MODELLING ADAPTIVE ROUTING IN WIDE AREA NETWORKS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF SCIENCE
AT THE UNIVERSITY OF CAPE TOWN
IN FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Andrew Hutchison
September 1991

Supervised by
Prof P.S. Kritzinger

Abstract

This study investigates the modelling of adative routing algorithms with specific reference to the algorithm of an existing Wide Area Network (WAN). Packets in the network are routed at each node on the basis of routing tables which contain internal and external delays for each route from the node. The *internal* delay on a route represents the time that packets queued for transmission will have to wait before being transmitted, while the *external delay* on a route represents the delay to other nodes via that route.

Several modelling methods are investigated and compared for the purpose of identifying the most appropriate and applicable technique. A model of routing in the WAN using an *analytic* technique is described. The hypothesis of this study is that dynamic routing can be modelled as a sequence of models exhibiting fixed routing. The modelling rationale is that a series of analytic models is run and solved. The routing algorithm of the WAN studied is such that, if viewed at any time instant, the network is one with static routing and no buffer overflow. This characteristic, together with a real time modelling requirement, influences the modelling technique which is applied. Each model represents a routing update interval and a multiclass open queueing network is used to solve the model during a particular interval.

Descriptions of the design and implementation of *Xwan*, an X Window based modelling system, are provided. A feature of the modelling system is that it provides a Graphical User Interface (GUI), allowing interactive network specification and the direct observation of network routing through the medium of this interface. Various applications of the modelling system are presented, and overall network behaviour is examined. Experimentation with the routing algorithm is conducted, and (tentative)

recommendations are made on ways in which network performance could be improved. A different routing algorithm is also implemented, for the purpose of comparison and to demonstrate the ease with which this can be affected.

Acknowledgements

For their involvement and association with this study I would like to express gratitude to:

- André van der Walt and Ryn Bodde of Network Systems who have been particularly helpful with information on the Distributed Nodal Network, and with comments on the DNN Hardware and Software descriptions.
- Graham Wheeler, Hylton Donnelly, Riël Smit, Guido Zsilavecz, Donald Cook, and Michael "B" of UCT, for their assistance at various stages of the exercise.
- the Abteilung Informatik of the Universität Dortmund, with the tacit agreement of the Deutsche Bundepost, for making MACOM available to me. I would like to thank Heinz Beilner, Bruno Müller-Clostermann and in particular Michael Szczittnick, for their cooperation.
- Prof Kritzinger, for consistently optimal 'thesis throughput' under heavy utilisation; and for excellent project guidance.
- the family Hutchison, and Lisa Horne (my 'S.O.'), for their support and for helping me retain my sense of humour through it all!

Contents

List of Figures

Chapter 1

Adaptive Routing

A progression from isolated computing systems to systems communicating with each other around the world was inevitable. There are already numerous computer networks in existence, but *connectivity* is still a topic of interest and remains a challenge in that the *performance* of many of these networks needs assessment and improvement. In some senses the theory of networking has followed the practice, and many principles have been extracted from the experience gained with actual networks, notably ARPANET and TYMNET [Tym80] [Tym81].

Nothing is as crucial to the realisation of the ideal of inter-connectivity, as the wide area networks which facilitate this. With the advent of gateways across numbers of networks it is even more crucial than ever that the conceptual design of a network and the algorithms employed are effective, efficient and accommodating enough to handle growing user demands.

This places great responsibility on network designers and controllers, who require foresight and vision to allow for enough growth while still retaining service levels. The use of *network models* is crucial in helping the network designer assess future demands and what their impact might be on a network. This study specifically addresses the design and construction of a system to enable the modelling of adaptive routing in a wide area network. Such a modelling system can be invaluable in helping a network planner, but there are difficulties with modelling a system too.

Standardisation attempts have helped to define protocols and inter-network standards, but these standards are often open to various interpretations, thereby differing in practical implementation from their theoretical specification. For the purposes of this study we begin by examining the Open Systems Interconnection (OSI) model as proposed by the International Standards Organisation (ISO).

1.1 The ISO Reference Model

Computer networks are usually organised as a series of layers. The ISO OSI model consists of seven layers: The *Application* Layer, the *Presentation* Layer, the *Session* Layer, the *Transport* Layer, the *Network* Layer, the *Data Link* Layer and the *Physical* Layer (see Figure 1). Each layer performs particular functions, offering certain services to the higher layers. In this way higher layers need not be concerned with how lower layers actually implement the functions which they perform. On each of the communicating machines the same layered structure exists, and corresponding layers on different machines are known as *peer processes.*

A process can be viewed as having three interfaces. *Physical interfaces* exist with the higher and lower layers, implying those above $(n + 1)$ and below $(n - 1)$ any particular layer (n). In addition to these two physical interfaces, a *logical interface* exists with the peer process at the corresponding layer. These interfaces are defined by the various international standards which exist, enabling network designers to change the software of any layer and maintain functional equivalence without affecting the overall network architecture, since the interfaces are standard.

The division of a network into a number of layers is essential for manageability. In this way design problems can be addressed over smaller, more understandable domains using top-down, successive refinement techniques. Once the individual layers have been designed or implemented they can be put together to provide the coherent whole. Zimmermann [Zim80] gives an account of the principles which guided the ISO subcommittee (SC16) in defining the specific set of layers in the OSI architecture. The principles he describes were used in determining layer boundaries, interfaces and functions. Zimmermann identifies the most fundamental aspect of the network

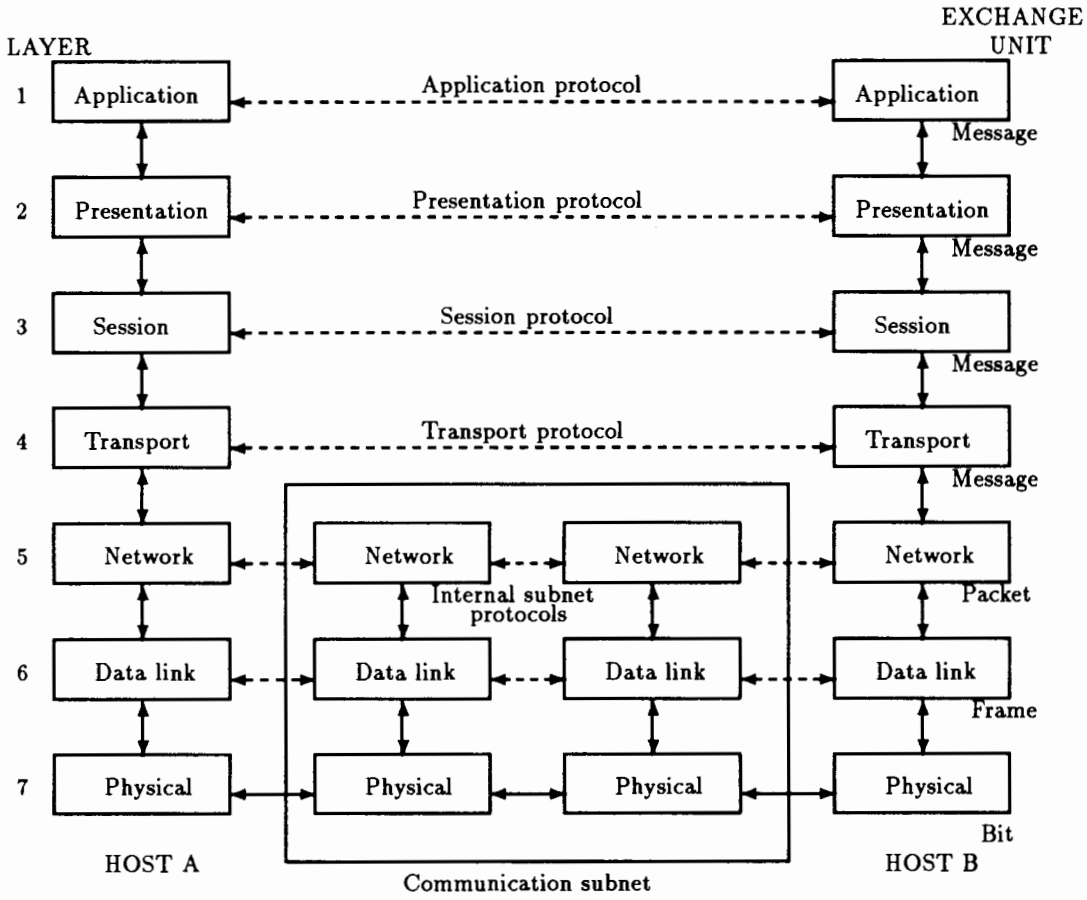


Figure 1: The ISO OSI reference model

layering concept as being that "...each layer adds value to services provided by the set of lower values in such a way that the highest layer is offered the set of services needed to run distributed applications"(p. 426).

1.2 The Network Layer

The ISO OSI layer most pertinent to this study is the *network layer*. It is thus expedient to consider some of the issues relating to this layer in more detail, and in the following sections this is addressed.

1.2.1 Responsibilities

At the network layer peer processes work together in implementing routing and flow control for the network. In this way transport entities, at the layer above this one, are not concerned with routing considerations. New packets from external sites (*exogenous arrivals*) enter the network layer from the transport layer. At external sites, higher level control packets enter the network layer from the transport layer too. At a node, control packets for routing and flow control functions can be generated within the network layer. Packets also arrive from other nodes in the network (*endogenous arrivals*), at the network layer.

Network layer processes perform a routing function in deciding where to send all arriving packets. If a packet is destined for another node, the link on which to forward it is chosen, and the packet is sent via the data link layer for that link. If a packet has arrived at the site it has been sent to, it is passed up to the transport layer. If the packet is a control packet, for routing or flow control, it is processed within the network layer module.

The network layer process determines when to accept packets from a higher layer and when to transmit packets. These decisions constitute the provision of *flow control* in the network. Because of these delays, and delays by the data link layer in accepting packets, *buffering* must be provided at the network layer.

It has been pointed out that "The network layer is conceptually the most complex

of the layered hierarchy since *all* the peer processes at this layer must work together ... [It is] the only layer in which the overall algorithms are distributed between many geographically separated processes" [Ber87, pp.23–24].

1.2.2 Network Services

Packets can be transported through the internal subnet of a network using virtual circuits or datagrams. When *virtual circuits* are used to convey packets through the network a route is chosen from source node[1] to destination node when a circuit is established. Packets on a designated virtual circuit always take the same route through the subnet, and each node on the path of a virtual circuit holds a circuit table enabling correct forwarding of packets. Each packet travelling through the subnet contains a virtual circuit number in its header. In contrast, when *datagrams* are used nodes have tables specifying which outgoing line to select, according to a packet's destination. Using this approach, each packet must contain full destination and source addresses as part of its header.

Inside the subnet a design choice has to be made between virtual circuits and datagrams. A trade off between *nodal memory space* and *bandwidth* is necessary, taking the following points into account. If virtual circuits are implemented packets do not need to contain full destination and source addresses, and can just be numbered. In a datagram approach full destination and source addresses are required in each packet and this may represent a significant amount of overhead, consuming a portion of the available bandwidth. A certain amount of table overhead is required in the case of virtual circuits. Vulnerability has to be considered in selecting the use of virtual circuits, since if a node crashes all virtual circuits have to be abolished. In a datagram implementation a node outage will only affect those packets queued in the node. Similarly, in a datagram environment loss of a communication line can be accommodated by the system, whereas with virtual circuits this is catastrophic. A certain amount of overhead is required to establish a virtual circuit, and in certain circumstances this is justified. A major advantage of datagrams is that each one

[1]The term *node* is used to mean Data Circuit Terminating Equipment (DCE)

is routed individually, enabling traffic to be balanced through the network and the network considered in this study is '*packet switched*' employing a datagram approach.

1.2.3 Packet Lengths

Having distinguished between datagram and virtual circuit implementations, and identified that the WAN examined uses the former approach, it is informative to consider other implications of this particular approach. *Packet length* can have ramifications for packet switched network performance, and some consideration is given here to choice of packet length.

The choice of a maximum packet length is essentially determined by the overhead which is associated with each packet, and the amount of processing that is required for each packet.

As maximum packet length is decreased, messages are split into more packages with the result that any packet overhead, such as headers and trailers, must be repeated many times. This would appear to suggest that a large maximum packet size would be ideal.

The amount of processing required for each packet also has to be considered. If a message is transmitted as one packet, then the full message has to be received at each node before it can be forwarded to the next node. From this scenario a small maximum packet size would seem to be ideal.

Bertsekas and Gallager [Ber87] describe a method for calculating packet length, taking the above factors into consideration. Their final conclusion is that as *packet overhead* increases, packet length should be increased, but as *path length* increases, packet length should be decreased. Using this information some sort of a trade-off has to be made. These authors demonstrate that "a high variability in packet lengths (which occurs with a large packet size) increases delay" [Ber87, p. 84], thus arguing in favour of short packets.

1.3 Routing Algorithms

It has been mentioned that routing is a major concern of the network layer, and network layer design determines the algorithm used for route selection, as well as the data structures which are used to advise the routing process. The successful operation of a packet switched data communication network is dependent on the routing algorithm which is implemented. As the heart of the network, it is essential that the routing algorithm satisfies a number of properties for successful functioning. It should be recognised that successful or adequate functioning may be very different from optimal functioning.

In the following sections we first consider generic design criteria for good routing algorithms, then look at some specific algorithms after which we examine congestion control and its relationship to routing.

1.3.1 Design Criteria

At a local level routing entails "the decision making process in which a node of a packet switched network selects one or more of its outgoing lines on which to forward a packet making its way to an ultimate destination"[Bel86, p. 34]. On a more global level "the effect of good routing is to increase throughput for the same value of average delay per packet under high offered load conditions, and to decrease average delay per packet under low offered load conditions"[Ber87, p. 302]. These two perspectives of routing give us a specific view, as well as the overall vision, of the concerns of network routing.

Before looking at some specific criteria for a good routing algorithm, it is worthwhile to outline the general constraints within which a routing algorithm must function.

Computer networks are *bursty* by nature, with varying customer demands. This implies that the network has to function under varying loads. Not only is a network subject to fluctuating demands on its bandwidth, but it may also suffer node or line failures. Should an *outage* occur, it is desirable that the whole network should not cease functioning, but that traffic should be re-directed around the area where the

fault is. Traffic should also be re-directed around congested nodes, and in general the network should minimise congestion and queueing delay. If buffer space at a node is critically low or unavailable, then the routing algorithm should institute recovery action. It is also important that packets do not cycle eternally through the network. Some control has to be kept on stray packets, should they occur.

All of the abovementioned concerns are the responsibility of the routing algorithm. It is also desirable to minimise the hardware and software requirements at each node, since all of the tasks required of the routing algorithm must be performed within the constraining parameter of processor overhead. It is within these contradictory constraints that the algorithm designer must work.

Tanenbaum [Tan89] has proposed six properties which the ideal routing algorithm should possess. Bell and Jabour [Bel86] have elaborated on these properties, and they are presented here with the insights of both proponents.

The ideal routing algorithm should exhibit:

- **Correctness** — At a most fundamental level an algorithm must work with some measure of competence to be considered successful.[2]

- **Computational simplicity** — So as to minimise packet delay, a routing algorithm should ensure that the processing time at each node is kept to a minimum.

- **Adaptiveness to changing traffic and topologies: robustness** — A routing algorithm must be able to cope with the outage or re-appearance of nodes and links. The algorithm should also be able to cope with varying traffic levels.

- **Stability** — An effective algorithm must not display excessive oscillation, and must converge to some acceptable level of stability while still being sensitive enough to adapt to traffic and topological change efficiently.

- **Fairness** — Within the priorities that exist on a network, the routing algorithm should be fair to all users.

[2]This is, as alluded to previously, a vague notion and lies somewhere on a continuum between adequacy and optimality.

- **Optimality** — A routing algorithm should provide optimal routing by *minimising mean packet delay* and by *maximising total network throughput.*

It is perhaps prudent to add a further routing algorithm characteristic which Pickholtz and McCoy [Pic79] identify. They propose that the provision of a technique for detecting loops and 'ping-ponging' is also an important requirement for an adaptive routing strategy. With these general principles in mind, we now turn to a taxonomy of routing algorithms. Routing algorithms can be broadly classed into two categories and most authors discriminate on the basis of this division.

The first of these categories is distinguished by algorithms which are *non adaptive*, or *deterministic*, in nature. Algorithms of this type calculate routes based on some rule formulated in advance. Such algorithms do not respond to changing traffic or topology.

The second category of algorithms consists of those algorithms which do respond to conditions in the network, being known as *adaptive*, *dynamic* routing algorithms. These algorithms typically gather information about packet delays, queue lengths, line utilisations and the status of nodes and links, subsequently using such information as the basis for routing decisions.

The way in which information is collected by these adaptive algorithms determines a further subdivision into *centralised*, *isolated*, and *distributed* adaptive routing. With centralised adaptive routing, routing decisions are made at a Routing Control Centre (RCC) somewhere in the network. Using isolated adaptive routing, nodes make routing decisions based on information they have collected – without reference to, or communication with, other nodes. Distributed adaptive routing makes use of information gathered from other nodes, and routing decisions are made at individual nodes based on information about the network that they have received and assimilated. A further distinction which is sometimes made under adaptive routing algorithms is the identification of those algorithms which perform *hybrid* routing – a combination of centralised and isolated routing.

When networks become very large, nodal delay tables can consume a lot of memory at each node, while also requiring a lot of CPU time for scanning. A further problem is that as network size increases, so too does the bandwidth needed to exchange status

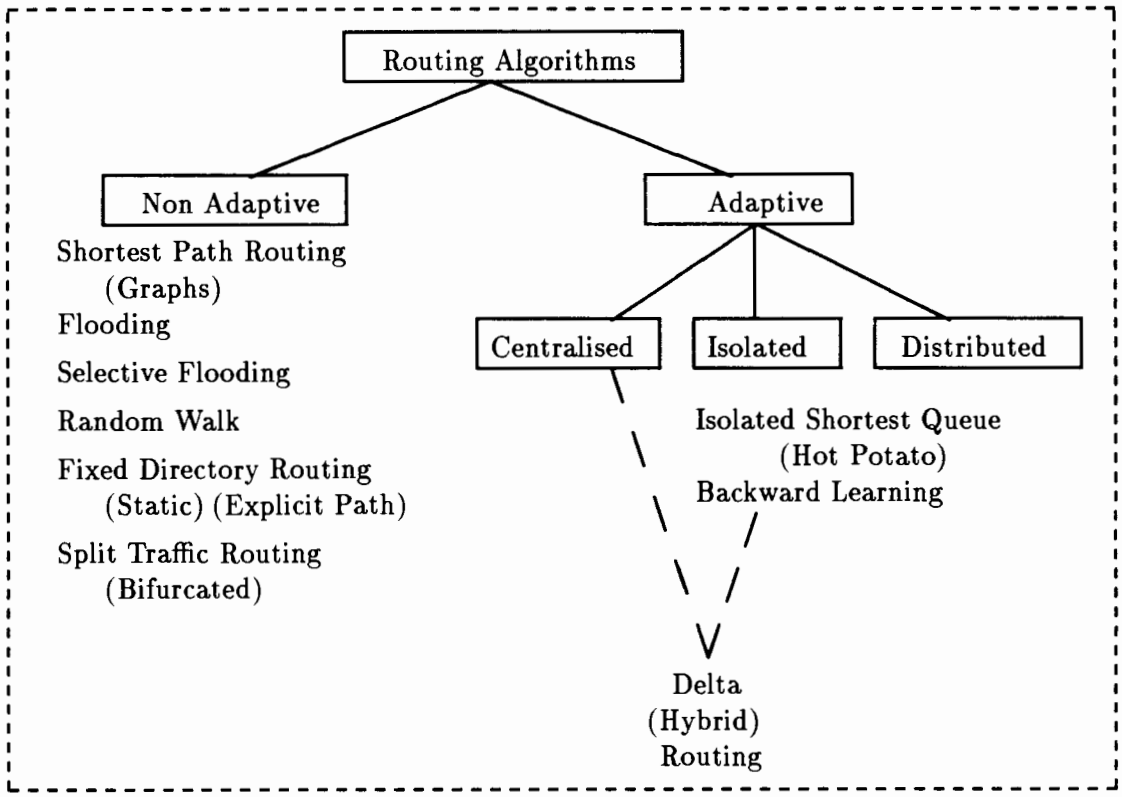


Figure 2: Routing algorithm taxonomy

reports. For this reason *hierarchical* routing has been proposed as a means of dividing large networks into *regions* and routing packets first to the appropriate region and then to the specific node. Kamoun and Kleinrock have investigated the problem of finding an optimal *clustering* structure [Kam77], and examined the performance of large networks with hierarchical routing [Kam79]. Work is still being done on the general problem of routing table size vs. efficiency, and this topic was examined more recently by Peleg and Upfal [Pel89].

1.3.2 Examples

Having identified broad categories of routing algorithm we now consider examples of each routing strategy, noting the specific routing philosophies of these various algorithms. Figure 2 shows how these algorithms fall into an overall taxonomy of

routing algorithms.

We begin by elaborating on some of the *non adaptive* routing algorithms which are used:

- **Shortest Path Routing:** To implement this method of routing a graph is constructed from which shortest paths are determined using number of hops, geographic distance or queueing delay on each arc as the cost measure. Davies [Dav79] correctly points out that a set of shortest path routing tables does not necessarily spread traffic evenly over the network, and Bertsekas and Gallager [Ber87] identify that shortest path routing may cause low throughput, poor response to traffic congestion and oscillatory behaviour.

- **Flooding:** When routing using this technique each incoming packet is sent out on every outgoing line except the one it arrives on. This is a very primitive routing algorithm, but a very robust one. The generation of vast numbers of duplicate packets has to be dealt with. It does have the advantage of a guaranteed shortest path and may be useful as a metric.

- **Selective Flooding:** This is a variation on flood routing in which packets are forwarded only on lines going in the general direction of the packet's destination node.

- **Random Walk:** This routing method consists of the random choice of an outgoing line at intermediate nodes, with packets being forward on the particular selected line.

- **Fixed Directory Routing:** This is also referred to as *static* or *explicit path* routing. In this scheme nodes keep tables specifying the best, second best, etc. outgoing lines for a particular destination. Before forwarding a packet, a node generates a random number and chooses among the alternatives using the weights as probabilities.

- **Split Traffic Routing:** This type of routing is sometimes referred to as *bifurcated* routing, and the philosophy behind this type of routing is that better

network performance can be obtained by splitting traffic over several paths to reduce the load on each of the communication lines.

We now turn our attention to some of the *adaptive* routing algorithms:

- **Isolated:** Isolated routing occurs when routing decisions are taken based on local information only.
  - **Isolated Shortest Queue:** This algorithm is also known as the *hot potato algorithm* since outgoing packets are placed on the shortest output queue without regard for where the line leads to.
  - **Backward Learning:** In this routing scheme delays are learnt from arriving packets in which a counter keeps track of the number of hops the packet has made. If a packet arrives on a route from a certain source with fewer hops than the route on which packets *to* that destination are being routed, then the route changes. This algorithm runs into problems if a line goes down or becomes overloaded, and tables have to be purged regularly.
- **Distributed:** Routing decisions are distributed and take place asynchronously at each node in the network according to this routing scheme. Routing tables are kept at each node of the network and information regarding the status of the node is transmitted to neighbouring nodes. In this way information trickles through the network.
- **Centralised:** When centralised routing occurs routing tables are generated by a Routing Control Centre (RCC). The nature of this routing technique makes it very susceptible to failures of the RCC, as well as consuming a great deal of bandwidth through having to regularly distribute routing tables to all network nodes.
- **Delta Routing:** This is also known as *hybrid* routing, and it is a combination of centralised and isolated routing. The technique, proposed by Rudin [Rud76], consists of *isolated* nodes collecting information which is periodically transmitted to a *central* RCC. Changing the value of *delta* determines how much influence the RCC has in routing decisions.

1.3.3 Flow Control and Congestion

Flow control is applied in situations where the offered load is greater than the receiver can cope with. In some senses it shifts delay from the network layer to higher layers in an attempt to prevent data from entering the subnet faster than it can be accommodated. In this view flow control is an end to end phenomenon. The objective of flow control is to restrict subnet entry while still keeping delays reasonable.

Various schemes for effecting flow control have been proposed, among these *isarithmic flow control* whereby the number of packets in the subnet is limited, and *window flow control* where an upper bound (window size) exists specifying the number of unacknowledged packets that can be sent.

Congestion occurs when packets arriving at a node demand more buffer space than is available. Various strategies for preventing congestion exist, with the intention of preventing *deadlock* – congestion developed to its fullest extent. A recent discussion of congestion problems and solutions is provided in [Jai90].

Chapter 2

Distributed Nodal Network (DNN): Hardware

The Network Processing Units (NPUs) in the WAN under study (Figure 3) are based on the use of the industry standard IEEE 1296 Parallel System Bus. One of the most significant features of this bus is that it allows the independent operation of several autonomous processors. It provides a bandwidth of 40Mbytes per second.

The configuration of the NPUs, with the Parallel System Bus providing connectivity, is as follows:

A *Peripheral Processor* is present with the function of providing peripheral functions such as system diagnostics, start-up and monitoring. The peripheral processor also provides general purpose processing capability for editing configuration files, and for system software support. The Peripheral Processor provides the function of monitoring the system, and also of automatically restarting after a failure. A standard Intel 186 based single board computer constitutes the Peripheral Processor in the DNN NPUs.

The MINIX [Tan87] Operating System is employed on the Peripheral Processors, providing editing capability via *mined.*

A *Central Services Module* provides bus control functions such as clock generation, reset control and error monitoring for the Parallel System Bus.

Up to two *Ethernet Processors* can be included in the hardware configuration,

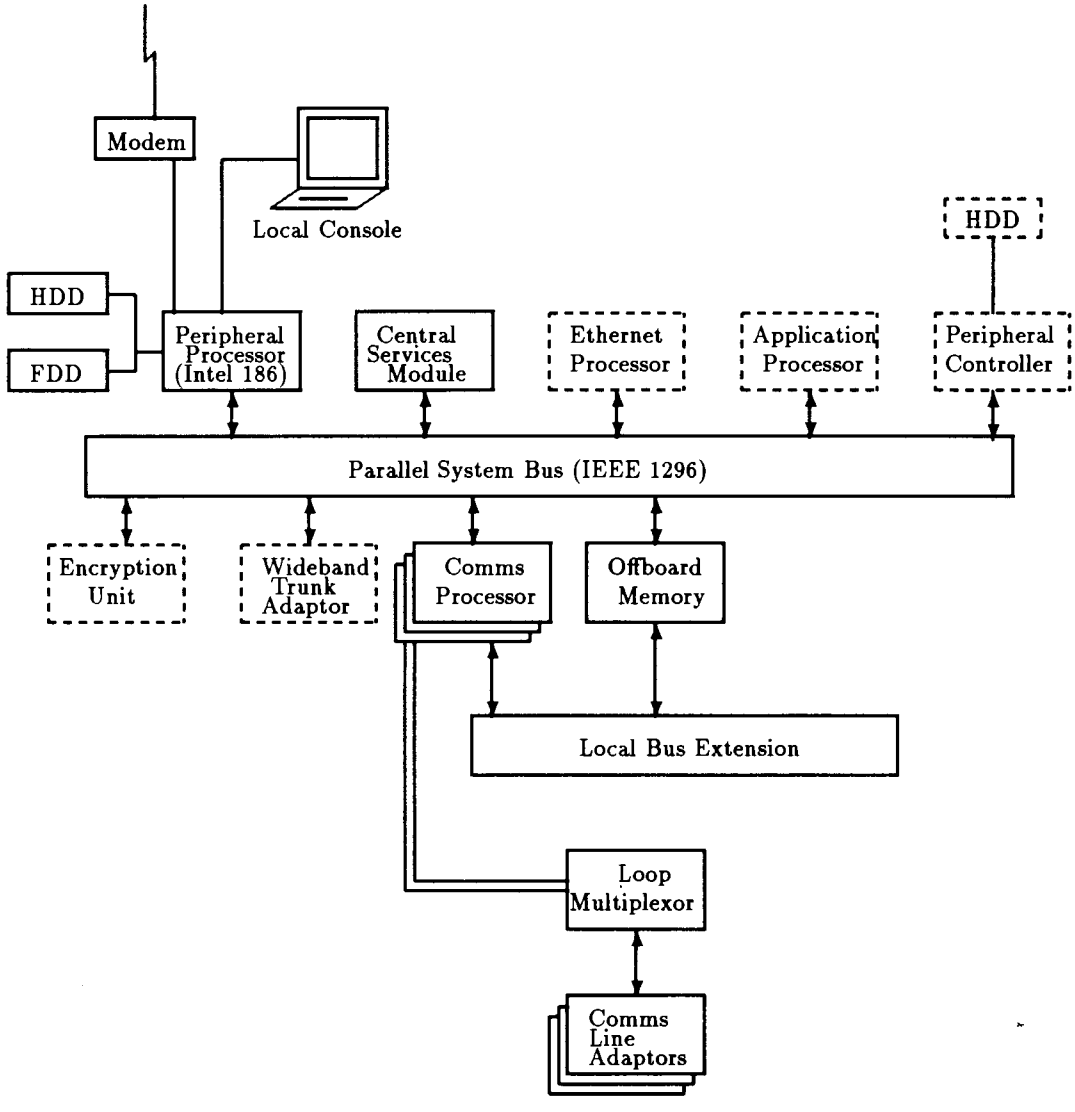


Figure 3: NPU configuration

providing optional Local Area Network (LAN) interfaces. Each Ethernet Processor provides IEEE 802.3 compatible LAN capability and a Parallel System Bus interface with full message passing.

An *Application Processor* is an optional unit for most nodes, but is required for the Network Management Centre node in large networks. In such situations it is used for on-line network management functions. In smaller networks an 80386 PC could be used as a substitute for the Application Processor. The Application Processor may also be used for network resident, value added facilities such as e-mail for customer specific application systems.

A *Peripheral Controller Unit* provides I/O capability, and allows the Application Processor to access SCSI peripherals. On most nodes this unit is optional, but if an Application Processor is present, then it is a requirement. An onboard 16MHz, 386 CPU with 1 Mbyte of onboard memory constitutes the Peripheral Controller Unit.

Up to four *Communications Processors* can be configured into each NPU, depending on the required throughput. The Communications Processor is a micro-programmable high-speed processor. Due to the fact that it is micro-programmable, it is capable of performing many operations in parallel. The microcode resides in 16K words of Writable Control Store (WCS) and each word is 96 bits wide. WCS access is 45 nanoseconds. The designers of the system have found the micro instruction set particularly well suited to data communications with, for example, the inclusion of Cyclic Redundancy Check computations.

There are three Printed Circuit Boards (PCBs) which constitute the Communications Processor.

- The *Memory Interface* provides memory management and access to off-board memory via the Parallel System Bus or a Local Bus Extension.

- The *Arithmetic and Logic Unit* provides the main processor circuitry of the Communications Processor. Separate arithmetic and multiplication units, an instruction sequencer, an interrupt controller and macro instruction decoding hardware make up the Arithmetic and Logic Unit.

- The *Serial Loop Interface* is the main data communications controller, providing

access to the communications subsystem via a high speed (16MHz) serial fibre optic loop.

The Communications Processor together with one or more Loop Multiplexors and a number of Communication Line Adapters are used to provide the data communications capability of the NPU.

The fibre optic loop from the Serial Loop Interface connects to one or more *Loop Multiplexers.* Loop Multiplexers provide the interface between the fibre optic serial loop and the Communications Line Adapters. Each Loop Multiplexor can accommodate up to 8 Communication Line Adapters.

The *Communication Line Adapters* provide an intelligent interface between the NPU and the external data communications lines. Each card contains 8 ports. Different interface cards are available to support RS232, X.21 and V.35 electrical interfaces.

Off-board memory is used as a memory extension which can be configured as a common memory module accessed by two Communication Processors. Off-board memory can be provided as $\frac{1}{2}$, 1, 2 or 4 Mbytes of RAM. The Communications Processor normally uses the *Local Bus Extension* interface which significantly reduces contention on the Parallel System Bus. Cache memory of 8 Kbytes is configured for each unit.

An optional *Encryption Unit* can be included for security of data.

A *Wideband Trunk Adaptor* is an optional unit which will allow operation of four ports at sustained speeds of up to 2048 Mbps.

Chapter 3

Distributed Nodal Network (DNN): Software

The DNN software is organised in layers which approximate those of the ISO OSI model, but different terminology is used. Figure 4 shows the relationship between the ISO OSI model and the DNN model with the DNN terminology which is used.

A software *Trunk Control Module* is responsible for all DNN trunk transmissions. The primary function of the Trunk Control Module is to provide an error-free transport mechanism for the *Network Access Module* to transport packets from source to destination. The Trunk Control Module is concerned with packet transmission, including control of the trunk lines, trunk protocol and error recovery, as well as the adaptive routing scheme which controls packet flow. The handling of packets on an end-to-end basis is a higher level function, performed by the Network Access Module.

At the Network Access Module level, the X.25 protocol is used to control packet transmission between source and destination nodes. At this level packet sequence numbers are used to resequence packets that may have arrived out of sequence. Large, 15 bit, sequence numbers are used since the number of packets in transit may not exceed the maximum sequence number. The use of a 15 bit sequence number makes the network suitable for satellite and other wideband communications too, where propagation delays warrant having large numbers of unacknowledged packets in the

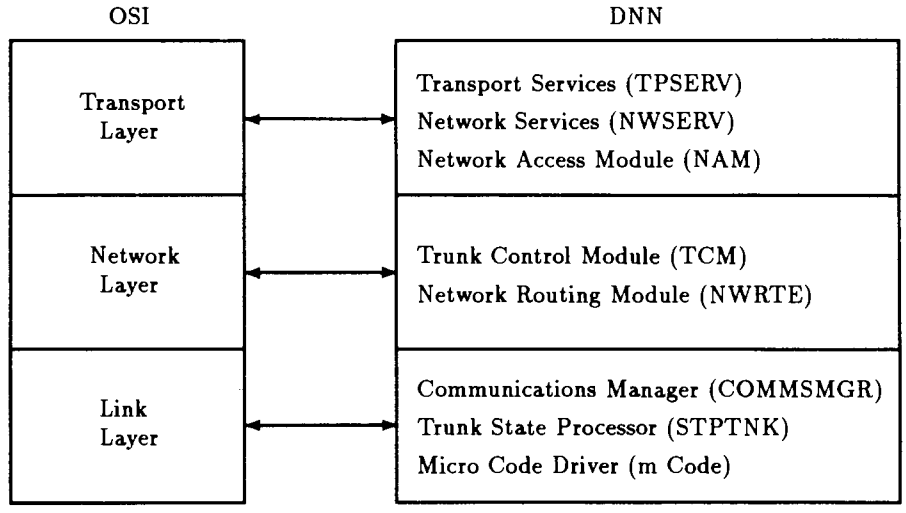


Figure 4: Relationship between DNN layers and ISO OSI model layers

network. In order to deliver packets in the correct sequence for processing, out-of-sequence packets are held until the missing packets are received, and only then will packets be released for processing.

There is a priority facility in the Network Access Module, and priority packets are always processed first. It is also this module which prevents internal queue build up by instituting flow control procedures when necessary. The Network Access Module forwards packets to the Trunk Control Module for transmission. The Network Access Module also forwards received in-sequence packets to higher network levels.

This discussion of the network software used in the DNN will be orientated around the Trunk Control Module (OSI Network) layer, as it is with the routing aspect of the network that this study is primarily concerned. This discussion should be seen in the context of an X.25 like logical connection between source and destination nodes at the Network Access Module (OSI Transport) layer.

At the network layer a *Network Routing Module* exists. An adaptive routing technique is utilised, and a datagram approach facilitates this. Messages are broken down into packets, and these smaller units are routed through the network. In the DNN packets are 128 bytes, and the length of a packet in the network is given in units of 16 bytes, known as Packet Length Units (PLUs).

Routing tables (or delay vectors) exist at each node of the network, and it is on

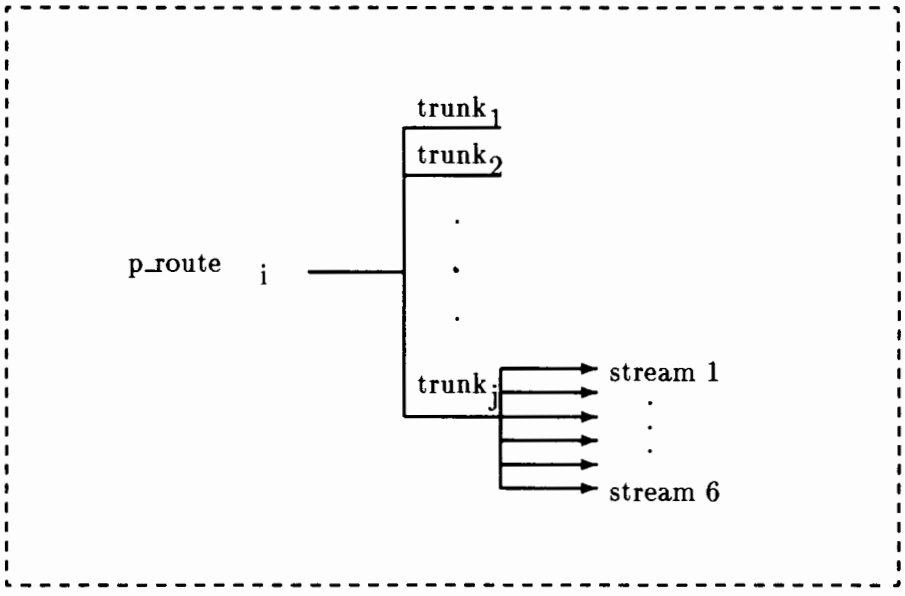


Figure 5: DNN physical route structure

the basis of information in these tables that routing is carried out. Tables are updated asynchronously, and delay information is transmitted to adjacent nodes. In this way knowledge of a node's status ripples to all other nodes in the network.

3.1 Physical Routes

Nodes are said to be connected to other nodes on different physical routes (*p_routes).* If a node is linked to n other nodes then there are n *p_routes* emanating from that node. In this sense *p_routes* exist from Node A to adjacent Node B, and from Node A to adjacent Node C. Each route consists of a number of physical *trunks*, and the DNN uses multiple physical trunks to constitute a *p_route.* These trunks are divided into logical *streams*, and when examining the scheduling of packets queued on a *p_route*, a fuller explanation of the stream implementation will be given. At this point it suffices to say that there are six logical streams on each trunk.

3.2 Logical Routes

A logical route (*l_route*) can be described as a virtual connection between any two nodes, through the network. An *l_route* is analogous to a path between nodes. This means that, although there may be only p *p_routes* emanating from a node A, there may exist $n-1$ logical routes from node A for an n node network.

3.3 Physical Route Service Time

In order to calculate delays in the network, *p_route* service times are required. The *p_route* service time is influenced mainly by the *capacities* of the trunks constituting the *p_route*. The *p_route* service time is also affected by trunk error rates, and these have to be considered in calculating a *p_route* service time which accurately reflects current delay on a *p_route*. The following formula shows how transmission error is incorporated into the calculation of Working Trunk Speed.

- Let C be defined as the trunk capacity derived when a trunk is first sensed (worked out implicitly during the sending of the startup message on that trunk).
- Let u be the count of unsuccessful, and s the count of successful transmissions on the trunk. We define $\alpha = \frac{u}{s}$. This is the maximum allowable error ratio and is set to $x\%$ so that the moment $\alpha > x\%$ the trunk is declared unusable.

Then

$$\textit{Working Trunk Speed} = C \times (1 - \alpha) \tag{1}$$

The ***badness probability*** is the probability that a transmission will result in an error, and that a packet will have to be retransmitted. The badness probability should be 1 to 2 percent for a normal running network.

Working Trunk Speed is stored in *characters per second* (i.e. bytes per second). For a b bps line, the trunk speed would thus be represented as $\frac{b}{8}$ characters per second. As soon as retransmissions are picked up, the effective data rate is slowed down and the Working Trunk Speed calculation will reflect this. The delay processor looks at

the Working Trunk Speed when calculating delays, and will thus take the decreased performance into account when recalculating the trunk and *p_route* service times.

After calculating the Working Trunk Speed for each of the t trunks, a total *p_route* speed in characters per second is gathered for the *p_route*:

$$P\_Route\ Speed = \sum_{i=1}^{t} Working\ Trunk\ Speed_i \tag{2}$$

We can now calculate the *P_Route* Service Time in Packet Length Units (PLUs) per second.

$$P\_Route\ Service\ Time\ (PLUs/sec) = \frac{P\_Route\ Speed}{16} \tag{3}$$

Division by the number of bytes in a PLU allows conversion of the *P_Route* Service Time to PLUs per second. To find the time, in milliseconds to transmit one PLU we divide 10^3 (the number of milliseconds in a second) by the *P_Route* Service Time in PLUs per second. This amount in ms/PLU is then the time taken to transmit one PLU.

Assuming there are t trunks with capacity C_i on a particular *p_route*, the final calculation is:

$$P\_Route\ Service\ Time\ (ms/PLU) = \frac{10^3}{\frac{\sum_{i=1}^{t} C_i \times (1-\alpha_i)}{8} \times \frac{1}{16}} \tag{4}$$

3.4 Buffers

When messages are broken down into packets, these packets are stored in buffers. In this section we consider the allocation and arrangement of buffers in memory.

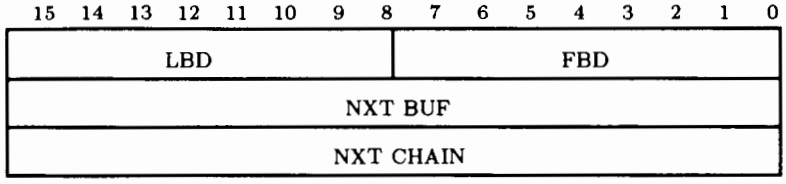
3.4.1 Structure

Depending on the configuration of the Nodal Processing Unit, there can be up to 4 MBytes of offboard memory. This memory is divided into big and small buffers. Big buffers are 64 bytes in size, and small buffers are 32 bytes. Packets are stored in 64 byte buffers, with enough buffers being requested to accommodate the amount of

data in the packet. The buffers are chained together in memory. There are 3 control words (occupying 6 bytes) at the start of every buffer.

Packet headers are stored in small buffers, and the buffer next chain pointer points to the next packet header buffer (or zero in the case of the last packet).

The structure of a big buffer is as follows:



LBD: Last byte displacement

FBD: First byte displacement

NXT BUF: ID (address) of next buffer in chain

NXT CHAIN: Points to next packet header when buffers are joined with small buffer Packet Headers for transmission.

3.4.2 Buffer Regulation Threshold

When packets are stored in the buffers of a node, the node's available buffer space is decreased. A buffer threshold exists specifying the amount of memory which should remain available at each node in the network. If the available buffer space goes below this threshold (expressed as a percentage of the available buffer space) then recovery action must take place. This action is initiated by a delay table update, and the announcement and effect of this *regulation* will be described when considering the delay tables.

3.5 Packets

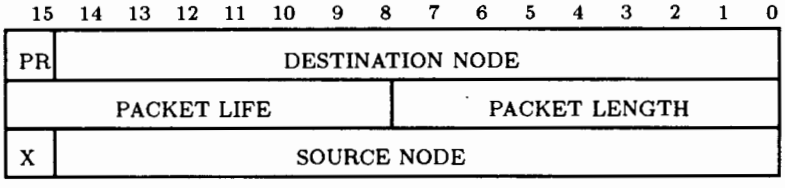
The nodal buffers are used for storing packets during a *packetize* operation. A packet remains in the buffers of a node until an *acknowledgement* has been received. A *packet*

life is associated with every packet to prevent eternal packet looping and in order to initiate retransmissions. In the following sections these are described.

3.5.1 Packetizing

The DNN makes use of the software *Network Access Module* to packetize a Transport Data Unit (message). The packetize operation merely adds a packet header (small) buffer to the front of the data buffer chain, and then interposes intermediate packet header buffers between every two data (big) buffers. The packet headers are chained together by means of a forward pointer in the buffer (NXT CHN).

The 'packetize' instruction adds a 6 byte packet header containing:



PR: Priority bit / flag

Destination Node: The destination node is set to zero in control packets. Control packets only travel between adjacent neighbours.

Packet life: This is stored as the number of 200 millisecond units.

Packet length: This is stored as the (rounded up) number of 16 byte Packet Length Units (PLUs) in the packet.

X: Reserved

Source node: Originating node of this packet

The packet header comprises a small buffer which links to the big buffers that constitute the packet. The packet header is also stored in memory at the source node until the ACK is received. It is from the packet life stored in the packet header, and kept in the source buffer, that it can be determined whether the ACK arrived back

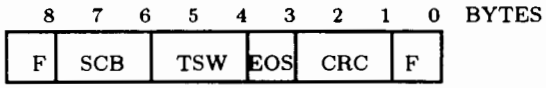
within the packet's lifetime. The packet life in the header is decremented by 5 every second.

Packets in DNN are a relatively short 128 (2^7) bytes, allowing more frequent breaks for ACK transmission across communication lines because of this short package length. Each packet consists of 8 PLUs.

3.5.2 Acknowledgements

Before discussing the use of Acknowledgements in the DNN their structure, and that of their subcomponents, is described.

The structure of an Acknowledgement (ACK) is as follows:



F: Opening / Closing Flag

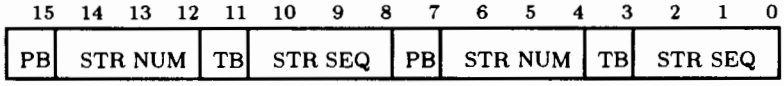
SCB: Stream Control Bytes

TSW: Trunk Status Word (included if the TB in the SCB is set)

EOS: End of Stream

CRC: Cyclic Redundancy Check

There are two Stream Control Bytes in the Acknowledgement, and these are mirror images of each other. The Stream Control Bytes are structured in the following way:



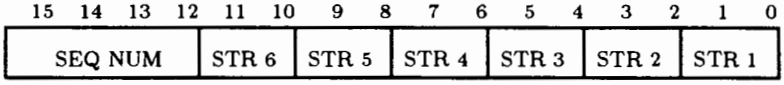
PB: Parity Bit

STR NUM: Stream Number

TB: Trunk Status Word Present Bit

STR SEQ: Stream Sequence Number

The structure of the Trunk Status Word is as follows:



SEQ NUM: Sequence number between 0 and F

STR 1 - 6: Streams 1 - 6, two bits per stream consisting of a TFLAG toggle bit (toggled when something different is being sent permitting discrimination between new and old AFLAG information) and a second bit, the AFLAG (acknowledge flag), which is zero if it is an ACK, and 1 if a NAK.

When starting the network, everything in the Trunk Status Word is zero. When status for a stream is gained, the toggle bit for that stream is set.

For every complete packet received from an adjacent node on a stream, the AFLAG associated with that stream must be set to ACK or NAK to indicate acceptance or rejection, respectively, of the received packet.

The TFLAG must be toggled, and the TB set in the Stream Control Byte. The Trunk Status Word is transmitted to the adjacent node, where it is determined which TFLAGS have changed, identifying streams for which responses have arrived.

As soon as a packet has been transmitted from a sending node, it is placed on an *unacknowledged packet chain* where it remains, until acknowledgement is received that it has arrived correctly at the receiving node.

Acknowledgements are sent back for each packet. ACKs have priority status for sending. The internal delay at a source node is reduced upon receiving an ACK from the next node on the route. If the node receiving the ACK is not the source node, packets are dropped from the node buffers and the amount of memory available is incremented. All trunks are full duplex, so if there is no traffic coming back along a trunk we can assume (on a 9600 bps line) that an ACK will come back in 9 or 10 milliseconds. On a 64 Kbps line, an ACK will return in about 1 millisecond, if there is no other traffic on the line. If there is other traffic on the line, the ACK can be *piggybacked*, maintaining very fast response on a loaded link.

ACKs can be used as a means of flow control and piggybacked packet ACKs can be suspended for through traffic. The Trunk Status Word, which runs at trunk level once every second, will carry an ACK if it has been delayed, so the longest an ACK can be delayed is 1 second.

At this juncture we need to distinguish between a trunk ACK and a higher level destination–source ACK.

On receiving a packet at an *intermediate node*, and having verified by means of a cyclic redundancy check (CRC) that the packet has been received correctly, a trunk ACK is sent to the preceding, forwarding node. If the preceding, forwarding node is not the source node, the node will then drop the packet from its buffers.

If I_1 forwards a packet from S to I_2, the packet will only remain in the buffers of I_1 until the trunk ACK is received from I_2. S (the source) will not drop the packet from its buffers until it receives a higher level ACK from D (the destination). At the higher level, there is an X.25 window mechanism between source and destination providing *window flow control.* It is at this level that packets are inserted into or removed from buffers at the source and destination.

3.5.3 Packet Life

Each packet is given a packet life to prevent eternal looping through the network, or the 'ping-pong' effect of packets moving backwards and forwards between nodes. At each node traversed packet life is decreased, until finally packet life expires and the packet is discarded.

Packet life is adjusted dynamically by transmitting nodes, and is halved for new packets every time the ACK for transmitted packets arrives back within the packet life. This continues until a suitable figure for packet life is arrived at. After a time-out at a node, the timed-out packet is retransmitted with packet life reset to the maximum configured value.

At the source node, packet life is decremented every second and if it goes below zero before receiving an ACK, the packet will be retransmitted. There is an 'unacknowledged' queue at the source node, and it is on this queue that packets have their packet life decremented.

3.6 Tables

It has been stated that there are routing tables at each node, and indeed these tables form the kernel of the routing mechanism. The routing tables occupy some of the memory of each node, but there are other tables which also reside in memory at each node.

Amongst these other tables at each node are those which keep track of the routes, trunks and streams at that particular node. After loading the network software into memory, the first thing that happens at the node is that space is reserved in memory for tables. The amount of memory allocated is determined by the parameters in a configuration file which exists for each node.

To co-ordinate all of the tables in memory, a Table Pointer Vector exists at each node, with entries specifying:

- the first word address of each table
- the number of entries in the particular table
- the size of each entry in the table (this is kept mainly for system integrity and protection against different versions, programmer error etc.)

3.6.1 Delay Tables

At each node there are two delay tables (see Figure 6). There is a *received* (RX) delay table containing delay information received on its various *p_routes*, as well as a *transmitted* (TX) delay table which is built up for transmission to adjacent nodes. The delay vector transmitted on a *p_route* is a one dimensional vector containing the average of the delays to different destinations from this node on that particular

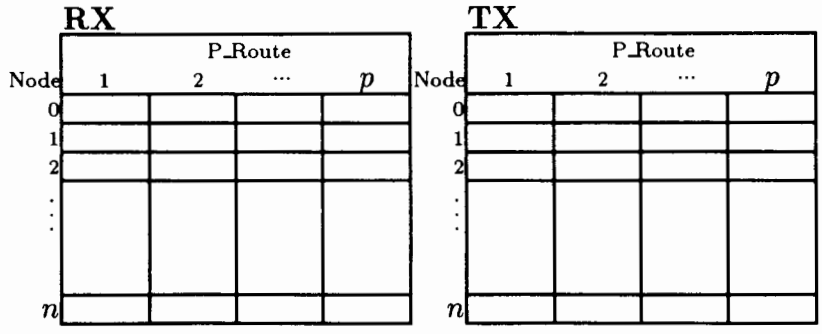


Figure 6: Delay tables for Node i

p_route. If we view the complete transmitted delay vector as a two dimensional array, then only one column of the array is sent on any particular *p_route*. The column that will be transmitted on a *p_route* is the one corresponding to that particular *p_route*.

Bodde [Bod81] describes how a 'directional approach' to routing involves each node viewing its *L_routes* as convergent or divergent with respect to any other given node. There will be decreasing delay table values along a convergent route, and in the DNN the designers refer to 'conceptual arrows' on each route, indicating the direction of the route with respect to a particular node.

Four classes of *L_route* exist in the delay tables:

1. An *L_route* can be *divergent* to a particular node (i.e it is not to be taken into consideration, since the destination along this *L_route* has an infinite delay), in which case it is represented by FFFF. All *L_routes* start off as FFFF.

2. The *L_route* can be to an *adjacent node*, in which case the network delay is considered to be zero, and is represented as 0.

3. An *L_route* can be *convergent* to a particular node (i.e there is a finite delay to the destination along this *L_route*), in which case it is represented by a positive delay value, below some maximum convergent value. This maximum convergent value (which is 3FFF) exists so that addition of delays will not result in negative value delays due to overflow.

4. An *L_route* can be *congested* if, in terms of routing, delay exceeds a certain threshold. In this case a value of 7FFF is used for *regulation*. By setting the

delay to this value, the node will effectively be ignored as a possible route for other nodes. Regulation originates at the buffer level. If a node's available buffer space goes below b% (a configuration specified percentage/footnoteThis threshold is set to 30% in the DNN.) then that node's network delays are all set to 7FFF.

If a node goes into regulation, it will not accept any more input on trunks (*endogenous* arrivals), and it will simply discard any packets from other nodes. It will still however accept information from external sources connected to this particular node (*exogenous* arrivals).

In the case of endogenous arrivals, no NAKs will be sent to the sending node, since on receipt of a NAK the sending node will immediately resend the packet on which it is awaiting acknowledgement. By just throwing away the received packet, the sending stream will time-out and the packet will be re-routed, hopefully via another *l_route*. A node in regulation will still accept control packets, but it will not accept any through traffic.

The events at the sending node, when no ACKs are received, are as follows:

- send packet: start timer
- after 3 seconds: timeout
- send packet again: re-start timer
- after 3 seconds: timeout
- dequeue packet
- pass packet back to router
- if packet life has not expired: re-route

Before going into regulation, the node will slow down its ACKs by disabling piggybacked ACKs and only sending ACKs once a second with the Trunk Status Word that is transmitted on a trunk every second. The slowing down of ACKs in this way represents the first level of recovery action, while entering the state of regulation

constitutes the second, more drastic, level. Higher level flow control should come into effect long before these lower level recovery techniques become necessary.

Whether to make the delay table comprise of the minimum (best route), maximum or average delays is a design issue, but in the DNN an *average* value is used. On this issue Bodde reports that "The effect is that at distant nodes, delay tables are inaccurate, but a fair approximation. As the packet gets closer to its destination the decisions become more accurate" [Bod80, p. 14].

Internal delay: This is the delay on a particular *p_route* attributable to packets already queued on the *p_route*, and the *p_route* service time. It is calculated as the product of these two quantities and represents PLUs/ms.

The internal delay at a node is stored in row 0 of the RX table, and reflects the current utilisation of particular *p_routes*, because if a large number of packets are queued on a particular *p_route*, any new traffic will have to wait for those packets to be transmitted before being sent. In calculating the best *L_route* to use, a node will add the internal delay on a *p_route* to the network delay and then arrive at an overall delay factor.

3.6.2 *P_Route* and Trunk Tables

The existence of each trunk on a *p_route* is recorded in a trunk table, and a chain of trunks is kept by the *p_route* table. As the presence of a trunk is detected, an entry will be made in the *p_route* table (if it is the first trunk on a *p_route*), and in the trunk table. The trunk will be added to a chain of trunks kept for each *p_route* in the *p_route* table.

If a trunk goes down, the trunk table is updated, and the trunk is flagged as being off-line. It is also necessary to update the *p_route* table and delink the trunk from the chain of trunks on the *p_route*. The *P_Route* Service Time must also be recalculated.

If the trunk is the only trunk on the *p_route* (or all the other trunks on the *p_route* have also gone down) then *p_route* table maintenance will also be necessary.

The DNN delays telling the higher (session) level that a *p_route* is down for *s*

seconds[1], in the hope that the circuit will come up again.

The table updates that occur on losing a *p_route*, are:

1. In the *p_route* table, information is zeroed for the *p_route* which has been lost,
2. In the transmitted (TX) delay table all nodes on that *p_route* are flagged as being divergent (FFFF).

3.6.3 Exchange of Delay Tables

Delay tables are exchanged between nodes under certain conditions:

1. When trunks come up or go down. Such conditions will trigger the immediate sending of delays by all nodes.
2. Every 5 seconds (on a 'quiet' network)
3. When there is a direction change in the RX delay vector. (i.e the route to a node has changed from convergence to divergence or vice-versa.)
4. When there is an internal delay swing of more than p percentage change[2].

Tables are not actually transmitted on a *p_route* unless the *transmit bit* on that particular *p_route* is set. This bit will be set the moment anything changes on that *p_route* in the routing table. In this way unnecessary network load is reduced, since redundant tables which are the same as those previously sent will not be transmitted because their transmit bits will not have been set.

The transmitted (TX) delay table is built, starting from *p_route* 1 and working through to *p_route p*. In this manner, columns of the table are built up. A more detailed explanation follows later.

It has been said that the network reflects average delay, and delays trickle through the network. Along a particular *l_route* Node i's information will reach Node $(i+1)$

[1]This is typically 15 seconds in the DNN

[2]This percentage is currently set to 25% in the DNN

after 5 seconds, however it will be 10 seconds before the information reaches Node $(i+2)$.

Tables are sent asynchronously, and travel as priority packets to their neighbours. Effectively they are put at the front of the priority queue, and are routed to adjacent nodes. If there is an earlier delay table still at the front of the priority queue, the later table overwrites the earlier one and supersedes it.

3.6.4 How the Delay Tables are Updated

We need to look at two cases for updating:

- How a received delay vector is interpreted.
- How the delay vectors for transmission on each *p_route* are calculated.

3.6.4.1 Receiving a Delay Vector on a *P_Route*

At the receiving node, for each *l_route*:

- First look at the last transmitted delay value
- If it was DIVERGENT (FFFF), then accept new delay value
- If the last transmitted delay value was NOT divergent then:
  - if the new delay value is divergent, make delay divergent (FFFF)
  - if the new delay is NOT divergent
    - if the sending node is adjacent (delay value is 0), make delay 0
    - if the new delay is *regulation*, make received delay *regulation*
    - if the new delay value < the last transmitted value, make delay the new delay value
    - if the new delay value = the last transmitted value then:
      - if the sending node number > receiving node number, use the new delay value

- if the sending node number < receiving node number, force divergence (FFFF) /* in this way 'conceptual arrows' are placed on the network */

* if the new delay value > the last transmitted value, force divergence (FFFF)

3.6.4.2 Transmitting a Delay Vector

In order to transmit delay vectors, the internal delay vector on each *p_route* has to be updated.

For each *p_route* we calculate the *P_Route* Service Time (as described in Section 3.3). To calculate the internal *p_route* delay, we multiply the time it will currently take to transmit one PLU (the *P_Route Service Time*) by the total number of PLUs queued on the particular *p_route*. This information is then used to update the internal delay vector on the particular *p_route*.

When calculating the delay values to send to other nodes, the internal delay is added to the external delay using the following algorithm.

Procedure for building TX table at Node i:

For each *l_route* from Node i:

- Sum the convergent *p_routes* only, and add the internal delay on all *p_routes*.
- For entering the actual value for each *p_route* of an *l_route* in the TX table:
  - exclude the *p_route* that is being dealt with (by subtracting the external and internal delays on that *p_route*)
  - divide by the number of convergent *p_routes* (less the one we are dealing with) UNLESS
    * If external RX delay is 0, place FFFF in TX.
    * If external RX delay is 7FFF or FFFF then place the average delay over all (convergent) *p_routes* in TX.
- Delay to Node i is 0 (since *p_routes* connect to an adjacent neighbour).

In this way the TX table is built up. Once values have been calculated, they are checked against the existing values in the TX table. The transmit bit for a *p_route* is only set when a value in that column (*p_route*) changes in the TX table. On completion of calculation, the columns to be transmitted are moved into buffers, the router is notified and told that there are control packets to be sent. The *p_route* information is only sent out on the *p_route* which has changed. If *p_route p* has changed in the TX table, then the column in the delay vector applicable to *p_route p* gets sent on *p_route p* to the particular neighbour who is on that route.

3.6.5 Delay Table Update: Worked Examples

3.6.5.1 Worked Example 1

Figure 7 shows the calculated delay table for transmission (TX) built up using the external and internal delays held in the received (RX) delay table. The values in Row 0 of the RX table represent the internal delay on each respective *p_route*.

In this example the tables are those of Node 3.

RX

Node	P_Route 1	2	3
0	5	6	7
1	0	8	20
2	10	0	22
3	FFFF	FFFF	FFFF
4	12	7FFF	24
5	14	FFFF	0

TX

Node	P_Route 1	2	3
0			
1	FFFF	16	10
2	18	FFFF	11
3	0	0	0
4	31	24	17
5	7	13	FFFF

Figure 7: Transmitted delay table calculation at Node 3

3.6.5.2 Worked Example 2

This example illustrates the initialisation of the tables at different nodes as the network starts up. The network will eventually consist of 3 nodes as shown in Figure 8. The tables comprise Figures 9 to 16.

Initially Nodes 1 and 2 will come up, and later Node 3 will be added with the trunk from Node 1 to Node 3 coming up first, and the trunk from Node 2 to Node 3

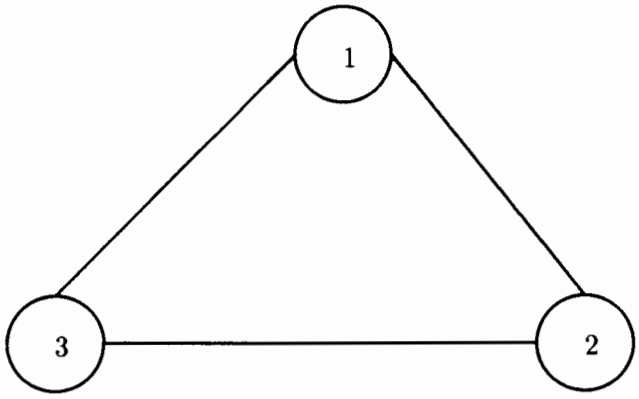


Figure 8: Three node network used in Worked Example 2

following.

All entries in the received and transmitted delay tables at each node are FFFF initially (Figure 9).

Node 1 detects a *p_route* to Node 2, after receiving Node 2's Trunk Status Word (TSW). Nodes transmit TSWs every second, so as soon as Node 2 came up it would have received Node 1's TSW, and it would have started sending TSWs on all trunks.

Assuming there is nothing queued on this route from Node 1 to Node 2, Node 1 updates its internal delay to the time for one PLU to be transmitted on that *p_route*.

Node 1 builds its delay vector for transmission on Route 1. The delay to the sending node is zero, so on Route 1 the delay to Node 1 is entered as zero. Because the delay vector on Route 1 has been updated, the transmit bit is set, and column 1 of the transmitted (TX) delay table is sent to Node 2. Figure 10 shows the updated tables.

Node 2 receives the delay table from Node 1, and now goes about updating its received (RX) delay table. Because the previous delay to Node 1 on Route 1 was the initial FFFF, Node 2 replaces FFFF on *p_route* 1 to Node 1 with 0. This value indicates that Node 1 is an adjacent node. Node 2 makes its internal delay the time for one PLU on that *p_route*, as shown in Figure 11.

Node 2 goes about building its delay vector for transmission. Since only one *p_route* exists so far, we will be concerned only with *p_route* 1 (to Node 1).

Figure 12 shows that the delay to Node 2 is 0, since Node 1 is adjacent. The delay

RX

Node	P_Route 1	2	3
0	FFFF	FFFF	FFFF
1	FFFF	FFFF	FFFF
2	FFFF	FFFF	FFFF
3	FFFF	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

TX

Node	P_Route 1	2	3
0	FFFF	FFFF	FFFF
1	FFFF	FFFF	FFFF
2	FFFF	FFFF	FFFF
3	FFFF	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

Figure 9: Initial tables at Node 1

RX

Node	P_Route 1	2	3
0	14	FFFF	FFFF
1	FFFF	FFFF	FFFF
2	0	FFFF	FFFF
3	FFFF	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

TX

Node	P_Route 1	2	3
0	FFFF	FFFF	FFFF
1	0	FFFF	FFFF
2	FFFF	FFFF	FFFF
3	FFFF	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

Figure 10: Tables at Node 1 after detecting Node 2

RX

Node	P_Route 1	2	3
0	14	FFFF	FFFF
1	0	FFFF	FFFF
2	FFFF	FFFF	FFFF
3	FFFF	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

TX

Node	P_Route 1	2	3
0	FFFF	FFFF	FFFF
1	FFFF	FFFF	FFFF
2	FFFF	FFFF	FFFF
3	FFFF	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

Figure 11: Tables at Node 2 after receiving delay vector from Node 1

RX

Node	P_Route 1	2	3
0	14	FFFF	FFFF
1	0	FFFF	FFFF
2	FFFF	FFFF	FFFF
3	FFFF	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

TX

Node	P_Route 1	2	3
0	FFFF	FFFF	FFFF
1	FFFF	FFFF	FFFF
2	0	FFFF	FFFF
3	FFFF	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

Figure 12: Tables at Node 2 before transmitting first delay vector

to Node 1, via Node 2, is reflected as FFFF (divergent) since Node 1 is adjacent.

The transmit bit on column one of the transmitted (TX) delay table is set, and the delay table is transmitted to Node 1.

Node 1 receives the delay vector from Node 2, and updates its received delay table to reflect that there is zero external delay to Node 2, since it is adjacent.

After calculating its transmitted delay table (Figure 13), there are no changes to *p_route* 1's delay information. The transmit bit is not set, so no new information is sent to Node 2.

Node 3 comes up, and Node 1 receives a TSW from Node 3. It updates *p_route* 2 (to Node 3) to reflect a delay of zero to Node 3, on *p_route* 2, and records the time for one PLU to be transmitted, for the internal delay on *p_route* 2.

The transmitted delay table is updated (Figure 14) to reflect zero external delay to Node 1, from Node 2 on *p_route* 1 and Node 3 on *p_route* 2, and to show a delay of 14 to Node 2 on *p_route* 2 (from 3), and a delay of 14 to Node 3 on *p_route* 1 (from 2).

Because both *p_routes* in the transmitted delay table have changed, transmit bits are set for columns 1 and 2 and the columns are sent on *p_routes* 1 and 2 respectively.

Node 2 receives this information regarding a route to Node 3 via Node 1, and updates its received delay table accordingly (Figure 15). The delay value to Node 3 on *p_route* 1 was previously FFFF, so the received value of 14 is accepted and replaces the divergence indicator.

Node 2 now builds its transmitted delay table, but reflects the delay to Node 3 as FFFF. This is because the *p_route* to Node 3 is divergent via Node 2. This information must be passed to Node 1 (on *p_route* 1 from Node 2).

A trunk now comes up between Nodes 2 and 3. Node 2 now opens a second *p_route* going to Node 3. After updating its received delay table (Figure 16), it reflects the new *p_route* in its delay vector for Node 1 (on *p_route* 1). Node 2 also tells Node 3 that it has a convergent *l_route* to Node 1.

These new delay vectors are transmitted on their respective *p_routes*.

RX

Node	P_Route 1	2	3
0	14	FFFF	FFFF
1	FFFF	FFFF	FFFF
2	0	FFFF	FFFF
3	FFFF	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

TX

Node	P_Route 1	2	3
0	FFFF	FFFF	FFFF
1	0	FFFF	FFFF
2	FFFF	FFFF	FFFF
3	FFFF	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

Figure 13: Tables at Node 1 after receiving delay vector from Node 2

RX

Node	P_Route 1	2	3
0	14	14	FFFF
1	FFFF	FFFF	FFFF
2	0	FFFF	FFFF
3	FFFF	0	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

TX

Node	P_Route 1	2	3
0	FFFF	FFFF	FFFF
1	0	0	FFFF
2	FFFF	14	FFFF
3	14	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

Figure 14: Tables at Node 1 after trunk Node 1 – Node 3 comes up

RX

Node	P_Route 1	2	3
0	14	FFFF	FFFF
1	0	FFFF	FFFF
2	FFFF	FFFF	FFFF
3	14	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

TX

Node	P_Route 1	2	3
0	FFFF	FFFF	FFFF
1	FFFF	FFFF	FFFF
2	0	FFFF	FFFF
3	FFFF	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

Figure 15: Tables at Node 2 after trunk Node 2 – Node 3 comes up

RX

Node	P_Route 1	2	3
0	14	14	FFFF
1	0	14	FFFF
2	FFFF	FFFF	FFFF
3	14	0	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

TX

Node	P_Route 1	2	3
0	FFFF	FFFF	FFFF
1	FFFF	14	FFFF
2	0	0	FFFF
3	14	FFFF	FFFF
4	FFFF	FFFF	FFFF
5	FFFF	FFFF	FFFF

Figure 16: Tables at Node 2 after trunk Node 2 – Node 3 comes up

3.7 Routing

In a normal store-and-forward network, packets must be received completely - and correctly - at a node before they are routed on to the next node. This means that a receiving node has to wait for a complete packet to arrive before it can transmit the packet.

The designers of NETEX, the predecessor of DNN, came up with the idea of sending packets on to the next node as soon as the packet header had been received at the intermediate node [Bod80, Bod81].

Routing calculations were done using the information contained in the packet header, and network throughput was greatly improved by sending packets on the moment the header arrived. This meant that error checking was only completed after the packet was transmitted onwards, so techniques had to be developed to deal with this. This through-routing technique was very good in theory, but the possibility arises that a node sending data out on a high speed trunk could be being supplied by a low speed trunk with the result that the node ends up waiting for data on the low speed trunk, because it is able to immediately send data onwards, faster than it is receiving data. To eliminate this situation where a node is sitting idle, the concept of *multistreaming* or *stream-switching* was borne.

3.7.1 Multistreaming

In a multistreaming environment, each trunk is viewed as comprising of a number of logical streams. This was depicted in Figure 5. Viewing a trunk in this way, there can be more than one packet queued for sending on a trunk, and a number of packets can be put on streams awaiting transmission. The implication of this is that as soon as one stream runs out of data to send, a *stream switch* can occur, and data from another stream can be sent before switching back to the stream that ran out of data, keeping the outgoing line busy at all times. Some sort of scheduling occurs, giving each stream service according to some service discipline. Priority and normal classes can be implemented using priority streams, creating priority by servicing certain streams more regularly than others. In this way the trunk protocol logically time-shares the

trunk facility amongst multiple users, with the users being the streams on the trunk.

Kermani and Kleinrock [Ker79] have recognised the success and potential of this multistreaming approach, and not long after stream switching was implemented in NETEX, they pronounced the virtues of what they called a 'Virtual Cut Through' approach.

On a busy network, where streams have to wait because of heavy traffic, the performance of multistreaming will be the same as that of a normal store-and-forward network. Packets will be received completely at nodes before being forwarded, because of queues existing on routes and the packets having to wait for streams. On a quiet network with light load, multistreaming can have a great impact in improving system throughput. Kleinrock points out this load phenomenon as influencing 'Virtual Cut Through', and the designers of DNN recognise the effect of network load on their multistreaming too.

3.7.2 Streams in the DNN

In the DNN each trunk is divided into 6 streams in each direction: three priority streams and three normal streams. Each stream is controlled by a half-duplex protocol. Utilising 6 half-duplex stream protocols in each direction makes efficient use of the trunks and provides a full duplex protocol at the trunk level. Combining the ACK responses on all 6 streams in a particular direction makes economical use of the trunk bandwidth too.

A micro-code scheduler transmits the various streams, looking for the next ready stream after completing transmission of a particular stream. To speed transmission up at the data link level, the data link protocol allows the closing flag of one packet to act as the starting flag for the next. The scanning algorithm is such that after the transmission of a normal stream, priority streams are checked for readiness. If there are priority streams ready then these (up to) three streams are transmitted first before the next normal stream is transmitted. We can think of priority and normal classes as being 'throughput classes'. The end user selects priority or normal when setting up a session. For example, an interactive terminal user would desire priority service, whereas a user requiring file transfer or printing jobs could request normal.

There is a third conceptual class of stream, namely a control stream on which the startup packet and delay vectors are queued. This queue will only consist of one customer at a time. The reason for calling this a "conceptual class" is that effectively control packets are put at the front of the priority queue for transmission, and this is the way in which the class is implemented. This control stream will still have to wait for the end of a stream that is busy sending before it can be transmitted.

Routing is done as soon as the header of a packet is received. In the case of a 9600 bps trunk feeding into a high speed 64 Kbps trunk[3] the situation may arise that there are no bits waiting to be sent on the particular stream because the first 10 bytes have been sent, and now the stream is idle awaiting more bytes to send. To eliminate this situation where a stream is sitting idle, the scheduler will switch to the other streams awaiting service and route their packets, creating a multiplexing effect, before returning to the low-into-high speed situation and routing the next group of bytes on this stream. When a stream runs out of data to send, a 'partial end of stream' (NULL) is sent to the next node, and a stream switch occurs.

It should be seen from the above scenario that in this sort of situation, the listed internal delay stored at a node may be less than the actual delay which occurs. The listed internal delay works on the assumption that the messages will be transmitted without running out of information to send, however when a low-into-high speed connection occurs, a forwarding node will have to wait for data to arrive.

3.7.3 Error Recovery

When multistreaming is used error recovery becomes a complex issue. If an error is discovered in a packet once the whole packet has been received, a *downline abort* is necessary to tell 'downline' nodes (to which the packet has already been routed by virtue of the stream switching) that they should discard that packet from their buffers. This is actually a data link control *abort* indication, and once the downline node detects this status on a line it will discard data received by resynchronising the Communication Line Adapter (CLA), and waiting for a startup on that stream. The

[3]The typical capacity of a high speed DIGINET trunk is 64 Kbps with an effective 48 Kbps capacity due to the fact that 12 Kbps of the bandwidth is required for control information.

CLA will not pass any more data through until an opening flag is seen.

This is only applicable when multistreaming is activated. Without multistreaming downline aborts are unnecessary as the network performs in a standard store-and-forward manner, checking each packet for errors at each node before forwarding it to the next node. If the line error ratio on a trunk passes a certain threshold, multi-streaming is automatically inhibited, and onward bound packets are delayed until the packet has been received completely and correctly. Multi-streaming is re-enabled when an acceptable error ratio is reached again.

3.7.4 The Router

The router looks at the destination node number of the packets which arrive for routing, and then consults its delay tables and decides on an outgoing *p_route*. The router will calculate the time it will take to transmit the packet, based on its length (in packet length units) given in the packet header. The router then updates the internal *p_route* delay on that particular *p_route*, increasing it by the number of PLUs multiplied by the *p_route* delay, as stored in its received (RX) delay vector.

This is, however, an optimistic view of the *p_route* delay for two reasons:

1. ACK return time is not considered (although this is negligible).

2. More significantly, if multistreaming is enabled and a slow trunk is feeding into a higher speed trunk, the high speed trunk may actually have to wait for bytes to arrive, obviously increasing the actual time that transmission of that packet will take. Stream-switching ensures that the higher speed trunk is kept busy sending from other streams, but the internal delay does not reflect possible delays arising from having to wait for bytes to arrive.

We can thus say that the internal delay value is an optimistic view of transmission delay on that line.

In terms of the network's division into *p_routes*, trunks and streams, the routing procedure takes place in the following manner:

1. Select best *p_route*

2. Select trunk

3. Select stream

If a packet is not allocated to a stream, it is queued on a *p_route*. A chain of packets can be queued on a *p_route* meaning that they have been allocated to a *p_route*, but have not yet been assigned to a stream. When a packet has been transmitted and an ACK received, a stream on one of the trunks becomes available, upon which the first queued packet is dequeued and placed on the free stream on that particular trunk.

If a packet on a stream is not acknowledged at an intermediate node (for example, the destination node may have gone down), then:

- if the packet life has been exceeded, it is discarded.
- if there is packet life left, the packet is put back on the queue for routing.

At the source node, messages remain on a source retention queue until a high level ACK comes back to the source. This is where packet life comes in to play, and if the packet is discarded, it is at the transport level of end-to-end recovery that the packet will be re-sent.

3.8 DNN Implementation of X.25

It has been mentioned that an X.25 like logical connection between source and destination hosts exists. The *Network Access Module* co-ordinates the transmission of packets between source and destination nodes.

In the DNN there are some variations from the X.25 protocol as it is defined by CCITT.

1. The most significant difference is in the *CALL REQUEST* and *CALL ACCEPTED* packets. The Network Access Module differs from X.25 in that it does not establish a point-to-point connection. This means that each Network Access Module in each node does not know which logical channels each other Network Access Module has free.

To obviate this limitation, the originating node's logical channel (or connection) number is sent in the internal *CALL REQUEST* packet, and the *CALL ACCEPTED* packet is sent containing the connection numbers of source and destination nodes. These two connection numbers are used by both ends, and define one logical channel. For all subsequent transmissions, the transmitter inserts the receiver connection number in the X.25 logical (virtual) channel field.

2. Sequence numbering in DNN is done using 15-bit sequence numbers (modulus 32768) as opposed to the usual 3 or 7-bit sequence numbers (modulus 8 and 128, respectively) of standard X.25.

   The rationale for using extended sequence numbers is that in large networks a great number of packets can be in transit at any time, and unique sequence numbers are necessary. The ability to handle large quantities of packets is essential for satellite and wideband communication in particular.

At the data link level, implementation of multi-streaming prevents the use of the LAPB protocol, however bit oriented HDLC framing conventions are used. Effectively, frames have the same structure, but the *address* and *control* fields have been redefined.

3.9 Configuration File

There is a configuration file at each node, in which constants *routes* (maximum number of neighbours) and *nodes* (maximum number of nodes in the network) are defined. These constants are used to define the size of the delay tables and vector. The delay vector increases in proportion to the number of nodes, with an increase of 2 bytes per node on each route. A constant *trunk* defines the maximum number of trunk circuits to cater for. Receiving window size (*RWSZ*) and maximum packet life (*MPL*) (in 200 millisec units) are defined for each node, and when internal clocking is used, the transmission speed (*TXSPD*) on each line (trunk) is defined. Normally *TXSPD* is set to 0 for external (or modem) clocking.

Chapter 4

Network Modelling

In carrying out any modelling exercise it is essential to have a clear understanding of the model requirements, since these will determine the nature and focus of the modelling system. This chapter begins by presenting the requirements identified for a system modelling adaptive routing. An investigation of solution techniques and tools is then made in order to determine an approach which will best satisfy the model requirements. A comparison of modelling tools is presented, and from this experience the proposed solution technique is identified and motivated.

4.1 Modelling System Requirements

The primary purpose of this modelling exercise is to develop a modelling system which can model adaptive routing. In order to achieve this there are certain requirements. A fundamental requirement is that a network should be specified, with provision of the network topology and load. This specification should be transformed into a model which is solved and the results should be reported back to the user.

Two classes of modelling system user are identifiable. These are the *network designer*, who makes use of the modelling system for network performance evaluation and capacity planning, and the *routing algorithm designer*, who makes use of the modelling system for investigating the performance of routing algorithms. An understanding of these two different user perspectives is helpful in classifying the

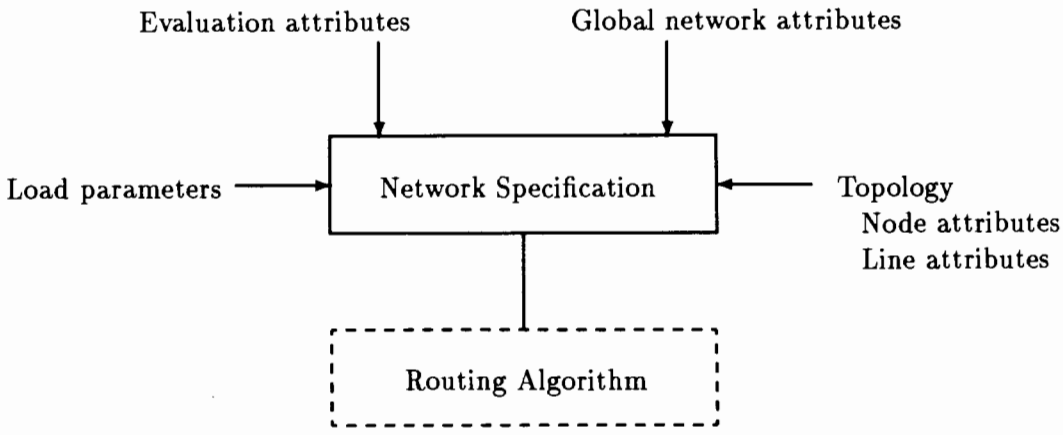


Figure 17: Network specification

requirements of each. Figure 17 shows the inter-relation of the two perspectives in providing a network specification.

The network designer will typically wish to see the effect of adding and removing network trunks and nodes. Changing the capacities of these will be a further area of interest, as will the variation of packet arrival rates and destination probabilities. Ideally a modelling session of this nature should proceed in an interactive 'what if' fashion. This suggests a real time modelling system, as a batch processed model solution does not facilitate exploration of the type described. It is desirable that results are presented in such a way that the network designer can easily relate these to the network specification.

Routing algorithm designers may use the system in the same manner as network designers, but these users will be more interested in observing the effects of routing algorithm modification. The modelling system should not only be capable of modelling adaptive routing, but also of allowing the routing algorithm to be changed.

From this description it can be seen that for non network-experts *operational*, *tactical* and *strategic* decision making [Ter87] should be possible, suggesting the clear and straightforward presentation of utilisations, delays and bottlenecks to such users. Network experts such as routing algorithm designers will require more detailed output such as delay table values, and route selection reporting which will be of interest to them.

With the identification of these modelling system requirements we now approach performance modelling with the intention of identifying the solution technique which will best satisfying them.

4.2 Performance Modelling

A standard method for analyzing the behaviour of a complex system is to identify those parameters that have a significant effect on the performance of the system, and then to find a set of relationships or equations that relate system performance to these parameters. These equations define a model of the real system. The equations can then be solved to calculate system performance as a function of the defining parameters. The model can thus be used to understand the behaviour of the system without having to experiment with the actual system.

There are two major approaches towards solving the equations that define a model: analytic techniques, and simulation. An analytic solution seeks equations relating the performance measures to the parameters. Simulation may be used when the equations are insoluble, or when it is not even obvious what the equations of the system are. Everything else being equal, analytic solution methods are preferable for their relative speed of computation.

4.2.1 Analytic Methods

A number of theoretical techniques are appropriate to analytical modelling. The most prominent such technique is queueing theory. Since 1975 the theory of multiclass queuing networks has assumed a position of central importance in the modelling of computer system performance.

The most common queueing networks have come to be called BCMP-networks [Bas75][1] or *product form* networks. The latter term derives from the fact that, informally, the solution of a queueing network is given by

[1] Named after Baskett, Chandy, Muntz and Palacios-Gomez who are the four authors of the seminal work on the topic.

$$P(S_1, S_2, \ldots, S_M) = \frac{1}{G} P_1(S_1) P_2(_2) \ldots P_M(S_M)$$

where $P(S_1, S_2, \ldots, S_M)$ is the probability of a network state in a network of M queues. $P_m(S_m)$ is a factor reflecting the probability that queue m is in state S_m and G is a constant which ensures that the probabilities sum to 1.

The author had access to a modelling tool called MicroSnap with which one is able to specify and solve product form queuing network models of systems. MicroSnap is discussed in Section 4.3.1.

Only certain attributes of real systems can be modelled accurately by *exact* product form networks. Typical of those which cannot be modelled in this way are

- congestion leading to blocking and customer loss
- state dependent routing
- batch arrivals of customers
- forking or joining of customers
- simultaneous resource possession.

Exact solutions of analytic models which do represent most of these phenomena are possible by direct Markovian analysis. The tool MACOM allows one to define and solve such models from a graphical description of the system being modelled. The solution involves solving a set of simultaneous linear equations.

The problem with this direct approach is that the system of simultaneous equations can become too large to solve with known methods. An alternative is to build an extended queueing network model of the system and to solve the model by simulation. Extended queueing networks allow many of the phenomena which product form networks cannot cope with. RESQ [Sau81, Sau83], is one such system.

4.2.2 Simulation

A common approach to modelling a computer system is to simulate it. In other words, to build a program that behaves like the system and to observe the behaviour of the program. Note the subtle difference between this and the use of simulation to solve an extended queueing network model of the system as is the case for the modelling tool RESQ.

There are three main problems with this technique: the expense of building and debugging a simulation program, the computational expense of executing the program and the statistical nature of the observed results.

Naylor et al. contend that "simulation is ... essentially a technique that involves setting up a model of a real situation and then performing experiments on the model" [Nay66, p. 2].

Various views have been proposed as to what should constitute a simulation model, and the process which should be followed to build a simulation model. Two such views will be presented here as a means of insight, and also for purposes of comparison.

Gordon [Gor69] suggests that there are 3 major tasks which comprise a simulation:

1. Generating a model: a set of numbers representing the state of the system (system image) must be found. Activities of the system must be represented as routines that are to carry out the changes to the system image.

2. Programming the Model: the procedure that executes the cycle of actions involved in carrying out the simulation (the simulation algorithm) must be programmed. Gordon makes the point that, while the routines representing the system activities are specific to the system being simulated, the simulation algorithm need not be.

3. Generation of an output report: Statistics gathered during the simulation run will usually be provided by a report generator.

Pritsker, Sigal and Hammesfahr [Pri89] have more recently proposed a simulation strategy. The phases which Pritsker et al. identify are more detailed than those of Gordon, although Pritsker's stages could be viewed in a top-down manner as a second

level breakdown of Gordon's proposed stages. The philosophies of both of these authors suggest an iterative approach to the construction of a simulation model. The modelling process proceeds by designing and building a model, and then refining this model until additional detail is unnecessary, and validation can show the model to be a realistic representation of the system being modelled.

Pritsker et al. use the term "evolutionary problem solving" (p. 15) which encapsulates the philosophies of Pritsker et al. [Pri89], Gordon [Gor69], Naylor et al. [Nay66] and others.

4.2.2.1 The Realism vs. Simplicity Dilemma

It has been pointed out by Naylor et al. that a balance needs to be attained in setting up a simulation model. The model needs to be detailed enough to capture all the salient features of what is being modelled, but also simple enough to be well understood and clearly defined. This important issue is reiterated by Pritsker et al. as well, who remark that a model should be "easy to understand, but realistically detailed" [Pri89, p. 27].

Davies warns that a simulation model "should in no sense be an emulation, seeking to duplicate every small detail of networked operation" [Dav79, p. 88] and he cites the work of Slyke, Chou and Frank in which the dangers of a model which seeks to cover every detail are expounded. Expense, waste and slow-running execution are the result of too detailed a simulation. Davies, guarding his position, recognises that elimination of too much detail from a simulation model is dangerous. He suggests that in assessing a routing algorithm, specifically, a simulation model should describe nodal queue handling procedures and the routing process itself in full detail.

4.2.2.2 Simulation Languages

Gordon identifies a simulation language as "a language with which to describe the system, and a programming system that will establish a system image and execute a simulation algorithm" [Gor69, p. 134]. In a simulation language routines are typically supplied for such functions as the scanning of events, clock updates, gathering of

statistics and maintaining events in time and priority sequence. Entities of a model are usually specified together with their attributes.

Numerous simulation languages have been designed but two of these languages, in particular, have been very widely used those being General Purpose Simulation System (GPSS) and SIMSCRIPT.

Some of the major objections to using a simulation language are that there is generally

1. Reduced flexibility in models.

2. Increased computer running time.

3. Restrictions on the format (and content) of output reports.

The disadvantage of these simulation languages is that in their generality is their specificness. They are unsuitable for the level of detail required in a large simulation model. Another difficulty with simulation languages is that it is rare, and usually impossible, to be able to get at measures other than those provided by the system. Model development time could be decreased considerably if a simulation language is used, and this is a distinct advantage of their use.

4.2.2.3 General Purpose Programming Languages for Writing Simulations

An alternative approach is to write a simulation using a general purpose programming language. If a general purpose programming language is used the model may require long development time, since the modeller will have to implement a scheduler, a system image, and so on, which would not be required if a simulation language was used. While predefined routines to create different arrival and service distributions exist implicitly in a simulation language, routines to generate variables according to particular distributions have to be implemented explicitly if a general purpose programming language is used.

4.2.3 Hybrid Models

In a hybrid modelling approach analytic and simulation techniques are combined. A simulation module may be used to replace a subsystem of an analytical model, or an analytical sub-module may be incorporated into a simulation model. In this way a simulation model could be made more efficient and less time consuming, or an analytical model may incorporate simulated results in place of some aspect of a network for which equations may be difficult to determine. HIT [Bei90] provides such a modelling facility.

4.3 A Comparison of Performance Modelling Tools

We now investigate some of the tools which are available for the purposes of performance modelling. The motivation for this investigation is to explore the suitability of available modelling tools to the task of modelling adaptive routing.

4.3.1 MicroSnap

The MicroSnap package developed at the University of Cape Town allows the modelling of BCMP[Bas75] or queueing networks which have a product form solution. These networks cannot be used to model blocking (which may result in the loss of a customer) or state-dependent routing, amongst others. Resources in the real system are represented by service centres, and the load by customers in the queueing network.

In BCMP networks, customers are grouped into *chains*. A chain $j, j = 1, \ldots, J$ may be closed with a finite population K_j or open, meaning that customers arrive and leave the chain. Customers belonging to a particular chain may in turn belong to different *classes* depending upon their resource demands and a customer may change class within a chain as its resource demands vary from one visit to the next to a particular centre.

A *validation algorithm* ensures that a model as described is a valid one in that there are no centres to which customers in a particular chain (or class, if applicable) arrive, but do not depart. The validation algorithm also checks that there are no

centres in an open chain which are not visited at all from an entry point in the network, and so on.

Central to the theory for closed networks is the concept of *relative throughput* of each class of customer at a centre. In MicroSnap these throughputs are determined by solving a set of simultaneous, sparse linear equations using a version of LU-Decomposition known as *Crout's method with Partial Pivoting* [Pre87]. As with most linear equation solvers numerical stability problems may arise with this technique in certain circumstances.

MicroSnap solves a validated queueing network model by applying the *Mean Value Analysis (MVA)* algorithm found by Reiser and Lavenberg [Rei80]. This algorithm is based upon a recursive expression for the response time of a service centre in terms of measures for the network with one less customer [Lav83]. The time complexity of the algorithm is proportional to $\Pi_{j=1}^{J} K_j$ and soon becomes prohibitive for networks with large population sizes. The MVA algorithm, theory and associated calculations are described in Lazowska et al. [Laz84].

4.3.1.1 The Modelling Paradigm

A queueing network consists of a series of *centres* which represent the resources of the system being modelled. The centres are defined with different service disciplines depending upon their purpose and function. Those service disciplines supported by MicroSnap are:

- **FCFS** (First-Come-First-Served) — customers receive service in the same order as they entered the queue.
- **PS** (Processor Sharing) — customers receive service immediately, but if n customers are being served, each customer receives service at $1/n$ times the rate at which a single customer would receive service. This is identical to the discipline described in Section 4.3.2.
- **LCFS** (Last-Come-First-Served-Preemptive-Resume) — a customer entering the queue receives service immediately, to the completion of its request or until

another customer enters the queue. If another customer arrives, the customer which was in service is taken out of service and put at the front of a waiting queue, while the new customer enters service. If another customer arrives, then the customer being served will be put on the waiting queue ahead of the one it interrupted. Once an interrupting customer has completed service, service of the customer at the front of the queue is resumed rather than repeated.

- **DELAY** (Infinite Server) — this service discipline is such that there is always a server available for a customer entering the queue. A customer will only be delayed for a time equal to the service it demands at the centre, and no longer.

- **MICRO** — this is not a service discipline in the usual sense, but rather an attribute associated with a centre which serves merely as a reference point in the network model, typically to measure the throughput on a particular route.

- **PRIORITY** — customers are either priority or not, and are served FCFS within priority. Since BCMP networks do not allow priority servers, the results in this case are approximate.

Customers are the entities demanding service, and represent the load upon the system. MicroSnap allows customers to be grouped into *workloads* (*chains*), and customers belonging to the same class in a workload exhibit similar behaviour and place similar demands on the centres. MicroSnap allows *multiclass workloads* for situations where the service demands of a customer change from one visit to a centre, to another visit to that same centre. Customers may change class membership any number of times as they proceed through the network, thereby representing the changing resource demands of customers in a workload.

Routes are the paths that workloads take through a network. In MicroSnap a customer representing a class of a particular workload at a centre is referred to as a *node* in the network. The routing description of a particular workload describes customers moving between nodes, rather than between routing centres. Probabilities are specified at a node, indicating how customers at a node will proceed to any other node specified in the routing description.

4.3.1.2 User Interface

Although an X-Window graphical interface for MicroSnap is under development, the current version of the tool uses a specification language interface. There is no interactive modelling facility as is the case with MACOM described in Section 4.3.2.

The language, MicroSnapL, is a functional subset of the language SNAPL/1, as described by Booyens and Kritzinger [Boo84b]. Any editor can be used to prepare the MicroSnapL model specification which is compiled and validated before the model is solved. A model specification consists of a definition section, describing the model structure, and an evaluation section describing the experiments and results required.

Results can either be printed or formatted in a Lotus worksheet for easy graphical output.

4.3.1.3 Experience with Using MicroSnap

There are certain phenomena commonly found in real systems that cannot be modelled accurately by the BCMP networks which MicroSnap solves. Systems in which phenomena such as those described in Section 4.2.1 are determinants of performance, cannot be modelled with MicroSnap. When MicroSnap is adequate, however, the models are solved quickly and accurately which is important when using the tool to make decisions in real time, as would be the case in managing a wide area computer network.

MicroSnap model descriptions have to be compiled until error free. This is time consuming, particularly since the current version has no error recovery. It would be very useful if, for example, MicroSnap would issue a warning when an expression defining a routing probability evaluates to zero. This error is currently very difficult to detect.

4.3.2 MACOM

MACOM is a software tool being developed at the Universität Dortmund by Sczittnick and Müller-Clostermann, and the Deutsche Bundespost [Scz90] for deriving and solving performance models of complex computer systems. Models are solved

by calculating the steady-state probability distribution of the finite, homogeneous, continuous-time Markov chain derived from a graphical model description of the system. Because the solution is derived directly from the Markov chain there is, in principle, no restriction on features such as

- flow congestion leading to blocking and losses of customers in a network
- batch arrivals of customers
- load balancing and adaptive routing in computer networks
- flow control
- circuit switching in teletraffic networks
- transient loads.

Such characteristics of the actual system can be accurately represented in a MACOM model. MACOM also allows for analyses of transients in the system being modelled.

Solving the models basically involves the solution of simultaneous linear equations represented by large (maximally, order 350 × 350), very sparse coefficient matrices. This model solution method is clearly superior to simulation in that none of the problems associated with the statistical nature of a solution by simulation arises.

For solving small systems of equations, an algorithm developed by Grassmann, Kumar and Billinton [Gra87] is used. This algorithm is numerically more stable than the conventional Gaussian elimination method, since it avoids errors arising from the subtraction of near equal values. For large matrices iterative techniques are employed, and a Gauss-Seidel technique or successive overrelaxation is used.

When the equations are solvable, the answers are theoretically exact. Depending upon the value of the coefficients the potential for rounding error in the computations is however, significant.

The major drawback of the entire approach is the potential intractability of models with too many states.

4.3.2.1 The Modelling Paradigm

MACOM, not surprisingly, borrows heavily from classical queueing network theory terminology. "Customers" are however, called "load units" which, depending on the context of the model, may refer very broadly to jobs, calls, packets or any other logical or physical entity.

The classical concepts of service stations and probabilistic routing have moreover been extended by the facility to limit buffer space at a service station so that *wait-loss stations* can be implemented in addition to pure *loss stations*. For each station, the number of load units that can be held simultaneously at a station is given to specify the capacity of the station. Load units enter the network through *sources* and exit through *sinks* or *loss exits*. *State dependent routing* is allowed by MACOM so that the movement of packets, for example, can be made dependent on the state of the system. Three central parameters are associated with every service station: capacity, number of servers and the service discipline. Service disciplines offered by MACOM are:

- **INFINITE** — newly arriving load units will always find a server available.
- **PS** (Processor Sharing) — all load units are always served, but the service rate is inversely proportional to the number of load units present.
- **RANDOM** — the number of servers is limited and when a server becomes free, a load unit is chosen at random from the queue.
- **NON-PREEMPTIVE PRIORITY** — this discipline is similar to the last one except that a customer which is of a priority class cannot be preempted and is served to completion.**PREEMPTIVE PRIORITY** also implements priority classes, but allows the preemption of a priority class customer.
- **POLLING** — customers are placed in different queues according to a polling number. Selection of the next customer to serve, is based on the polling number of the current customer. In this way a cyclic polling of queues can be

implemented. A matrix can be used to formulate a stochastic polling strategy. **POLLING EXHAUSTIVE** uses the same queueing mechanism, but the queue being served is only changed if the queue of a departing customer is empty. In this case customers on a queue are scheduled with equal probability.

Arrivals to the network can have *exponential* or *phase type* Coxian interarrival time distributions. A source may also generate *batch* arrivals to the network, with a group of load units entering the network at the same instant.

4.3.2.2 User Interface

The MACOM package makes use of SunView on a SUN workstation, and provides a graphical user interface (GUI) which is used to create, edit and run models.

Models consist of different types of objects. The user selects objects – such as sources, sinks, servers and lines – from a choice of icons, creating a complete, connected model. Each object has attributes associated with it, and the attributes are specified by invoking pop-up windows from each object. Through this combination of object selection, and attribute entry via pop-up windows, a complete model description can be entered without any programming at all. MACOM uses colour effectively to distinguish different components of the model. It is easy to modify a model, and the user can add, delete or move objects as desired. MACOM performs certain integrity checks to ensure that a valid model is specified.

Routing of load units is described by *links* and *conex* elements. A link represents an unconditional transition from one model element to another, whereas a conex is a switch comprising one entry and two exit points. The exits are referred to as the *then-exit* and the *else-exit* respectively.

At a *probabilistic conex*, a probability p determines the probabilistic choice of the then-exit, implying that the else-exit is chosen with probability $1 - p$. At a *state-dependent conex*, a boolean expression specified by the user determines the branch followed by load units.

4.3.2.3 Experience with Using MACOM

MACOM is unique in its objectives and is still under development. As is the case with all general purpose tools, there is an overhead involved when applying it to a specific problem for which a specific solution could be provided. The time complexity of even the best, known linear equation solvers is $O(n^3)$ and large models take a long time to solve. MACOM allows solutions to run as background processes however, so that large models can be started, and results examined at a later stage.

As with other graphical interfaces, a difficulty with the MACOM GUI is that very large models tend to be cumbersome to build and difficult to discern on the screen. Connecting lines are easily lost, although the use of colour in MACOM definitely helps to clarify the different routes in a model. It is possible to mechanically scroll through a model, but it would be helpful to have a zoom facility as well. The use of smaller icons in MACOM may also alleviate this problem.

Whilst MACOM does allow the specification of variable names for certain parameters, it does not have a facility whereby default attributes can be assigned to classes of objects. In large models this necessitates re-specifying parameter values for each object in a class. This can be tedious in models such as the computer communication network described in Section 4.3.4 in this paper, for which the same set of attributes must be set for about 50 objects. It would be useful if one could specify that service disciplines, arrival distributions or evaluation choices should apply to all objects of a particular class.

Another limitation of the GUI approach is that obtaining a hard copy of the model is not a straightforward process. This implies that routing or constant parameter values have to be queried or examined interactively. Where the user has specified variable names for parameters, these parameter values are however printed out with the results. Whilst MACOM models are specified graphically, results are given in tabular form without any reference to the GUI.

The factors mentioned lead one to consider whether, compared to a language approach such as that used in MicroSnap, a GUI is actually an advantage.

A feature which the author would have found very useful in modelling distributed adaptive routing would have been the facility to specify more complex state dependent

routing. This requirement is arguably beyond the design of a general tool such as MACOM however.

4.3.3 Pure Simulation

The third "tool" reported on is thus a pure simulation program built to simulate the various systems being modelled. The program was written in the language *C* and, although no attempt was made to design a general purpose simulator, some generality was provided to facilitate constructing the various models reported on in the next section. Equally it was decided not to use an existing general purpose simulation system such as GPSS [Sch74] since full control over the simulation was wanted, in addition to wanting to gain experience in the building of such a simulator. This program is referred to as the tool *Sandy*.

4.3.3.1 The Modelling Paradigm

Sandy is a discrete event simulator in that individual events which take place in a system are modelled by building a system image and monitoring the status of this system image whilst simulating arrivals, departures and service activities. Arrivals to the system are generated according to the specified distribution. The same holds for the service times which are randomly generated from a distribution with the mean service time specified. Also specified for each server is the service discipline.

Queues are maintained at each server for requests which arrive while the server is busy. Depending on the service discipline, the queued jobs will be scheduled in some manner by the server. A system clock keeps track of the simulated time, and a chain of future events is used to control the simulation. The simulation program keeps a chronological list of future events, and the simulation clock is set to the time of the next event, at which point any activities associated with that event are carried out. The activities associated with an event may include adding extra events onto the future events list for later processing.

While the system is executing, data are gathered at each server, and for the system as a whole, allowing one to compute the relevant statistics at the end of the simulation

run. No confidence intervals are computed.

Sandy stops after a certain simulation time or after a given number of events have been simulated. Although this subject is a research topic in itself, the author did not feel it justified to spent more time on this aspect of the simulation. The objective was to choose between the various modelling paradigms rather than to perfect any one of the tools.

4.3.3.2 Experience with Sandy

Constructing the program and generalised chain of events to implement discrete event simulation took a fair amount of time. Afterwards, it was straightforward to apply the program to the model.

With Sandy designed and implemented by the author it is clearly possible, as with all home grown discrete event simulators, to model any feature of the real system which is required. Equally, additional statistics such as coefficients of variation, standard deviations etc. can be computed from the data being collected.

4.3.4 Test Model

This section describes the results obtained by applying the various tools to a queueing model. The MACOM, Sandy and MicroSnap experiments were run on a SUN SPARCStation 1+ with 8 Mbyte of memory and no floating point accelerator.

The queueing model chosen was one where all the tools could model the system exactly. It could be argued that this makes the model solved trivial, and that the full capabilities of each modelling technique and tool are not illustrated. The reasons for using the chosen model were that:

- Model attributes had to be representable in each one of the chosen approaches.
- Model results had to be easily comparable in a quantitative manner which would not be possible if, for example, blocking was considered in MACOM and Sandy whilst it could not be handled in MicroSnap.

- Model solution time was one of the predominant features of comparison, and the WAN model was large enough to demonstrate significant solution time differences between the modelling tools used.

The definition of queue length, queueing time and waiting time, respectively, warrant some explanation, as interpretations of these measures differ in the different modelling tools:

Queue Length. In MACOM and MicroSnap queue length includes the customer in service. This definition was used in the solutions by simulation too.

Queueing Time. In general this is the average time spent waiting and being served, however in MicroSnap queueing time is the average time spent by a customer *before* being admitted to service. The latter interpretation of the term was adopted for consistency.

Waiting Time. In MicroSnap this time is understood to mean the average time spent waiting plus the average service time. For consistency, the MicroSnap interpretation was used throughout.

Three Node Computer Network

Figure 18 depicts the queueing model of a single nodal processor in a wide area computer network. Three such (sub-)models were inter-connected to form a 3 node network in the experiment for which the results are given in Table 1. It was felt that 3 nodes were adequate for the purposes of the investigation. Note that no attempt was made at this stage to model the routing and buffer mechanisms of the WAN.

In all cases the Nodal Processing Units (NPU) were modelled as Processor Sharing centres while the trunks were modelled as FCFS centres. Time-outs were modelled as DELAY servers in MicroSnap (Section 4.3.1) and INFINITE servers in MACOM (Section 4.3.2). While a MICRO server is provided by MicroSnap those reference points in the MACOM model were represented by INFINITE servers as well. Routing probabilities and average service times are given in the figure.

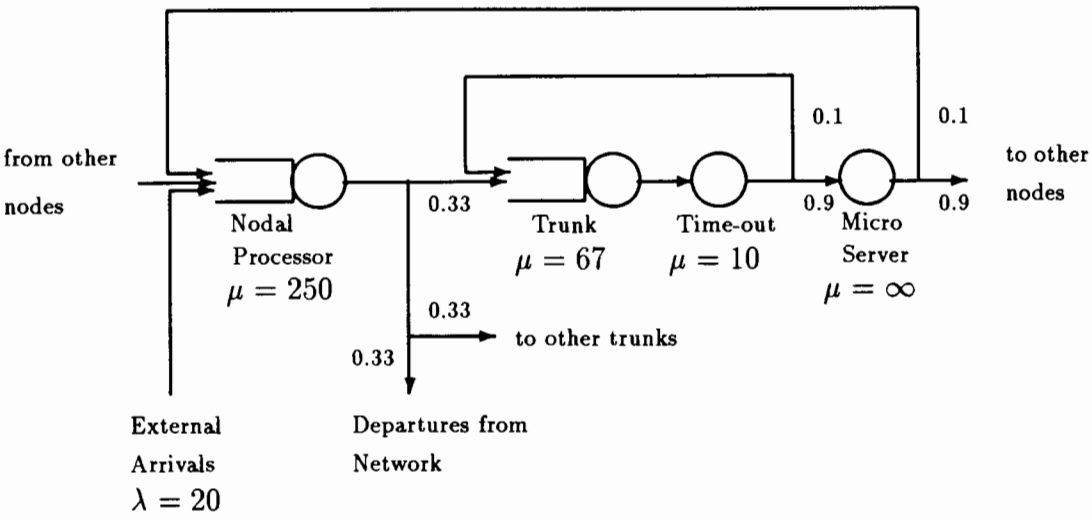


Figure 18: One node of the network

Measure	MicroSnap	MACOM	Sandy
NPU Utilisation	0.200	0.232	0.179
Trunk Utilisation	0.278	0.273	0.251
Nodal Delay	35.77	33.87	36.73
Network Delay	66.67	69.44	55.56
Solution time	0.7s	240.0s	16.0s

Table 1: Statistics for WAN Model

In the case of Sandy a total of 200 000 events was simulated. The Network Delay was computed from the total departure rate from the network.

4.3.5 Comparison of the Results

In evaluating the appropriateness of each technique and tool for modelling the adaptive routing computer network the following criteria were applied:

1. Availability of the software for incorporation into a larger modelling and network management system,

2. the appropriateness and ease of use of the various user interfaces,

3. how well the attributes of the WAN, particularly the routing algorithm and buffer management aspects, would be represented in the final system involving the model,

4. the computational accuracy of each tool, and

5. speed of computation.

All the tools, with the exception of possibly MACOM, were freely available in source code. The Markovian analysis applied by MACOM and its various solution algorithms are known [Scz90, Kri90] and could be implemented for this specific WAN modelling exercise if that technique were favoured in the end. With MACOM available it was, thankfully, possible to test the technique without having to actually build such a system.

The appropriateness of the user interfaces was not only of interest as it applied to the specific tools used, but the experience was equally valuable in deciding upon an interface for the WAN modelling system. In spite of some disadvantages an icon oriented graphical interface with pop-up windows to specify the system parameters, as implemented by MACOM, would seem to be the most appropriate. The potential difficulties with such an interface were described in Section 4.3.2.

The extent to which the theory underlying each tool allows the attributes of the WAN to be modelled has been mentioned in the text. If this were the only criterion for selection the choice would be Sandy, MACOM, MicroSnap – in that order.

The computational accuracy of Sandy clearly depends on the simulation time while that of MACOM on possible rounding errors incurred in the computation. The results for MicroSnap are exact. The largest observed deviation from the MicroSnap results was about 17 percent for the network delay in Table 1. This criterion would therefore not seem to be a determining factor in a choice between the various methods.

Speed of computation was by far the biggest variant with an order difference between the speed of MicroSnap and Sandy, and an order difference again between

Sandy and MACOM. The execution time of the three node model on a SUN SPARC-Station 1+ was 240 seconds in the case of MACOM. By far the greatest component of this time is spent in translating the graphical model description to the Markov transition probability matrix. The state space generator and quantitative analyzer components of MACOM have to be compiled, adding significant overhead to smaller models. This time is not proportional to the size of the model, so that for models with many states, MACOM would come into its own right.

Chapter 5

Design Principles

Having discussed the DNN in Chapters 2 and 3, and considered performance evaluation techniques and tools in the preceding chapter we now consider the general design principles which were identified to guide implementation of a WAN modelling facility.

5.1 Choice of Modelling Technique

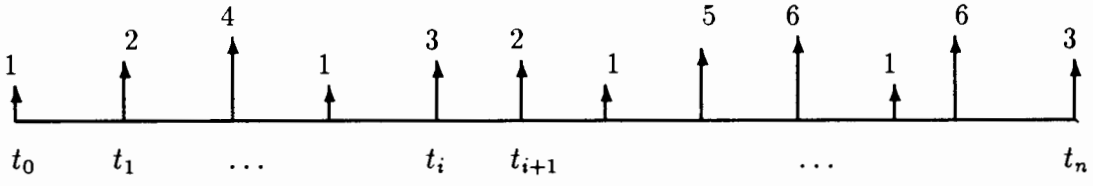
Ideally the modelling method chosen for the ultimate WAN model should be able to represent the attributes of buffer overflow and dynamic routing. Clearly then, the choice of modelling tool should be MACOM.

In addition, however, an important objective is to investigate various routing algorithms, and the ultimate model must allow us to change the routing algorithm. In order to study the effects of various algorithms, their effect on the performance of the network must be reflected.

While this is possible in Sandy (by issuing a call to a procedure performing the routing update), it is to some degree also within the capabilities of MACOM, while such a capability could be provided in MicroSnap too.

Another requirement of the final model is that we should be able to predict the performance of the WAN in real time: in other words execution time of the model is crucial.

Figure 19 represents a typical period of operation of the WAN, where the instants



delay table exchange due to

1 - under light load
2 - trunk coming up
3 - trunk going down
4 - internal delay change
5 - node into regulation
6 - route from divergent to convergent

Figure 19: Typical routing table update instants

$t_0 \ldots t_n$ indicate routing table updates arising from the normal process of table updates, routes going from convergent to divergent, nodes going into regulation etc. as described in Chapter 3. The average length of an interval $[t_i, t_{i+1}]$ is in the order of a few seconds.

Viewed at any such time instant t_i $(i = 1 \ldots n)$, the network is one with static routing and no buffer overflow. We can therefore conclude that during an interval $[t_i, t_{i+1}]$, the network can be modelled without accounting for dynamic routing and blocking.

In view of the requirement for real time execution, the computational speed of the various tools, and in light of the conclusion reached in the previous paragraph, it was clear that MACOM would be computationally too slow but that the modelling capabilities of MicroSnap would be adequate through running a sequence of models.

5.2 The Modelling Hypothesis

The nature of DNN routing is such that routing table exchanges and updates occur when certain events take place in the network. The routing algorithm prevents buffer overflow by forcing a *regulation* state at nodes whose available buffer space drops below a particular threshold. On a quiet network, where traffic is light, delay tables are exchanged every s seconds. When traffic gets heavier on the network, delay

tables are exchanged more regularly, specifically when there is a direction change in the RX delay vector or when there is an internal delay swing of more than a particular percentage change.

If one views the network at any instant t_i, where t_i is the instant immediately following a routing table update, the network routing remains static and no buffer overflow occurs during $[t_i, t_{i+1}]$. The duration of such an interval is in the order of seconds. This suggests that a sequence of models can be analysed, one model for each time interval $[t_i, t_{i+1}]$. These models can then be combined in such a way as to reflect the effect of adaptive routing and buffer overflow over a longer time interval (of the order of minutes).

The advantage of this approach is that it enables the use of analytic techniques to solve each of the static models. Since each of the static models exhibits fixed routing without buffer overflow, this removes two of the traditional analytic modelling difficulties. Having argued in Section 4.3 that an analytic modelling approach is preferable for reasons of efficiency and accuracy, the objective of the study was to determine whether a sequence of static models could be used to approximate the dynamic nature of the real network. If the approach could succeed in providing the model solution it would pave the way to further investigation of such a technique, giving new consideration to the use of analytic methods. As mentioned there are distinct advantages if analytic techniques can be employed, and this study attempts to exploit those advantages and overcome some of the limitations which queueing networks have traditionally posed. Whilst there is ongoing work attempting to extend queueing networks to handle non product form features of networks [Ser89, LeB86, Onv90] the approach of this study is to utilise the specific nature of the network under investigation and to use its characteristics to create an efficient modelling tool. The generality of such an approach will be considered and reviewed as an epilogue to the description of the model design and implementation, and the results which were achieved.

5.3 The Model

Having assessed the nature of the network, and made the hypothesis that the network routing can be modelled through a sequence of static models, it is necessary to consider how these models can be combined in such a way as to demonstrate the nature of the routing algorithm. It is important to reiterate that the purpose of this study was to model the *routing* in the WAN. This meant that the study was focussed on the OSI Network Layer, or DNN Network Routing Module. The implication of this is that higher and lower layers, with their associated concerns and special requirements, were not modelled. This was an assumption made in order to keep the study manageable, and so that attention could be given to the routing component which was the aspect under investigation.

5.3.1 Static Phase

Initialising the model, and attaining a state from which modelling of the dynamic nature of the network could proceed, constituted an important phase of the modelling process. Routing in the DNN is informed by tables which are kept at each node. It was necessary during the *static* phase of the modelling process to initialise these tables in such a way as to reflect the tables at each node of the network.

At a lowest level this meant that the *p_routes* connecting nodes had to be represented in these tables, implicitly storing information about the topology of the network being modelled. Each *p_route* constitutes an entry in the received (RX) and transmitted (TX) delay tables, so the tables manifest the physical connectivity.

A secondary level of information is required in order to initialise the model. While it is a primary requirement to represent adjacencies in the network, it is also necessary to represent the internal and external delay values at each node in the network. This information is needed as a preliminary stage from which the dynamic modelling can proceed.

In order to determine the initial routing table values a sequence of steps is carried out. Knowledge of *p_route* service times is used to determine shortest paths throughout the network. At this stage an omniscient view of the network is taken. We are

modelling a distributed system, in which such a calculation could not take place by the very nature of a distributed system. This omniscient approach has been used to *initialise the network*. The network establishes itself through a complex sequence of events which is performed in order for each node to become aware of other nodes in the network, and for preliminary *p_route* service times to become known through the external delays held at each node of the network. The assumption is that there is no traffic while the network initialises itself (besides control packets, which are insignificant). For these reasons it is feasible to use a static shortest path algorithm in order to approximate the routing table values, and such a method was used to initialise the routing tables.

In order for the routing tables to reflect the state of the tables in the network, a transmission of all routing tables is carried out. The resultant received delays reflect the delays after static *p_route* service time calculation, and also after the routing algorithm has been used to transmit the values of *l_route* service times and receive these values along the various *p_routes* according to the routing algorithm.

After the shortest path calculation and the calculation of received (RX) and transmitted (TX) delay tables, an exchange of delay tables is carried out, in order to reconstruct the RX tables using the routing algorithm. From these tables it is possible to determine which *p_route* will initially offer the best service for packets destined for any node in the network. These 'best' routes are referred to as *preferred routes*, and this terminology is used to indicate the *p_route* offering the best service time to a particular destination node at any stage during network operation. Internal and external delays are taken into consideration when selecting a *preferred route*. This is the way in which routing occurs in the network, and as packets arrive at intermediate nodes in the network, they are forwarded on the preferred route at that particular instant.

Routing in the network changes with the load that is applied to the network. The demands which the load makes on resources of the network vary according to packet arrival rate and the destination frequencies of these arriving packets. The *internal delays* stored in each RX table are the delays on particular *p_routes*, incorporating transmission and queueing time on a specific *p_route* in calculating the internal delay

of that *p_route*. As the load moves through the network according to the routing algorithm, so the internal delay on *p_routes* changes. This *internal delay* on a *p_route* is a function of the packets queued on the *p_route* and the route service time. Since the route service time is known, a network model should attempt to calculate the queue lengths through determining what the preferred routes are, examining where packets are bound for, and then analytically solving the system to determine queue lengths on each trunk and at each node.

The nature of adaptive routing is that queues build up if all packets bound for a destination, d, are routed onto a preferred *p_route*. The situation may be compounded if a particular *p_route* provides a preferred route to multiple destinations. The routing algorithm monitors queue build up, and will start routing packets bound for d via one of the other *p_routes* if the delay on the preferred *p_route* (taking the time necessary to transmit all packets queued on this *p_route* into account) becomes longer than some other available *p_route*. In order to account for this *dynamic routing* a sequence of queue length calculations is carried out in order to model the action–reaction functioning of the network.

The routing algorithm directs packets along *p_routes* according to their final destination. During this process queues build up. Delay information is transmitted to adjacent nodes, and routing changes as this information proceeds through the network. This process is repeated continually during network operation. By modelling the network as a sequence of models with fixed routing in this manner, it should be possible to determine average queue lengths representing the state of the network at time intervals during network operation. This is the manner of implementation as a consequence of the modelling hypothesis.

5.3.2 Dynamic Phase

In order to update the modelled internal delays with values approximating those which appear in the received (RX) routing tables of the network, it is necessary to compute these values. It has been described how the approach to modelling is one of examining the network during an interval $[t_i, t_{i+1}]$. During this interval, before there is a change in routing at any node in the network, it is possible to solve the network

analytically using the knowledge of network topology, network routing and network load. The topology and load are specified by the user while the routing algorithm implemented is the determinant of the routing which occurs.

Analytic network solution is able to provide, amongst other measures, average queue lengths, average queueing times and utilisations at various points in the network. The time needed to transmit all the packets queued on a *p_route* is stored as the internal delay of that *p_route*. The *p_route* service time is multiplied by the number of packets queued in order to arrive at the internal delay. When viewing the network during a certain time interval we know what the *p_route* service time is during that period. If we know the average number of packets queued on a particular *p_route* it should be possible to determine the average internal delay (or an approximation thereof) using the product of the average queue length on the *p_route* and the *p_route* service time. The *average* queue length is appropriate here since we are viewing this measure as the average over a particular interval. In this way it is possible to update the internal delay on each *p_route* in an iterative fashion, through solving a sequence of static models. At each iteration of the model process internal delays are re-calculated as the outcome of the model solution.

Another consequence of the iterative modelling approach is that when representing the delay on each *p_route* in the *next* interval ($[t_{i+1}, t_{i+2}]$) this delay should reflect the delay on that *p_route* as influenced by the packets already queued there. To facilitate representation of this at each iteration in the model solution, a *p_route* delay at any particular iteration is determined by the *p_route* service time and the queue length of that *p_route* during the previous iteration.

Through a sequence of steps the dynamic nature of the network is modelled. A system model is constructed using the preferred routes as determined from the internal and external delays in the RX routing tables. The system model is solved and internal delays are updated as has been described. The routing algorithm is used to update TX tables and the appropriate columns of the tables are transmitted on the particular *p_routes* on which adjacency occurs. Tables are received, and again the routing algorithm is used to determine how received information is incorporated into the RX table. A manipulation of the algorithm is carried out in order that

the synchronous, ordered nature of the model updates produces a result akin to the asynchronous, unordered updates which occur in the DNN.

When a change occurs at a particular node, the TX table is constructed at that node, and the columns of the TX table are sent on their respective *p_routes*. At the receiving node the last transmitted TX table is considered when updating the RX table. This means that the timing and order of the updates will affect the way in which the updates occur. In order to model this distributed, asynchronous algorithm in some sort of a synchronous manner, tables are exchanged in the following way:

The TX tables of all nodes are constructed from the RX tables with their updated internal delays. In the model the nodes are updated in sequence. This means that when delay tables are received at another node, that node will interpret the received delay in the light of its latest updated TX table - taking into consideration the latest internal delay information. This means that every node is receiving information from other nodes as though it had initiated the routing table exchange.

5.3.3 Queueing Model

In order to 'solve' the network during a particular time interval it is necessary to represent the network in such a way that meaningful results can be established. A discussion of solution techniques has been conducted, and as was described, an analytic modelling approach is preferable. To this end the design principles of a queueing model are now presented.

5.3.3.1 Workloads, Classes, Service Centres and Routing

In order to model the traffic which typically makes demands on the DNN, two workloads were identified. These were *packets*, which constitute the major network load, and *control packets*, which constitute a very small part of the network load. Control packets only travel between adjacent nodes, whereas packets may travel between many nodes as they are routed to their destinations.

In a DNN model with n nodes, n classes are identified for the packets queueing model workload. These classes are named *N1bound*, *N2bound*, ..., *Nnbound*.

Through the use of these classes it is possible to route packets according to their destination. Packets arrive at nodes in the network, and the modeller specifies the destination frequencies of these packets as part of the network specification. If p percent of the external packets arriving at Node x are *Nnbound* then those packets will join the *Nnbound* class, which will determine their routing through the network and ensure that they exit the network through a sink to which they are directed on arriving at Node n.

At each node of the network routing is specified for all n classes of the workload *packets. Control packets* are routed to adjacent nodes, and this is done probabilistically, with control packets being sent with equal probability to each of the adjacent nodes.

Each network node is represented by a Processor Sharing service centre in the queueing model, as shown in Figure 18. The mean service time of these centres approximates the time which it takes to process each packet in the DNN. This time is user specifiable. Routing occurs from these service centres, and selection of the preferred route to a particular destination occurs here. The choice of preferred route results in packets being queued on a particular *p_route* with each end of the *p_route* being represented by a FCFS service centre. The mean service time of these centres approximates the delay which packets experience on that *p_route* during a particular iteration.

Routing is fixed for the duration of a particular iteration. It is possible that there may not be a way of reaching all other $n - 1$ nodes in the network from particular nodes. In such instances packets which are bound for an unreachable destination are channeled to a sink. The packet life would expire in the DNN, and the packets would have to be re-sent from the originating node, with the optimistic view that they will now be routed by some alternative path which will enable them to reach their ultimate destination.

5.3.4 Interface

The modeller provides information about the network to be modelled by means of a network specification. This would typically include the network topology (including

node and line attributes), load parameters, global network attributes and evaluation attributes as shown in Figure 17.

A Graphical User Interface (GUI) was considered to be the most appropriate means of allowing network specification. A GUI allows interactive network specification, eliminating the need for a language description of the network. It is easy to modify a network specification using a GUI and it is intuitively much clearer what the topology of a network is when this can be viewed graphically. A further advantage of a GUI is that model results can be displayed directly, giving the modeller immediate feedback on network performance in a graphical manner.

Some attempts to formally specify GUIs as part of a design phase have been made. One of these involves the application of *statecharts* to the design and specification of a GUI. Recent work [VaMi91] has taken Harel's methodology for specifying complex reactive systems [Har88] and used this in an attempt to model GUI functionality in terms of states and the transitions which can occur. In GUIs based on asynchronous events the use of graphical statecharts was found by these researchers to be a 'natural' way to model concurrency. While this design approach was not adopted in this study, it is recognised that a GUI formalism such as this could be very helpful in the design of a large direct-manipulation interactive user interface.

Chapter 6

Implementation: the Xwan

Having outlined the design principles of a WAN modelling facility, we now consider the implementation of this design, and the practical realisation of the *Xwan*.

It is convenient to discuss the implementation of the system in terms of its main components and these are identified in Section 6.2. Before identifying and describing the *Xwan* components we briefly consider the software and hardware environments on which the *Xwan* is based.

6.1 Software and Hardware Environments

The *Xwan* was implemented in *C*, while the *X Window System* (X11R4) with the Athena widget set was used to create the GUI. A UNIX environment was used for software development on a Sun SPARCstation 1+ (SunOS 4.1).

6.1.1 Choice of C as Programming Language

C was chosen as the implementation language for a number reasons. In a UNIX environment *C* has a natural advantage in terms of the wide availability of compilers and the portability that it allows. The Kernighan and Ritchie language allows great flexibility and enables manipulations which would not be possible in some other languages. Another consideration in choosing *C* was the fact that there is an X Window System interface for *C*. The author's familiarity with *C* was also an influential factor!

C++ was another possible candidate, and here again there were various reasons for *not* selecting this language. The lack of a clear standard at this stage weighs heavily against choosing *C++*. *C++* compilers are not as readily available as *C* compilers, and a *C++* implementation would not have allowed the required portability of the modelling system. Using *C++* as an interface to the X Window System is possible by making use of vanilla *C* function calls. A *C++* interface to the X Window System, where the X Window System makes implicit use of *C++* objects, is not currently available. The next release of the X Window System will probably address this.

LISP and ADA interfaces to the X Window System also exist, but *C* is arguably preferable to these languages too, which suffer a similar lack of compiler availability and conformity to that described for *C++*.

6.1.2 Choice of X Window System

The X Window System was developed jointly by MIT and DEC to enable programmers to develop portable GUIs. The device-independent architecture of the X Window System facilitates this. In contrast to window systems such as those used by Apple's MacIntosh or Microsoft Windows, which support a particular style of user interface, the X Window System "provides *mechanisms* to support many interface styles rather than enforcing any one *policy*" [You89, p. 2] The X Window System thus allows great flexibility while at the same time enabling usage in heterogeneous environments. The X Window System *per se* does not provide user interface components such as scroll bars, menus or dialog boxes. These components are usually contained in higher level libraries built on top of the *X Protocol.*

The X Window System architecture is based on a client-server model. This allows programs to be run on one machine but display on another. The server communicates with clients by means of events, and X Window System applications are *event driven.* Network transparency occurs in the X Window System, and this is facilitated by the architecture. It should be noted that the X Window System has been criticised by some authors, for example [Gaj90], who have identified several deficiencies in the X protocol.

Access to the X protocol is provided through Xlib, and Xlib is the lowest-level

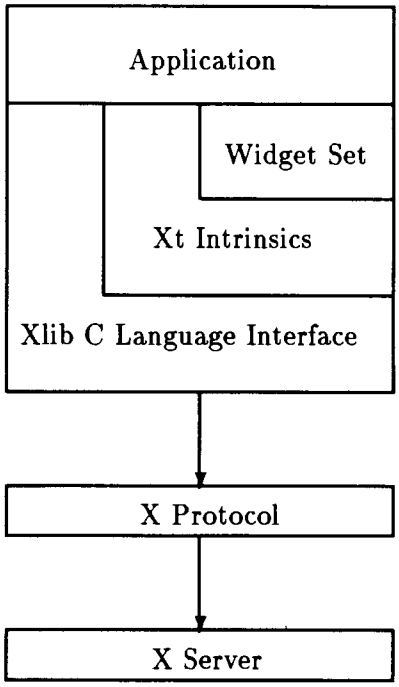


Figure 20: Architecture of the X Window System programming environment

C language interface to the X Window System, handling the interface between an application and the network. Because of the low-level nature of Xlib, higher level toolkits have been designed to be layered on top of Xlib. The most standard toolkit used by application programmers is the X Toolkit (*Xt*)[1]. Xt provides an object-oriented layer supporting the user-interface abstraction known as a *widget.* Various widget sets have been developed, providing sets of commonly used user-interface components. For the purposes of this project the Athena widget set was chosen. This widget set is the most readily available, and is distributed with the X Window system[2].

6.2 Components

[1]Some other toolkits available are the Stanford *InterViews* toolkit, the Carnegie Mellon *Andrew* toolkit, the Texas Instruments *CLUE* toolkit and the Hewlett Packard *Xray* toolkit.

[2]Commercial widget sets such as OSF's *Motif* and AT&T's *OPEN LOOK* are also available to developers.

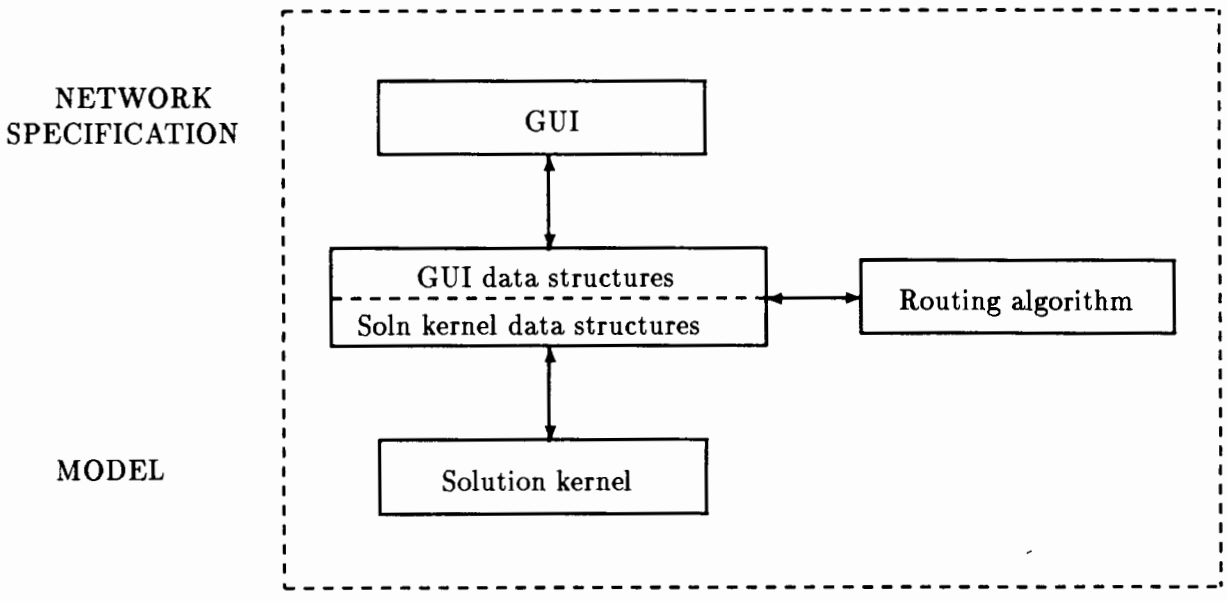


Figure 21: Components of the *Xwan* modelling system

There are two central components to the *Xwan* modelling system these being the GUI and the solution kernel. The GUI is the component with which the user interacts in describing the network to be modelled. This is the network specification phase. Once network specification is complete the modelling phase takes place. The *Xwan* takes the network specification and transforms it into a model which is solved in the solution kernel. Figure 21 shows these components and their interaction through the data structures of the GUI and the solution kernel. The network routing algorithm is identified as a separate component because it can be modified by routing algorithm designers who are using the *Xwan*. The routing algorithm 'communicates' with the solution kernel by means of the data structures which exist.

In the following sections these central components are described in detail.

6.3 Graphical User Interface

The GUI facilitates network specification. In order to introduce the GUI, and the way in which a network is specified, we begin by considering the user's view of the

Figure 22: The *Xwan* screen

GUI and the steps taken for a network specification. In the second part of this section we turn our attention to the implementation of the GUI.

6.3.1 User View

Figure 22 shows the *Xwan* screen constituting the GUI. The largest area of the screen is allocated to a *drawing palette* on which the network topology is drawn. An *icon panel* enables control of network specification through the selection of different modes. A *menu bar* appears across the top of the screen, and some of the network attributes are specified through the invocation of *popup menus*. The modelling system is activated by means of a *mouse* which is 'clicked' in various locations depending on the

particular operation. A *sprite* identifies the location of the mouse on the screen, and changes shape according to the operation being carried out. A *message line* appears at the bottom of the *Xwan* screen, and instructions and status reporting are given at this location. If a warning or error message is displayed a 'beep' accompanies display on the message line. The area below the icons on the icon panel is initially blank. During model solution this *result area* is used for reporting point to point delay and for cumulatively graphing these delays.

Having introduced the general terminology associated with regions of the *Xwan* we now concentrate our attention on each individual area in the order that a user would typically proceed.

6.3.1.1 Icons for Control

It has been said that the icon panel (Figure 23) allows the selection of different modes. The two modes of most interest to the modeller at the start of any network specification are those of *node* and *p_route* drawing. These are represented by the top left and top right icons respectively. The mode which is chosen will determine whether nodes or *p_routes* are drawn when the mouse is clicked on the drawing palette. At this early specification stage two other icons of interest are the *erase* icon and the *information* icon, located on the left and right, respectively, of the third icon row. In erase mode nodes and *p_routes* on the drawing palette can be deleted. Selecting the information icon invokes a popup window which enables the user to seek help with a particular operation.

The remaining icons are the *animation* and *point to point set* icons, in the second row, and the *delay reporting* and *utilisation reporting* icons in the fourth row. These icons are all related to specification of evaluation attributes, and will be discussed in the context of that section.

6.3.1.2 Network Topology

Depending on the mode selected, the user is able to create nodes or *p_routes* on the drawing palette. Nodes can by moved on the drawing palette and arranged to the user's satisfaction by pressing the middle mouse button while inside the region of the

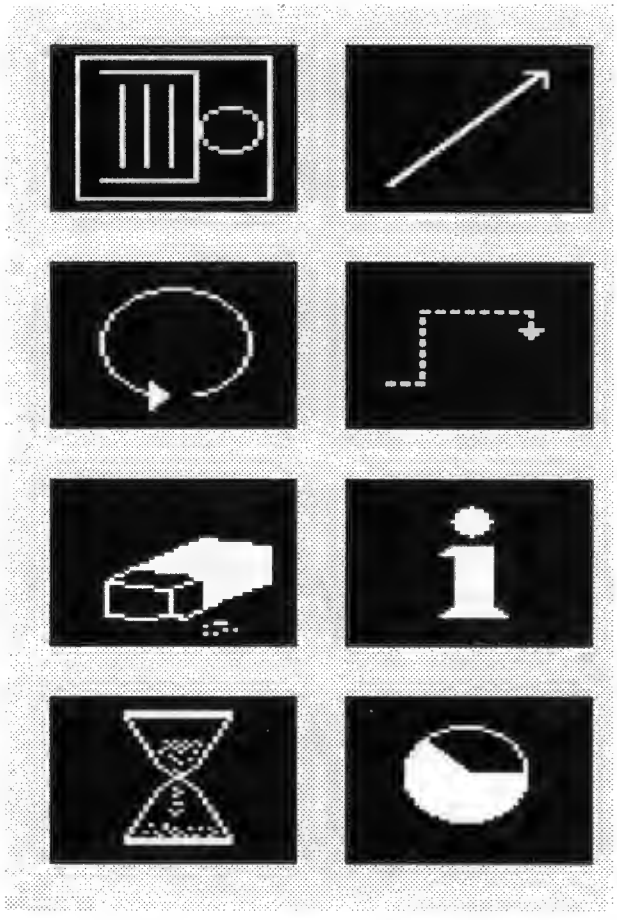


Figure 23: *Xwan* icon panel

Figure 24: Nodal attribute specification popup

node. *P_Route* drawing is carried out by selecting a starting node and then drawing a line from this node, in any pattern desired, to another finishing node. The technique of *rubber banding* is employed, allowing the user to 'stretch' the line as the *sprite* is moved around the screen.

Through this combination of node and *p_route* drawing the network topology is specified and appears on the drawing palette. Scrollbars are present on the drawing palette enabling the user to scroll networks with many nodes. If deletion of a node or *p_route* is required, the erase icon is selected followed by the object for deletion. A popup window requesting confirmation of the delete operation appears so that the modeller does not accidentally delete an object.

The *Xwan Print* menu provides users with the option of printing a hardcopy representation of the topology. An option to create a *PostScript* dump of the network topology facilitates easy generation of such a print file from within the modelling system.

Each node and *p_route* has attributes associated with it and, either during or after the node and *p_route* drawing, these must be specified to complete the topology description. In order to specify the attributes for a particular node or *p_route* the sprite is positioned on the object and the right button of the mouse is clicked. This invokes the nodal attribute specification popup (Figure 24) or the *p_route* attribute

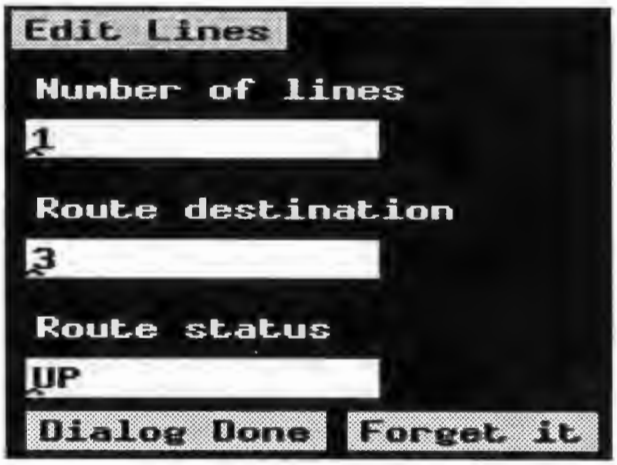


Figure 25: *P_Route* attribute specification popup

Figure 26: Line attribute specification popup

specification popup (Figure 25). The object to which the popup window refers changes colour, in the case of a node, or becomes dotted, in the case of a *p_route*.

Default attributes are provided for nodes and *p_routes* and these can be changed by invoking the node or *p_route* attribute popup from the icon panel. This is done by clicking the right mouse button on whichever object's default attributes are to be changed. When a new node or *p_route* is instantiated these defaults attributes will then apply.

Nodal attributes include the mean nodal service time, and the node capacity, representing the amount of buffer space available at the node. Node names can also be provided, giving qualitative information about a node. If specified, this name appears on the screen below the node. Once the node attributes have been specified the user is given the option of accepting (*Dialog Done*) or rejecting (*Forget it*) the attribute modification. This technique of allowing the user to 'roll back' changes is applied consistently throughout the *Xwan*.

P_Route attributes are more sophisticated than those of the node by virtue of the fact that the lines, or trunks, which physically constitute the *p_route* have to be specified. This mirrors the situation in the actual WAN, where multiple trunks may be used on any particular *p_route* to provide the desired capacity. This is a central aspect of the modelling system, which allows the user to experiment with different trunk combinations in order to determine how the network will perform when particular capacities are used. Figure 26 shows the line attribute specification popup which is invoked by selecting the *Edit lines* option on the *p_route* attribute specification popup.

A Create command button on the line attribute specification popup is used to add new lines to the *p_route*. Next and Previous command buttons allow the user to traverse the lines which they have specified, making amendments if necessary. Any amendments are stored with the use of the Save command button. A Delete command button allows the deletion of lines. As the user navigates through the different lines, using the command buttons, so the attributes of the various lines are displayed. The capacity and error rate on each line are among the attributes specified through this popup window. The user is informed of the number of lines in existence and the line

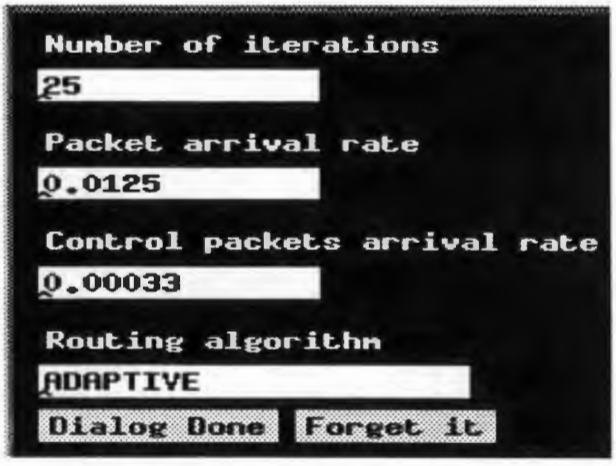


Figure 27: Global option specification popup

currently being displayed is specified.

6.3.1.3 Load Parameters

Having specified the network topology, the user would then usually specify network traffic through the load parameters. In order to represent the load on the network being modelled it is necessary for the user to specify the arrival rate of packets to nodes of the network, as well the frequencies with which these packets are destined for particular nodes in the network. The destination frequency information may be crucial to the correct functioning of the model since an assumption that exogenous packets are routed to all other nodes in the network with equal probability would be unrealistic.

Packet arrival rate is specified through a Global Options popup (Figure 27) which is invoked from the *Options* menu. The Destination Frequency Matrix (Figure 28) enables the modeller to specify what portion of the exogenous packets arriving at a node is bound for each other node in the network. The destination frequency matrix is displayed in a popup window which is also invoked from the *Options* menu.

In order to provide as much flexibility as possible, the system assumes by default that – from a given node – destinations for which a specific frequency is *not* specified are chosen with equal probability. If the user specifies that $d\%$ of packets should be

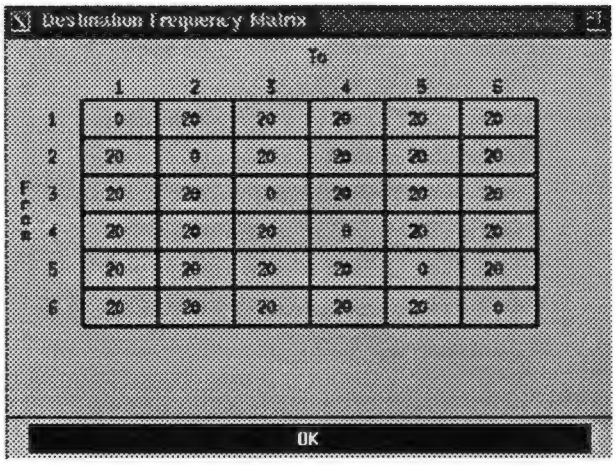


Figure 28: Destination Frequency Matrix

destined for Node x, and that $e\%$ of packets must be bound for Node y, then the destination frequency to each of the other nodes will automatically be recalculated as $\frac{100-(d+e)}{n-2-1}$. The user can specify particular frequencies to every other node, should this be desired, but in general there will be certain nodes which receive the majority of the exogenous packets from a particular node. The remaining packets will be bound for the other nodes with even probability.

6.3.1.4 Global Network Attributes

Other global network attributes can be specified by the user through the Network Options popup invoked from the *Options* menu. The specification of these attributes would typically be done at this stage and at this stage we have an example of attributes which can be considered to affect the routing algorithm too. The routing algorithm designer using the system may wish to alter, for example, the values of packet length, the regulation threshold or the trunk error threshold and these can be done here. If more than one routing algorithm is implemented in the system, then the routing algorithm to use can be specified too. This demonstrates the generality of the network specification which could be combined with other routing algorithms built into the *Xwan*.

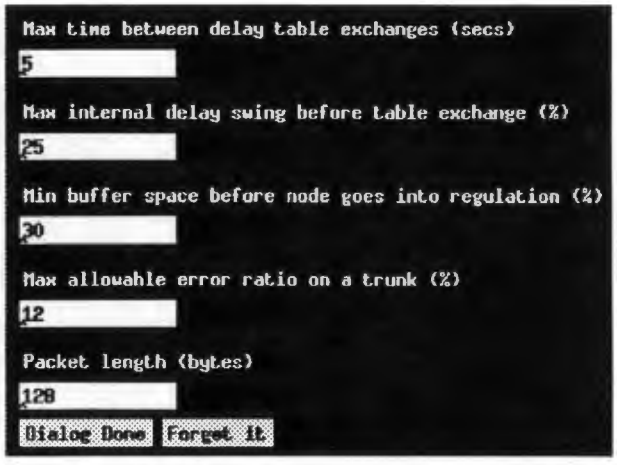


Figure 29: Network option specification popup

6.3.1.5 Evaluation Attributes

Depending on the interests of the *Xwan* user, evaluation attributes are specified before model solution is actually carried out.

In the *Xwan* the user can select the ***point to point set*** icon and then click on a starting and an ending node causing the system to show the route which packets take between these two nodes during each iteration. In this way the modeller can monitor packet flow through the network, and determine which *Lroutes* packets are following. It is possible to see the routing algorithm in action with this option and to visually observe changes in routing. Lines change colour on the screen, depicting the path between two nodes. If the situation occurs where there is no convergent *Lroute* to a destination node packets are discarded from the network. This is illustrated on the screen by routing the packets to a literal ***bit bucket*** which is displayed on the screen next to the node at which the divergence occurs.

In addition to displaying packet routing, point to point delay reporting (in milliseconds per packet) can be enabled by selecting the ***delay reporting*** icon.

It is likely that the modeller will wish to have graphs representing the outcome of different model runs. In keeping with the interactive, real-time philosophy of *Xwan* that it should provide the user with immediate, visual feedback, a graph history of the point to point delays at each iteration is displayed beneath the icon panel (Figure

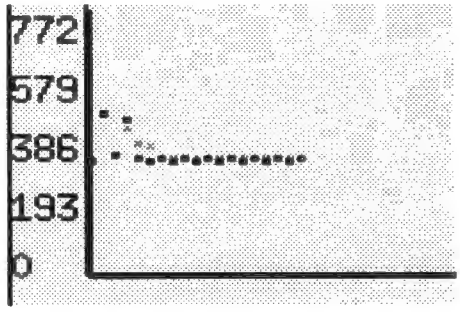


Figure 30: Graph display

30). A moving average is also plotted on the graph, enabling the user to see the average delay as influenced by the last v iterations. The graph is not meant to be a highly accurate delay indicator, but rather to provide the modeller with the general trend in the delay values. The graph is automatically rescaled and redrawn if delay values fall outside the range of an existing graph.

If more sophisticated and permanent display of delays is required, a Lotus compatible file can be generated. This can be imported into Lotus or another graphing package and high quality output can be produced. In a UNIX environment a package such as *gnuplot* draws very acceptable colour graphs, and generation of a *gnuplot* 'plot file' is also available as an *Xwan* output format. From *gnuplot PostScript* files for printing can be produced.

Utilisation reporting is another of the performance measures reported, and the modeller is able to observe utilisation on trunks and/or nodes during a model run by selecting the ***utilisation*** icon. Average utilisations are reported for trunks and nodes but, as with other averages, these can be disabled by the modeller. Similarly if specific values are not of importance in a modelling exercise then reporting can be restricted to averages or to sole graphical reporting. These choices are specified on the result display specification popup (Figure 31).

Utilisation on *p_routes* is reported graphically by the use of arrows denoting the current *p_route* utilisation (see Figure 32). One arrow represents a utilisation of less than 25%. Two arrows denotes utilisation between 25 and 50%, three arrows utilisation between 50 and 75% and four arrows indicates that utilisation is above 75%. From this graphical depiction it is very clear how *p_routes* are coping, and very

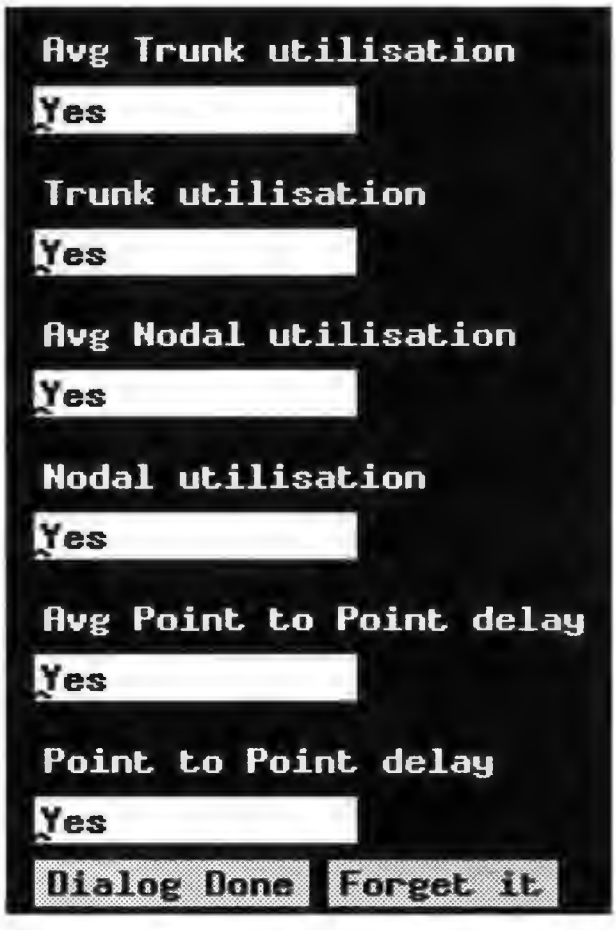


Figure 31: Result display specification popup

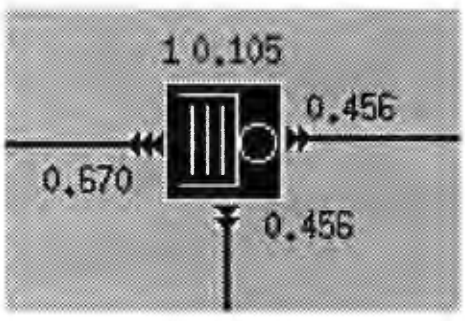


Figure 32: GUI utilisation display

high utilisation may precede the reporting of an infinite queue on a particular *p_route*. Nodal utilisation is shown above the node, to the right of the node number.

In the case of an infinite queue on a *p_route* the model stops and highlights the infinite *p_route* with four arrows in a different colour showing where the infinite queue occurred. The screen representation of such an occurrence is shown in Figure 41 in Chapter 7. If an infinite queue occurs at a node, the node changes colour and the model stops enabling the user to change trunk capacities, destination frequencies, arrival rates and nodal service times in order to see how the situation may be resolved. In this way capacity planning can be done, for a projected load, and the user can perform interactive "what-if" analysis to determine network performance with different topologies and loads.

The user can also examine files containing the routing tables at network nodes, and trace the table updates which occurred as model solution progressed. A Report option specification popup (Figure 33), invoked from the *Options* menu facilitates specification of column widths, page lengths etc. in these routing table reports.

If the modeller wishes to examine routing, or any of the reported metrics, more closely at each iteration then selection of the ***animation*** icon will pause the model after each iteration so that performance measures can be scrutinised.

The number of iterations which will be performed in solving the model should be specified by the user on the Global Options popup window. This determines the number of routing table updates that will be modelled during model solution.

6.3.1.6 Model Solution

Model solution starts with selection of the *Start solution* option on the *Results* popup menu. Before model solution can begin the network which the user has specified is checked. At a simplest level checking is carried out to ensure that all nodes are reachable, and that the user has specified at least one ***p_route*** to each node in the network. A ***p_route*** is made up of lines which have to be specified by the user. If lines have not been specified by the user model solution cannot begin.

A check is carried out to ensure that the user has set up the destination frequency matrix. The user has to set up destination frequencies at least once during network

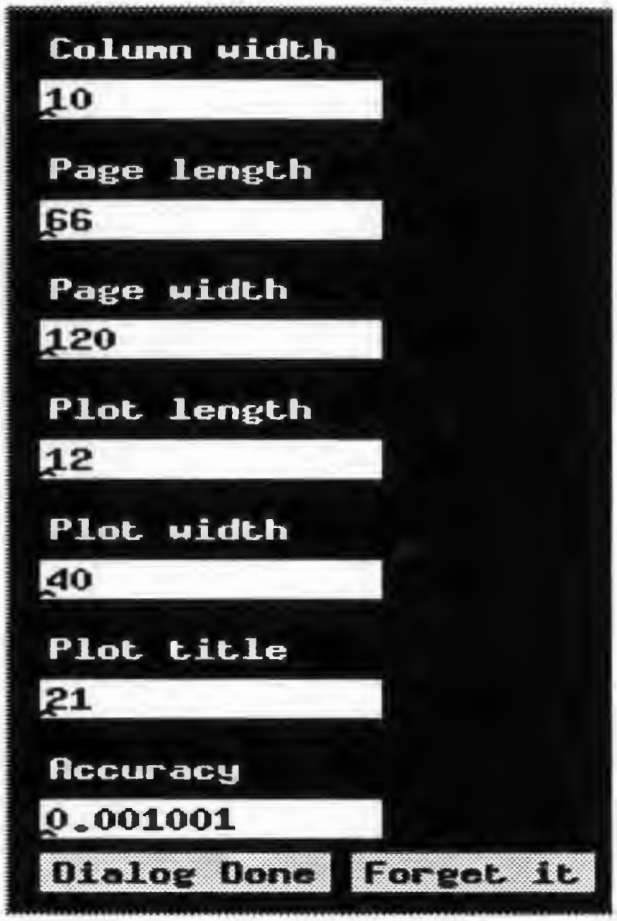


Figure 33: Report option specification popup

specification. The nature of the modelling system is such that, as model specification proceeds, the destination frequency matrix is automatically updated as nodes are added or deleted.

Model solution proceeds by transforming the network specification, which the user sets up in the manner described, into a model which is solved. Results are displayed on the GUI according to the evaluation attributes that the modeller has requested. After observing the results a user can change the topology, network load or any other aspect of the network specification (using the specification techniques over again) and respecify the network to be solved. In this way 'what if' exploration can be conducted.

6.3.2 Implementation

In terms of implementing the interface for the modelling system there are certain features of the GUI which warrant explanation, and brief description of the way in which they are implemented. Three items have been identified for brief presentation. The first describes *improvements* which the *Xwan* interface makes, when compared with other graphical systems. The second describes the X Window *widget tree* used in *Xwan* implementation, while the third presents the *GUI data structures.*

6.3.2.1 Improving on Other GUIs

An examination of other GUIs was undertaken, specifically examining GUIs of modelling systems in which similar information was specified. This examination influenced *Xwan* implementation in that positive features of graphical modelling systems were adopted and negative ones were improved, where possible.

GUI for a Specific Purpose

The fact that the *Xwan* is aimed at representing one specific network, and one particular level of that network, made GUI implementation easier than for a generalised modelling tool. In the DNN nodes are alike in their function and purpose and, while they may have slightly different attributes, they are characteristically similar and can be modelled by one object. This is in contrast to modelling tools such as CACI's

COMNET II.5[3] where, for example, nodes can be any of a number of devices such as multiplexors, concentrators, front-end processors or even gateways, each with very different meanings in terms of the model being constructed. The MACOM modelling tool (section 4.3.2) provides a GUI which allows the user to set up a network, with various elements presented to facilitate conditional routing and other characteristics. Here again there are a variety of elements which a user can select from in the MACOM *model world.*

The *Xwan* modelling system insulates the user from the low level model since this information is implicitly built into the modelling system. While the *Xwan* modelling tool is less general than others it is also simpler to use because users can concentrate their attention on salient issues of the modelling exercise rather than having to set up low level model solution components themselves every time they wish to model a network.

Default Values and Ease of Use

Default values are provided throughout *Xwan*, in contrast to some other modelling systems examined. Every dialog box which exists in a popup window or menu provides a *default value* which the user need only change if the particular attribute value is unacceptable. The concept of this approach is that the user should have to do as little mandatory work as possible – but that the user should have flexibility in experimenting with different values if so desired. A facility for re-setting default attributes was felt to be lacking in some systems tested (for example MACOM), and the *Xwan* enables users to change the default values for node and line attributes.

Another issue relating to *Xwan* ease of use stems from the fact that during the course of a modelling session a modeller may add nodes to, or delete nodes from, the network topology. Such activities are an integral part of the network specification. However these activities have implications for the destination frequencies specified for external packets arriving at each node.

In implementing destination frequency specification, an overriding principle of 'ease of use' was maintained. This suggested making extensive use of default values

[3]CACI produce a range of generalised modelling packages for LANs, WANs etc.

– which the user has freedom to change and modify, should they be inappropriate. The specific approach implemented assumes, by default, that packets are destined for the $(n-1)$ other nodes with equal probability. This saves the user from having to specify these frequencies for each model. The user can rather specify only specific frequencies, allowing the remaining packets to be routed equally between 'unspecified' destinations. Frequencies which have been specified by the user are flagged, facilitating program identification as user specified. A count of user specified frequencies is also maintained. When a node is added, user specified frequencies do not change, but the default frequencies to nodes with unspecified frequency do change and these are adjusted. The unspecified frequencies are also adjusted when a node is deleted, but here we may have the situation where packets are being sent to a node which does not exist. To counter this situation the destination frequency of packets bound for a deleted node is removed for every other node in the network. The default destination frequency is recalculated without the influence of the deleted node. In order that specified destinations are preserved when a node is deleted, nodes are re-numbered to exclude the number which has been removed. Destination frequencies are adjusted accordingly, so that user specified frequencies are maintained.

The destination frequency considerations are described as much for their demonstration of providing 'ease of use' through default values, as they are for showing the considerations which are necessary in a system where users can add and delete objects at any time. The ramifications of a topology change have to be considered in GUI implementation.

An allied problem with the addition and deletion of nodes is that the matrix used to represent destination frequencies has to change size according to the number of nodes in the network. A matrix resize operation is conducted in order to keep the destination frequency matrix at the correct dimensions and the popup window size is adjusted accordingly. Dynamic resizing ensures that whenever the destination frequency matrix is invoked it is up to date.

Icon Size

A problem identified with the icons of some of the other modelling systems was that they were too large and that it was difficult to fit large networks onto the screen. In the *Xwan* modelling system this has been alleviated to some extent by making the icons fairly small. This is in no way restrictive, nor does it detract from the visibility and ease of use. Because all nodes in *Xwan* are similar, it is not as important to have large, clear depictions of server type as may be the case in a more generic modelling system.

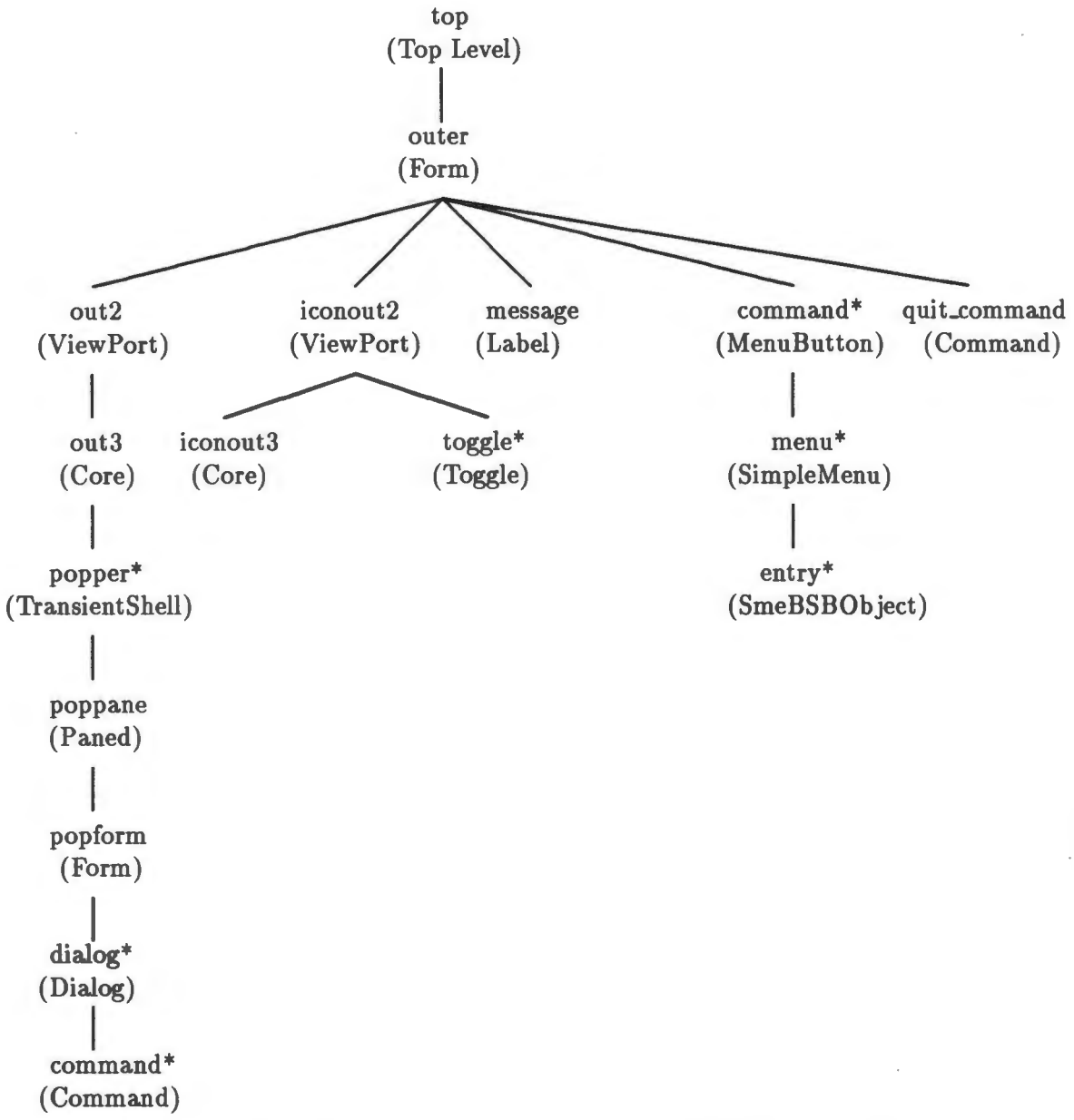
The *Xwan* icons are implemented as bit-maps. They were created using the X Window System's *bitmap* editor. Implementing the icons as bit maps was viewed as being preferable to presenting these graphics as a series of Xlib line draws through the medium of primitive graphics commands. Whereas a bitmap can be displayed as a single object, for example after an *expose* event, it would be time consuming to have to re-draw each icon using the graphics primitives. Using a bitmap implementation also allows easy editing of the icons, should this be required. Modifying graphics primitives is also more complex than interactively drawing a new bitmap.

6.3.2.2 Widgets

Figure 34 provides a composite indication of the central widget hierarchy which constitutes the GUI. This should provide an indication of the way in which the GUI was structured using X Windows, and of the relationship between different units.

6.3.2.3 GUI Data Structures

Figure 35 shows the model data structures used to represent the network topology specified by means of the GUI. An array of *node* structures is used to store information associated with each node, including the delay tables which are used by the routing algorithm. Linked list implementation of *p_routes* provides dynamic memory allocation as the user specifies *p_routes* on the drawing palette. Line segments representing the 'pieces' comprising a *p_route's* image on the screen, are stored as a list associated with the particular *p_route*. A number of lines may be specified as the



command* = file_command, options_command, run_command, modify_command

menu* = file_menu, options_menu, run_menu, print_menu

toggle* = nodecomm, linecomm, stepcomm, setcomm, delcomm, infocomm, utilcomm, delaycomm

popper* = popper, rtpopper, lnpopper, confpopper, globpopper, reppopper, netpopper, animpopper, helppopper, resdisppopper

Figure 34: *Xwan* central widget tree

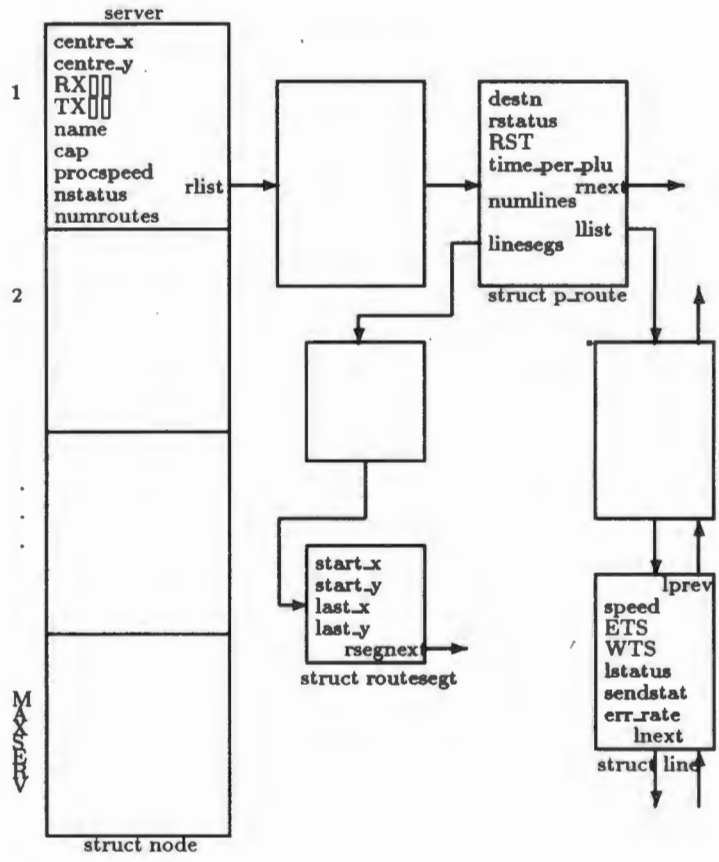


Figure 35: Data structure representation of nodes and *p_routes*

constituents of a *p_route*, and these are stored as another linked list allowing easy addition and removal of lines on a *p_route*.

These are the central data structures used to represent the system image, and its display on the GUI. The screen locations of line segments and nodes are stored so that screen refreshing is facilitated, and so that the offsets for text can be determined. These are also necessary for saving a model in order to restore the GUI display of the model accurately.

6.4 Solution Kernel

As described in the discussion of model design principles, a multiclass queueing model representation of the network forms the basis of the solution technique. Through solving the queueing model analytically, and using the results to update the nodal delay tables using the network routing algorithm, it is possible to model adaptive routing.

An open *packets* workload with n classes is used to represent an n node network, and packets are routed according to these classes. A second open *control packets* workload is used to represent control packets which travel between adjacent nodes in the network. Queueing network service centres are used to represent the nodes and trunks in the network in the manner depicted in Figure 18. The queueing model represents each node as a PS service centre where the mean nodal service time is user specified. Trunk delays are represented by FCFS service centres with mean service times representing the applicable trunk delay.

Packets are routed according to the nodes and *p_routes* created in the network specification. Exogenous packets are probabilistically allocated to one of the $n-1$ classes from a particular node depending on the user specified destination frequencies. Packets are routed by the class to which they belong, and depending on the preferred route at a particular iteration, one of the *p_routes* emanating from the node will be selected for packet forwarding.

The queueing model results are obtained by calling the MVA solution routines of MicroSnap. The information returned from these routines is used to update the

routing tables in order that routing information can be recalculated. In this way routing table updates are automated, using the information from model solution at each iteration.

Consideration is now given to the interaction between the *Xwan* and the MVA solution routines of MicroSnap. Two approaches were identified and these are described in terms of an *indirect* and a *direct* solution technique.

6.4.1 Indirect vs. Direct MVA Solution

One technique which could be employed is to have the *Xwan* generate SNAP/L [Boo84b] code. This code could then be used as a MicroSnap specification, being parsed and providing a file of results. This would however be unsatisfactory in terms of the DNN modelling facility required. It would be a very inefficient solution for a system which attempts to provide results in real time. In some senses this indirect approach would be an easy one, in that generating SNAP/L code would be a simple matter. The most fundamental problem with this approach is that it would be difficult to incorporate MicroSnap results into the modelling system in order to update the routing tables and generate the revised model. This sequence of generating files, parsing, solving, scanning a results file and updating the routing tables is far too clumsy and inefficient to merit implementation.

A preferred approach – and the one implemented – is to interface directly to the MicroSnap MVA solution routines, bypassing the parsing of a language description. Since the MicroSnap source code was available this was a realistic possibility.

As the MicroSnap recursive descent parser proceeds, it issues function calls which build up an image of the network being modelled in the MVA data structures. This image of the network, once complete, is solved. As the result producing statements of the language description are examined, they too call functions which determine the desired results. By virtue of this structure, and a degree of modularity contained within the MicroSnap design, it was possible to generate the appropriate function calls required to build up a system image directly, without going through the language interface. In order to realise this approach it was necessary to generate class names according to the user's network specification.

Information about network topology and load is provided by the user, and this information can be transferred directly into the MicroSnap data structures by emulating the sequence of calls to the functions which build up the MicroSnap model image. This direct approach is a very effective way of speeding up the modelling process. Another advantage of this approach is that model results can be directly manipulated and interpreted within the system enabling results to be incorporated into the routing tables and other *Xwan* data structures, with these being updated directly at each iteration.

Since the topology of the network and the applied load are assumed to be constant, the changes that occur between iterations of the network pertain to the routing which occurs for particular classes at particular nodes, and also to the service times which change for different iterations. The fact that only *aspects* of a model are changing between iterations can be exploited, and it is in fact possible to modify an existing queueing model, changing the routing and service times, rather than having to reconstruct the whole model from the start. This corresponds well to the iterative approach which has been outlined, in that the queueing network can be solved for a particular iteration of the routing algorithm model, the routing tables can be directly updated with the results of MVA solution, the routing algorithm can be called – making use of this latest information – and the queueing model can be modified in order to represent the new state of the system. The process can be repeated as many times as required, with the queueing model being modified at each iteration. This approach enhances efficiency and sustains the real time attainment considerations.

6.4.2 Automation of Routing Table Updates

In order to automate the routing table updates it is necessary to represent the network in a way which will facilitate it's modelling. To this end various tables have been used (Figure 36).

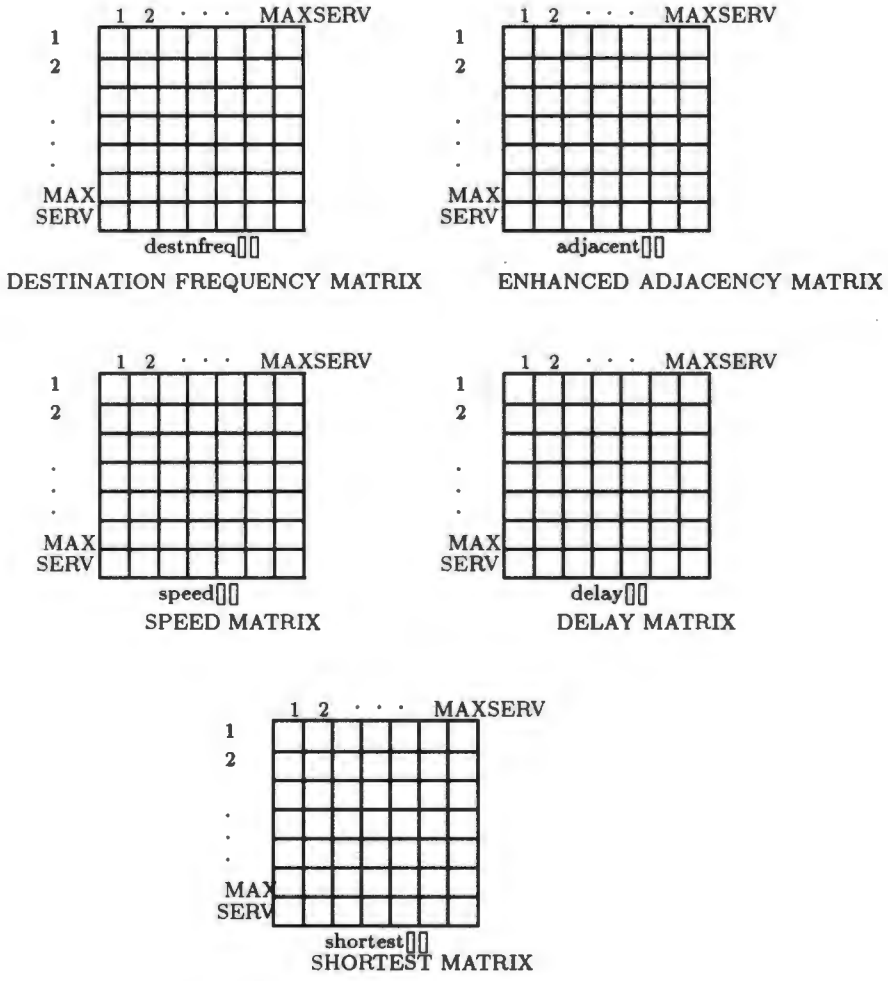


Figure 36: Matrices used in model solution

Enhanced Adjacency Matrix

In order to know which nodes of the network are adjacent an adjacency matrix is used. The matrix has been modified so that instead of just representing adjacency or non-adjacency (1 or 0) it also indicates the *p_route* on which the adjacency occurs. This is important for relating the correct *p_routes* on the routing tables when delay information is exchanged.

Delay Matrix

A delay matrix is used to store the waiting times on each trunk. The values in this matrix are initially the route service times (equal in both directions), but as the model proceeds the values represent the internal delay as calculated for each *p_route*. These delays are placed into the RX table at each node when it is updated at each t_i.

Speed Matrix

A speed matrix is used to store the trunk speeds. This matrix is used in the initial calculation of delays, and may be used at some stage for the modelling of error rates too.

Shortest Path Algorithms

As part of the update problem it is necessary to initialise the model with delay values, and after calculating trunk waiting times at t_0 shortest paths are calculated using these delays. Since the network uses adaptive routing, it is not a case of selecting the best *Lroutes* and then transmitting all packets on those *Lroutes*. Each node has *p_routes* to adjacent nodes. When routing is determined, consideration is given to the delay offered on each of the *p_routes*. In order to initialise the model, and to use shortest path – via a particular *p_route* – as the measure of delay, it was necessary to calculate shortest path delays, and then to use these in the following way. Node n has known delays to adjacent Nodes $n+1$ and $n+2$. To calculate the delay to each node in the network from Node n, the delay from n to $n+1$ (or $n+2$) is added to the shortest paths to other nodes in the network as calculated at $n+1$ (or $n+2$). In

this way an approximation of delays is made – using the adjacent node's information and delay to the adjacent node – in such a way that the shortest path *by routing on a particular p_route* is chosen to represent delay.

Various shortest path algorithms were examined [Nol81, Deo74, SwTh81] and three algorithms were considered:

- The Bellman-Ford Algorithm. This algorithm finds the shortest paths from a given source node to all other nodes, iterating on the number of arcs in a path. The algorithm proceeds by finding shortest path lengths with paths of at most one arc, then shortest path lengths with two arcs, and so on. In the worst case the amount of computation required for this algorithm is $O(N^3)$ where the algorithm is iterated $N-1$ times, each iteration is done for $N-1$ nodes, and for each node $N-1$ alternatives may have to be considered.

- The Dijkstra Algorithm. Again this algorithm finds shortest paths from a given source node to all other nodes, but in this case path length is iterated on. In determining shortest path lengths, the algorithm finds shortest paths in order of increasing path length. Each step of this algorithm requires a number of operations proportional to N, and steps are iterated $N-1$ times. This creates a worst case where the amount of computation required is $O(N^2)$.

- The Floyd-Warshall Algorithm. This algorithm finds the shortest paths from all nodes to all other nodes, iterating on the set of nodes allowed as intermediate nodes on the paths. This algorithm proceeds by calculating shortest paths where only node 1 can be used as an intermediate node, then calculating shortest paths where nodes 1 and 2 can be used, and so on. Each of the N steps must be executed for each pair of nodes, creating computational requirements of $O(N^3)$.

It was decided to implement the Dijkstra algorithm, and this algorithm is used to build up the RX table at each node in order to initialise the modelling process. The algorithm makes use of the values in the Delay Matrix, and places the shortest path lengths in the matrix Shortest.

6.4.3 Modelling Procedure

Figure 37 portrays the modelling procedure, with reference to the tables described in the previous section. The procedure described represents the static and dynamic phases referred to in the discussion of modelling system design.

The demarcated top half of the figure depicts the sequence of events constituting the *static* phase. Initialisation of the various matrices is a central activity. Dijkstra's shortest path algorithm is used to make an initial approximation with which to initialise the delay tables. The enhanced adjacency matrix is referred to during the static and dynamic phases, and facilitates correct *p_route* updating and communication. Initial RX tables are constructed as the final outcome of the static phase, using the delay table value for the initial internal delay, and shortest path information to find the shortest *l_routes* from the adjacent node on each *p_route* to all other nodes in the network.

The *dynamic* phase of model solution is represented by the 'looped' activities which are iterated as many times as the modeller requires. TX table construction forms the first stage of a delay table exchange and is the first step of the dynamic phase. Table exchange occurs, followed by the determining of *preferred* routes from the RX tables. These preferred routes are determined by finding an *l_route* with least delay to each destination. When constructing the queueing model – setting up the system image for MVA solution – the preferred routes are used to determine which *p_route* *n*bound packets are queued on at each node. The delay matrix provides trunk service times, representing the delay incurred on the trunk as the FCFS queueing centre service time. The queueing model is solved, and average queue lengths are determined according to the routing specified in the model at this iteration. The average queue length is multiplied by the *p_route* service time to provide an approximation of the average delay incurred by packets with the current routing scheme. The internal delay values in the RX tables are updated with this product. The average waiting time on each *p_route* is placed into the delay matrix, for use as the trunk service time during the next iteration.

Various other results are stored, and utilisations and point to point delays are calculated at each iteration too. Checks are carried out to see that no infinite queues

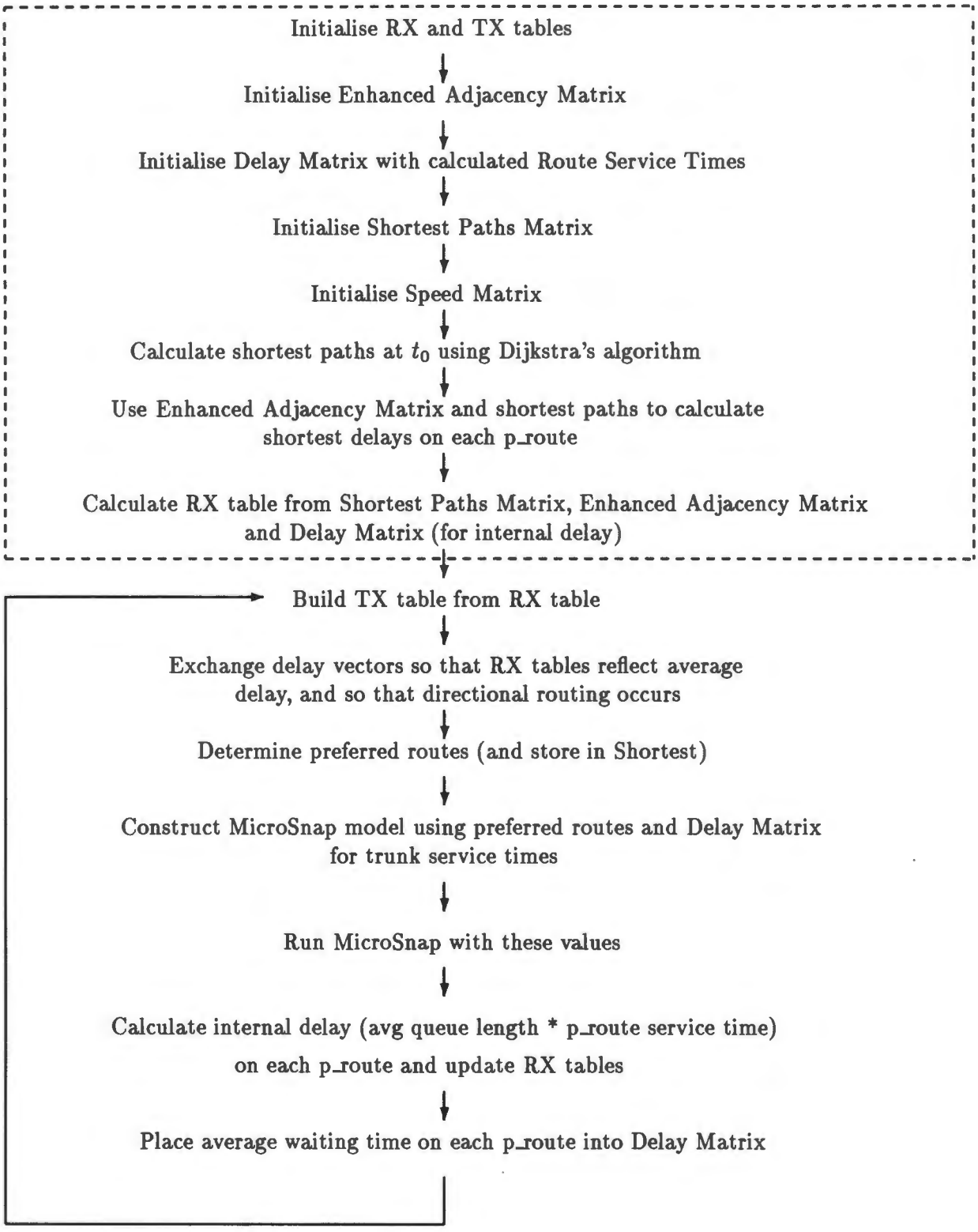


Figure 37: Modelling procedure

have occurred at trunks or nodes, and regulation state checking is also done. Regulation triggers a routing table update, as it does in the network.

Chapter 7

Model Application

In order to test the modelling system it was decided to construct a number of network models. For each of these models a series of experiments was conducted in order to observe network behaviour under circumstances typical of network operation. It was unfortunately not possible to verify these experimental results with measurements from the DNN since test data were unavailable, but experience with the behaviour of the network and insight into network functioning have been used to critically assess the results produced.

Experiments were carried out interactively, in the manner in which a modeller would typically proceed with network investigation. The immediate reporting of results and the display of network status as model solution progresses, enables the modeller to identify bottlenecks, heavily utilised trunks or nodes, the availability of buffer space and other such characteristics of interest in determining network performance. Through observing that, for example, a *p_route* is being heavily utilised, the modeller can modify the capacity of the trunks constituting the *p_route* and determine how this will affect network performance. Metrics such as utilisations, delays and buffer space are typically of interest to network managers.

As has been suggested, the use of a modelling system such as *Xwan* is not restricted to the *operational* and *tactical* decisions relating to the implementation and tuning of specific networks, and it can also be used for *strategic* decision making, for example in long term capacity planning [Ter87]. In addition such a modelling system

is also a valuable *engineering tool* for use in optimising the performance of network software. With the application of a steady and constant load, such as *Xwan* allows, it is possible to observe the performance of the network software in a *controlled* environment. This suggests that the routing algorithm can be changed, with the effects of a modification examined using the modelling system, before going to the lengths of implementing the change in a live network. Without a modelling tool the effort required to investigate a small, but possibly significant, routing algorithm change is of such a magnitude that routing algorithm designers are unlikely to be able to experiment with the optimisation of routing algorithm (and network) performance very easily.

This chapter proceeds with an initial discussion of model validation followed by a description of experimentation illustrating the utility of the modelling tool. Examples of routing algorithm manipulations are then described demonstrating the ease with which the effect of a routing algorithm change can be explored, and the interesting results which can be obtained. It was stated in the early pages of this discourse that much of the theory relating to networks has *succeeded*, rather than *preceded* the practice. In this light it is proposed that the theoretical modelling of the DNN has revealed some potential changes which could be made in order to improve the performance of this Wide Area Network.

It should be mentioned at the outset of this chapter that the experience with the modelling system has been that the modeller builds up an intuitive understanding of the network by selectively examining the routing between different points in the network. In this way the modeller can ascertain which *Lroutes* packets are choosing. By modelling a number of source–destination pairs, and observing the routing between them as it is animated on the GUI, the user can build up a complex picture of packet behaviour. It is difficult to convey the subtleties of characteristics observed during first hand modelling experience, but the graphs which are presented in this chapter represent selected behavioural patterns which are discussed in the context of the rest of the network for illumination.

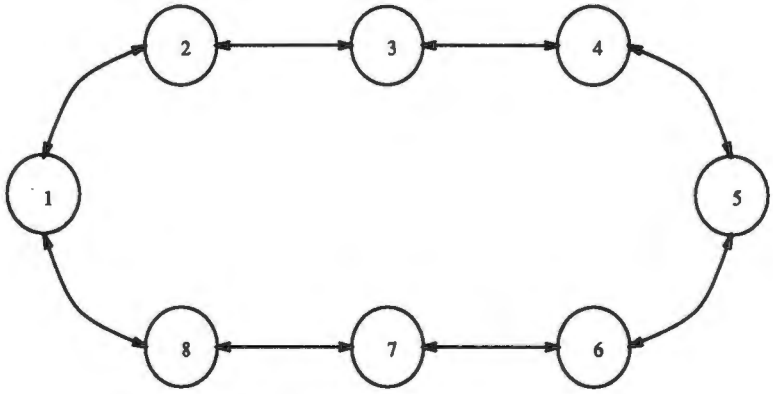


Figure 38: 8 node test network

7.1 Validation

The first network model introduced is used to demonstrate that the routing algorithm, and the modelling system, are intuitively correct and that their results are as would be expected under the circumstances tested.

The network model is an 8 node network arranged topologically in a ring (Figure 38). The destination frequency of traffic entering the network at each node has in all cases been modelled with $\frac{1}{n-1}$ percent of the exogenous packets being sent to every other node in the n node network. In this model high speed trunks were modelled, with each trunk having a data rate of 64 Kbps. Trunk error rates of 2 percent were specified on each trunk. The trunk error rate was calculated from a bit error rate of approximately 2×10^{-5} if

$$p = 1 - (1 - e)^l$$

where p is the trunk error rate, e is the bit error rate and l is the packet length. An arrival rate of 0.00125 packets per millisecond was used in the first application, while a rate of 0.125 was used in the second. A mean service time of 5 ms was specified for each network node.

The ring topology of Figure 38 was tested in order to observe direction changes in the routing algorithm, and to verify that the routing algorithm was operating as anticipated, in a topology where there is symmetric connectivity and equal *p_route* speed.

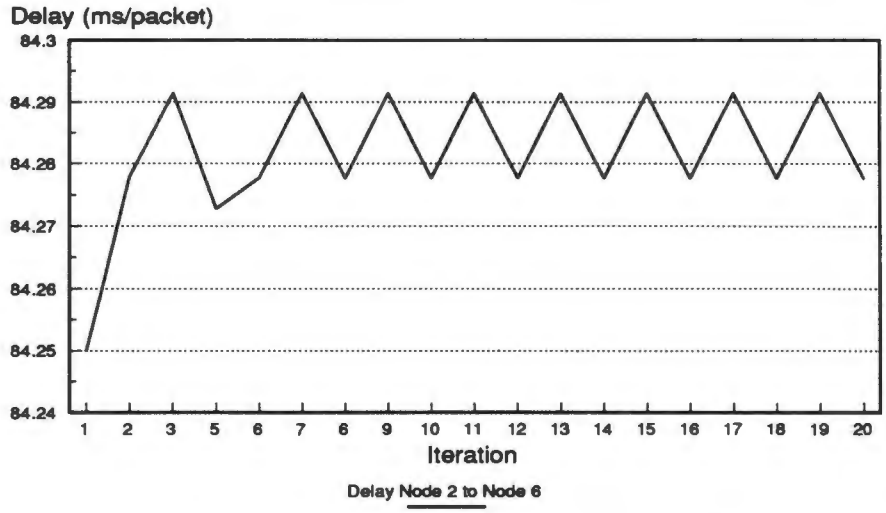


Figure 39: Adaptive routing from Node 2 to Node 6

7.1.1 Alternate *L_Route* Selection

This experiment was conducted to verify that when equal length *Lroutes* (paths) exist, packets will be routed alternately along these two equal length *Lroutes*. This behaviour is shown in Figure 39. As packets are routed along the *p_routes* of the first *Lroute* so internal delay increases. This makes the second *Lroute* the preferred route at the next iteration, when a new routing decision occurs based on the state of the system during the current iteration. The initial fluctuation occurs as load spreads through the network and the pattern of routing stabilises.

7.1.2 Increased Load Leading to an Infinite *P_Route*

This experiment shows the effect of increased load, where the load sharply becomes unmanageable and leads to infinite delay. The steepness of the graph (Figure 40) shows how the *p_route* between Node 2 and Node 6 becomes rapidly unusable until an infinite delay results.

The GUI displays very clearly which *p_routes* have resulted in infinite queues (as shown in Figure 41), and the modeller is easily able to identify these network components and remedy the situation by making appropriate changes. These can

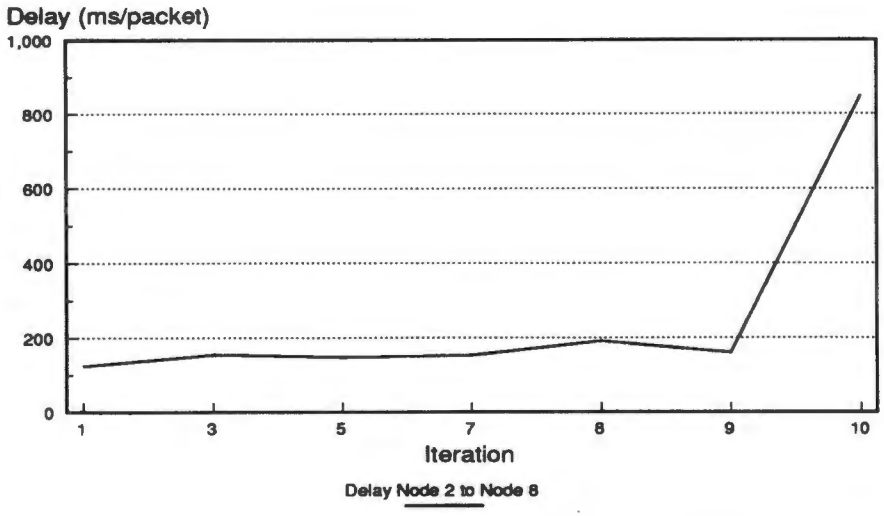


Figure 40: Routing from Node 2 to Node 6 leading to an infinite *p_route*

then be tested. In this case the provision of increased capacity will facilitate decreased queue lengths and shorter delay times.

7.2 Experimentation

The second model introduced is a 6 node network, with the topology carefully chosen to represent various connectivities. The topology displays different levels of *reachability* in order to portray varying connectivity. As in the 8 node network described earlier, the destination frequency of exogenous traffic is $\frac{1}{n-1}$ percent to every other node in the network. Unless otherwise stated, single trunks were used to constitute each *p_route*, with data rates of 9600 bps and trunk error rates of 2 percent. A mean service time of 5 ms at each network node was specified and an initial arrival rate of 0.0125 packets per millisecond was modelled.

Figure 42 shows the chosen topology of the 6 node network. Node 1 is reachable via two *p_routes*, while Nodes 3 and 4 are each adjacent to three other nodes, and Node 5 is adjacent to four other nodes. Node 2 is connected to every other node in the network and Node 6 is only reachable via one *p_route*. In constructing a topology which allows demonstration of the routing algorithm and the modelling tool

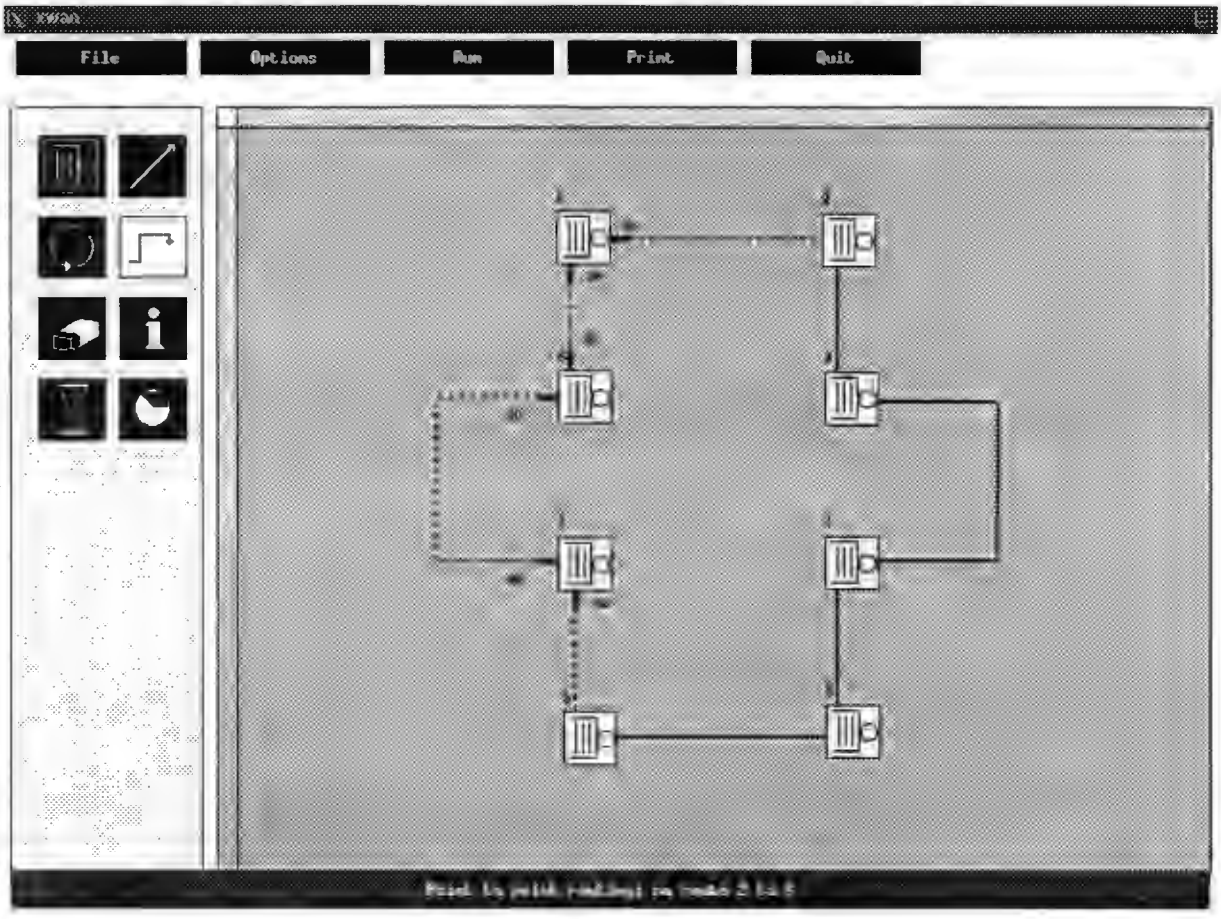


Figure 41: GUI display of infinite *p_routes*

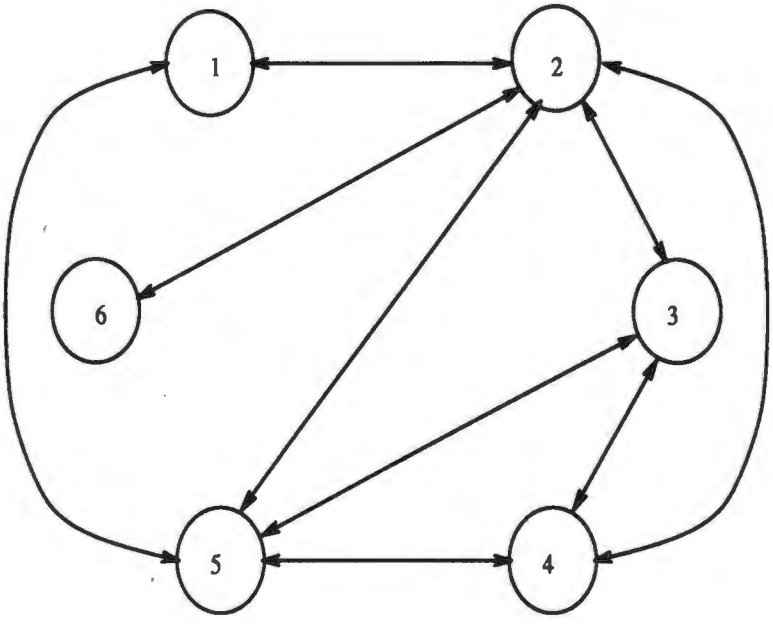


Figure 42: 6 node test network

under various conditions, it was felt that the topology described had the potential for incorporating a wide range of interesting scenarios.

Figure 43 shows the GUI display with the described network topology specified.

7.2.1 Constant *L_Route* Selection

The graph of the point to point delay between Nodes 1 and 4 of the six node network (Figure 44) depicts an initial oscillation between different *Lroutes* before the packets between these nodes settle into a constant pattern. The early oscillations, where packets are routed to Node 4, first via Node 2 and then via Node 5, are exaggerated as the other traffic in the network also attempts to determine *Lroutes* which will minimise delay and maximise throughput. In this experiment the topology and load are such that a levelling out of the graph occurs and packets follow one *Lroute*, via Node 5, without changing or being affected by changing utilisation on *p_routes* constituting the *Lroute*.

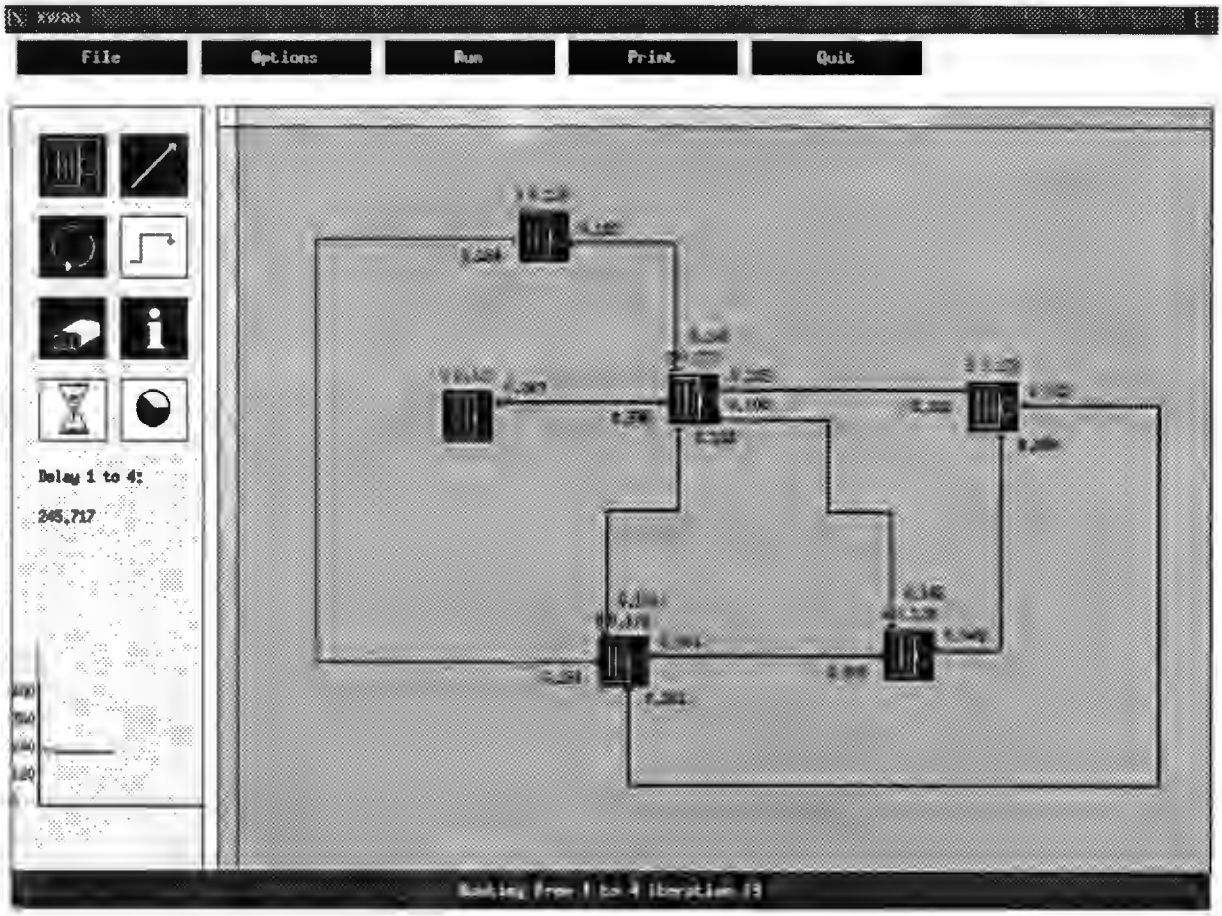


Figure 43: Screen display of 6 node test network

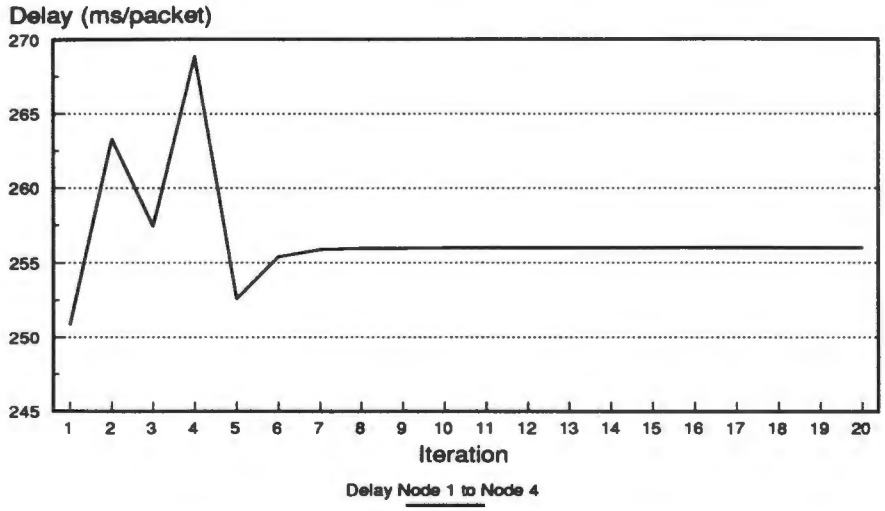


Figure 44: Constant *Lroute* selection

7.2.2 Single *P_Route* Delay

The *Lroute* chosen by packets bound for Node 3 from Node 5 is a direct *p_route* on which packets are routed without change. Figure 45 shows how a decrease in delay occurs as other packets are routed along *Lroutes* that do not include the *p_route* from Node 3 to Node 5. An increase in delay occurs again as the traffic which moved onto another *Lroute* moves back onto the *Lroute* incorporating the Node 5 – Node 3 connection.

7.2.3 The Effect of Divergence

This experiment examining routing from Node 2 to Node 6 demonstrates the effect of a divergent *Lroute* existing – the reason for the decrease in delay during the second iteration (Figure 46). Node 5 becomes divergent thus decreasing the load, and hence the delay, on the *p_route* from Node 2 to Node 6. Node 4 routes to Node 6 via Node 5, as do Nodes 3 and 1 during the second iteration so none of these nodes is able to reach Node 6 during the second iteration. The situation improves, however, with all nodes finding convergent *Lroutes* to Node 6 in subsequent iterations, and a plateau occurs with an associated levelling out of the delay from Node 2 to Node 6. The fact that

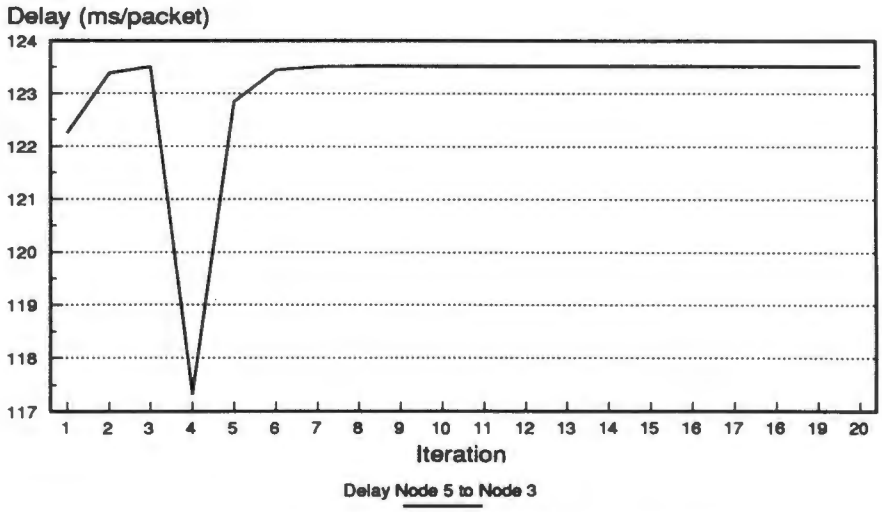


Figure 45: Single *p_route* from Node 5 to Node 3

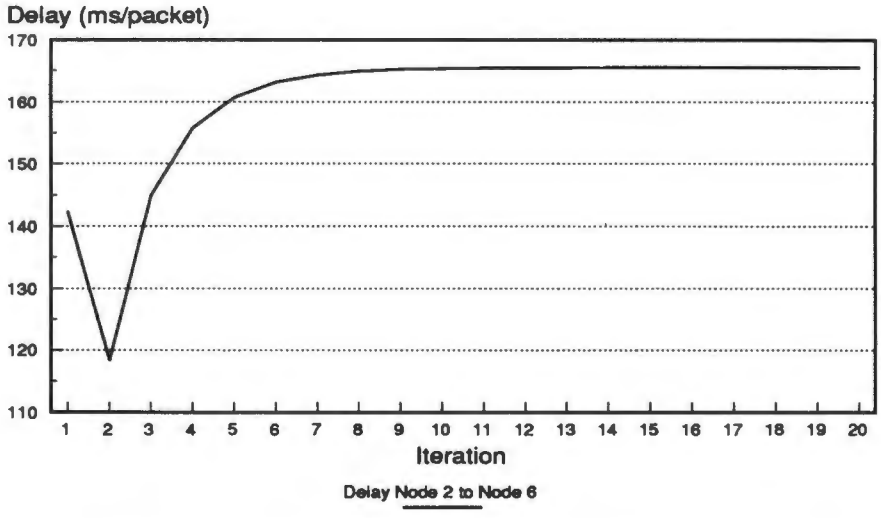


Figure 46: Routing from Node 2 to Node 6 with the effect of divergence

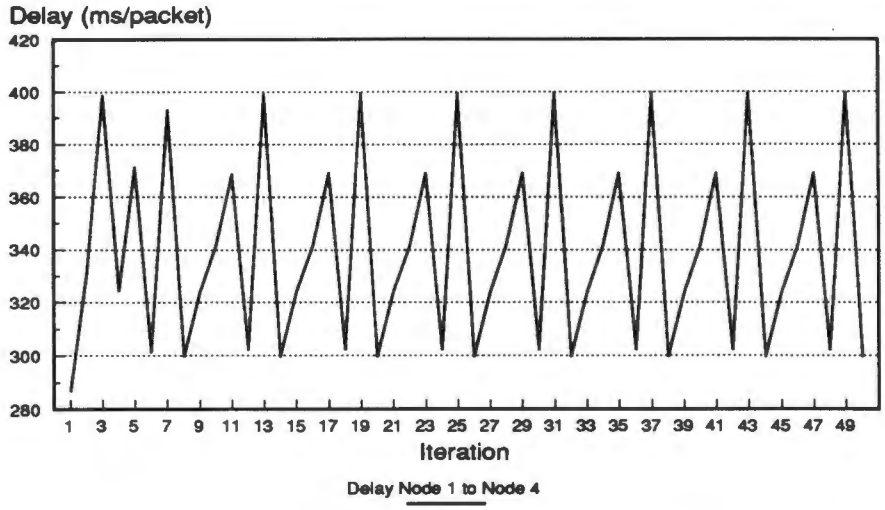


Figure 47: Routing from Node 1 to Node 4 under increased load

divergence to Node 6 occurs, is as a result of Node 5 initially informing its adjacent nodes that it has a convergent *l_route* to Node 6 but then receiving information from Node 2 which causes it to make its received delay information *divergent* on the *p_route* to Node 2. This results in packets bound for Node 6 being discarded from the network at Node 5, with a resultant improvement in delays (such as that between Node 2 and Node 6) because packets are not being forwarded on that *p_route* during that iteration.

7.2.4 Increasing Network Load

This experiment, in which the packet arrival rate is doubled, demonstrates oscillations occurring as packets are routed along alternate *p_routes*. Utilisations increase, with all *p_routes* still able to accommodate this traffic except for the *p_route* between Nodes 2 and 6. This *p_route* develops infinite queues necessitating an increase in trunk capacity. The addition of a second 9600 bps trunk enables the network to continue functioning without any infinite queues developing. As the graph (Figure 47) shows, alternate *p_routes* are selected, and on the *Xwan* GUI we are able to observe the packets being routed via Node 5, and then via Node 2. It is interesting to contrast the delays and *l_routes* experienced in this experiment with those of Section

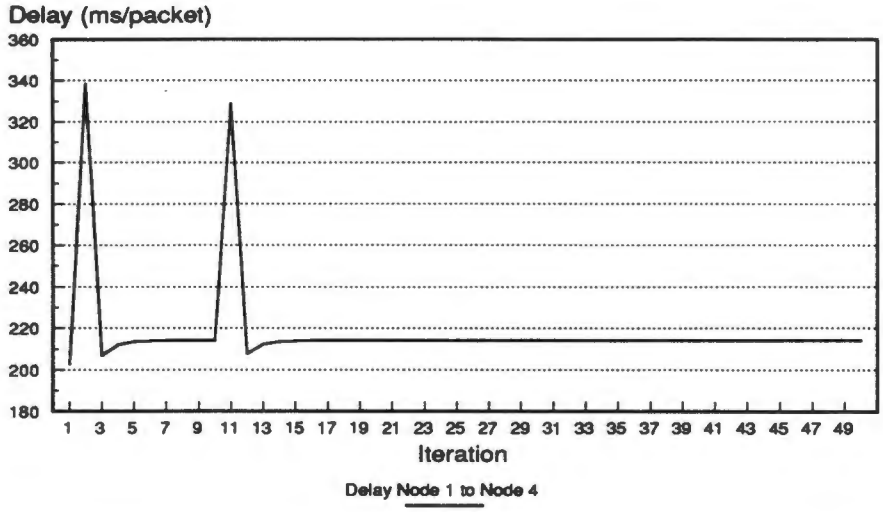


Figure 48: Routing from Node 1 to Node 4 with increased load and increased capacity

7.2.1, which showed the delay between Nodes 1 and 4 under light load settling to a constant value with packets following a single preferred path.

7.2.5 Increasing Capacity to Accommodate Load

Various candidate trunks were identified for capacity increase in order to eliminate the oscillatory behaviour between Node 1 and Node 4 which occurred under the increased load described in section 7.2.4. The *p_route* between Node 4 and Node 2 was given an extra 9600 bps line, but this failed to produce satisfactory behaviour. This extra trunk was then removed from the *p_route* between Nodes 4 and 2, and placed on the *p_route* connecting Nodes 1 and 2. Again this did not create the desired effect and further experimentation seemed necessary. Placing a second 9600 bps line on the *p_route* connecting Nodes 1 and 5 was discovered to exhibit the best improvement in stability and in delay. Increasing the capacity of this *p_route* specifically, was found to take some of the load off Node 2 and improve network performance by strategically removing packet build up occurring at Node 2.

A sharp increase in the delay (Figure 48) at iteration 10 occurs when routing from Node 1 to Node 4 is conducted via Node 2 for one iteration. As the network settles it

can be observed that a far more stable nature prevails in the routing from Node 1 to Node 4 if comparison is made with Figure 47. This characterises the improvement in the rest of the network as well where Node 2, by virtue of its high connectivity (and the fact that it is the only node from which Node 6 can be reached), is now able to cope with traffic demands more easily because of the diffusion of packets onto other *Lroutes*.

7.3 Routing Algorithm Investigation

In order to demonstrate ways in which the *Xwan* modelling system facilitates routing algorithm investigation and exploration, three modifications are presented. In the first investigation the *trunk error threshold* is investigated. The second investigation focuses on the *regulation threshold* applied by the routing algorithm. A third routing algorithm modification was conducted to ascertain the effect of modifying the internal delay *bias factor*. A final investigation demonstrates the application of a different adaptive routing algorithm.

7.3.1 Trunk Error

In the network studied, a trunk error threshold of 12 percent is specified, above which trunks becomes unusable. Figure 49 shows that for a 9600 bps trunk with an error rate above 10 percent delays start increasing very quickly, and by the time an error rate of 12 percent occurs, a trunk is already unusable in practical terms. Specifying 12 percent as the error threshold appears to be too high, and the modelling endeavour suggests that a threshold of 10 percent (or below) would be more effective in eliminating the need for 'recovery' action once a trunk has been rendered unusable. In this way the routing scheme could act *pro*actively rather than *re*actively and, anticipating the loss of a trunk, route packets along alternate trunks without a large number of retransmissions becoming necessary. If there is only one trunk on a *p_route* then loss of the trunk obviously causes loss of the whole *p_route*, with potentially serious consequences.

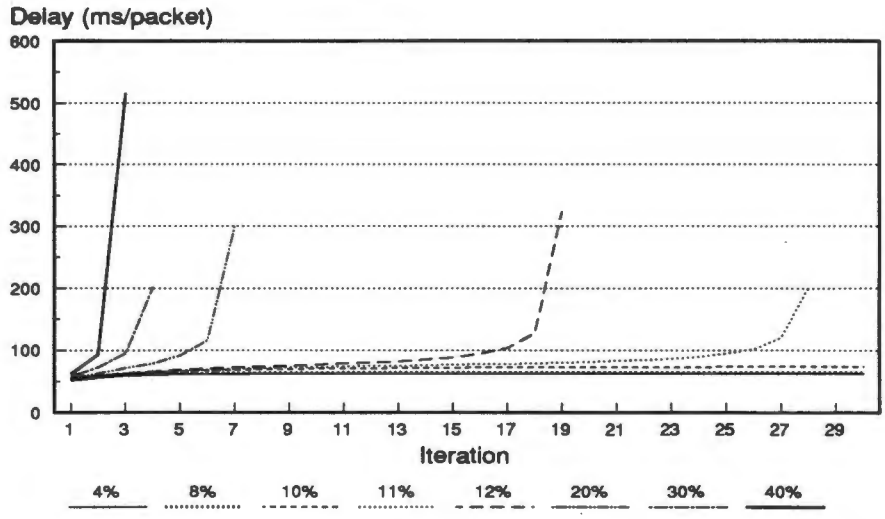


Figure 49: The effect of trunk error

7.3.2 Regulation Threshold

Regulation occurs when a node's buffer space becomes critically low. When this happens a node informs its adjacent nodes that it is no longer able to accept *endogenous* traffic so packets should be routed elsewhere. In the network studied, a threshold of 30 percent has been specified meaning that as soon as less than 30 percent of the buffer space is available at a node, the node should go into regulation. One of the experiments was to examine this threshold and to determine whether a threshold of 30 percent is in fact the most effective in terms of minimising network delay and maximising throughput. Whatever margin is specified, allowance must be made for the buffering of exogenous arrivals (a node in regulation continues to accept packets arriving from external sources) and of in-flight packets which were routed to the regulation node before news of its regulation state reached other nodes.

In order to investigate the setting of this threshold value, the 6 node network was modelled with the buffer capacity of Node 2 being reduced in order to ensure that regulation would occur. By varying the regulation threshold it was possible to see the effect that this has on delay in the network. The delay from Node 3 to Node 1 is presented in Figure 50, being representative of the way in which network delay responds to the regulation state of Node 2.

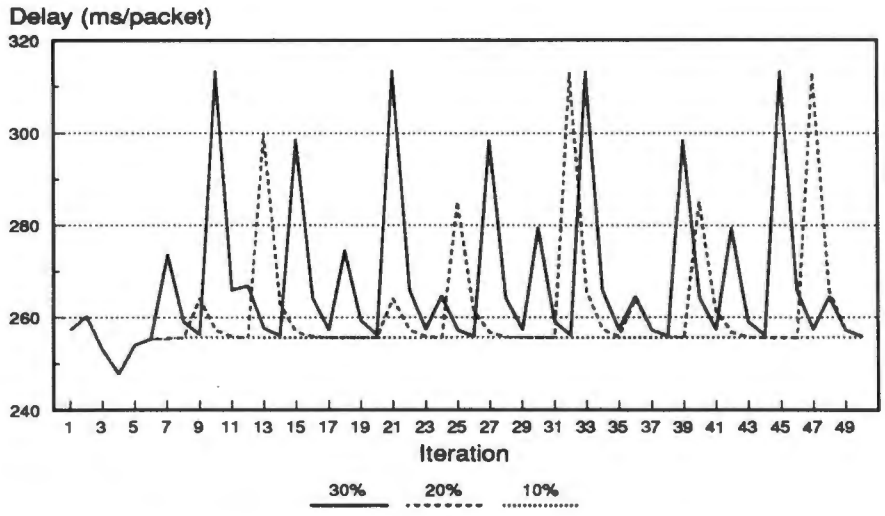


Figure 50: The effect of changing the regulation threshold

Initially the threshold of 30 percent was modelled, demonstrating the situation as it exists in the network studied. It should be noted that at this level Node 2 moved in and out of regulation with regularity, as shown by the high delays which result in the rest of the network which now has to accommodate the additional load that Node 2 will not accept. Next, the threshold was reduced to 20 percent so that 80 percent of the buffer space could be used before a node would go into regulation. At this level regulation still occurred, but less frequently with an average delay below that with a threshold of 30 percent. Finally a regulation threshold of 10 percent was attempted revealing that under the offered load, the node was able to accommodate all packets without buffer overflow occurring.

This indicates that a threshold of 30 percent may be too conservative, and that this should possibly be decreased to avoid the situation where regulation prohibits use of a node in cases where this is not actually necessary. It is likely that prudence prevailed in the network implementation, where it would not be possible to examine the threshold as accurately as is possible in a modelling situation where a constant load can be applied. From this investigation it would seem that a regulation threshold of even 10 percent enables the network to function without danger of buffer overflow and resultant loss of packets. As long as the load being modelled represents the

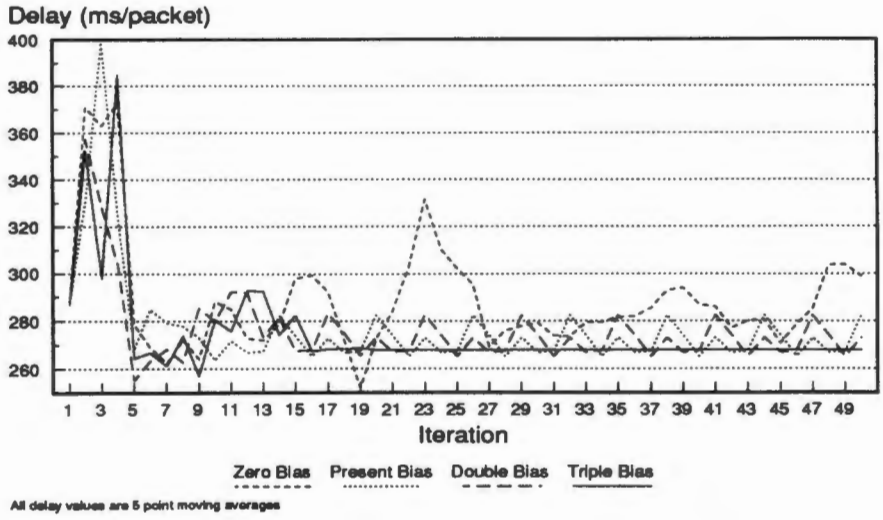


Figure 51: Routing 1 to 4 with a *bias factor* incorporated into the routing algorithm

heaviest bursts of traffic on the actual network, it should be possible to calculate the buffer size which will enable the network to operate at this level of utilisation. It is clearly desirable that a node should go into regulation as rarely as possible, since regulation degrades performance throughout the rest of the network where routes adapt in order to convey traffic along alternate *L_routes*.

7.3.3 Bias Factor

As was shown in section 7.2.4, a great deal of oscillation can occur with packets being routed rapidly over different paths when the network is dealing with high traffic volumes. Oscillatory behaviour was identified as a problem with the initial ARPANET routing algorithm [Ber87], and subsequent versions of the ARPANET routing algorithm attempted to remedy this behaviour [Kha89] in order to provide the routing algorithm with more stability. It should be reiterated that *stability* was one of the desirable features identified when discussing routing algorithms in an earlier section.

In the ARPANET an increased *bias factor* was used to provide the desired routing stability, and this same approach warranted investigation in the network studied to see whether this could be used to stabilise the routing situation portrayed in Figure

47, and also to examine what effect this would have on network delay. Another approach to reducing oscillations would be to maintain some sort of average delay value calculated over a number of routing table updates, in order to arrive at a smoothed routing decision value. The algorithm should not become too static, since it would then be insensitive to changes in traffic, and it is evident that trade-offs have to be made in order to attain what was optimistically described as the *optimum* performance of an adaptive routing algorithm.

A bias factor of the line capacity is included in the algorithm studied, so it was decided to test the effect of removing this bias factor (*zero* bias) and also of increasing it to *double* the line capacity, and *triple* the line capacity to see what effect this would have.

The graph of Figure 51 shows the original delays from Node 1 to Node 4 (as a five point moving average), representing the existing bias in the network. Against this are plotted the delays with the experimental biases. The zero bias graph shows the delays oscillating wildly. A distinct deterioration over the bias factor which is used. Doubling the bias factor had the result of exhibiting better stability – but worsened delay. Tripling the bias factor had very interesting consequences. As the graph shows, not only was the triple bias most successful in eliminating oscillation but it also reduced the average delay to the lowest of any of the tested biases.

The experiments conducted indicate that a routing algorithm modification may improve network performance. A bias increase could improve stability and result in decreased delay and increased throughput.

7.3.4 Implementation of the Hot Potato Algorithm

In order to compare the performance of the implemented routing algorithm with that of another adaptive routing algorithm, the DNN algorithm was replaced by the *hot potato* algorithm. By this routing scheme packets are forwarded on the *p_route* with the shortest queue. This algorithm is also known as the *isolated shortest queue* algorithm and, as the term 'isolated' suggests, routing decisions are made autonomously at nodes without any information from other nodes.

In its most primitive form the number of packets queued on a *p_route* is used to

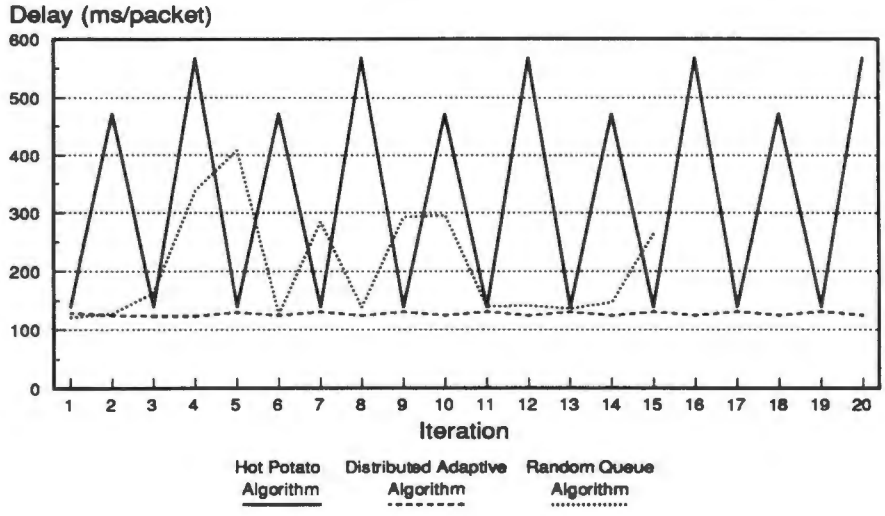


Figure 52: Routing algorithms compared

determine the outgoing line which is selected. A variation, which is slightly more sophisticated, assigns a weight to each outgoing line based on trunk speed, thus arriving at a variation of *static directory routing.* This is the algorithm implemented for demonstration.

It should be noted that in view of the iterative modelling approach, where the routing during an interval remains fixed, the choice of a new *shortest queue* would signify a new modelling interval.

The hot potato algorithm is a very inefficient one, and when observing the routing which is taking place in the network it can be seen that poor routing choices are made – sometimes leading to cycles. Figure 52 shows the hot potato algorithm, when routing from Node 3 to Node 2 of the 6 node network, against the distributed adaptive DNN algorithm.

In order to contrast the isolated shortest queue algorithm with a static algorithm, an algorithm which chooses outgoing lines randomly was tested. The random queue algorithm results in infinite queues during the sixteenth iteration, and modelling did not proceed any further than that. The random queue delays from Node 3 to Node 2 are, for a while, better than those of the isolated adaptive algorithm, but at a particular interval it indiscriminately chooses an *Lroute* already heavily loaded and

it 'falls over'.

This is just one 'random' selection, and is not necessarily reproducible. Figure 52 clearly demonstrates that this algorithm fails to meet the identified properties of an ideal routing algorithm (Section 1.3.1) – except perhaps for *computational simplicity.* The isolated shortest queue algorithm adds an *adaptiveness to changing traffic* (at a local level), but still fails to offer *stability, fairness* or *optimality.* The isolated shortest queue algorithm settles into a pattern of increasing and decreasing delays as packets are routed from Node 3 to Node 2 over alternate *L_routes.*

This comparison shows that, as expected, the distributed, adaptive algorithm is distinctly better than the other algorithms tested. While hesitantly acknowledging that a degree of *correctness* exists in the distributed adaptive algorithm, the extent to which it is *optimal* remains elusive. A modelling tool such as the *Xwan* is, however, a useful first means of exploring the *optimality* of different features and aspects of the routing algorithm. With the recommendations made it is hoped that, through the use of the *Xwan*, the quest for optimality can now be taken another step forward.

Chapter 8

Conclusion

The aim of this study was to explore the modelling of adaptive routing in wide area networks. With the objective of implementing an interactive modelling system for a specific wide area network, the task of designing and constructing such a system was researched and carried out. From this experience it is possible to draw some general conclusions, and to reflect critically on the overall modelling endeavour.

8.1 Successes of the Study

- A preliminary investigation into modelling techniques and tools prevented reaching a dead end with, for example, solution times which were inappropriate in a real time system. The efficiency of the analytic solution technique contributed to the success of the modelling approach.

- Limiting the scope of the modelling exercise to *adaptive routing* was successful because the modelling objective was well defined and this removed the potential complexities of modelling, for example, flow-control or low level individual trunk transmissions at the Data Link layer.

- The interactive GUI approach was very effective in allowing the modeller to understand the system in a way which would not be possible with printed reports. It is very clear how packets are being routed, and the direct display of

results while model solution progresses makes modelling far more meaningful than with other systems where results are only reported at the end of a model run. The use of X Windows enabled the development of a GUI which satisfied the requirements for network specification and result display.

To a large extent the feasibility of result display at each iteration is a result of the analytic solution technique, viewing the model in terms of a sequence of discrete iterations. If a simulation solution approach had been employed it would not be obvious at which points the GUI should be updated to portray packet routing, particularly considering the overhead of updating the GUI.

- A feature of the modelling system is its ease of use, facilitated by the use of icons, default values and popup windows. Network specification can be done quickly, while still allowing very specific modelling to be carried out. Interactive error reporting is conducted, and errors in model specification are conveyed through the message line eliminating error reports and line numbers.

- The modularity of MicroSnap, allowing *seamless integration* of the MVA solution routines into the *Xwan* added to the success of the particular implementation.

- The separation of network specification, routing algorithm and model solution components within the *Xwan* means that experimentation with different networks, routing algorithms – or solution techniques – can be done.

8.2 Limitations of the Study

- Multistreaming is used in the DNN, yet this was not incorporated into the *Xwan* model. It is not clear how this could be accounted for in the modelling approach employed but by using a shorter nodal processing time the user can to some extent approximate the situation where nodal delay of all packets is very small. Under high error rates, when multistreaming is automatically inhibited, the user

could increase the nodal processing time to account for the normal 'store-and-forward' operation which occurs. Under heavy load the network functions in a standard store-and-forward manner anyway since packets arrive completely at intermediate nodes before being forwarded.

- Something that this study did not consider was flow control. Flow control occurs at the layer above the network layer, the transport layer – or Network Access Module in the DNN, so exogenous traffic must be viewed in the context of this flow control at the higher layer.

- A possible drawback of this modelling approach is that the modelling hypothesis uses mean value analysis as opposed to the time dependent modelling of individual events. This would only be possible with a simulation, which it was decided takes too long to execute.

- It was reported that results were not compared with test data from the wide area network upon which this study is based, and when such data become available it will be very interesting to see how well the model predicts actuality.

8.3 Improvements in the Network Studied

A tentative recommendation for improving the performance of the network is that the trunk error threshold should be decreased to 10 percent so that trunks are declared unusable before they become unusable. A further recommendation is that the nodal regulation threshold could probably be reduced to 20 percent (or below) reducing unnecessary regulation states at nodes. An increased bias factor is a third possibility for further investigation, and for testing on a live network to compare the results of the ***Xwan*** with those observed in the network.

8.4 Future Work and General Applicability of the Modelling Technique

Experimentation with different algorithms is a very useful exercise, and the *Xwan* modelling system facilitates testing and comparison of routing algorithms. This is an example of future work with *Xwan* specifically, but a consideration of future work should be concerned with continued work on the modelling philosophy, as it is this contribution which is potentially of most benefit to other researchers and to progress in the field of performance modelling.

In terms of future work on iterative analytic models for the modelling of wide area networks it will be interesting to see if other modelling work verifies the success of this approach. It is possible that the modelling approach employed for the network in this study may not be suitable for other wide area networks. It is hoped that this work will encourage other researchers to employ iterative analytic techniques, with their advantages over other solution methods, and investigate this approach in other contexts.

The modelling of a wide area network is a daunting task, and it is necessary to set limits on the features modelled. Through adopting a particular perspective, salient network features can be identified as concerns of a modelling exercise.

Ten years ago a paper on the DNN reflected the main objection to "the development of a model to observe the effectiveness of the routing algorithm" being the "cost of such a model due to the complexity of the problem" [Bod81, p.20]. A modelling system has been built attempting to represent the complexity identified. What remains now is to determine how successfully the *Xwan* has addressed this discerned complexity, and whether the experience of building this modelling system can be used by others through the application of interactive network specification and the use of iterative, analytic solution techniques.

Bibliography

[Ahu79] Ahuja, V. "Routing and Flow Control in Systems Network Architecture". *IBM Systems Journal*, **18**, pp. 298–314. 1979.

[Atk80] Atkins, J.D. "Path Control: The Transport Network of SNA". *IEEE Transactions on Communications*, **COM-28**, pp. 527–538. 1980.

[Bas75] Baskett, F. et al. "Open, Closed and Mixed Networks of Queues with Different Classes of Customers". *Journal of the ACM*, **22**, 2, pp. 248 – 260. April 1975.

[Bei90] Beilner, H. "Structured Modelling and Tool Support". *Performance 1990*, Johannesburg. 1990.

[Bel86] Bell, P.R. and Jabbour, K. "Review of Point-to-Point Network Routing Algorithms". *IEEE Communications Magazine*, **23**, pp. 34–38. Jan 1986.

[Ber87] Bertsekas, D. and Gallager, R. *Data Networks.* Prentice-Hall, Englewood Cliffs, NJ. 1987.

[Bod80] Bodde, R. "Network Software Design". *Working Papers, State of the Art Seminar on Computer Performance Predictions of Networks.* Stellenbosch University, Sept 1980.

[Bod81] Bodde, R. "Adaptive Routing Techniques Used in NETEX", Working Document, Network Systems. 1981.

[Bod90] Bodde, R. "Meeting Performance Criteria Demanded by Internetworked LANs". *Working Papers, Performance 1990.* Johannesburg. March 1990.

[Boo84a] Booyens, M. et al. "SNAP: An Analytical Multiclass Queueing Network Analyzer". *Proceedings of the International Conference on Modelling Techniques and Tools for Performance Analysis*, Paris. 1984.

[Boo84b] Booyens, M. and Kritzinger, P.S. "SNAPL/1: A Language to Describe and Evaluate Queueing Network Models". *Performance Evaluation*, **Vol 4**, 3. 1984.

[Cov67] Coveyou, R.R. and MacPherson, R.D. "Fourier Analysis of Uniform Random Number Generators". *Journal of the of ACM*, **XIV**, pp. 100–119. 1967.

[Dav79] Davies, D.W et al. *Computer Networks and their Protocols.* Wiley. 1979.

[Deo74] Deo, N. *Graph Theory with Applications to Engineering and Computer Science.* Prentice-Hall, Englewood Cliffs, NJ. 1974.

[Eph86] Ephremides, A."The Routing Problem in Computer Networks" in Blake, I.F. and Poor, H.V. (eds), *Communication and Networks.* New York. Springer-Verlag, pp. 299–324. 1986.

[Fra71] Frank, H. and Frisch, I.T. *Communication, Transmission and Transportation Networks.* Addison-Wesley. 1971.

[Gaj90] Gajewska, H. et al. "Why X Is Not Our Ideal Window System". *Software–Practice and Experience* ,**Vol 20**, S2. Oct 1990.

[Gor69] Gordon, G. *System Simulation.* Prentice-Hall, Englewood Cliffs, NJ. 1969.

[Gra87] Grassmann, W., Kumar, S., Billinton, R. "A Stable Algorithm to Calculate the Steady-State Probability and Frequency of a Markov System". *IEEE Transactions on Reliability*, **36**, 198, pp. 58–61. 1987.

[Hal88] Halsall, F. *Data Communications, Computer Networks and OSI*, Addison-Wesley, 2nd Edition. 1988.

[Har88] Harel, D. "On Visual Formalisms". *Communications of the ACM*, **31**, 5. 1988.

[Jai90] Jain, R. "Congestion Control in Computer Networks: Issues and Trends". *IEEE Network*, **Vol 4**, No 3. May 1990.

[Kam77] Kamoun, F. and Kleinrock, L. "Hierarchical Routing for Large Networks: Performance Evaluation and Optimisation". *Computer Networks*,**Vol 1**, pp. 155–174. 1977.

[Kam79] Kamoun, F. and Kleinrock, L. "Stochastic Performance Evaluation of Hierarchical Routing for Large Networks". *Computer Networks*, **Vol 3**, pp. 337–353. Nov 1979.

[Ker79] Kermani, P. and Kleinrock, L. "Virtual Cut-Through: A New Computer Communication Switching Technique". *Computer Networks*, **Vol 3**, No 4. 1979.

[Kha89] Khanna, A. and Zinky, J. "The Revised ARPANET Routing Metric". SIGCOMM, Communications Architectures and Protocols. Austin, Texas. Sept 19–22, 1989.

[Kle76a] Kleinrock, L. *Queueing Systems, Vol 1: Theory.* John Wiley, New York. 1976.

[Kle76b] Kleinrock, L. *Queueing Systems, Vol 2: Computer Applications.* John Wiley, New York. 1976.

[Kri79] Kritzinger, P.S. et al. "Modelling of Computer Communication Networks". Institute for Applied Computer Science, University of Stellenbosch, *Technical Report ITR 80-03-01.* Oct 1979.

[Kri83] Kritzinger, P.S. and Van Wyk, S. "MEAN VALUE ANALYSIS: A Collection of the Results". Institute for Applied Computer Science, University of Stellenbosch, *Technical Report ITR 82-14-01.* Aug 1983.

[Kri90] Krieger, U.R., Müller-Clostermann, B. and Sczittnick, M. "Modelling and Analysis of Communication Systems Based on Computational Methods for Markov Chains". *IEEE Journal on Selected Areas in Communications*, **Vol 8**, 9. 1990.

[Krz80] Krzesinski, A. "An Introduction to Computer Performance Modelling". Institute for Applied Computer Science, University of Stellenbosch, *Technical Report ITR 80-01-01.* June 1980.

[Lav83] Lavenberg, S.S. (ed.) *Computer Performance Modeling Handbook*, Academic Press. 1983.

[Laz84] Lazowska, E.D et al. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice-Hall, Englewood Cliffs, NJ. 1984.

[LeB86] Le Boudec, J.Y. "A BCMP Extension to Multiserver Stations with Concurrent Classes of Customers". 1986 ACM Sigmetrics Conference, Performance Evaluation Review, **Vol 14**, 1. May 1986.

[Mül90] Müller-Clostermann, B. and Sczittnick, M. "MACOM - A Tool for the Markovian Analysis of Communication Systems". 1990.

[Nay66] Naylor, T.H., Balintfy, J.L. Burdick, S.D. and Chu, K. *Computer Simulation Techniques.* John Wiley & Sons, New York. 1966.

[Nol81] Noltemeier, H. *Graphtheoretic Concepts in Computer Science*, Proc. of the International Workshop WG80, Springer-Verlag. June 15–18, 1980.

[Onv90] Onvural, R.O. "Survey of Closed Queueing Networks with Blocking". *ACM Computing Surveys*, **Vol 22**, No2. June 1990.

[Paw90] Pawlikowski, K. "Steady-State Simulation of Queueing Processes: A Survey of Problems and Solutions". *ACM Computing Surveys*, **Vol 22**, No 2. June 1990.

[Pel89] Peleg, D. and Upfal, E. "A Trade-Off between Space and Efficiency for Routing Tables". *Journal of the ACM*, **Vol 36**, No 3. July 1989.

[Pic79] Pickholtz, R. and McCoy, C. "Effects of a Priority Discipline in Routing for Packet-Switched Networks". *IEEE Transactions on Communications*, **Vol COM-24**, No 5. May 1979.

[Poo73] Pooch, U.W. and Nieder, A. "Survey of Indexing Techniques for Sparse Matrices". *ACM Computing Surveys*, **Vol 5**, 2. June 1973.

[Pre87] Press, W. et al. "Numerical Recipes – The Art of Scientific Computing", Cambridge University Press. 1987.

[Pri77] Price, W.L. "Adaptive Routing in Store-and-Forward Networks and the importance of load splitting". *Proc. IFIP Congress 77*, Toronto, pp. 309–313. August 1977.

[Pri89] Pritsker, A.A. et al. *Slam II: Network Models for Decision Support.* Prentice-Hall, Englewood Cliffs, NJ. 1989.

[Pro89] Product Description of the Network Systems (Pty) Limited Distributed Nodal Network System. July 1989.

[Rei80] Reiser, M. and Lavenberg, S.S. "Mean Value Analysis of Closed Multichain Queueing Networks". *Journal of the ACM*, **Vol 27**, 2. April 1980.

[Rud76] Rudin, H. "On Routing and Delta Routing: A Taxonomy and Performance Comparison of Techniques for Packet-Switched Networks". *IEEE Trans. Commun*, **COM-24**. 1976.

[Sau81] Sauer, C.H. and Chandy, K.M. *Computer Systems Performance Modeling.* Prentice-Hall, Englewood Cliffs, NJ. 1981.

[Sau83] Sauer, C.H. and MacNair, E.A. *Simulation of Computer Communication Systems.* Prentice-Hall, Englewood Cliffs, NJ. 1983.

[Sch74] Schriber, Thomas J. "Simulation Using GPSS". Wiley, New York. 1974.

[Sch80] Schwartz and Stern. "Routing Techniques Used in Computer Communication Networks". *IEEE Trans. Commun*, **COM-28**, pp.539–552. 1980.

[Scz90] Szczittnick, M. and Müller-Clostermann, B. "MACOM - A Tool for the Markovian Analysis of Communication Systems". *Proceedings of the Fourth International Conference on Data Communication Systems and their Performance*, Barcelona. 1990.

[Ser89] Serfozo, R.F. "Markovian Network Processes: Congestion-Dependent Routing and Processing". *Queueing Systems*, **Vol 5**, 1989.

[Sta85] Stallings, W. *Data and Computer Communications.* Macmillan, New York. 1985.

[SwTh81] Swamy, M.N.S. and Thulasiraman, K. *Graphs, Networks and Algorithms.* John Wiley and Sons, New York. 1981.

[Tan87] Tanenbaum, A.S. *Operating Systems: Design and Implementation.* Prentice-Hall, Englewood Cliffs, NJ. 1987.

[Tan89] Tanenbaum, A.S. *Computer Networks*, Prentice-Hall, Englewood Cliffs, NJ. 1989.

[Ter87] Terplan, K. *Communication Networks Management.* Prentice-Hall, Englewood Cliffs, NJ. 1987.

[Tsa89] Tsai, W.T. et al. "An Adaptive Hierarchical Routing Protocol". *IEEE Transactions on Computers*, **Vol 38**, No 8. August 1989.

[Tym80] Tymes, L. "The New Routing Algorithm for ARPANET". *IEEE Transactions on Communications*, **COM-28**. 1980.

[Tym81] Tymes, L. "Routing and Flow Control in TYMNET", *IEEE Transactions on Communications*, **COM-29**. 1981.

[VaMi91] Van Zijl, L. and Mitton, D. "Using Statecharts to Design and Specify a Direct-Manipulation User Interface". *Proceedings of the 6th Southern African Computer Symposium*, Caledon. 1991.

[Won78] Wong, J.W. "Queueing Network Modelling of Computer Communication Networks". *Computing Surveys*, **Vol 10**, No 3, pp. 343–351. September 1978.

[You89] Young, D. *X Window Systems Programming and Applications with Xt.* Prentice-Hall, Englewood Cliffs, NJ. 1989.

[Zim80] Zimmermann, H. "OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection". *IEEE Transactions on Communications*, **COM-28**, 4. April 1980.