(node, buffersinuse);
monitor.entry(message, node);
message.INTO(L3waiting);
message :- NONE;
emptyL3waiting;
End of Procedure L3Receive;

comment ------------------------------------------------------------------;
Procedure EmptyL3waiting;
comment This procedure takes packets from their source,
        sets up layer 3 retransmission then sends the packet
        on its way. This procedure maintains the layer 3 VS;
Begin
    REF (packet) temp, message;
    REF (retransmitter) retransmission;
    integer i;

message :- L3waiting.FIRST;
While (message =/= NONE) do
    begin
    temp :- NONE;
    if L3VS(node, message.destination) <>
            L3sendwindow(node, message.destination) then
        begin
        message.PS := L3VS(node, message.destination);
        L3VS(node, message.destination) :=
                            L3VS(node, message.destination) + 1;
        if L3VS(node, message.destination) >
                                                maxsequencenum then
            L3VS(node, message.destination) := 0;

        comment set up layer 3 retransmission;
        retransmission :- NEW retransmitter(node);
```

```
        retransmission.message := NEW packet;
        retransmission.message.L2acknowledgement  := false;
        retransmission.message.L3acknowledgement  := false;
        retransmission.message.error := false;
        retransmission.message.retransmission := true;
        retransmission.message.layer3 := true;
        retransmission.message.source := message.source;
        retransmission.message.destination :=
                                          message.destination;
        retransmission.message.size := message.size;
        retransmission.message.PS := message.PS;
        retransmission.message.transtime :=
                                           message.transtime;
        for i := 1 step 1 until maxnode do
            begin
            retransmission.message.entrytime(i) :=
                                          message.entrytime(i);
            retransmission.message.totalinternaltime(i,1) :=
                                  message.totalinternaltime(i,1);
            retransmission.message.totalinternaltime(i,2) :=
                                  message.totalinternaltime(i,2);
            end;
        retransmission.message.number := message.number;
        retransmission.timer := L3timerdelay /
                                       outputline(1).linespeed;
        retransmission.needed := true;
        retransmission.layer3 := true;
        retransmission.sentto := message.destination;
        retransmission.INTO(L3unacknowledged);
        ACTIVATE retransmission;
        retransmission :- NONE;

        temp :- message;
        end;
    message :- message.SUC;
    if temp =/= NONE then
        begin
        temp.INTO(input@);
        temp :- NONE;
        Route;
        end;
    end of L3waiting;
End of EmptyL3waiting;

comment ------------------------------------------------------------;
Procedure Acknowledged (message, from, layer);
    REF (packet) message;
    integer from, layer;
comment This procedure searches the queue unacknowledged
        for a retransmission Process waiting to retransmit
        any of the packets for which an acknowledgement has
        been received. By setting the boolean needed to
        false this procedure stops the retransmission;
Begin
    REF (retransmitter) retransmission, temp;
    integer count;
    boolean found, advance;

if message.destination = node then
```

```
begin
if layer = 2 then
   HOLD(acknowledgetimer(1))
else
   HOLD(acknowledgetimer(2));
comment search for pending retransmission;
count := 0;
found := false;
advance := false;

if layer = 2 then
   begin
   comment layer 2 acknowledgement;
   retransmission := DSE(node).L2unacknowledged.FIRST;
   While (retransmission =/= NONE) and
         not found do
      begin
      temp := NONE;
      if (retransmission.sentto = message.source) then
         begin
         if retransmission.message.NS <> message.NR then
            begin
            if retransmission.needed then
               begin
               comment layer 3 acknowledgement frees
                        buffers at source;
               if retransmission.message.source <>
                                                  node then
                  count := count + 1;
               retransmission.needed := false;
               temp := retransmission;
               advance := true;
               end;
            end
         else
            found := true;
         end;
      retransmission := retransmission.SUC;
      if temp =/= NONE then
         temp.OUT;
      end of unacknowledged 0;
   end;

if layer = 3 then
   begin
   comment layer 3 acknowledgement;
   retransmission := DSE(node).L3unacknowledged.FIRST;
   While (retransmission =/= NONE) and
         not found do
      begin
      temp := NONE;
      if retransmission.sentto = message.source then
         begin
         comment layer 3 acknowledgement;
         if retransmission.message.PS <> message.PR then
            begin
            if retransmission.needed and
                  retransmission.layer3 then
               begin
```

```
                    comment layer 3 acknowledgement frees
                            buffer at source;
                    count := count + 1;
                    retransmission.needed := false;
                    temp := retransmission;
                    advance := true;
                    end
                else
                    temp := NONE;
                end
            else
                found := true;
            end;
        retransmission := retransmission.SUC;
        if temp =/= NONE then
            temp.OUT;
        end of unacknowledged Q;
    end;

if advance then
    begin
    comment advance send window and release the
            unneeded buffers;
    if layer = 2 then
        begin
        L2sendwindow(node, from) := message.NR +
                                        L2sendwindowsize;
        if L2sendwindow(node, from) > maxsequencenum then
            L2sendwindow(node, from) :=
                                L2sendwindow(node, from) -
                                    maxsequencenum - 1;
        end
    else
        begin
        comment layer 3 acknowledgement;
        L3sendwindow(node, message.source) :=
                                message.PR + L3sendwindowsize;
        if L3sendwindow(node, message.source) >
                                    maxsequencenum then
            L3sendwindow(node, message.source) :=
                        L3sendwindow(node, message.source) -
                                    maxsequencenum - 1;
        emptyL3waiting;
        end;
    buffersinuse := buffersinuse - count;
    monitor.bufferutilisation(node, buffersinuse);
    end;
monitor.acknowledgements(message.source,
                            message.destination,
                            layer);
end;
End of Procedure Acknowledged;

comment ------------------------------------------------------------;
Procedure Route;
comment This process determines which output line the
        incomming packet should be transmitted on by
        looking up the routing table;
Begin
```

```
REF (packet) message;
integer outputlinenum;

comment - - - - - - - - - - - - - - - - - - - - - - - - ;
Integer Procedure FindLine(message);
    REF (packet) message;
comment this procedure returns the line number to be
          used (note that L2acknowledgements are not
          rerouted);
Begin
    real prob, minQ;
    integer immeddest, i, posline;

if message.L2acknowledgement then
    immeddest := message.destination
else
    begin
    comment select immediate destination;
    if routingmethod = 1 then
        begin
        comment Static Routing;
        immeddest := routingtable(message.destination, 1);
        end
    else
        begin
        if routingmethod = 2 then
            begin
            comment Biased shortest queue routing (ignoring
                    line speed);
            minQ := 9999;
            for i := 1 step 1 until 5 do
                begin
                if routingtable(message.destination,i) <> 0 then
                    begin
                    posline := linechoice(routingtable(
                                          message.destination,i));
                    if (outputQ(posline).CARDINAL +
                        priorityoutputQ(posline).CARDINAL) <
                            minQ then
                        begin
                        minQ := outputQ(posline).CARDINAL;
                        immeddest :=
                            routingtable(message.destination,i);
                        end;
                    end;
                end;
            end of Biased Shortest Q (ignoring line speed)
        else
            begin
            if routingmethod = 3 then
                begin
                comment Biased shortest queue routing;
                minQ := 9999;
                for i := 1 step 1 until 5 do
                    begin
                    if routingtable(message.destination,i)
                                                        <> 0 then
                        begin
                        posline := linechoice(routingtable(
```

```
                                        message.destination,i));
                                if (outputQ(posline).CARDINAL +
                                   priorityoutputQ(posline).CARDINAL) /
                                      outputline(posline).linespeed (
                                        minQ then
                                   begin
                                   minQ := outputQ(posline).CARDINAL;
                                   immeddest :=
                                     routingtable(message.destination,i);
                                   end;
                                end;
                             end;
                          end of Biased Shortest Q;
                       end;
                    end;
                 end;

    FindLine := linechoice(immeddest);
    End of Integer Procedure FindLine;

comment - - - - - - - Body of Route - - - - - - - - - - - - - - - - - - ;

While not inputQ.EMPTY do
    begin
    message :- inputQ.FIRST;
    message.OUT;
    if message.callrequest then
       HOLD (routingalgtime);
    if message.destination = node then
       begin
       comment arrived at correct DSE;
       message.INTO(DTEQ);
       message :- NONE;
       ACTIVATE DTEsink;
       end
    else
       begin
       comment forward to next DSE on route;
       outputlinenum := FindLine(message);
       comment acknowledgements and retransmissions get
                higher priority than I frames;
       if message.L2acknowledgement or
             message.L3acknowledgement or
             message.retransmission then
          message.INTO(priorityoutputQ(outputlinenum))
       else
          message.INTO (outputQ(outputlinenum));
       comment note time that message enters output queue;
       message.waitingtime := TIME;
       message :- NONE;
       ACTIVATE outputline (outputlinenum);
       end;
    end of inputQ;
End of Procedure Route;

comment ------------------------------------------------------------;
Procedure Transmit (message, line, fromnode);
    REF (packet) message;
    integer line, fromnode;
```

```
comment This procedure carries out the actual mechanics of
        the transmission. It sets up the layer 2
        retransmission, acquires the line, holds the line
        for the required ransmission time,releases the line
        and then decides if there has been a transmission
        error. Finally the receive procedure of the
        destination node is called to add the message to
        its input queue.;
Begin
    REF (Retransmitter) Retransmission;
    REF (packet) ack;
    real linedelay, waitingtime;
    integer i;

if not (message.L2acknowledgement or
        message.L3acknowledgement or
        message.retransmission) then
    begin
    comment retransmit only 'I packets';
    Retransmission :- NEW Retransmitter (fromnode);
    retransmission.message :- NEW packet;
    retransmission.message.L2acknowledgement := false;
    retransmission.message.L3acknowledgement := false;
    retransmission.message.error := false;
    retransmission.message.retransmission := true;
    retransmission.message.layer3 := false;
    retransmission.message.source := message.source;
    retransmission.message.destination := message.destination;
    retransmission.message.size := message.size;
    retransmission.message.NS := message.NS;
    retransmission.message.PS := message.PS;
    retransmission.message.transtime := message.transtime;
    for i := 1 step 1 until maxnode do
        begin
        retransmission.message.entrytime(i) :=
                                        message.entrytime(i);
        retransmission.message.totalinternaltime(i,1) :=
                                    message.totalinternaltime(i,1);
        retransmission.message.totalinternaltime(i,2) :=
                                    message.totalinternaltime(i,2);
      end;
    retransmission.message.number := message.number;
    retransmission.timer := L2timerdelay /
                                        outputline(line).linespeed;
    retransmission.needed := true;
    retransmission.layer3 := false;
    retransmission.sentto := outputline(line).linkedto;
    retransmission.INTO(DSE(fromnode).L2unacknowledged);
    ACTIVATE retransmission;
    retransmission :- NONE;
    end;

linedelay := message.size / outputline(line).linespeed;

channel.acquire (fromnode, outputline(line).linkedto);

if UNIFORM (0.0, 1.0, seed(1)) <
      (outputline(line).errorprob * message.size) then
    message.error := true
```

```
else
    begin
    message.error := false;
    if not (message.L2acknowledgement or
            message.L3acknowledgement) then
       monitor.departure(message, fromnode);
    end;

if not (message.L2acknowledgement or
        message.L3acknowledgement) then
    begin
    waitingtime := TIME - message.waitingtime;
    monitor.waiting(fromnode, outputline(line).linkedto,
                            waitingtime);
    end;

if debugon then
    debug("      send", fromnode,
                        outputline(line).linkedto, message);

HOLD (linedelay + outputline(line).propagationtime);

channel.release (fromnode, outputline(line).linkedto);

DSE(outputline(line).linkedto).L2receive (message, fromnode);
End of Procedure Transmit;

comment ------------------------------------------------------------;
Procedure Debug(sendreceive, node, from, message);
    value sendreceive; TEXT sendreceive;
    integer node, from;
    REF(packet) message;
comment This procedure prints the debugging output to the
        debugfile.;
Begin
if (node = rightnode) and (from = leftnode) then
    begin
    comment right to/from left;
    debugfile.OUTTEXT(blanks(39));
    debugfile.OUTTEXT("|");
    if sendreceive = "  receive" then
       debugfile.OUTINT(message.source, 2)
    else
       debugfile.OUTINT(message.destination, 2);
    if message.L2acknowledgement then
       begin
       debugfile.OUTTEXT("         RR");
       debugfile.OUTINT(message.NR, 5);
       end
    else
       begin
       if message.L3acknowledgement then
          begin
          debugfile.OUTTEXT("      L3 RR");
          debugfile.OUTINT(message.PR, 5);
          end
       else
          begin
          debugfile.OUTINT(message.NS, 5);
```

```
            debugfile.OUTINT(message.number, 5);
            if message.retransmission then
               begin
               if message.layer3 then
                  debugfile.OUTTEXT("   +   ")
               else
                  debugfile.OUTTEXT("   *   ")
               end
            else
               debugfile.OUTTEXT(blanks(5));
            end;
         end;
      if sendreceive = "   receive" then
         debugfile.OUTINT(message.destination, 2)
      else
         debugfile.OUTINT(message.source, 2);
      debugfile.OUTTEXT("|");
      debugfile.OUTINT(L2VS(rightnode, leftnode), 5);
      debugfile.OUTINT(L2VR(leftnode, rightnode), 5);
      debugfile.OUTINT(L2sendwindow(rightnode, leftnode), 5);
      if sendreceive = "   receive" then
         begin
         debugfile.OUTINT(L3VS(message.destination,
                                          message.source), 5);
         debugfile.OUTINT(L3VR(message.source,
                                       message.destination), 5);
         debugfile.OUTINT(L3sendwindow(message.destination,
                                          message.source), 5);
         end
      else
         begin
         debugfile.OUTINT(L3VS(message.source,
                                       message.destination), 5);
         debugfile.OUTINT(L3VR(message.destination,
                                          message.source), 5);
         debugfile.OUTINT(L3sendwindow(message.source,
                                       message.destination), 5);
         end;
      debugfile.OUTTEXT(sendreceive);
      debugfile.OUTIMAGE;
      end of right to/from left;

   if (node = leftnode) and (from = rightnode) then
      begin
      comment left to/from right;
      debugfile.OUTTEXT(sendreceive);
      debugfile.OUTINT(L2VS(leftnode, rightnode), 5);
      debugfile.OUTINT(L2VR(rightnode, leftnode), 5);
      debugfile.OUTINT(L2sendwindow(leftnode, rightnode), 5);
      if sendreceive = "   receive" then
         begin
         debugfile.OUTINT(L3VS(message.destination,
                                          message.source), 5);
         debugfile.OUTINT(L3VR(message.source,
                                       message.destination), 5);
         debugfile.OUTINT(L3sendwindow(message.destination,
                                          message.source), 5);
         end
      else
```

```
            begin
            debugfile.OUTINT(L3VS(message.source,
                                        message.destination), 5);
            debugfile.OUTINT(L3VR(message.destination,
                                        message.source), 5);
            debugfile.OUTINT(L3sendwindow(message.source,
                                        message.destination), 5);
            end;
        debugfile.OUTTEXT("|");
        if sendreceive = "   receive" then
            debugfile.OUTINT(message.destination, 2)
        else
            debugfile.OUTINT(message.source, 2);
        if message.L2acknowledgement then
            begin
            debugfile.OUTTEXT("          RR");
            debugfile.OUTINT(message.NR, 5);
            end
        else
            begin
            if message.L3acknowledgement then
                begin
                debugfile.OUTTEXT("      L3 RR");
                debugfile.OUTINT(message.PR, 5);
                end
            else
                begin
                debugfile.OUTINT(message.NS, 5);
                debugfile.OUTINT(message.number, 5);
                if message.retransmission then
                    begin
                    if message.layer3 then
                        debugfile.OUTTEXT("   +   ")
                    else
                        debugfile.OUTTEXT("   *   ")
                    end
                else
                    debugfile.OUTTEXT(blanks(5));
                end;
            end;
        if sendreceive = "   receive" then
            debugfile.OUTINT(message.source, 2)
        else
            debugfile.OUTINT(message.destination, 2);
        debugfile.OUTTEXT("|");
        debugfile.OUTIMAGE;
        end of left to/from right;
    End of Procedure Debug;

comment ----------------------------------------------------------------------;

End of CLASS NodeClass;

comment **********************************************************;
Process Class TimerProcess(node, from, layer);
    integer node, from, layer;
comment This Process sends an acknowledgement if the window
        timer expires (ie. the gap between incomming packets
        is too large);
```

```
Begin
    REF (packet) ack, temp;
While true do
    begin
    PASSIVATE;

    ack :- NEW packet;
    INSPECT ack do
        begin
        if layer = 2 then
            begin
            L2acknowledgement := true;
            L3acknowledgement := false;
            NR := L2VR(from, node);
            end
        else
            begin
            L3acknowledgement := true;
            L2acknowledgement := false;
            PR := L3VR(from, node);
            end
        error := false;
        retransmission := false;
        callrequest := false;
        network := false;
        source := node;
        destination := from;
        size := 48;
        end of INSPECT ack;

    if layer = 2 then
        begin
        L2receivewindow(from, node) :=
                                    L2receivewindow(from, node) +
                                                L2receivewindowsize;
        if L2receivewindow(from, node) > maxsequencenum then
            L2receivewindow(from, node) :=
                                    L2receivewindow(from, node) -
                                                maxsequencenum - 1;
        end
    else
        begin
        L3receivewindow(from, node) :=
                                    L3receivewindow(from, node) +
                                                L3receivewindowsize;
        if L3receivewindow(from, node) > maxsequencenum then
            L3receivewindow(from, node) :=
                                    L3receivewindow(from, node) -
                                                maxsequencenum - 1;
        end
    temp :- DSE(node).input0.FIRST;
    if temp =/= NONE then
        ack.PRECEDE(temp)
    else
        ack.INTO(DSE(node).input0);
    ack :- NONE;
    temp :- NONE;
    DSE(node).Route;
    end of while true;
```

```
End of Process Class TimerProcess;

comment ************************************************************;
Process Class OutputProcess (node, linenum);
    integer node, linenum;
comment This process takes a packet from the output queue or
        the priority output queue (if not empty) for its
        line and calls the procedure transmit to transmit it
        to the node that this line is connected to.;
Begin
    REF (packet) message;
    real linespeed, propagationtime,
         errorprob;
    integer linkedto;

    comment ------------------------------------------------------------;
    Procedure PriorityPackets;
    comment layer 2 and 3 acknowledgements and retransmissions
            get priority over I frames;
    Begin
        REF (packet) message, temp;

    message :- DSE(node).priorityoutputQ(linenum).FIRST;
    While message =/= NONE do
        begin
        temp :- NONE;
        if message.L2acknowledgement or
              message.L3acknowledgement then
            begin
            temp :- message;
            DSE(node).transmit(temp, linenum, node);
            end
        else
            begin
            if message.retransmission then
                begin
                comment stop non-retransmissions until an
                        acknowledgement;
                if message.layer3 then
                    begin
                    if L2VS(node, linkedto) <>
                                    L2sendwindow(node, linkedto) then
                        begin
                        L3sendwindow(message.source,
                                        message.destination) :=
                                              L3VS(message.source,
                                                   message.destination);
                        message.NS := L2VS(node, linkedto);
                        L2VS(node, linkedto) :=
                                               L2VS(node, linkedto) + 1;
                        if L2VS(node, linkedto) > maxsequencenum then
                            L2VS(node, linkedto) := 0;
                        temp :- message;
                        DSE(node).transmit(temp, linenum, node);
                        end;
                    end
                else
                    begin
                    L2sendwindow(node, linkedto) :=
```

```
                                                    L2VS(node, linkedto);
                    temp :- message;
                    DSE(node).transmit(temp, linenum, node);
                    end;
                end;
            end;
        message :- message.SUC;
        if temp =/= NONE then
            temp.OUT;
        end of priorityoutputQ;
    End of Procedure PriorityPackets;

comment ----------Body of OutputProcess----------;
While true do
    begin
    PASSIVATE;

    if not DSE(node).priorityoutputQ(linenum).EMPTY then
        prioritypackets;

    while not DSE(node).outputQ(linenum).EMPTY and
          (L2VS(node, linkedto) <>
                          L2sendwindow(node, linkedto)) do
        begin
        if not DSE(node).priorityoutputQ(linenum).EMPTY then
            prioritypackets;
        if (L2VS(node, linkedto) <>
                          L2sendwindow(node, linkedto)) then
            begin
            message :- DSE(node).outputQ(linenum).FIRST;
            message.OUT;
            comment assign and increment layer 2 VS;
            message.NS := L2VS(node, linkedto);
            L2VS(node, linkedto) := L2VS(node, linkedto) + 1;
            if L2VS(node, linkedto) > maxsequencenum then
               L2VS(node, linkedto) := 0;
            DSE(node).transmit(message, linenum, node);
            message :- NONE;
            end;
        end of outputQ;
    end of while true;
End of Process Class OutputProcess;

comment **************************************************;
Process Class Source (node, intermesstime, packetgentime,
                      avgnumpackets);
    integer node;
    real intermesstime, packetgentime, avgnumpackets;
comment This process generates traffic for this node of the
        network. It calls the process submit to 'send' each
        packet to its node;
Begin
    REF (packet) message;
    integer numpackets, sent, dest, i, j;

While true do
    Begin
    HOLD (NEGEXP (1.0 / intermesstime, seed(2)));
    numpackets := NEGEXP(1.0 / avgnumpackets, seed(3)) + 0.5;
```

```
    dest := RANDINT (minnode, maxnode, seed(4));
    sent := 1;

    While sent <= numpackets do
        begin
        HOLD (NEGEXP (1.0 / packetgentime, seed(5)));
        message :- NEW packet;
        monitor.creation(node, dest);
        INSPECT message do
            begin
            number := monitor.numberpackets(node, dest);
            L2acknowledgement := false;
            L3acknowledgement := false;
            error := false;
            retransmission := false;
            source := node;
            destination := dest;
            if sent = 1 then
               callrequest := true
            else
               callrequest := false;
            network := false;
            size := 1072;
            for i := 1 step 1 until maxnode do
               begin
               for j := 1 step 1 until 2 do
                  totalinternaltime(i,j) := 0;
               entrytime(i) := 0.0;
               end;
            end of INSPECT message;
        sent := sent + 1;
        message.INTO (DSE(node).sourceQ);
        message :- NONE;
        ACTIVATE DSE(node).submit;
        end of group of packets;
    End of while true;
End of Process Class Source;

comment ************************************************;
Process Class Submitter(tonode);
    integer tonode;
comment This process takes packets from the source queue and
        calls the procedure L3receive to add them to the
        network. If there is no spare buffer at the node this
        process waits for a 'short time' and then resumes
        submitting packets;
Begin
    REF (packet) message;
    real timer;

While true do
    begin
    PASSIVATE;
   While not DSE(tonode).sourceQ.EMPTY do
        begin
        if DSE(tonode).buffersinuse <
                                    DSE(tonode).totalbuffers then
            begin
            message :- DSE(tonode).sourceQ.FIRST;
```

```
            message.OUT;
            message.transtime := TIME;
            DSE(tonode).L3receive(message);
            end
        else
            begin
            monitor.bufferlack(tonode, tonode);
            HOLD(timer);
            end;
        end of sourceQ;
        message :- NONE;
    end of while true;
End of Process Class Submitter;

comment ***************************************************;
Process Class Sink(node);
    integer node;
comment This process removes traffic from the network when it
        has reached its destination, collecting statistics as
        it does so. It also maintains the layer 3 VR, generates
        and processes layer 3 acknowledgements.;
Begin
    REF (packet) message, ack, temp;

While true do
    Begin
    PASSIVATE;
    While not DSE(node).dteQ.EMPTY do
        begin
        message :- DSE(node).dteQ.FIRST;
        message.OUT;
        if message.L3acknowledgement then
            begin
            comment process layer 3 acknowledgement that
                    has arrived;
            DSE(node).acknowledged(message, node, 3);
            message :- NONE;
            end
        else
            begin
            comment process received I frame;
            monitor.departure(message, node);
            monitor.packettransfer(message);
            if message.PS = L3VR(message.source, node) then
                begin
                REACTIVATE DSE(node).L3timer(message.source)
                        DELAY (L3windowtimer /
                                DSE(node).outputline(1).linespeed);
                L3VR(message.source, node) :=
                                L3VR(message.source, node) + 1;
                if L3VR(message.source, node) > maxsequencenum then
                    L3VR(message.source, node) := 0;
                if L3VR(message.source, node) =
                        L3receivewindow(message.source, node) then
                    begin
                    comment create layer 3 acknowledement (RR frame);
                    ack :- NEW packet;
                    INSPECT ack do
                        begin
```

```
                    L2acknowledgement := false;
                    L3acknowledgement := true;
                    error := false;
                    retransmission := false;
                    callrequest := false;
                    network := false;
                    source := node;
                    destination := message.source;
                    PR := L3VR(message.source, node);
                    size := 48;
                    end of INSPECT ack;
                temp := DSE(node).inputQ.FIRST;
                while (temp =/= NONE) andthen
                      (temp.L2acknowledgement or
                       temp.L3acknowledgement) do
                    temp := temp.SUC;
                if temp =/= NONE then
                    ack.PRECEDE(temp)
                else
                    ack.INTO(DSE(node).inputQ);
                ack := NONE;
                temp := NONE;
                L3receivewindow(message.source, node) :=
                          L3receivewindow(message.source, node) +
                                                  L3receivewindowsize;
                if L3receivewindow(message.source, node) >
                                                  maxsequencenum then
                    L3receivewindow(message.source, node) :=
                          L3receivewindow(message.source, node) -
                                                  maxsequencenum - 1;
                DSE(node).route;
                end of L3acknowledgement creation;
            if message.source <> node then
                begin
                DSE(node).buffersinuse :=
                                          DSE(node).buffersinuse - 1;
                monitor.bufferutilisation(node,
                                             DSE(node).buffersinuse);
                end;
            end
        else
            begin
            comment discard packet at layer 3;
            message := NONE;
            end;

        message := NONE;
        end;
    end of dteQ for this node;
  End of while true;
End of Process Sink;

comment ************************************************;
Process Class Retransmitter(node);
   integer node;
comment This process is activated each time a packet is
        transmitted between two nodes. If the procedure
        Acknowledged has not set the boolean needed to
        false before the timer expires the procedure
```

```
        route is called to retransmit the packet;
Begin
    REF (packet) message, temp;
    real timer;
    integer sentto;
    boolean needed, layer3;

HOLD(timer);
While needed do
    begin
    monitor.retransmissions(node, layer3);
    message.retransmission := true;
    message.layer3 := layer3;
    comment retransmissions get priority;
    temp :- DSE(node).input0.FIRST;
    while (temp =/= NONE) andthen (temp.retransmission or
          temp.L2acknowledgement or temp.L3acknowledgement) do
       temp :- temp.SUC;
    if temp =/= NONE then
       message.PRECEDE(temp)
    else
       message.INTO (DSE(node).input0);
    DSE(node).Route;
    HOLD(timer);
    end;
CURRENT.OUT;
End of Process Class Retransmitter;

comment ************************************************************;
CLASS StatsCollecter;
comment This Process Class contains all the data collection
        procedures;
Begin
    real array utilStartTime(1:maxnode, 1:maxnode),
               utilisation(1:maxnode, 1:maxnode),
               waittime(1:maxnode, 1:maxnode),
               waittimeSq(1:maxnode, 1:maxnode),
               maxwait(1:maxnode,1:maxnode),
               transfertime(1:maxnode, 1:maxnode),
               transtimeSq(1:maxnode, 1:maxnode),
               mintransfer(1:maxnode,1:maxnode),
               maxtransfer(1:maxnode,1:maxnode),
               bufferutil(1:maxnode), lasttime(1:maxnode),
               minbufferutil(1:maxnode),
               maxbufferutil(1:maxnode),
               internaltime(1:maxnode, 1:2),
               internaltimeSq(1:maxnode, 1:2),
               mininternaltime(1:maxnode, 1:2),
               maxinternaltime(1:maxnode, 1:2);
    integer array numacks(1:maxnode,1:maxnode,1:2),
                  numwaits(1:maxnode, 1:maxnode),
                  numberpackets(1:11, 1:11),
                  numpacketstrans(1:maxnode, 1:maxnode),
                  packeterrors(1:maxnode, 1:maxnode),
                  numretrans(1:maxnode, 1:2),
                  bufferslacking(1:maxnode, 1:2),
                  numthrough node(1:maxnode, 1:2);

    comment ------------------------------------------------------------;
```

```
Procedure UtilStart (x, y);
    integer x, y;
comment monitors the start of transmission on the line
        between x and y;
Begin
utilStartTime(x, y) := TIME;
End of Procedure UtilStart;

comment ----------------------------------------------------------------;
Procedure UtilEnd (x,y);
    integer x, y;
comment monitors the end of transmission on the line
        between x and y;
Begin
utilisation(x,y) := utilisation(x,y) + TIME -
                                        utilStartTime(x,y);
End of Procedure UtilEnd;

comment ----------------------------------------------------------------;
Procedure Acknowledgements(x,y, layer);
    integer x,y, layer;
comment counts the number of acknowledgements between
        each pair of nodes at each layer;
Begin
if layer = 2 then
    numacks(x,y,1) := numacks(x,y,1) + 1
else
    numacks(x,y,2) := numacks(x,y,2) + 1;
End of Procedure Acknowledgements;

comment ----------------------------------------------------------------;
Procedure Waiting (x, y, waitingtime);
    integer x, y;
    real waitingtime;
comment monitors the time each packet spends waiting for the
        line between node x and y;
Begin
if waitingtime > 0.0 then
    begin
    waitingtime := waitingtime * 1000;
    numwaits(x,y) := numwaits(x,y) + 1;
    waittime(x,y) := waittime(x,y) + waitingtime;
    waittimeSq(x,y) := waittimeSq(x,y) + waitingtime *
                                        waitingtime;
    if waitingtime > maxwait(x,y) then
       maxwait(x,y) := waitingtime;
    end;
End of Procedure Waiting;

comment ----------------------------------------------------------------;
Procedure Creation (source, dest);
    integer source, dest;
comment This procedure monitors the number of packets
        created at each source;
Begin
numberpackets(source, dest) :=
                                numberpackets(source, dest) + 1;
numberpackets(11, dest) := numberpackets(11, dest) + 1;
numberpackets(source, 11) := numberpackets(source, 11) + 1;
```

```
numberpackets(ii, ii) := numberpackets(ii, ii) + 1;
End of Procedure Creation;

comment ------------------------------------------------------------;
Procedure Entry(message, x);
    REF (packet) message;
    integer x;
comment notes the entry of a packet into a node;
Begin
    integer y;
if x = message.destination then
    y := 1
else
    y := 2;
numthroughnode(x,y) := numthroughnode(x,y) + 1;
message.entrytime(x) := TIME;
End of Procedure Entry;

comment ------------------------------------------------------------;
Procedure Departure(message, x);
    REF (packet) message;
    integer x;
comment notes the departure of a packet from a node;
Begin
    real inttime;
    integer y;
if x = message.destination then
    y := 1
else
    y := 2;
inttime := (TIME - message.entrytime(x)) * 1000;
message.totalinternaltime(x,y) :=
                    message.totalinternaltime(x,y) + inttime;
if inttime < mininternaltime(x,y) then
    mininternaltime(x,y) := inttime;
if inttime > maxinternaltime(x,y) then
    maxinternaltime(x,y) := inttime;
End of Procedure Departure;

comment ------------------------------------------------------------;
Procedure PacketTransfer (message);
    REF (packet) message;
comment this procedure collects the data on packet
        transfer times;
Begin
    real transtime;
    integer x,y;
transtime := (TIME - message.transtime) * 1000;
x := message.source;
y := message.destination;
numpacketstrans(x,y) := numpacketstrans(x,y) + 1;
transfertime(x,y) := transfertime(x,y) + transtime;
transtimeSq(x,y) := transtimeSq(x,y) +
                                transtime * transtime;
if transtime < mintransfer(x,y) then
    mintransfer(x,y) := transtime;
if transtime > maxtransfer(x,y) then
    maxtransfer(x,y) := transtime;
```

```
for x := 1 step 1 until maxnode do
    begin
    if x = message.destination then
       y := 1
     else
       y := 2;
    internaltime(x,y) := internaltime(x,y) +
                         message.totalinternaltime(x,y);
    internaltimeSq(x,y) := internaltimeSq(x,y) +
                          message.totalinternaltime(x,y) *
                           message.totalinternaltime(x,y);
    end;

if dumpon then
    begin
    for i := 1 step 1 until message.source - 1 do
       dump(message.destination).OUTTEXT(blanks(4));
    dump(message.destination).OUTINT(message.number,4);
    dump(message.destination).OUTIMAGE;
    end of dump;
End of procedure packettransfer;

comment ----------------------------------------------------------------;
Procedure Error(x,y);
    integer x,y;
comment counts the number of transmission errors between
        each pair of nodes;
Begin
packeterrors(x,y) := packeterrors(x,y) + 1;
End of Procedure Error;

comment ----------------------------------------------------------------;
Procedure Retransmissions(x, layer3);
    integer x;
    boolean layer3;
comment counts the number of retransmissions from each
        node at each layer;
Begin
if layer3 then
    numretrans(x,2) := numretrans(x,2) + 2
else
    numretrans(x,1) := numretrans(x,1) + 1;
End of Procedure Retransmissions;

comment ----------------------------------------------------------------;
Procedure BufferUtilisation(x, number);
    integer x, number;
comment measures the utilisation of the buffers at each node;
Begin
bufferutil(x) := bufferutil(x) + number *
                                  (TIME - lasttime(x));
if number > maxbufferutil(x) then
    maxbufferutil(x) := number;
if number < minbufferutil(x) then
    minbufferutil(x) := number;
lasttime(x) := TIME;
End of Procedure BufferUtilisation;

comment ----------------------------------------------------------------;
```

```
Procedure Bufferlack(x,y);
    integer x,y;
comment counts the number of times a packet can not be received
        because there is no spare buffer at its immediate
        destination;
Begin
comment local or non-local packet unable to be stored;
if x = y then
    bufferslacking(x,1) := bufferslacking(x,1) + 1
else
    bufferslacking(x,2) := bufferslacking(x,2) + 1;
End of Procedure Bufferlack;

comment ----------------------------------------------------------;
Procedure Reset;
comment This procedure initialises all the counters used
        by the monitor;
Begin
    integer i,j,k;
for i := minnode step 1 until maxnode do
    begin
    for j := minnode step 1 until maxnode do
        begin
        utilStartTime(i,j) := utilisation(i,j) := 0.0;
        waitTime(i,j) := waitTimeSq(i,j) := 0.0;
        maxwait(i,j) := 0.0;
        numwaits(i,j) := 0;
        transferTime(i,j) := transTimeSq(i,j) := 0.0;
        mintransfer(i,j) := 999999999;
        maxtransfer(i,j) := 0.0;
        numberpackets(i,j) := numberpackets(i,11) := 0;
        numberpackets(11,j) := numpacketstrans(i,j) := 0;
        packeterrors(i,j) := 0;
        numacks(i,j,1) := numacks(i,j,2) := 0;
        end;

    bufferutil(i) := 0.0;
    lastTime(i) := TIME;
    minbufferutil(i) := 99999999;
    maxbufferutil(i) := 0.0;
    for k := i step 1 until 2 do
        begin
        numretrans(i,k) := 0;
        bufferslacking(i,k) := 0;
        numthroughnode(i,k) := 0;
        internaltime(i,k) := internaltimeSq(i,k) := 0.0;
        mininternaltime(i,k) := 999999999;
        maxinternaltime(i,k) := 0.0;
        end;
    end;
numberpackets(11,11) := 0;
End of Procedure Reset;

comment ----------------------------------------------------------;
Procedure Report;
comment prints report on statistics gathered by the
        StatsCollecter;
Begin
    comment - - - - - - - - - - - - - - - - - - - - - - - - - - - -;
```

```
Real Procedure Mean (sigma, number);
    real sigma;
    integer number;
Begin
if number > 0 then
    Mean := sigma / number
else
    Mean := 0.0;
End of Real Procedure Mean;

comment - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -;
Real Procedure StDev(sigma, sigmaSq, number);
    real sigma, sigmaSq;
    integer number;
Begin
    real sigmaTimesSigma;
sigmaTimesSigma := sigma * sigma;
StDev := 0.0;

if (number > 1) then
    begin
    if sigmaSq >= sigmaTimesSigma / number then
        StDev :=
               SQRT( (sigmaSq - sigmaTimesSigma / number) /
                              (number - 1) );
    end;
End of Real Procedure StDev;

comment - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ;
Procedure Title;
Begin
    integer i;
outf.OUTTEXT("Report on the");
outf.OUTIMAGE;
outf.OUTTEXT("==============");
outf.OUTIMAGE;
outf.OUTTEXT(networkname);
outf.OUTIMAGE;
outf.OUTTEXT(" ");
outf.OUTIMAGE;

outf.OUTTEXT("  Presimulation Time, Simulation Time");
outf.OUTTEXT(" (seconds)");
outf.OUTIMAGE;
outf.OUTFIX(presimtime, 2, 12);
outf.OUTFIX(simtime, 2, 19);
outf.OUTIMAGE;
outf.OUTTEXT("                          Random Number Seeds");
outf.OUTIMAGE;
for i := 1 step 1 until 5 do
    outf.OUTINT(initseed(i), 8);
outf.OUTIMAGE;
outf.OUTTEXT(" ");
outf.OUTIMAGE;
End of Procedure Title;

comment - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ;
Procedure Created;
Begin
```

```
    integer i,j;
outf.OUTTEXT("Number of packets created");
outf.OUTIMAGE;
outf.OUTTEXT("-------------------------");
outf.OUTIMAGE;
outf.OUTTEXT("Source             Destination");
outf.OUTIMAGE;
outf.IMAGE.SUB(7, 12) := DSE(minnode).nodename;
for i := minnode + 1 step 1 until maxnode do
    outf.IMAGE.SUB(7 * i, 5 + 7 * i) := DSE(i).nodename;
outf.IMAGE.SUB(7 * (maxnode + 1),
                   5 + 7 * (maxnode +1)) := "Total ";
outf.OUTIMAGE;
for i := minnode step 1 until maxnode do
    begin
    outf.OUTTEXT(DSE(i).nodename);
    outf.OUTTEXT(" ");
    outf.OUTINT(numberpackets(i,1), 3);
    for j := minnode + 1 step 1 until maxnode do
        outf.OUTINT(numberpackets(i,j), 7);
    outf.OUTINT(numberpackets(i,11), 7);
    outf.OUTIMAGE;
    end;
outf.OUTTEXT("Total   ");
outf.OUTINT(numberpackets(11,1), 3);
for j := minnode + 1 step 1 until maxnode do
    outf.OUTINT(numberpackets(11,j), 7);
outf.OUTINT(numberpackets(11,11), 7);
outf.OUTIMAGE;
outf.OUTTEXT(" ");
outf.OUTIMAGE;
End of Procedure created;

comment - - - - - - - - - - - - - - - - - - - - - - - - - - ;
Procedure TransferStats;
Begin
    real totaltrans;
    integer i,j;
outf.OUTTEXT("Packet transfer times (milliseconds)");
outf.OUTIMAGE;
outf.OUTTEXT("------------------------------------");
outf.OUTIMAGE;
outf.OUTTEXT("From     To      Number     Mean        ");
outf.OUTTEXT("StDev     ");
outf.OUTTEXT("Minimum   Maximum");
outf.OUTIMAGE;
totaltrans := 0;
for i := minnode step 1 until maxnode do
    for j := minnode step 1 until maxnode do
        begin
        outf.OUTTEXT(DSE(i).nodename);
        outf.OUTTEXT(" ");
        outf.OUTTEXT(DSE(j).nodename);
        outf.OUTINT (numpacketstrans(i,j), 5);
        outf.OUTFIX (Mean (transfertime(i,j),
                           numpacketstrans(i,j)), 5, 12);
        outf.OUTFIX (StDev (transfertime(i,j),
                            transtimeSq(i,j),
                            numpacketstrans(i,j)), 5, 12);
```

```
        if numpacketstrans(i,j) <> 0 then
           outf.OUTFIX (mintransfer(i,j), 5, 10)
        else
           outf.OUTFIX(0.0, 5, 10);
        outf.OUTFIX (maxtransfer(i,j), 5, 12);
        outf.OUTIMAGE;
        totaltrans := totaltrans + numpacketstrans(i,j);
        end;
outf.OUTTEXT("Total");
outf.OUTINT(totaltrans,13);
outf.OUTIMAGE;
outf.OUTTEXT(" ");
outf.OUTIMAGE;
End of Procedure TransferStats;

comment - - - - - - - - - - - - - - - - - - - - - - - - ;
Procedure TimeInNode;
Begin
    integer i, j;
outf.OUTTEXT("Time spent in each node (milliseconds)");
outf.OUTIMAGE;
outf.OUTTEXT("-------------------------------------------");
outf.OUTIMAGE;
for j := 1 step 1 until 2 do
    begin
    if j = 1 then
       outf.OUTTEXT("                          Local")
    else
       outf.OUTTEXT("                         Nonlocal");
    outf.OUTIMAGE;
    outf.OUTTEXT("Node     Number     Mean          StDev");
    outf.OUTTEXT("              Minimum     Maximum");
    outf.OUTIMAGE;
    for i := 1 step 1 until maxnode do
       begin
       outf.OUTTEXT(DSE(i).nodename);
       outf.OUTTEXT(" ");
       outf.OUTINT(numthroughnode(i,j), 5);
       outf.OUTFIX(Mean (internaltime(i,j),
                         numthroughnode(i,j)), 5, 12);
       outf.OUTFIX( StDev( internaltime(i,j),
                           internaltimeSq(i,j),
                           numthroughnode(i,j)), 5, 12);
       if mininternaltime(i,j) < 99999 then
          outf.OUTFIX(mininternaltime(i,j), 5, 12)
       else
          outf.OUTFIX(0.0, 5, 12);
       outf.OUTFIX(maxinternaltime(i,j), 5, 12);
       outf.OUTIMAGE;
       end;
    end;
outf.OUTTEXT(" ");
outf.OUTIMAGE;
End of Procedure TimeInNode;

comment - - - - - - - - - - - - - - - - - - - - - - - - ;
Procedure Errors;
Begin
    integer i, j, total;
```

```
outf.OUTTEXT("Number of packets with errors received");
outf.OUTIMAGE;
outf.OUTTEXT("-------------------------------------------------------");
outf.OUTIMAGE;
outf.OUTTEXT("From     At     Number");
outf.OUTIMAGE;
total := 0;
for i := minnode step 1 until maxnode do
    for j := minnode step 1 until maxnode do
        begin
        outf.OUTTEXT(DSE(i).nodename);
        outf.OUTTEXT(" ");
        outf.OUTTEXT(DSE(j).nodename);
        outf.OUTINT(packeterrors(i,j), 5);
        total := total + packeterrors(i,j);
        outf.OUTIMAGE;
        end;
outf.OUTTEXT("Total");
outf.OUTINT(total, 13);
outf.OUTIMAGE;
outf.OUTTEXT(" ");
outf.OUTIMAGE;
End of Procedure Errors;

comment - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ;
Procedure Retransmissions;
Begin
    integer totretrans2, totretrans3, i;
outf.OUTTEXT("Number of Retransmissions");
outf.OUTIMAGE;
outf.OUTTEXT("-----------------------------------------");
outf.OUTIMAGE;
outf.OUTTEXT("From   Layer 2  Layer 3");
outf.OUTIMAGE;
totretrans2 := totretrans3 := 0;
for i := minnode step 1 until maxnode do
    begin
    outf.OUTTEXT(DSE(i).nodename);
    outf.OUTINT(numretrans(i,1), 4);
    outf.OUTINT(numretrans(i,2), 9);
    totretrans2 := totretrans2 + numretrans(i,1);
    totretrans3 := totretrans3 + numretrans(i,2);
    outf.OUTIMAGE;
    end;
outf.OUTTEXT("Total");
outf.OUTINT(totretrans2, 5);
outf.OUTINT(totretrans3, 9);
outf.OUTIMAGE;
outf.OUTTEXT(" ");
outf.OUTIMAGE;
End of Procedure Retransmissions;

comment - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - ;
Procedure Acknowledgements;
Begin
    integer i,j, totaltrans2, totaltrans3;
totaltrans2 := totaltrans3 := 0;
outf.OUTTEXT("Number of acknowledgements");
```

```
outf.OUTIMAGE;
outf.OUTTEXT("----------------------------------------");
outf.OUTIMAGE;
outf.OUTTEXT("From    To       Layer 2   Layer 3");
outf.OUTIMAGE;
for i := minnode step 1 until maxnode do
    for j := minnode step 1 until maxnode do
        if (numacks(i,j,1) > 0) or
                (numacks(i,j,2) > 0) then
            begin
            outf.OUTTEXT(DSE(i).nodename);
            outf.OUTTEXT(" ");
            outf.OUTTEXT(DSE(j).nodename);
            outf.OUTINT(numacks(i,j,1), 4);
            outf.OUTINT(numacks(i,j,2), 10);
            outf.OUTIMAGE;
            totaltrans2 := totaltrans2 + numacks(i,j,1);
            totaltrans3 := totaltrans3 + numacks(i,j,2);
            end;
outf.OUTTEXT("Total");
outf.OUTINT(totaltrans2, 12);
outf.OUTINT(totaltrans3, 10);
outf.OUTIMAGE;
outf.OUTTEXT(" ");
outf.OUTIMAGE;
End of Procedure Acknowledgements;

comment - - - - - - - - - - - - - - - - - - - - - - - - - - - - -;
Procedure LineUtilisations;
Begin
    integer i,j,k;
outf.OUTTEXT("Line Utilisation");
outf.OUTIMAGE;
outf.OUTTEXT("----------------------------");
outf.OUTIMAGE;
outf.OUTTEXT("From     To        Utilisation (%)");
outf.OUTIMAGE;
for i := minnode step 1 until maxnode do
    for j := minnode step 1 until maxnode do
        if utilisation(i,j) > 0 then
            begin
            outf.OUTTEXT(DSE(i).nodename);
            outf.OUTTEXT(" ");
            outf.OUTTEXT(DSE(j).nodename);
            utilisation(i,j) := utilisation(i,j) /
                                              simtime * 100;
            outf.OUTFIX(utilisation(i,j), 5, 12);
            outf.OUTIMAGE;
            end;
outf.OUTTEXT(" ");
outf.OUTIMAGE;
End of Procedure LineUtilisations;

comment - - - - - - - - - - - - - - - - - - - - - - - - - - - - -;
Procedure WaitTimes;
Begin
    integer i,j,k;
outf.OUTTEXT("Wait times for lines (milliseconds)");
outf.OUTIMAGE;
```

```
outf.OUTTEXT("------------------------------------");
outf.OUTIMAGE;
outf.OUTTEXT("From     To     Number   Mean       StDev");
outf.OUTTEXT("     Maximum");
outf.OUTIMAGE;
for i := minnode step 1 until maxnode do
   for j := minnode step 1 until maxnode do
      if numwaits(i,j) > 0 then
         begin
         outf.OUTTEXT(DSE(i).nodename);
         outf.OUTTEXT(" ");
         outf.OUTTEXT(DSE(j).nodename);
         outf.OUTINT (numwaits(i,j), 5);
         outf.OUTFIX ( Mean (waittime(i,j),
                              numwaits(i,j)), 5, 12);
         outf.OUTFIX ( StDev (waittime(i,j),
                               waittimeSq(i,j),
                               numwaits(i,j)), 5, 10);
         outf.OUTFIX (maxwait(i,j), 5, 12);
         outf.OUTIMAGE;
         end;
outf.OUTTEXT(" ");
outf.OUTIMAGE;
End of Procedure WaitTimes;

comment - - - - - - - - - - - - - - - - - - - - - - ;
Procedure BufferUtilisation;
Begin
   integer i;
outf.OUTTEXT("Buffer Utilisation");
outf.OUTIMAGE;
outf.OUTTEXT("-------------------------------");
outf.OUTIMAGE;
outf.OUTTEXT("Node  Utilisation (%)  Minimum    ");
outf.OUTTEXT("Maximum  Available");
outf.OUTIMAGE;
for i := minnode step 1 until maxnode do
   begin
   outf.OUTTEXT(DSE(i).nodename);
   outf.OUTFIX( (bufferutil(i) / simtime /
                 DSE(i).totalbuffers * 100), 5, 12);
   if minbufferutil(i) < 99999 then
      outf.OUTFIX(minbufferutil(i), 5, 12)
   else
      outf.OUTFIX(0.0, 5, 12);
   outf.OUTFIX(maxbufferutil(i), 5, 12);
   outf.OUTINT(DSE(i).totalbuffers, 8);
   outf.OUTIMAGE;
   end;
outf.OUTTEXT(" ");
outf.OUTIMAGE;
End of BufferUtilisation;

comment - - - - - - - - - - - - - - - - - - - - - - ;
Procedure Bufferlack;
Begin
   integer i;
outf.OUTTEXT("Packets rejected due to lack of buffers");
outf.OUTIMAGE;
```

```
        outf.OUTTEXT("-----------------------------------------------------------------");
        outf.OUTIMAGE;
        outf.OUTTEXT("Node     Local   Nonlocal");
        outf.OUTIMAGE;
        for i := minnode step 1 until maxnode do
            begin
            outf.OUTTEXT(DSE(i).nodename);
            outf.OUTINT(bufferslacking(i,1), 5);
            outf.OUTINT(bufferslacking(i,2), 9);
            outf.OUTIMAGE;
            end;
        outf.OUTTEXT(" ");
        outf.OUTIMAGE;
        End of Procedure Bufferlack;

    comment - - - - -Report Body- - - - - - - - - - - - - - - - - - - - ;

    SYSOUT.OUTTEXT("Simulation complete, writing report.");
    SYSOUT.OUTIMAGE;

    Title;
    created;
    transferstats;
    timeinnode;
    errors;
    retransmissions;
    acknowledgements;
    lineUtilisations;
    waitTimes;
    bufferUtilisation;
    bufferlack;

    End of procedure Report;

comment ---------------------------------------------------------------------;
End of CLASS StatsCollecter;

comment ***************************************************************;
Procedure Initialise;
comment Main initialisation routine;
Begin
    integer i, j, dest;

    comment ---------------------------------------------------------------;
    REF (NodeClass) Procedure EstablishNode (node);
        integer node;
    comment This procedure initialises a node and activates
            all the processes associated with it.;
    Begin
        REF (NodeClass) temp;
        integer numlines, i,j, k;

    temp :- NEW NodeClass(node);
    inf.INIMAGE; inf.INIMAGE; inf.INIMAGE;
    INSPECT temp do
        begin
        nodename :- BLANKS(6);
        nodename := inf.IMAGE.SUB(1,6);
```

```
inf.INIMAGE;
buffersinuse := 0;
totalbuffers := inf.ININT;
inf.INIMAGE;
numlines := inf.ININT;
inf.INIMAGE;

for i := 1 step 1 until numlines do
    begin
    outputQ(i) :- NEW HEAD;
    priorityoutputQ(i) :- NEW HEAD;
    inf.INIMAGE;
    outputline(i) :- NEW outputprocess(node, i);
    ACTIVATE outputline(i);

    INSPECT outputline(i) do
        begin
        linkedto := inf.ININT;
        linechoice(linkedto) := i;
        linespeed := inf.INREAL;
        propagationtime := inf.INREAL * inf.INREAL /
                                                1000000;
        errorprob := inf.INREAL;
        end of INSPECT outputline(i);
    end;

inputQ :- NEW HEAD;
DTEQ :- NEW HEAD;
L2unacknowledged :- NEW HEAD;
L3unacknowledged :- NEW HEAD;
SourceQ :- NEW HEAD;
L3waiting :- NEW HEAD;

inf.INIMAGE;
inf.INIMAGE;
DTEsource :- NEW source (node, inf.INREAL, inf.INREAL,
                         inf.INREAL);

Submit :- NEW Submitter(node);
Submit.timer := inf.INREAL;
ACTIVATE Submit;

DTEsink :- NEW sink(node);
ACTIVATE DTEsink;

inf.INIMAGE;
for i := minnode step 1 until maxnode do
    begin
    inf.INIMAGE;
    if routingmethod = 1 then
        k := 1
    else
        k := 5;
    dest := inf.ININT;
    for j := 1 step 1 until k do
        routingtable(dest, j) := inf.ININT;
    end;

end of inspect temp;
```

```
    EstablishNode := temp;
    End of EstablishNode;

comment ----------Body of Initialise----------;

comment open the configuration file;
SYSOUT.OUTTEXT("Enter name of configuration file");
SYSOUT.OUTIMAGE;
SYSIN.INIMAGE;
inf := NEW INFILE (SYSIN.IMAGE.STRIP);
inf.OPEN (blanks(80));

comment open the stats file;
SYSOUT.OUTTEXT("Enter the name of the statistics file");
SYSOUT.OUTIMAGE;
SYSIN.INIMAGE;
outf := NEW OUTFILE(SYSIN.IMAGE.STRIP);
outf.OPEN(blanks(80));

comment dump files required?;
SYSOUT.OUTTEXT("Enter y for dumps");
SYSOUT.OUTIMAGE;
SYSIN.INIMAGE;
if (SYSIN.IMAGE.STRIP = "Y") or (SYSIN.IMAGE.STRIP = "y") then
    dumpon := true
else
    dumpon := false;

comment debugging required?;
SYSOUT.OUTTEXT("Enter y for debugging");
SYSOUT.OUTIMAGE;
SYSIN.INIMAGE;
if (SYSIN.IMAGE.STRIP = "Y") or (SYSIN.IMAGE.STRIP = "y") then
    begin
    debugon := true;
    SYSOUT.OUTTEXT("Enter numbers of left and right nodes");
    SYSOUT.OUTIMAGE;
    SYSIN.INIMAGE;
    leftnode := SYSIN.ININT;
    rightnode := SYSIN.ININT;
    end
else
    debugon := false;

comment get the random number seeds, the presimulation and
        simulation times from the terminal;
SYSOUT.OUTTEXT("Enter 5 random number seeds");
SYSOUT.OUTIMAGE;
SYSIN.INIMAGE;
for i := 1 step 1 until 5 do
    seed(i) := initseed(i) := SYSIN.ININT;

SYSOUT.OUTTEXT ("Enter Presimulation and Simulation time ");
SYSOUT.OUTTEXT("(in seconds)");
SYSOUT.OUTIMAGE;
SYSIN.INIMAGE;
presimtime := SYSIN.INREAL;
simtime := SYSIN.INREAL;
```

```
comment read the network name, number of nodes from the
        configuration file, and routing method;
inf.INIMAGE;
networkname :- BLANKS(80);
networkname := inf.IMAGE;
inf.INIMAGE;
maxnode := inf.ININT;
minnode := 1;
inf.INIMAGE;
routingmethod := inf.ININT;

maxsequencenum := 7;
channel :- NEW ServerClass;
monitor :- NEW statscollecter;

comment establish each node;
for i := minnode step 1 until maxnode do
    DSE(i) :- EstablishNode(i);

comment read the routing algorithm time,
                    frame check sequence time,
                    acknowledgement processing time,
                    layer 2 and 3 send window size,
                    layer 2 and 3 receive window size,
                    layer 2 and 3 retransmission timer,
                    and layer 2 and 3 acknowledgement delay time
                    from the configuration file;
inf.INIMAGE; inf.INIMAGE; inf.INIMAGE;
routingalgtime := inf.INREAL;
inf.INIMAGE; inf.INIMAGE;
FCStime := inf.INREAL;
inf.INIMAGE; inf.INIMAGE;
acknowledgetimer(1) := inf.INREAL;
acknowledgetimer(2) := inf.INREAL;
inf.INIMAGE; inf.INIMAGE;
L2sendwindowsize := inf.ININT;
L3sendwindowsize := inf.ININT;
inf.INIMAGE; inf.INIMAGE;
L2receivewindowsize := inf.ININT;
L3receivewindowsize := inf.ININT;
inf.INIMAGE; inf.INIMAGE;
L2timerdelay := inf.ININT;
L3timerdelay := inf.ININT;
inf.INIMAGE; inf.INIMAGE;
L2windowtimer := inf.ININT;
L3windowtimer := inf.ININT;

comment open the dump files;
if dumpon then
    begin
    for i := 1 step 1 until maxnode do
        begin
        dumpfile :- blanks(6);
        dumpfile.SUB(1,4) := "dump";
        if i < 10 then
            dumpfile.SUB(5,1).PUTINT(i)
        else
            dumpfile.SUB(5,2).PUTINT(i);
        dump(i) :- NEW OUTFILE(dumpfile);
```

```
        dump(i).OPEN(blanks(80));
        dump(i).OUTTEXT("Packets received by Node");
        dump(i).OUTINT(i,3);
        dump(i).OUTTEXT(". From Nodes:");
        dump(i).OUTIMAGE;
        for j := 1 step 1 until maxnode do
           dump(i).OUTINT(j,4);
        dump(i).OUTIMAGE;
        for j := 1 step 1 until maxnode do
           dump(i).OUTTEXT("----");
        dump(i).OUTIMAGE;
        end;
    end of opening dump files;

comment open debug file;
if debugon then
    begin
    debugfile :- NEW OUTFILE("debug");
    debugfile.OPEN(blanks(130));
    debugfile.OUTTEXT(blanks(30));
    debugfile.OUTTEXT(DSE(leftnode).nodename);
    debugfile.OUTTEXT(blanks(28));
    debugfile.OUTTEXT(DSE(rightnode).nodename);
    debugfile.OUTIMAGE;
    debugfile.OUTTEXT(blanks(9));
    debugfile.OUTTEXT(" L2VS L2VR Send L3VS L3VR Send");
    debugfile.OUTTEXT(blanks(6));
    debugfile.OUTTEXT("NS number NR");
    debugfile.OUTTEXT(blanks(3));
    debugfile.OUTTEXT(" L2VS L2VR Send L3VS L3VR Send");
    debugfile.OUTIMAGE;
    debugfile.OUTTEXT(blanks(19));
    debugfile.OUTTEXT("window          window");
    debugfile.OUTTEXT(blanks(30));
    debugfile.OUTTEXT("window          window");
    debugfile.OUTIMAGE;
    end of opening debug file;

comment initialise sliding window protocol values;
for i := 1 step 1 until maxnode do
    for j := 1 step 1 until maxnode do
        begin
        L2VS(i,j) := L2VR(i,j) := 0;
        L3VS(i,j) := L3VR(i,j) := 0;
        L2sendwindow(i,j) := L2sendwindowsize;
        L3sendwindow(i,j) := L3sendwindowsize;
        L2receivewindow(i,j) := L2receivewindowsize;
        L3receivewindow(i,j) := L3receivewindowsize;
        DSE(i).L2timer(j) :- NEW timerprocess(i,j,2);
        DSE(i).L3timer(j) :- NEW timerprocess(i,j,3);
        ACTIVATE DSE(i).L2timer(j);
        ACTIVATE DSE(i).L3timer(j);
        end;
inf.CLOSE;
inf :- NONE;
End of Procedure Initialise;

comment ************************************************;
Process Class RunningMessage;
```

```
comment This process displays on the screen a message
        indicating how far the model has got;
Begin
While true do
    begin
    HOLD(5.0);
    SYSOUT.OUTTEXT("Running!   ");
    SYSOUT.OUTINT (TIME, 3);
    SYSOUT.OUTTEXT(" seconds elapsed");
    SYSOUT.OUTIMAGE;
    end;
End of Process Class RunningMessage;

comment ******************** MAIN ****************************;

initialise;

for i := minnode step 1 until maxnode do
    ACTIVATE DSE(i).DTEsource;

running :- NEW runningmessage;
ACTIVATE running;

monitor.reset;

HOLD (presimtime);

monitor.reset;

HOLD (simtime);

monitor.report;

outf.CLOSE;
if dumpon then
    for i := 1 step 1 until maxnode do
       dump(i).CLOSE;
if debugon then
    debugfile.CLOSE;

END of Simulation;
END of Program;
```

Appendix 2 Code of EM, the source emulator

```
Begin
Simulation Begin
comment ************************************************
        *                                              *
        *     Source Emulator                          *
        *     ===============                          *
        *                                              *
        *          Written by Paul Reynolds            *
        *                                              *
        *               Date  1/9/1984                 *
        *                                              *
        ************************************************
        This program produces statistics on the
        number of packets generated by a source;

    REF (source) DTE;
    REF (OUTFILE) outf;
    TEXT filename;
    real array counts(1:61), bounds(1:60);
    real intermesstime, packetgentime, avgnumpackets,
         simtime, classwidth;
    integer array seed(1:3), initseed(1:3);
    integer length, totalsent;

    comment ************************************************************;
    Process Class Source (intermesstime, packetgentime,
                          avgnumpackets);
       real intermesstime, packetgentime, avgnumpackets;
    comment This process generates traffic for a node of the
            network. It then calls histo to increment the
            frequency count for the present time interval.;
Begin
    integer numpackets, sent;

While true do
    Begin
    HOLD (NEGEXP (1.0 / intermesstime, seed(1)));
    numpackets := NEGEXP (1.0 / avgnumpackets, seed(2)) + 0.5;
    sent := 1;

    While sent <= numpackets do
       begin
       HOLD (NEGEXP (1.0 / packetgentime, seed(3)));
       sent := sent + 1;
       totalsent := totalsent + 1;
       comment increment counts;
       HISTO (counts, bounds, TIME, 1.0);
       end of group of packets;
    End of while true;
End of Process Class Source;

comment ************************************************************;
Procedure Report (counts, bounds, length, filename);
    value filename; TEXT filename;
    real array counts, bounds;
    integer length;
comment This procedure writes out the report for this
        program including the histogram.;
Begin
```

```
    integer i;

    comment ----------------------------------------------------------------;
    Procedure Plot (counts, bounds, length, width);
        real array counts, bounds;
        integer length, width;
    comment This procedure plots a histogram;
    Begin
        integer i, j;

    for i := 1 step 1 until length do
        begin
        outf.OUTTEXT("      |");
        if counts(i) > width then
            begin
            outf.OUTTEXT(" Overflow");
            outf.OUTTEXT(blanks(width - 9));
            end
        else
            begin
            for j := 1 step 1 until counts(i) do
                outf.OUTTEXT("*");
            outf.OUTTEXT(blanks(width - counts(i)));
            end;

        outf.OUTINT(counts(i),3);
        outf.OUTIMAGE;

        outf.OUTFIX(bounds(i),2,5);
        outf.OUTTEXT("|");
        outf.OUTIMAGE;
        end of row;
    End of Procedure Plot;

comment ----------------Body of Report----------------------------------------;
comment output to screen or file?;
if filename = blanks(10) then
    outf :- SYSOUT
else
    begin
    outf :- NEW OUTFILE(filename);
    outf.OPEN(blanks(80));
    end;

outf.OUTTEXT("Random Number Seeds");
outf.OUTIMAGE;
for i := 1 step 1 until 3 do
    outf.OUTINT(initseed(i), 8);
outf.OUTIMAGE;
outf.OUTTEXT("Intermessage and Interpacket time");
outf.OUTIMAGE;
outf.OUTFIX(intermesstime,5,10);
outf.OUTFIX(packetgentime,5,10);
outf.OUTIMAGE;
outf.OUTTEXT("Average number of packets per message");
outf.OUTIMAGE;
outf.OUTFIX(avgnumpackets,2,6);
outf.OUTIMAGE;
outf.OUTTEXT("Simulation time");
```

```
outf.OUTIMAGE;
outf.OUTFIX(simtime,2,6);
outf.OUTIMAGE;
outf.OUTTEXT(blanks(80));
outf.OUTIMAGE;
outf.OUTTEXT(" Time");
outf.OUTIMAGE;

plot(counts, bounds, length, 30);

comment print average packet generation figures;
avgnumpackets := totalsent / simtime;
outf.OUTTEXT("Average number of packets generated per second");
outf.OUTFIX(avgnumpackets,2,7);
outf.OUTIMAGE;
avgnumpackets := avgnumpackets * 3600;
outf.OUTTEXT("Average number of packets generated per hour");
outf.OUTFIX(avgnumpackets,2,10);
outf.OUTIMAGE;
End of Procedure Report;

comment ************************************************************;
Procedure Initialise;
Begin
    integer i;
filename := blanks(10);
SYSOUT.OUTTEXT("enter name of output file (nothing for screen)");
SYSOUT.OUTIMAGE;
SYSIN.INIMAGE;
filename := SYSIN.IMAGE.STRIP;

SYSOUT.OUTTEXT("enter width of histogram classes");
SYSOUT.OUTIMAGE;
SYSIN.INIMAGE;
classwidth := SYSIN.INREAL;

SYSOUT.OUTTEXT("enter 3 random number seeds");
SYSOUT.OUTIMAGE;
SYSIN.INIMAGE;
for i := 1 step 1 until 3 do
    seed(i) := initseed(i) := SYSIN.ININT;

SYSOUT.OUTTEXT("enter intermessage and interpacket times");
SYSOUT.OUTIMAGE;
SYSIN.INIMAGE;
intermesstime := SYSIN.INREAL;
packetgentime := SYSIN.INREAL;

SYSOUT.OUTTEXT("enter average number of packets per message");
SYSOUT.OUTIMAGE;
SYSIN.INIMAGE;
avgnumpackets := SYSIN.INREAL;

SYSOUT.OUTTEXT("enter simulation time");
SYSOUT.OUTIMAGE;
SYSIN.INIMAGE;
simtime := SYSIN.INREAL;

comment ensure room on histogram for whole simulation;
```

```
length := simtime / classwidth + 0.5;
if length > 60 then
   begin
   length := 60;
   simtime := length * classwidth;
   SYSOUT.OUTTEXT("Simulation time TOO LONG, shortened to:");
   SYSOUT.OUTFIX(simtime, 2,8);
   SYSOUT.OUTIMAGE;
   end;

comment initialise bounds of histogram classes;
bounds(1) := classwidth;
for i := 2 step 1 until length do
   bounds(i) := bounds(i-1) + classwidth;
totalsent := 0;
End of Procedure Initialise;

comment ************************ MAIN ********************************;
initialise;

DTE :- NEW source(intermesstime, packetgentime, avgnumpackets);
ACTIVATE DTE;

HOLD(simtime);

report(counts, bounds, length, filename);

if filename <> blanks(10) then
   outf.CLOSE;
End of Simulation;
End of Program;
```