

APPENDIX A

Link: Connects two adjacent nodes, and allows packets to be transferred between the nodes.

Link connectivity: The minimum number of nodes that must be removed to disconnect two nodes in the network.

Looping: Looping occurs when a packet circulates indefinitely around the network and does not reach its destination.

Messages: The data at a node to be that is to be transferred to one or more nodes in the network. A message may be broken in to one or more packets to be transmitted to the destination node(s).

Node: This is often referred to as an inter message processor and is responsible for generating messages (data), relaying messages to other nodes and for consuming messages.

Packet: A ensemble of data (represented in bits) that is transferred along a link.

Packet delay: The delay experienced by a packet travelling from the source node to the destination node in the network, and is calculated as the time the packet is consumed by the destination node minus the time at which the packet was created at the source node.

Protocol: A set of rules that must be adhered by all the nodes in the network so that data may be successfully transferred between the nodes in the network.

Routing algorithm: Responsible for determining which output path, or paths, an incoming packet should be sent.

APPENDIX A

Sink: The node responsible for consuming the data or packet, that is the node for which the data is intended.

Source: The node responsible for the generation of data to be transferred to a destination node.

Throughput: The number of packets per second successfully arriving at their destination nodes in the network.

Traffic: The number of packets per second that are transmitted across the links in the network.

Virtual circuit: In a virtual circuit the path from the source node to the destination node remains constant throughout the duration for which the user is connected to the network. The packets are guaranteed to be successfully transmitted to the destination.

APPENDIX A

SYMBOLS

(i,k)	represents link from node i to node k .
μ_i	the average service rate of node i .
$dD/dr_{ik}(j)$	the marginal delay due to the traffic generated at node i .
$\theta_{ik}(j)$	the probability of a packet at node i , destined for node j being route via link (i,k) .
$B_{ik}(j)$	The set of blocked paths k at node i for destination node j .
C_{ik}	represents the capacity of link (i,k) .
$dD/dfk(j)$	the marginal link delay of link (i,k) due to the traffic flow in link (i,k) .
D_{ik}	the average packet delay of link (i,k) .
D_T	the total average packet delay of the network.
f_{ik}	the traffic flow in link (i,k) .
i	Current node number.
j	represents the destination node number.
k	Usually represents the neighbouring node to node i .
L	set of all links in the network.
N	The total number of nodes in the network.
n	The number of packets over which the incoming traffic is averaged at a node.
p	The minimum number of remaining buffers in the buffer pool of a node i before incoming packets at node i , generated at node i , are discarded.
$r_{ik}(j)$	traffic entering the network at node i destined for node j .
$t_{ik}(j)$	the total traffic passing through node i destined for node j .

APPENDIX B

LITERATURE SURVEY

B.1 Literature survey: Routing algorithms

Static routing algorithms

Name	Refer.	Description	Advantages	Disadvantages	Applications
Flooding	[43]	Packets are transmitted along every out-going path at a node, except the path on which the packet arrived.	Simple, robust, easy to implement and allows broadcasting.	Great deal of wasted bandwidth due to the duplication of packets in the network.	Used in small static networks where the traffic is low and remains constant. May be used as a backup algorithm. Often used as a reference to which the performance of other algorithms are compared.
Proportional	[37]	Packets are sent on the outgoing links at a node on an equiprobable basis.	Simple and shows a slight improvement compared to the flooding algorithm.	Highly inefficient.	Used in small networks with low traffic rates.
Random	[37]	Packets are sent on the outgoing links at a node with differing probabilities, the probabilities being determined by the routing control center (RCC) in the network.	Simple and requires minimal intelligence.	The slight improvement on the performance compared to the proportional algorithm does not usually justify the added cost of a RCC in the network.	Used in small networks with low to medium traffic rates, and where the traffic varies extremely slowly with time.
Least cost	[40]	Packets are routed via the least cost path to a destination. Can be made adaptive if the costs of the links are made proportional to the traffic in the links.	Simple, most widely used and is used to minimize the cost of running a network.	Does not adapt to changes in traffic, unless it is adaptive, where it adapts slowly.	Used in networks where minimizing the cost of transferring packets across the network is essential. Only useful in networks with a fairly constant input traffic rate.

Adaptive routing algorithms

Name	Refer.	Description	Advantages	Disadvantages	Applications
Delta routing	[37]	Combination of centralized and decentralized routing. Overall routing paths are determined by the RDC in the network. Individual nodes may make local routing decisions to prevent congestion.	Ultra-dynamic and varies rapidly to changes in the traffic patterns in the network.	Suffers from the vulnerability and problems of a centralized routing system.	For small networks of less than 20 nodes. The performance of the algorithm in larger networks requires further investigation.
Hierarchical	[23,26]	Large networks require large routing tables at each node. The algorithm has developed to reduce the table lengths required.	Reduced table lengths at each node in the network.	Only useful in large networks, in smaller networks the performance degrades as the table lengths required at each node are reduced.	For large distributed networks consisting of more than 100 nodes.
Relaxation routing	[41]	Used to route packets around congested regions in the network and to minimize the packet delays in the network.	Reacts rapidly to changes in traffic patterns in the network.	Large amount of routing information must be exchanged between the nodes in the network.	For highly interconnected networks with rapidly varying traffic rates. The performance of the algorithm in various network topologies requires further investigation.
Gradient projection	[39]	Minimizes the packet delay in the network.	Rapid convergence and compares favourably with other algorithms in small networks.	Complex computations required.	Suitable for small networks consisting of less than 20 nodes.
Minimum delay distributed algorithm	[11]	Minimizes the TOTAL average packet delay in the network, rather than the individual packet delays.	Only a small amount of routing information must be exchanged between the nodes in the network. Iterative and loop free.	Only suitable for 'quasi-static' networks. The performance of the algorithm in a dynamic network is subject to further investigation.	Suitable for irregular type networks. The size of the network is limited by the adaptability required and memory available at each node.

Name	Refer.	Description	Advantages	Disadvantages	Applications
Extremal flow	[8]	Minimizes the packet delay.	Guaranteed to converge to an optimal solution. Flexible and may be adapted to any type of network. Performance compares favourably with other algorithms.	Computationally intensive.	Due to computational intensity the algorithm is limited to smaller networks.
Second derivative	[5]	Minimizes the average packet delay, but computations are based on the second derivative of the traffic flow in the network.	Suitable for 'quasi-static' networks. Computational requirements are directly proportional to the adaptability of the algorithm.	Complex algorithm and large memory requirements.	Small networks, but the performance of the algorithm in various networks requires further study.

B.2 Literature survey: Networks

Name	Reference	Network size	Description
ARPANET	[3], [2] and [40]	100 nodes	The network was developed by the 'Advanced Research Project Agency' of the United States department of defense. The network started in 1969 as a four node network. Currently the network consists of more than 100 nodes distributed worldwide. Originally a distributed algorithm based on a shortest path routing algorithm was used. The algorithm displayed numerous problems. The main problem of the algorithm was the poor adaptability to changes in traffic in the network. Packet frequently looped indefinitely around the network. The algorithm also adapted extremely slowly to changes in the topology of the network. The algorithm was replaced by a more dynamic distributed algorithm. Priority is given to routing packets. Routing packets are transferred across the network using a flooding algorithm. The new algorithm reacts fairly rapidly to changes in traffic in the network.

Name	Reference	Network Size	Description
TYNNET	[40]	300 nodes	The TYNNET network of Tynesware Incorporated was developed in 1970 and consists of more than 300 nodes. Routing is performed by one of four routing computers present in the network. A virtual circuit is set up for each source destination node pair in the network. The virtual circuit is set up using a least cost algorithm. The cost of the links are proportional to the load on each link. The problem with the network occurs when a virtual circuit is set up during a busy period. The virtual circuit allocated may then be a long delay path. The path remains unchanged even if the traffic in the network changes. The network operates effectively under lightly loaded situations. The delay increases dramatically as the number of users in the network increases.
TRANSFAC	[40]	300-400 nodes	The network was developed in France. A distributed routing algorithm is used to route packets in the network. The network performs satisfactorily. The performance of the network under full load conditions is not known.
TIDAS	[41]	20 nodes	The TIDAS network is a Swedish network. The network consists of 20 terminals connected to 20 nodes. Routes between the nodes are established using the public telephone system. The routing algorithm used is similar to that of the ARPANET network. The algorithm used is well suited to the TIDAS network consisting of only 20 nodes.
DECNET	[42]	-	The DECNET network was developed by the Digital Equipment Corporation. The network allows small private networks to be implemented. The network operates on a datagram based system. Each node in the network contains the path length and cost to each destination node in the network. Congestion control is provided by limiting the number of packets that may be transferred for each source destination node pair. A hop counter is used to remove circulating packets in the network. The algorithm operates fairly efficiently except for the following problems: - The algorithm does not adapt to any changes in traffic in the network, - Packets may loop in the network, and - The algorithm adapts slowly to changes in the topology of the network.

APPENDIX C

THE MINIMUM DELAY ROUTING ALGORITHM

APPENDIX C

C.1 R G Gallager's minimum delay routing algorithm

A description of R G Gallager's routing algorithm is provided. The reader is referred to reference [11] for a more detailed derivation of the algorithm. The reader is also referred to Appendix A for an explanation of the terminology and notation used in the report.

The object of the algorithm is to minimize the total expected packet delay for the entire network. The total expected packet delay D_T is defined as the sum of the packet delays on the individual links of the network. To minimize the expected delay D_T each node i must incrementally decrease the probability $\theta_{ik(s)}$ of a packet being sent on a link for which the marginal delay $(dD/df_{ik} + \lambda D_T/r_{ik(s)})$, is large and increase the probability for which the marginal delay is small. The algorithm may be divided into two parts: A protocol for exchanging the marginal delays between the nodes in the network and an algorithm for updating the routing variables.

C.1.1 Calculation of marginal delays

Each node can as a time average estimate dD/df_{ik} . This is the marginal delay on link (i,k) as a result of the flow $\{f_{ik}\}$ on link (i,k) . The algorithm adopted estimate the value for dD/df_{ik} is based on an estimation procedure developed by Segall [38] and is described in detail in section C.3.

To calculate the value of $\lambda D_T/r_{ik(s)}$, each node i in the network waits until it has received $\lambda D_T/r_{ik(s)}$ from each neighbouring node for which $\theta_{ik(s)} > 0$. Using the fact that $\lambda D_T/r_{ij(s)} = 0$ node i then calculates $\lambda D_T/r_{ik(s)}$ and

APPENDIX C

broadcasts this to all its neighbours except destination node j . The following formula is used to calculate $\partial D_T / \partial r_{k(i,j)}$ [11]:

$$\partial D_T / \partial r_{k(i,j)} = \sum_{k \in N(i,j)} [dD / df_{ik} + \partial D_T / \partial r_{k(i,j)}]$$

Where dD / df_{ik} is the marginal delay of link (i,k) , and $\partial D_T / \partial r_{k(i,j)}$ is the marginal delay received from each neighbouring node k , for which $\theta_{ik(i,j)} > 0$ and k not equal to j .

C.1.2 The 'update' algorithm

Once all the necessary routing data for destination node j has been received at node i the routing variables $\theta_{ik(i,j)}$ are updated.

Firstly a set of blocked paths $B_{k(i,j)}$ must first be defined in order to maintain loop freedom. The set $B_{k(i,j)}$ contains the set of nodes k for which the value $\theta_{ik(i,j)}$ cannot be increased from zero. The algorithm for determining the set of blocked paths is given in section C.2.

The algorithm used to update the routing variable $\theta_{ik(i,j)}$ is defined as follows:

For all neighbouring nodes k :

If k is an element of $B_{k(i,j)}$
then

$\theta_{ik(i,j)} = 0$ and
 $\theta_{ik(i,j)} = 0$

APPENDIX C

If k is not an element of $B_{i,j}$,

then

$$\begin{aligned} \alpha_{ik}(j) &= dD/df_{ik} + \{D_r/\delta r_{ik}(j)\} \\ &\quad - \min[dD/df_{ik} + \{D_r/\delta r_{ik}(j)\}, \\ \beta_{ik}(j) &= \min[\alpha_{ik}(j), \alpha_{ik}(j)/t_{ik}(j)] \text{ and} \\ k_{min} &= \text{the value of } k \text{ that achieves} \\ &\quad \text{minimization} \end{aligned}$$

Finally

$$\begin{aligned} \theta_{ik}(j) &= \alpha_{ik}(j) - \beta_{ik}(j) \quad \text{for } k \neq k_{min} \\ \theta_{ik}(j) &= \alpha_{ik}(j) + \beta_{ik}(j) \quad \text{for } k = k_{min} \end{aligned}$$

The algorithm reduces the amount of traffic sent on the non optimal links and increases the traffic sent on the best link. The value of α determines the convergence of the algorithm. For small values of α the convergence of the algorithm is slow but guaranteed. As α is increased the convergence rate of the algorithm increases but then the probability of no convergence also increases. Using a simulation program suitable values for α may be determined.

C.2 Calculation of blocked paths

The marginal delay $\{D_r/\delta r_{ik}(j)\}$ decreases towards the destination node, and therefore if the marginal delay increases along a path the path cannot be a valid path to the destination. An improper routing variable is therefore defined if $\theta_{ik}(j) > 0$ and $\{D_r/\delta r_{ik}(j)\} \leq \{D_r/\delta r_{ik}(j)\}$.

The set of blocked paths has been fully defined by R G Gallager as follows: The set $B_{i,j}$, contains the set of nodes k for which the link (i,k) is not a set of L . A node k is blocked relative to destination node j if k has a path to j

APPENDIX C

containing some link (l,m) for which $\Theta_{lm}(s)$ is improper and satisfies

$$\Theta_{lm}(s) \geq [dD/df_{lm} + sD\tau/3r_{lm}(s) - sD\tau/3r_{l(s)}]/t_{l(s)}$$

The derivation of this definition may be found in reference [11].

The protocol adopted to calculate the set of blocked paths is as follows:

- Node l when it calculates $sD\tau/3r_{l(s)}$, determines for any $\Theta_{lm}(s) > 0$ if m is a blocked path, using the formula given above.
- If a blocked path is detected node l adds a special tag to its broadcast of $sD\tau/3r_{l(s)}$.
- If node i receives the value of $sD\tau/3r_{l(s)}$, which is tagged then node l is added to the blocked path set $B_{l(s)}$.
- If node i receives the value of $sD\tau/3r_{l(s)}$, which is tagged then node i also adds a special tag to its broadcast of $sD\tau/3r_{l(s)}$.

Using this protocol and assuming that the initial values of $\Theta_{lm}(s)$ are loop free the algorithm is loop free at each iteration. The derivation of the algorithm is given in detail in reference [11].

C.3 Estimation of incremental delays $dD/df_{lm}(s)$

The estimation of the incremental delays is performed using an algorithm developed by Segall [38]. The procedure has been adopted since it requires no assumptions about the nature of the traffic, is recursive and therefore requires little

APPENDIX C

memory and may be implemented in a totally decentralized manner. The algorithm may be summarized as follows:

If a_n represents the arrival of the n th packet on link (i,k) in the network and d_n represents the departure of the n th packet from link (i,k) then the following recursive formula may be defined.

$$\begin{aligned} C_m^n &= 0 && \text{for } m < n, \\ C_m^n &= d_n - a_n && \text{for } m = n, \text{ and} \\ C_m^n &= \min(C_{m-1}^n, d_{m-1} - a_m) && \text{for } m > n + 1 \text{ and,} \\ &\text{where } C_{n-1}^n = d_n - \max(a_{n+1}, d_{n-1}) \\ &\text{and } d_0 = 0. \end{aligned}$$

During the same period consisting of N packets the quantity dD/df is estimated as:

$$dD/df = \sum_{n=1}^N \sum_{m=n}^N C_m^n / N$$

This formula has been used in the simulation program as shown in the program listing given in Part II, Appendix D.

C.4 Calculation of event times

Another important aspect that must be considered is the calculations involved in determining the time at which the various events occur during the simulation. The times which have to be calculated are the arrival times, service times and departure times of the packets in the network.

APPENDIX C

Consider the pair of nodes illustrated below:

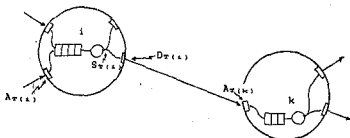


Figure C.1 Diagram of a typical pair of nodes in the network.

If one assumes a server with a constant service rate the times can be calculated as follows:

Let:

- $A_T(i)$ represent the arrival time at node i,
- $S_T(i)$ represent the service time at node i,
- $D_T(i)$ represent the departure time at node i,
- $TS(i)$ be the time required to service all the packets in the input queue of the node i,
- $IT_1(i)$ be the fraction of time required to insert the packet into the input queue,
- $IT_2(i)$ be the fraction of time required to insert the packet into the output queue,
- μ be the service rate of node i,
- n be the number of packets in the input queue of node i and,
- m be the number of packets in the output queue of node i destined to node k

The service time (represents end of service time):

$$S_T(i) = A_T(i) + n/\mu + 1/\mu + IT_1(i)$$

APPENDIX C

The departure time:

$$D_{T(i)} = S_{T(i)} + (m \times \text{bits per packet} / \text{capacity of link } (i,k))$$

The arrival time:

$$A_{T(k)} = D_{T(i)} + \text{transmission delay.}$$

If random service rates for each packet are assumed for individual packets serviced at node i then the end of service time $S_{T(i)}$ may be calculated as:

$$S_{T(i)} = A_{T(i)} + iTe(i) + iTs(i) + \text{random time for service.}$$

The calculations defined above determine the delay experienced by a packet moving across a network. The delay is related to the number of packets present in the network, the service rate of the nodes, the length of the packets and the capacity of the links in the network. The calculations defined above may be modified to include extra delays to improve the accuracy of the simulation program. The formulae are however sufficient and include most of the delays experienced by a packet travelling across a network.

C.5 Other calculations

The most significant calculations performed in the simulation program have been explained. Other important calculations performed by the simulation program are the estimation of the traffic at a node in the network and the calculation of the network performance results obtained from the simulation. The estimation of the traffic at a node is explained in chapter 6 of the report. The calculations involved in determining the final results from the simulation are provided in Part II, Appendix C.

APPENDIX D

MEMORY AND COMPUTATIONAL REQUIREMENTS

APPENDIX D

Memory and computational requirements

The memory and computational requirements of the routing algorithms implemented in the simulation program are specified. Only the significant memory and computational requirements are listed.

D.1 Memory requirements

Firstly the memory required by each algorithm is discussed. Memory is required to store variables such as the topology of the network, the list of paths to each destination node and other variables required by the specific routing algorithm. These variables must often be retrieved, examined and updated frequently and it is therefore of essential that the variables required by a routing algorithm be stored in random access memory (RAM). This usually limits the amount of memory a routing algorithm may use. It has been assumed in the calculations given below that all the variables used by the various routing algorithms require the same amount of memory of one unit. The memory calculated below is the amount of memory required at each node in the network.

D.1.1 Flooding algorithm

The memory required by the flooding algorithm is negligible compared to the other routing algorithms employed in the simulation program. The flooding algorithm requires only list of the output paths at each node in the network, and for a fully inter-connected network, the maximum memory required is $(N-1)$, where N is the number of nodes in the network.

APPENDIX D

Memory and computational requirements

The memory and computational requirements of the routing algorithms implemented in the simulation program are specified. Only the significant memory and computational requirements are listed.

D.1 Memory requirements

Firstly the memory required by each algorithm is discussed. Memory is required to store variables such as the topology of the network, the list of paths to each destination node and other variables required by the specific routing algorithm. These variables must often be retrieved, examined and updated frequently and it is therefore of essential that the variables required by a routing algorithm be stored in random access memory (RAM). This usually limits the amount of memory a routing algorithm may use. It has been assumed in the calculations given below that all the variables used by the various routing algorithms require the same amount of memory of one unit. The memory calculated below is the amount of memory required at each node in the network.

D.1.1 Flooding algorithm

The memory required by the flooding algorithm is negligible compared to the other routing algorithms employed in the simulation program. The flooding algorithm requires only list of the output paths at each node in the network, and for a fully inter-connected network, the maximum memory required is $(N-1)$, where N is the number of nodes in the network.

APPENDIX D

D.1.2 Least cost routing algorithm

The following data is required by the least cost algorithm:

- A cost matrix consisting of $N(N - 1)$ entries,
- A list of the output path for each destination node j , consisting of $(N - 1)$ entries.

Using this data the total number of memory units required is:

$$\begin{aligned}\text{Total memory required} &= N(N - 1) + (N - 1) \\ &= N^2 - 1\end{aligned}$$

Therefore the amount of memory required by the least cost routing algorithm is directly proportional to N^2 .

D.1.3 Adaptive routing algorithm

The following minimum data is required by the adaptive routing algorithm used in the simulation program:

- The network topology or link capacities in the network requiring $N(N - 1)$ units of memory,
- a list of output paths, or output probabilities $Q_{ik(s)}$, which requires a maximum of $(N - 1)(N - 1)$ amount of memory,
- a list of marginal delays dD/df_{ik} , utilizing $(N - 1)$ units of memory,
- a list of marginal delays $dD/dr_{ik(s)}$, requiring $(N - 1)(N - 1)$ entries,
- a set of blocked paths $B_{ik(s)}$, using $(N - 1)(N - 1)$ units of memory, and

APPENDIX D

- a set of routing data received from each neighbouring node for each destination node in the network requiring $(N - 1)(N - 1)$ units of memory.

The total amount of memory required can easily be calculated to be:

$$\begin{aligned}\text{Total memory required} &= N(N-1) + 4[(N-1)(N-1)] + (N-1) \\ &= N^2 - N + 4N^2 - 8N + 4 + N - 1 \\ &= 5N^2 - 8N + 3.\end{aligned}$$

The total memory is therefore, for large values of N , directly proportional to N^2 . The maximum amount of memory required has been calculated. The amount of memory required can be reduced by defining a maximum number of nodes each node in the network may be connected to. Above a fully interconnected network has been considered. The exact amount of memory required by the adaptive routing algorithm is also directly dependent on the actual implementation of the algorithm.

D.2 Computation requirements

The computation requirements of a routing algorithm is directly dependent on the rate at which the algorithm is updated, the amount of variables that must be processed, the computational complexity of the algorithm and the accuracy required by the algorithm. A rough estimation on the order of the computations required at each node in the network is given.

APPENDIX D

D.2.1 Flooding algorithm

For the flooding algorithm routing paths are selected each time a packet is to be routed to a destination node. Then the packet is simply transmitted along each outgoing link of the node except the link on which the packet arrived. No actual computations are performed by the flooding routing algorithm. For this reason the implementation of the flooding routing algorithm is extremely simple requiring very little memory.

D.2.2 Least cost algorithm

Each time the least cost algorithm is updated a path for each destination node in the network is calculated. For a network consisting of N nodes of which each node is a possible destination node the number of computations performed by the least cost algorithm is of the order N^2 (Of the order N^2 may be written as $O(N^2)$). The least cost algorithm need only be updated if the topology of the network or the cost of a link in the network changes.

D.2.3 The adaptive routing algorithm

The adaptive routing algorithm is made up of several independent algorithms, for example each time a packet arrives at a node i in the network from a neighbouring node k a new value for dD/df_{ik} must be calculated. Also if routing information is obtained from a neighbouring node the routing tables at the node must be updated accordingly. The number of computations required for a single iteration of the adaptive routing algorithm is $O(N)$, and the number of computations required to establish a new set of routing paths at a node

APPENDIX D

D.2.1 Flooding algorithm

For the flooding algorithm routing paths are selected each time a packet is to be routed to a destination node. Then the packet is simply transmitted along each outgoing link of the node except the link on which the packet arrived. No actual computations are performed by the flooding routing algorithm. For this reason the implementation of the flooding routing algorithm is extremely simple requiring very little memory.

D.2.2 Least cost algorithm

Each time the least cost algorithm is updated a path for each destination node in the network is calculated. For a network consisting of N nodes of which each node is a possible destination node the number of computations performed by the least cost algorithm is of the order N^2 (Of the order N^2 may be written as $O(N^2)$). The least cost algorithm need only be updated if the topology of the network or the cost of a link in the network changes.

D.2.3 The adaptive routing algorithm

The adaptive routing algorithm is made up of several independent algorithms, for example each time a packet arrives at a node i in the network from a neighbouring node k a new value for $d(i)/d(k)$ must be calculated. Also if routing information is obtained from a neighbouring node the routing tables at the node must be updated accordingly. The number of computations required for a single iteration of the adaptive routing algorithm is $O(N)$, and the number of computations required to establish a new set of routing paths at a node

APPENDIX D

for each destination node in the network is $O(N^2)$. The frequency at which the routing paths are updated is directly dependent on the rate at which routing information is exchanged between the nodes in the network, the traffic flow in the network and the convergence rate of the algorithm. In a real network it is necessary to ensure that the algorithm is not updated too frequently as to consume a large proportion of the valuable processing time available at the node.

D.3 Conclusions

The rapidly decreasing costs of memory allows algorithms requiring numerous variables to be implemented in large modern computer networks. The computational efficiency of a routing algorithm may be greatly improved by implementing a large portion of the algorithm in hardware (or even in a low level programming language) rather than implementing the algorithm in a high level language such as PASCAL. The disadvantage of implementing a routing algorithm in hardware is the increased complexity, reduced flexibility and the limited modifications that may be made to the routing algorithm.

REFERENCES

REFERENCES

- [1] Agnew C B and Mark J W, "Performance Modelling for Communication Networks at a Switching Node", IEEE Transactions on Communications, Vol. COM-32, No. 8, August 1984, pp. 197-211.
- [2] Anderson G A and Jensen L D, "Computer Interconnection Structures, Taxonomy Characteristics and Examples", Computing Survey, Vol. 7, No. 4, December 1973, pp. 179-211.
- [3] Appel J, "Network Operations - A Major Opportunity in Evolving Digital Networks", IEEE Communications Magazine, Vol. 24, No. 1, January 1986, pp. 39-42.
- [4] Bell P R and Jabbour K, "Review of Point to Point Network Routing Algorithms", IEEE Communications Magazine, Vol. 24, No. 1, January 1986, pp. 34-39.
- [5] Bertsekas D P, Gafni E M and Gallager R G, "Second Derivative Algorithms for Minimum delay Distributed Routing in Networks", IEEE Transactions on Communications, Vol. COM-32, No. 8, August 1984, pp. 911-919.
- [6] Boffey T B, "Graph Theory in Operations Research", MacMillan Computer Series, The MacMillan Press Ltd. Hong Kong, 1984.
- [7] Booch G, "Object-Orientated Development", IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, February 1986, pp. 211-221.

- [8] Cantoor D G and Gerla M, "Optimal Routing in a Packet Switched Computer Network", IEEE Transactions on Computers, Vol. C-23, No. 10, October 1974, pp. 1062-1069.
- [9] Cegral T, "A Routing Procedure for the TIDAS message switching network", IEEE Transactions on Communications, Vol. COM-23, June 1975, pp. 575-585.
- [10] Even, S "An Algorithm for Determining if the Connectivity of a Graph is at Least k", SIAM J. Computing, Vol. 4, September 1975, pp. 393-396.
- [11] Gallager R, "A Optimum Routing Algorithm Using Distributed Computation", IEEE Transactions on Communications, Vol. COM-25, January 1977, pp. 73-85.
- [12] Garcia J M et al, "A model for Telephone Networks and it's use for Routing Optimization Purposes", IEEE Journal on Selected Areas in Communications, Vol. SAC-4, No. 6, September 1986, pp. 966-1211.
- [13] Gerla M, "Controlling Routes, Traffic Rates and Buffer Allocation in Packet Networks", IEEE Communications Magazine, Vol. 22, No. 11, November 1984, pp. 11-23.
- [14] Gerla M and Kleinrock L, "Flow Control: A Comparative Survey", IEEE Transactions on Communications, Vol. COM-28, No. 4, April 1980, pp. 553-574.
- [15] Glen M and Zimmerman H, "Design Principles for Network Interconnection", Proceedings 6th Data Communications Symposium 1979, pp. 109-120.

- [15] Gitman I et al. "Routing in Packet Switching Broadcast Radio Network", IEEE Transactions on Communications, Vol. COM-24, August 1979, pp. 926-930.
- [17] Gitman I, "On the Capacity of Slotted ALOHA Networks and some Design Problems", IEEE Transactions on Communications, Vol COM-22, No. 3, March 1975, pp. 305-317.
- [18] Grossman G R, "Issues in International Public Data Networks", Computer Networks, Vol. 3, November 1979, pp. 259-266.
- [19] Hahn J J and Stolle D W, "Packet Radio Network Routing Algorithms: A Survey", IEEE Communications Magazine, Vol. 22, No. 11, November 1984, pp. 41-47.
- [20] Harrison P G, "An Analytical Model for Flow Control Schemes in Communication Network Nodes", IEEE Transactions on Communications, Vol. COM-32, No. 9, September 1984, pp. 1013-1019.
- [21] Hwang F K, "Some Optimal Partition Problems with Application to Switching Networks", IEEE Transactions on Communications, Vol. COM-26, No. 11, November 1979, pp. 1761-1764.
- [22] Iiyas M and Mouftah H T, "Performance Evaluation of Computer Communications Networks", IEEE Communications Magazine, Vol. 23, No. 4, April 1985, pp. 18-29.
- [23] Kamoun F and Kleinrock L, "Stochastic Performance Evaluation of Hierarchical Routing for Large Networks", Computer Networks, Vol 3, November 1979, pp. 337-353.

- [24] Kleinrock L, "Queueing Systems: Volume 1: Theory", John Wiley and Sons Inc. New York, 1975.
- [25] Kleinrock L, "Queueing Systems: Volume 2: Computer Applications", John Wiley and Sons Inc. New York 1976.
- [26] Kleinrock L and Kamoun F, "Hierarchical Routing for Large Networks", Computer Networks, Vol. 1, January 1977, pp. 155-174.
- [27] Kleitman D, "A method for Investigating the Connectivity of Large Graphs", IEEE Transactions on Circuit Theory, Vol. CT-16, May 1969, pp. 232-233.
- [28] Landweber L H et al. "Research Computer Networks and their Interconnection, IEEE Communication Magazine, Vol. 24, NO. 6, June 1986, pp. 5-16.
- [29] Malhotra V M, "An $O(|V|)$ Algorithm for finding Flows in Networks", Information Processing Letters, Vol. 7, No. 6, October 1978, pp. 277-278.
- [30] McGregor P and Shen D, "Network Design: An Algorithm for Access Facility Problem", IEEE Transactions on Communications, Vol. COM-25, January 1977, pp. 61-73.
- [31] McQuillan J M et al. "The New Routing Algorithm for ARPANET", IEEE Transactions on Communications, Vol. COM-28, May 1980, pp. 711-719.
- [32] McQuillan J M et al. "A Review of the Development and Performance of the ARPANET Routing Algorithm", IEEE Transactions on Communications, Vol. COM-26, December 1978, pp. 1802-1811.

- [33] Price W L, "Data Network Simulation Experiments at the National Physical Laboratory 1968-1976", Computer Networks, Vol. 1, No. 4, 1977, pp. 199-210.
- [34] Robertazzi T G and Sarachik P E, "Self-Organising Communications Networks", IEEE Communications Magazine, Vol. 24, No. 1, January 1986, pp. 28-33.
- [35] Roberts N et al. "Introduction to Computer Simulation: A System Dynamics Modelling Approach", Addison-Wesley Publishing Company, London 1983.
- [36] Rubinstein R Y, "Monte Carlo Optimization, Simulation and Sensitivity in Queueing Networks", John Wiley and Sons Inc. New York, 1986.
- [37] Rudin H, "On Routing and Delta Routing", IEEE Transactions on Communications, Vol. COM-24, No. 1, January 1976, pp. 43-59.
- [38] Segall A, "Modelling of Adaptive Routing in Data Communications Networks", IEEE Transactions on Communications, Vol. COM-25, January 1977, pp. 85-95.
- [39] Shwartz M and Cheung C, "The Gradient Projection Algorithm for Multiple Routing in Message Switched Networks", IEEE Transactions on Communications, Vol. COM-28, April 1976, pp. 449-456.
- [40] Shwartz M and Stern T, "Routing Techniques used in Computer Communication Networks", IEEE Transactions on Communications, Vol. COM-28, April 1980, pp. 539-552.

- [41] Stern T E, "A Class of Decentralized Routing Algorithms using Relaxation", IEEE Transactions on Communications, Vol. COM-25, October 1977, pp.1092-1102.
- [42] Swiderski J, "Unified Analysis of Local Flow Control Mechanisms in Message Switched Networks", IEEE Transactions on Communications, Vol. Com-32, No. 12, December 1984, pp. 1286-1295.
- [43] Tanenbaum A S, "Computer Networks", Prentice Hall International Editors, New York 1981.
- [44] Wasserman A I, "Principles of Systematic Data Design and Implementation", Tutorial on Software Design Techniques, edited by P Freeman and A I Wasserman, 3rd edition, IEEE Computer Society, 1980, pp. 287-293.
- [45] Watson R W, "An Architecture for Support of Network Operation System services", Computer Networks, Vol. 4, February 1980, pp. 33-49.
-

ROUTING STRATEGIES IN DIGITAL
NETWORKS

PART II: NETWORK ROUTING
SIMULATION PROGRAM

Marc Leon Francois

Part II of a project report submitted to the Faculty of
Engineering, University of the Witwatersrand, Johannesburg,
in partial fulfilment of the requirements for the degree of
Master of Science in Engineering

Johannesburg, 1988

CONTENTS

Page

PART I: ROUTING STRATEGIES

DECLARATION	ii
ABSTRACT	iii
ACKNOWLEDGEMENTS	iv
CONTENTS	
Part I	v
Part II	xiii
LIST OF FIGURES	
Part I	xvi
Part II	xvii
LIST OF GRAPHS	
Part I	xviii
LIST OF TABLES	
Part I	xx

1	INTRODUCTION	1
1.1	Project description	1
1.2	Layout of the report	2
2	BACKGROUND THEORY	3
2.1	Digital networks	3
2.2.1	Uses and advantages of digital networks	4
2.3	The OSI reference model	5
2.3.1	The Physical layer	6
2.3.2	The Data link layer	7
2.3.3	The Network layer	7
2.3.4	The Transport layer	8
2.3.5	The Session layer	9
2.3.6	The Presentation layer	9
2.3.7	The Application layer	9
2.4	Conclusion: Network Definition	9

CONTENTS	Page
2.5 Queuing theory	10
2.5.1 Introduction to queuing theory	10
2.5.2 Queuing systems in digital networks	11
2.5.3 Conclusion: Queuing theory	13
2.6 Routing algorithms	13
2.6.1 Properties of routing algorithms	13
2.6.2 Types of routing algorithms	15
Static routing algorithms	15
Adaptive routing algorithms	16
Centralized routing systems	16
Decentralized routing systems	17
2.7 Conclusion: Background theory	18
 3 LITERATURE SURVEY	 19
3.1 Routing algorithms	19
3.2 Networks	20
3.3 Conclusion: The literature survey	21
 4 PERFORMANCE EVALUATION TECHNIQUES	 22
4.1 Performance evaluation criteria	22
4.1.1 User-Orientated criteria	23
4.1.2 Manager-Orientated criteria	23
4.1.3 Designer-Orientated criteria	23
4.2 Performance measurements	23
4.2.1 Time delays	24
4.2.2 Throughput	24
4.2.3 Cost	24
4.2.4 Percentage use of the network capacity	25
4.2.5 Number of blocked packets	25
4.2.6 Use of buffers	25

CONTENTS

Page

4.2.7	Average number of hops	26
4.2.6	Other performance measurements and conclusions	26
4.3	Performance evaluation: Analytical techniques	27
4.3.1	Queueing theory analysis	27
	Packet bit lengths	29
	Traffic rate results	31
	Various link capacity results	32
4.3.2	Single node analysis	33
4.3.3	Electrical analog analysis	34
4.3.4	Analysis of quasi-static networks	34
4.3.5	Conclusion, analytical methods	35
4.4	Performance evaluation: Experimental techniques	35
4.4.1	Measurement techniques	35
4.4.2	Conclusion, Experimental techniques	36
4.5	Performance evaluation: Simulation techniques	37
4.6	Conclusion: Performance evaluation techniques	38
5	NETWORK SELECTION	39
5.1	Network requirements	39
5.2	Selected network	40
5.2.1	Network topology	40
5.2.2	Network protocols	41
	Type of service	42
	Circuit, packet or message switched network	42
5.2.3	Node specifications	43
5.2.4	Link specifications	44
5.2.5	Packet specifications	45
5.3	Routing requirements	46
5.3.1	The least cost routing algorithm	46
5.3.2	Adaptive routing algorithm	47
5.3.3	Conclusion: Routing algorithms	47

CONTENTS

Page

5.4	Congestion protocols and buffer allocation techniques	48
5.4.1	Congestion control	48
	Hop Level Flow Control	50
	Entry to Exit Level Flow Control	51
	Network Access Level Flow Control	51
	Transport Level Flow Control	52
5.4.2	Buffer allocation techniques	52
	Complete partitioning	53
	Sharing with maximum queues	54
	Sharing with minimum allocation	54
	Sharing with minimum allocation and maximum queue	54
5.4.3	Conclusion: congestion control and buffer allocation	55
5.5	Conclusion: Network selection	55
6	NETWORK SIMULATION	56
6.1	Program requirements	56
6.1.1	Program inputs	57
6.1.2	Program outputs	58
6.2	Simulation of a digital network	58
6.2.1	The event queue	59
6.2.2	Operation of a network node	60
	Arrival process	61
	Service process	62
	Departure process	62
6.2.3	Network operation	63
6.3	Static routing algorithms simulated	63
6.3.1	Flooding routing algorithms	63
6.3.2	Least cost routing algorithms	64

CONTENTS	Page
6.4 Adaptive routing algorithm	65
6.4.1 Algorithm protocol	65
6.4.2 Incremental delays	66
6.4.3 Calculation of blocked paths	67
6.4.4 Estimation of traffic flow at each node	68
6.4.5 Conclusion: Adaptive routing algorithm	70
6.5 Program development	70
6.5.1 Hardware and software requirements	71
6.5.2 Program variables	71
6.5.3 Program procedures	72
Input routines	72
Initialization routines	73
Simulation routines	73
Output routines	73
6.6 Assumptions made in the simulation program	74
6.7 Conclusion	75
 7 SIMULATION RESULTS	 78
7.1 Network topologies simulated	76
7.2 General results	78
7.2.1 Service rate tests	79
7.2.2 Input buffer length tests	80
7.2.3 Length of the output buffers at the nodes	82
7.3 Optimization of the adaptive routing algorithm	82
7.3.1 Exchange rate of routing data between nodes	83
Exchanging routing information at regular intervals	83
Informing the network of changes in traffic	85
7.3.2 Convergence rate of the algorithm	86
7.3.3 Traffic estimation and prediction	86
Estimation of the traffic in the network	88
Traffic prediction	89

CONTENTS

Page

7.3.4	Implementation of 'piggy-backing'	91
	The 'piggy-backing' protocol	91
	The updated 'piggy-backing' protocol	92
	'Piggy-backing' conclusion	94
7.3.5	Limiting the input traffic to the network	94
7.3.6	The final product	96
7.4	Problems and limitations of the adaptive routing algorithm	97
7.4.1	Generation of routing paths	97
7.4.2	Loop elimination	97
7.4.3	The transfer of routing data in the network	99
7.4.4	General performance of the adaptive routing algorithm	102
7.5	Comparison of the various routing algorithms	103
7.5.1	Average packet delays	103
7.5.2	Number of blocked packets	106
7.5.3	Average percentage use of the network	108
7.5.4	Total cost of running the network	109
7.5.5	Delay distributions	110
7.5.6	Network performance at extreme conditions	112
	Network performance at high traffic loads	113
	Performance of the network at high service rates	115
7.5.7	Memory and computational requirements	116
	Memory requirements	116
	Computational requirements	116
	Simulation program computational and memory requirements	117
7.6	Other network topologies simulated	117
7.7	Topology changes, link and node failures in the	118
7.8	Conclusion: Network simulations	121
7.8.1	The network routing simulation package	121
7.8.2	Routing algorithms	123

CONTENTS	Page
8 FINAL CONCLUSIONS AND RESULTS	125
8.1 The adaptive routing algorithm	125
8.1.1 Uses and limitations of the adaptive routing algorithm	125
8.1.2 Comparison with the ARPANET network	127
8.1.3 Conclusion: The adaptive routing algorithm	128
8.2 The simulation of networks	129
8.2.1 Features and limitations of the simulation program	129
8.2.2 Conclusion: Simulation of networks	130
8.4 Recommendations for future work	131
8.4.1 Topology changes	131
8.4.2 Removal of the limitations of the program	132
8.4.3 Congestion control and buffer allocation techniques	132
8.4.4 Data link protocols	132
8.4.5 Simulation of additional routing algorithms	133
8.5 Final words	134
APPENDIX A TERMINOLOGY AND SYMBOLS	136
APPENDIX B LITERATURE SURVEY	142
APPENDIX C THE MINIMUM DELAY ROUTING ALGORITHM	147
C.1 R G Gallager's minimum delay routing algorithm	148
C.1.1 Calculation of marginal delays	148
C.1.2 The 'update' algorithm	149
C.2 Calculation of blocked paths	150
C.3 Estimation of incremental delays $dB/df_{in}(s)$	151
C.4 Calculation of event times	152
C.5 Other calculations	154

CONTENTS

Page

APPENDIX D	MEMORY AND COMPUTATIONAL REQUIREMENTS . . .	155
D.1	Memory requirements	156
D.1.1	Flooding algorithm	156
D.1.2	Least cost routing algorithm	157
D.1.3	Adaptive routing algorithm	157
D.2	Computation requirements	158
D.2.1	Flooding algorithm	159
D.2.2	Least cost algorithm	159
D.2.3	The adaptive routing algorithm	159
D.3	Conclusions	160
REFERENCES		161

CONTENTS

Page

PART II: SIMULATION PROGRAM

CONTENTS	Part I	ii
	Part II	x
LIST OF FIGURES	Part I	xiii
	Part II	xiv
LIST OF GRAPHS	Part I	xv
LIST OF TABLES	Part I	xvii

1	INTRODUCTION	1
1.1	The network routing simulation program	1
1.2	Layout of Part II	2
2	ROUTING SIMULATION PROGRAM USER'S MANUAL	3
2.1	Introduction	3
2.1.1	Program requirements and specifications	3
2.2	Program operation	5
2.2.1	Getting started	5
2.2.2	Running the program	7
2.2.3	The output file	10
2.3	Conclusion	12
2.4	Input file format	13
2.4.1	Notation	14
2.5	Network example	14
2.5.1	The input file	16
2.5.2	The output file	17

CONTENTS

Page

3	DESIGN OF THE NETWORK ROUTING SIMULATION PROGRAM	19
3.1	First level design	19
3.2	Second level design	20
3.2.1	The initialization process	20
3.2.2	The get data process	20
3.2.3	Simulation process	20
3.2.4	Output process	21
3.3	Third and subsequent level designs	21
3.3.1	Program operation	21
3.3.2	Program variables	22
3.3.3	The network simulation program	23
1.	Initialize data process	24
2.	Get data from file	24
3.	Get data from user	25
4.	Shortest path	25
5.	Initialize node data	25
6.	Find insert position	26
7.	Insert event	26
8.	Enter events	26
9.	Random arrivals	27
10.	Arrival process	27
11.	Adaptive algorithm	28
12.	Short path	29
13.	Flood packet	29
14.	Generate departure	30
15.	Service process	30
16.	Departure process	31
17.	Process events	32
18.	Calculate results	32
19.	Calculate incremental delays	33
20.	Calculation of blocked paths	34
21.	Calculation of traffic	35
22.	Calculate new $SD/IR_{L(n)}$	35

CONTENTS

Page

23. Update routing data	36
24. Service routing data	37
25. Transmit new gD/dt_{new}	37
26. Transmit new dD/dt_{new}	38
27. Transmit routing data	39
28. Find the sink tree	40
29. Prune dead ends	40
30. Loop elimination	41
31. Output procedure	42
3.4 Conclusion	42
 4 CONCLUSION	 43
 APPENDIX A PROGRAM FLOW CHARTS	 45
 APPENDIX B LIST OF PROGRAM VARIABLES	 53
 APPENDIX C PROGRAM CALCULATIONS AND RESULTS.	 68
 APPENDIX D PROGRAM LISTING	 73
D.1 Main program listing	74
D.2 Routine to initialize the routing data	121
D.3 Procedures required for the minimum delay routing algorithm.	123
D.4 Procedures required for the elimination of loops in the network.	136
D.5 Procedures required for transferring routing data between the nodes in the network.	143

LIST OF FIGURES

PART I

Figure	Page
Figure 2.1 Diagram illustrating various network topologies.	3
Figure 2.2 Schematic representation of the OSI reference model	6
Figure 2.3 Diagram of a simple M/M/1 queuing system. . .	11
Figure 2.4 Diagram of a typical node in a network. . .	11
Figure 4.1 A simple digital network under test.	20
Figure 5.1 Diagram of a typical distributed network. . .	41
Figure 5.2 Schematic representation of a node.	44
Figure 5.3 Typical packet or frame format.	46
Figure 5.4 Diagram of two inter-connected nodes. . . .	50
Figure 5.5 Typical buffer allocation at a node.	53
Figure 6.1 Diagram of a typical node	60
Figure 6.2 Two neighbouring nodes in a network.	66
Figure 6.3 A simple network.	68
Figure 7.1 Diagram of a simple four node network. . . .	77
Figure 7.2 An example of a sink tree for source node 1 and destination node 5	98
Figure C.1 Diagram of a typical pair of nodes in the network	153

LIST OF FIGURES

PART II

Figure	Page
Figure 2.1 Diagram of a simple network.	14
Figure 3.1 The network simulation process	19
Figure A.1 Flow chart of the simulation program	46
Figure A.2 Flow chart of the simulation process	47
Figure A.3 Flow chart of the arrival process.	48
Figure A.4 Flow chart of the service process.	49
Figure A.5 Flow chart of the departure process.	51

LIST OF GRAPHS

PART I

Graph		Page
Graph 4.1	The average packet delay as a function of the of the input traffic to the network for various packet lengths	29
Graph 4.2	The percentage use of the network as a function of the input traffic to the network for various packet lengths	30
Graph 4.3	The packet delay for various input traffic loads	31
Graph 4.4	The percentage use of the network for various input traffic loads	31
Graph 4.5	The packet delay versus the input traffic load for various link capacity values . . .	32
Graph 4.6	The percentage use of the network versus the input traffic load for various link capacity values	32
Graph 5.1	Throughput versus offered load for a uncontrolled network compared with the throughput of an ideal network	49
Graph 7.1	Graph illustrating average packet delay for various service rates.	79
Graph 7.2	Graph illustrating the number of blocked packets for various service rates	80
Graph 7.3	Average packet delay as a function of the length of the input buffer pool	81
Graph 7.4	The number of packets blocked as a function of the input buffer pool	81
Graph 7.5	Tolerance versus the average packet delay. .	86
Graph 7.6	The total average packet delay as a function of the convergence rate of the algorithm . .	87

Graph	Page
Graph 7.7 Graph showing the effect of n on the average packet delay	89
Graph 7.8 Graph of the average packet delay for a network with and without piggy-backing . . .	94
Graph 7.9 Graph illustrating delay and number of blocked packets as a function of p	95
Graph 7.10 The percentage number of routing packets transmitted as the algorithm converges . . .	102
Graph 7.11 The total average packet delay for each routing algorithm	104
Graph 7.12 Graph illustrating average packet delays in a network	105
Graph 7.13 The number of blocked packets for the adaptive and least cost routing algorithms .	107
Graph 7.14 Graph illustrating total cost of each algorithm.	110
Graph 7.15 Delay distribution for the minimum delay routing algorithm	111
Graph 7.16 The average packet delay as a function of the input traffic to the network	113
Graph 7.17 Graph illustrating the overall throughput of the network as a function of the input traffic	114
Graph 7.18 The actual duration of a simulation versus the simulated duration of a network.	122

LIST OF TABLES

PART I

Table	Page
Table 7.1 Delay distribution for the least cost and minimum delay routing algorithms	112

1 INTRODUCTION

Part I of the report dealt with the selection of a particular network topology and routing algorithm. The routing algorithm was then optimized, for the particular network selected, subject to various input traffic loads. The performance of the algorithm was established with the aid of a simulation program.

Part II of the project deals specifically with the network routing simulation program. Provided in this part of the report is:

- a user manual,
- the program design,
- a list of the calculations performed in the program,
- a list of the program variables, and
- the entire program listing.

Part II of the report is important to the reader wishing to use or modify the simulation program in any way.

1.1 The network routing simulation program

Before describing the actual design of the simulation program the purpose of the program is summarized. The network routing simulation program is designed to simulate the routing of packets around any type of digital network. The packets may be routed using any one of three routing algorithms:

- a flooding routing algorithm,
- a least cost algorithm, and
- an adaptive routing algorithm based on R G Gallager's minimum delay routing algorithm.

The objective of the simulation program is to be able to evaluate the overall performance of the network subject to various input traffic loads. The performance of the network is established by observing the average packet delays, the number of packets blocked and the overall throughput of the network. The performance of each routing algorithm in a particular network may be compared using the simulation program. The simulation is a relatively quick and inexpensive method of analysing the performance of a network.

1.2 Layout of Part II

Chapter 2 includes a detailed "User Manual" of the simulation program. The inputs required as well as the outputs provided by the program are described in this chapter.

Chapter 3 is devoted to the design of the program. A description as well as a program descriptive language (PDL) representation of each procedure used in simulation program is provided. Chapter 4 concludes Part II of the report with some of the more important uses and limitations of the routing algorithms. For a detail explanation on the uses and limitations of the program the user is referred to Part I of the report.

The appendices of Part II of the report include the following information:

- Simple flow charts of the more important processes included in the program,
- a list and description of all the variables used in the program,
- a list of the calculations used in obtaining the outputs and results from the program, and
- the entire program listing.

2 ROUTING SIMULATION PROGRAM USER'S MANUAL

The "User Manual" provides the reader with a step by step guide on how to use the network routing simulation program. Each input required by the program is explained. The significance of the outputs to the program are also explained.

2.1 Introduction

The network routing simulation package is used to determine the performance of a digital network subject to various types of routing algorithms. The performance of the network is determined by observing the packets delays, number of packets blocked, average percentage use of the network and other network statistics. The program allows any network topology subject to various input traffic load conditions to be simulated.

2.1.1 Program requirements and specifications

The various inputs required by the program may be briefly listed as follows:

- The duration of the simulation,
- the network topology,
- the cost and capacities of the links in the network,
- the expected traffic for each source destination pair in the network,
- the average service rate of each node in the network,
- the size of the data packets in bits per second,
- the buffer sizes present at each node and,
- the routing algorithms to be simulated.

The simulation package is currently able to simulate any one of three routing algorithms:

- A simple flooding algorithm,
- A static least cost algorithm and
- an adaptive minimum delay algorithm based on R G Gallager's minimum delay routing algorithm.

At the end of the simulation numerous averages and results are calculated. The results are used to determine the overall performance of the network. The basic outputs required from the program are:

- The average delay of the packets in the network,
- the total traffic in the network,
- the percentage use of the links in the network,
- the number of packets blocked in the network,
- the number of routing packets generated in the network,
- the average cost per packet in transferring packets across the network and,
- the average hop delay in the network.

The output data is calculated and displayed at discrete intervals of time during the simulation. The performance of the network as a function of time or traffic flow may be observed. Long term averages for the entire simulation are also calculated and provided in the output of the program.

2.2 Program operation

A step by step user's manual on how to use the program is now provided. The program is implemented on an APOLLO domain computer and it is assumed that the user has sufficient

Knowledge to edit files and to execute programs on this computer.

2.2.1 Getting started

Before the simulation package can be executed the user must first create an input file for the program. The input file is a normal ascii text file which may be entered using a standard edit window. A template on the required format of the input file is given on page 13.

The first entry in the input file is a brief description of the network to be simulated. A maximum of 80 characters may be used for the description and if a longer description is entered it is simply truncated to 80 characters.

The next entry in the file is the number of nodes N in the network to be simulated and must be an integer value in the range $0 < N \leq 20$.

The next set of entries in the input file is the capacity of the links in the network. The capacities are entered as a matrix as shown on page 13 where the entry C_{ij} represents the capacity of link from node i to node j .

If L represents the set of all possible links in the network then the following applies:

$C_{ii} = 0$ for all i , and

$C_{ij} = 0$ for link (i,j) not an element of the set L .

The subsequent set of entries represents the cost of transmitting a packet across the links in the network. The data is entered in the same format as that for the link

capacities, where $D_{i,j}$ represents the cost of link (i,j) . Note that:

$D_{i,i} = 0$ for all i , and

$D_{i,j} = 0$ for all (i,j) not an element of the set L .

The next set of entries in the input file represents the expected traffic for each source destination pair in the network at various time intervals. The input format is as follows:

Source Destination Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_n

Source represents the number of the source node and destination represents the number of the destination node and

Y_0 is the expected traffic for the time period $T_0 < T \leq T_1$

Y_1 is the expected traffic for the time period $T_1 < T \leq T_2$

.....

Y_n is the expected traffic for the time period $T_{n-1} < T \leq T_n$

where $T_n = (\text{Duration of simulation}/x)n$ and

x = number of intervals simulation is divided into.

The source destination pairs may be entered in any order. Any source destination pairs not entered will be assumed to have zero traffic for the entire duration of the simulation.

The final set of data to be entered is the node number, service rate, input buffer length and output buffer length of each node in the network. All the nodes in the network must be entered.

2.2.2 Running the program

Once the input file has been entered and saved the Network Routing Simulation Package can be executed by typing 'NRSP.BIN' and then return. The user is then prompted as follows:

Enter the duration of the simulation:

A value between 0 and 1000 is required. One must remember that the simulation is divided into time increments of 1/1000 and a duration of say 1000, results in 1 000 000 increments which requires a great deal of time to complete.

Enter input filename (default INPUT.DAT):

The user must enter the name of the input file. The default file name is 'INPUT.DAT'. If the input file does not exist the error message

Error>> File not found

is displayed on the screen and the program is aborted.

Enter a seed for the random number generator:

Any value real or integer may be entered. This value is used to generate random arrivals of packets at the nodes.

Enter events manually (y/n) ? :

This option allows the user to enter individual arrival of packets at any node in the network destined for any other destination node in the network. If this option is selected random arrivals are not generated by the program. This option is useful for debugging purposes as a single packet can be entered and traced through the network.

Select routing algorithm to be simulated

1. Flooding algorithm
2. Least cost routing algorithm
3. Adaptive routing algorithm

Enter Choice:

Here the user must enter 1, 2 or 3 indicating which routing algorithm is to be simulated. This allows the user to compare the performance of the algorithms using the same network.

Enter output filename (Default 'RESULT.DAT'):

The user must enter the name of the file into which the results are to be stored. If no filename is entered the filename 'RESULT.DAT' is assumed. If the file already exists on the disk then it is simply deleted and the new file is stored onto the disk.

If the adaptive routing algorithm is selected for the simulation, then the user is requested for additional information as follows:

Enter convergence rate:

The user must enter the convergence rate (α) of the algorithm. The value entered should preferably be in the range $0 \leq \alpha < 5$.

Enter incremental delay tolerance:

The user must enter the maximum value the marginal delay $dD/df_{i,j}$ is allowed to vary before the value is transferred to node i . The value is typically in the range, $0 \leq \text{Tolerance} < 1$.

Enter maximum number of packets between updates:

An integer value greater than or equal to zero must be entered. The number represents the maximum number of packets that can be transferred between two nodes

Select routing algorithm to be simulated

1. Flooding algorithm
2. Least cost routing algorithm
3. Adaptive routing algorithm

Enter Choice:

Here the user must enter 1, 2 or 3 indicating which routing algorithm is to be simulated. This allows the user to compare the performance of the algorithms using the same network.

Enter output filename (Default 'RESULT.DAT'):

The user must enter the name of the file into which the results are to be stored. If no filename is entered the filename 'RESULT.DAT' is assumed. If the file already exists on the disk then it is simply deleted and the new file is stored onto the disk.

If the adaptive routing algorithm is selected for the simulation, then the user is requested for additional information as follows:

Enter convergence rate:

The user must enter the convergence rate (α) of the algorithm. The value entered should preferably be in the range $0 \leq \alpha < 5$.

Enter incremental delay tolerance:

The user must enter the maximum value the marginal delay $dD/df_{i,j}$, is allowed to vary before the value is transferred to node i . The value is typically in the range, $0 \leq \text{Tolerance} < 1$.

Enter maximum number of packets between updates:

An integer value greater than or equal to zero must be entered. The number represents the maximum number of packets that can be transferred between two nodes

before routing information is transferred between the nodes.

Enter number of traffic averaging packets:

The user must enter a integer number greater than zero, representing the number of packets the traffic is to be averaged over at a node.

Once all the data has been entered the simulation of the network commences, firstly all the necessary data is initialized and random arrivals generated. The actual simulation of the network then begins. At regular intervals during the simulation the current time of the simulation is displayed onto the screen to inform the user on the progress of the simulation.

At the end of the simulation, and if no errors occurred during the simulation, the results from the simulation are calculated and saved on disk under the filename previously entered.

2.2.3 The output file

The output file contains the following information:

- A. A description of the network simulated and the routing algorithm used in the network.
- B. General results: These are the results calculated at various time intervals during the simulation and include:
 - The time the results were calculated,
 - the total delays of all the packets in the network during the time interval,

- the number of packets blocked during that time interval,
- the average delay per packet for that time interval,
- the average cost per packet for the time interval,
- the average hop delay and
- the number of packets generated during that time interval.

C. Interval results: These results are calculated at various time intervals during the simulation and the result referred to the time interval when the packet was actually generated. For example assuming the delay of a packet generated during interval a is calculated during interval b, then the delay will be represented at interval b. This is useful since the packets are generated for a constant time period determined by the duration of the simulation and all delays can be referred to the expected traffic in the network. Some of the interval results provided include:

- The interval number and time,
- the expected traffic for that time interval,
- the actual generated traffic for the time interval,
- the percentage use of the network and,
- the average packet delay.

D. Routing information results: If the adaptive routing algorithm is simulated the percentage number of routing packets generated, not including the routing packets piggy backed, is calculated at various time intervals during the simulation.

E. Overall average results: These results contain averages or totals calculated over the entire duration of the simulation and include the following results:

- The entered duration of the simulation,
- the actual duration of the simulation, that is the time until all the data packets are transferred to the destination nodes or are discarded,
- the total average packet delay,
- the total average hop delay,
- the maximum packet delay,
- the total number of packets generated,
- the total number of packets discarded due to looping,
- the total number of packets blocked in the network,
- the percentage number of packets blocked per number of hops in the network,
- the total number of packets blocked due to no path available from a particular source node to destination node,
- the percentage of the time spent by the packets waiting in the queues of the network, and
- the total cost of running the network.

It is important for the user to correctly interpret the results obtained from the simulation. For example the number of packets blocked and discarded in the flooding algorithm does not give a good indication of the actual performance of the flooding algorithm. Many of the packets blocked or discarded will be duplicate copies of the original packet. The number of blocked packets in this case is not an indication of the number of packets unable to reach the destination node.

The results obtained from the simulation program are often best understood or interpreted using a graphics program to graphically represent the data.

2.3 Conclusion

The program has been made as flexible as possible allowing various network topologies to be simulated under varying input traffic load conditions. At the same time the actual utilization of the program has been kept as simple as possible. An example of an input and output file of a typical network is given in section 2.5.

2.4 Input file format

<<Network description>>

No. of nodes.

N

Bits per packet

B

Link capacities

00	C ₁₂	C ₁₃	C ₁₄	.	.	.	C _{1N}
C ₂₁	00	C ₂₃	C ₂₄	.	.	.	C _{2N}
C ₃₁	C ₃₂	00	C ₃₄	.	.	.	C _{3N}
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
C _{N1}	C _{N2}	C _{N3}	C _{N4}	.	.	.	00

Link Costs.

00	D ₁₂	D ₁₃	D ₁₄	.	.	.	D _{1N}
D ₂₁	00	D ₂₃	D ₂₄	.	.	.	D _{2N}
D ₃₁	D ₃₂	00	D ₃₄	.	.	.	D _{3N}
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
D _{N1}	D _{N2}	D _{N3}	D _{N4}	.	.	.	00

Expected traffic

Source	Dest.	Y1	Y2	Y3	.	.	Y _N
1	2	f ₁₂₍₁₎	f ₁₂₍₂₎	f ₁₂₍₃₎	.	.	f _{12(N)}
1	3	f ₁₃₍₁₎	f ₁₃₍₂₎	f ₁₃₍₃₎	.	.	f _{13(N)}
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
a	b	f _{ab(1)}	f _{ab(2)}	f _{ab(3)}	.	.	f _{ab(N)}

00 {END MARKER}

Node	Service_rate	I/P_Buffer_length	O/P_buffer_length
1	S ₁	IBL ₁	OBL ₁
2	S ₂	IBL ₂	OBL ₂
.	.	.	.
.	.	.	.
N	S _N	IBL _N	OBL _N

2.4.1 Notation

The following notation has been used to describe the input file on the previous page:

- N = The total number of nodes in the network.
- B = The number of bits per packet.
- $C_{i,j}$ = The capacity of link (i,j) .
- $D_{i,j}$ = The cost of link (i,j) .
- $F_{i,j}(k)$ = The total expected traffic from source node i to destination node j during time interval k .
- x = The total number of intervals in the simulation.
- S_i = The service rate of node i .
- IBL_i = The input buffer length of node i .
- OBL_i = The output buffer length of node i .

The template given above illustrates the format of a general input file. An example of a typical input and output file is given below:

2.5 Network example

The actual format of the input and output files of the simulation program was best illustrated by the use of a simple example.

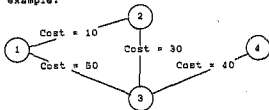


Figure 2.1 Diagram of a simple network.

The input file of this network is given on the following page. The network is then simulated for a duration of 20 units using a least cost routing algorithm. The output file or results from the simulation is shown on page 17.

2.5.1 The input file

Network A

No. of nodes.

4

Bits per packet

800

Link capacities.

00 20 20 00

20 00 20 00

20 20 00 20

00 00 20 00

Link costs.

00 10 50 00

10 00 30 00

50 30 00 40

00 00 40 00

Expected traffic.

Source	Dest.	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10
1	2	8	8	8	8	8	8	8	8	8	8
1	3	1	2	4	8	10	10	4	3	2	1
1	4	6	6	6	6	6	6	6	6	6	6
2	1	5	5	5	5	5	5	5	5	5	5
2	3	8	8	8	8	8	8	8	8	8	8
2	4	7	7	7	7	7	7	7	7	7	7
3	1	8	8	8	8	8	8	8	8	8	8
3	2	6	6	6	6	6	6	6	6	6	6
3	4	7	7	7	7	7	7	7	7	7	7
4	1	5	5	5	5	5	5	5	5	5	5
4	2	5	5	5	5	5	5	5	5	5	5
4	3	4	4	4	4	4	4	4	4	4	4

2.5.2 The output file

Simulation results

Algorithms --> SP_OR_IC

General Results

No.	Time	Delays	Pkts_blocked	Delay/Pkts	Cost/Pkts	Hop_delay	Mbps
1	0.0160	30.0660	0	0.3131	63.4375	0.1373	96
2	2.0050	97.0570	0	0.8589	41.8814	0.3913	113
3	4.0019	150.8339	15	1.3231	45.6140	0.5735	114
4	6.0018	179.8209	30	1.5896	48.5941	0.6455	113
5	8.0006	188.2830	51	1.8762	51.6414	0.7129	111
6	10.0096	176.4190	52	1.6488	54.6729	0.7346	107
7	12.0022	176.6371	36	1.4807	56.2393	0.6917	117
8	14.0096	185.3450	32	1.3492	46.9027	0.6745	113
9	16.0010	193.4510	29	1.2254	46.6387	0.6679	119
10	18.0003	173.3900	34	1.5163	51.3636	0.6708	116
11	20.0092	167.1670	0	1.4886	45.7576	0.6051	99

Interval Results

No.	Time	Exp_traffic	Gen_traffic	Loss	Inf_delay
1	0.0160	70	141	49.02	0.4619
2	2.0050	71	150	49.38	1.1783
3	4.0019	73	148	55.00	1.5707
4	6.0018	77	149	51.78	1.6769
5	8.0006	79	156	56.30	1.6528
6	10.0096	79	162	52.70	1.6731
7	12.0022	73	144	52.80	1.6635
8	14.0096	72	143	49.21	1.6381
9	16.0010	71	143	51.77	1.5407
10	18.0003	70	150	51.52	1.3915

Overall average results

Duration of newly generated arrivals	=	20.0000
Total duration of simulation	=	21.5707
Total average packet delay	=	1.4342
Delay per packet transmitted	=	1.1698
Total average delay per hop	=	0.5973
Maximum packet delay	=	2.1930
Total number of packets generated	=	1486
Total number of packets discarded	=	0
Total number of hops (pkts)	=	2174
Total number of blocked packets	=	274
Percentage of blocked packets/hop	=	12.59
Packets lost due to no path	=	0
Total S service + I/P queuing time	=	99.59
Total S O/P queuing time	=	6.17
Total S transmission time	=	0.24
Total cost of running network	=	57370.00

3 DESIGN OF THE NETWORK ROUTING SIMULATION PROGRAM

The design of the actual program is now described. A program descriptive language is used to describe the operations of the more complex procedures in the program. The simulation program was designed using a object-orientated approach. In an object-orientated design the resources (data) are first specified and then the operations to be performed on the resources are defined. A object-orientated approach was used since it leads to a maintainable and easy modifiable program [7].

3.1 First level design

The first part of the design is to consider the entire program as a 'black box' and to decide on the inputs and outputs to the program. At a first level design the process can be illustrated as,



Figure 3.1 The network simulation process

BEHAVIOUR: The behaviour of the process can be described briefly as follows: Data such as network topology, duration of the simulation etc. is entered into the process, the network is simulated using a particular routing algorithm and the results from the simulation are sent to a file or the screen.

3.2 Second level design

The process described above is now broken down further into smaller more well defined processes. This process is continued until each process is well defined and operates on well defined resources.

3.2.1 The initialization process

INPUTS: Data

OUTPUTS: Data

BEHAVIOUR: The behaviour of this process is to initialize all the necessary data of the program.

3.2.2 The get data process

INPUTS: Keyboard, File

OUTPUTS: Data, Screen

BEHAVIOUR: The purpose of this process is to obtain the data such as the network topology, link capacities, link costs etc. The data is entered via a file and keyboard.

3.2.3 Simulation process

INPUTS: Data, Keyboard

OUTPUTS: Data, Screen

BEHAVIOUR: This process forms the main part of the program and involves simulating the routing of data packets around a network. The main functions of this process is to:

- Simulate packets arriving, being serviced and departing for each node in the network.
- Update routing variables if required.
- Calculate which path or paths a packet should be transmitted on and forms part of the service process at a node.
- Calculate statistical information about the network.
- To accept and service interrupts from the user.

3.2.4 Output process

INPUTS: Data

OUTPUTS: Screen, file

BEHAVIOUR: The function of this process is to process the results obtained from the simulation and to send this information to the user via the screen or output file.

3.3 Third and subsequent level designs

The above second level processes are further subdivided into smaller more manageable processes. The actual design specifications of the third and subsequent levels of the design are not described in the report. The functions of the processes defined in the final level of the design are provided. Before discussing each of the processes used in the program a brief description of how the program operates is given.

3.3.1 Program operation

The task of simulating a network can be simplified by considering the events occurring in a network. Basically three events can occur:

1. The arrival of a packet at a node,
2. The servicing of a packet at a node and,
3. The departure of a packet from a node.

The main problem with simulating a network is that in a network numerous events occur simultaneously across the network. In a program it is only possible to handle one of the events at a time. Hence the events occurring in the network must be serialized. Serialization of the events is accomplished by inserting the events in chronological order

into a single queue. The events are then extracted in from the queue (in chronological order) and processed. The program commences by generating random packet arrivals to each node in the network. These arrivals are inserted into the event queue. During the simulation the events are extracted from the queue and any additional events are inserted into the appropriate position in the queue. For example suppose that the event extracted from the queue indicates the arrival of a packet at node i destined for node j. The next event that must occur is that the packet must be serviced by node i. The time of service is calculated and the event is inserted onto the event queue. When the service event is extracted from the queue the packet is serviced by node i and an output path or paths is calculated using a particular routing algorithm. The departure time of the packet from node i is then calculated and the event is inserted in the correct order into the event queue. The process continues for all the nodes in the network and events in the event queue.

3.3.2 Program variables

Before describing each of the processes used in the program in detail some of the more important classes of variables used in the program are briefly discussed:

Duration: This variable determines the duration of the simulation. The duration of the simulation is divided into small time increments. The smaller the increments the more accurate the simulation but the longer the time taken to complete the simulation.

Network: This variable contains information about the status of the network, such as the topology of the network, link capacities, link costs and the expected traffic in the network.

Node: This variable contains the information about the status of a particular node in the network. The variable contains data such as queue lengths of the node, the service rate of the node and the routing information contained at the node.

Event: This variable contains the information about an event in the event queue. The variable contains the time of occurrence of the event, the node number at which the event occurs, the type of event and also contains information about the packet associated with the event.

Packet: The packet contains information such as the destination address of the packet, the source address of the packet, the number of hops and whether the packet contains routing information or purely data.

Statistical information: This contains all the results such as average queue lengths, average number of hops, cost of running the network etc. in order that useful results may be obtained from the simulation program.

3.3.3 The network simulation program

A description of the processes used in the program is given. Where necessary a program descriptive language (PDL) explanation is provided for the processes. The processes described below are obtained by the breakdown of the processes illustrated in second level design given above into sequentially smaller processes. Only the more important processes are given here:

1. Initialize data process

INPUTS: Network definition, Node status and statistical results.

OUTPUTS: Network definition, Node status and statistical results.

BEHAVIOUR: The purpose of the process is to initialize all the necessary data such as queue lengths etc. to zero.

Process: Initialize data

```
BEGIN
    * Initialize expected traffic for all source *
    * destination pairs. *
    * Initialize queue lengths to zero. *
    * Initialize Routing variables. *
    * Initialize all statistical information *
END.
```

2. Get data from file

INPUTS: Filename

OUTPUTS: Number of nodes, Network definition, Status

BEHAVIOUR: The behaviour of this process is to read in the data from the input file.

Status: Successful

Unsuccessful: Input file not found or input file has incorrect format.

Process: Get data from file

```
BEGIN
    * Open input file *
    IF (* file successfully opened *)
    THEN
        BEGIN
            * Read in number of nodes *
            * Read in link capacities *
            * Read in link costs *
            * Read in expected traffic *
            * Read in service rates of each node *
            * Read in buffer sizes of each node *
        END
    END
END.
```

3. Get data from user

INPUTS: keyboard

OUTPUTS: Duration of simulation, Routing algorithm and Output filename.

BEHAVIOUR: The behaviour of this process is simple, the duration, name of the input file, type of routing algorithm to be simulated and the name of the file to which the output is to be sent is entered by the user.

4. Shortest path

INPUTS: Link costs, Source node, destination node.

OUTPUTS: least cost path.

BEHAVIOUR: The behaviour of this process is to determine the least cost path from the source node to the destination node. The process is used to route packets using the least cost routing algorithm. The algorithm used is based on A S Farnbaum's least cost routing algorithm [43].

5. Initialize node data

INPUTS: Number of nodes in network, network definition.

OUTPUTS: Node status.

BEHAVIOUR: At the beginning of each simulation it is assumed that each node has full knowledge of the topology of the network and if the adaptive algorithm is to be simulated then the routing tables at each node are initialized using a least cost routing algorithm.

Process: Initialize_node_data

BEGIN

* Initialize each node with link capacities.*

* Initialize routing tables using a least *

* cost algorithm.*

END.

6. Find insert position

INPUTS: Pointer to the front of the queue.

OUTPUTS: Pointer to the position where the event is to be inserted.

BEHAVIOUR: The procedure finds the correct chronological order in which an event should be inserted into the event queue.

7. Insert event

INPUTS: Pointer to position where the event is to be inserted into the event queue, the event to be inserted into the event queue, length of the event queue.

OUTPUTS: Length of the event queue, status.

BEHAVIOUR: The new event is inserted in chronological order into the event queue.

Status outcome: Successful

Unsuccessful: Event queue full.

8. Enter events

INPUTS: Number of nodes in the network, duration of simulation, event queue length.

OUTPUTS: Event queue length.

BEHAVIOUR: This procedure allows the user to enter arrivals manually into the event queue instead of letting the program generate random arrivals. This process is useful for debugging purposes and allows a single packet to be inserted into the network and traced through the net.

Process: Enter_event

```
BEGIN
  REPEAT
    REPEAT
      * Get time of arrival event from the user *
    UNTIL (* Time < duration *)

    REPEAT
      * Get source destination node from user *
    UNTIL (* source destination pair is valid *)

    * Insert event into the event queue *
  UNTIL (* User has entered all the events *)
END.
```

9. Random arrivals

INPUTS: Number of nodes, duration, expected traffic, length of event queue, pointer to front of the event queue, statistical information.

OUTPUTS: Length of the event queue.

BEHAVIOUR: Random arrival of packets to each node are generated, using a Poisson probability density function. The arrival rate is given by the expected traffic. In order to vary the arrival rates over the duration of the simulation the duration of the simulation is divided into intervals of equal length. The expected traffic for each source destination pair may vary from interval to interval. This allows the network to be simulated for a dynamic or varying input traffic load.

Process: Random_arrivals

```
BEGIN
  * Current clock := 0 *
  FOR * All valid source destination pairs * DO
    REPEAT
      * Generate random number *
      * Determine probability of arrival *
      IF (* Random number < Probability *)
        THEN
          BEGIN
            * Create arrival event *
            * Insert event onto the event queue *
          END
          * Increment the current clock *
        UNTIL (Current clock > duration)
    END.
END.
```

10. Arrival process

INPUTS: Interval number, Event definition, Pointer to front of the queue, Length of the event queue.

OUTPUTS: Length of the event queue, Statistical info.

BEHAVIOUR: This is one of the most important processes in the program and simulates the arrival of a packet at a node. The behaviour of this process is best described using PDL notation.

Process: Arrival_process

```
BEGIN
  IF (* Packet has not been generated at this
      node *)
  THEN * Update cost of transferring packet to *
      * the node.
  IF ((* Incoming pkt is a pure routing pkt *)
      AND (* I/P queue is not empty *))
  THEN
    BEGIN
      * Search event queue for pure data pkt to *
      * be serviced at node i.
      IF (* data packet found *)
      THEN
        BEGIN
          * Insert routing data into data pkt *
          * Finished := TRUE
        END
      END
    IF (* not finished *)
    THEN
      BEGIN
        IF (* Input queue is full *)
        THEN * Discard pkt and increment number *
            * of packets blocked.
        ELSE
          BEGIN
            * Decrement buffer pool & increment *
            * the input queue.
            * Calculate service time.
            * Insert service process into the *
            * event queue.
          END
        END
      END
    END
  END.
```

11. Adaptive algorithm

INPUTS: Event definition, node status.

OUTPUT: Output path number, node status.

BEHAVIOUR: The function of the adaptive algorithm process is to determine which output path number a data packet is to be sent on according to the adaptive routing algorithm employed by the simulation program. In the adaptive algorithm used in the simulation program a packet is routed along one of the output paths according to the output probabilities assigned to each path by the routing algorithm.

Status outcomes: Successful
Unsuccessful: No output path
found.

Process: Adaptive algorithm
BEGIN
 * get a random number *

 REPEAT
 * select an output path *
 Total probability := Total probability +
 probability for the output path

 IF (Random number < Total probability)
 THEN * path found *
 UNTIL ((Path found) or (no more output paths))
 END.

12. Short path

INPUTS: Event definition, Node status.

OUTPUTS: Output path, Status.

BEHAVIOUR: The packet is sent along an output path according to the least cost algorithm. Since only one path is available for the least cost algorithm the packet is sent across this link. The operation of the process is straight forward.

Status outcomes: Successful
Unsuccessful: No output path
found.

13. Flood packet

INPUTS: Event definition, Node status.

OUTPUTS: Output paths.

BEHAVIOUR: A copy of the packet is sent along each outgoing link of the node. The packet is not sent on the link on which it arrived. This simulates a flooding algorithm which has been inserted into the program so that the performance of other algorithms may be compared to the performance of the flooding algorithm in the network.

14. Generate departure

INPUTS: Output path number, time, interval number, duration, event definition, node status.

OUTPUTS: Event definition, node status, event queue length, statistical information.

BEHAVIOUR: The process simulates the queuing of a packet onto the output queue at a node.

Process: Generate_departure

```
BEGIN
  IF (* Output queue is full *)
    THEN * Packet blocked on the output queue *
    ELSE
      BEGIN
        * Calculate the departure time *
        * Increment number of packets in the *
        * the output queue. *
        * Update the event. *
        * Insert event into the event queue. *
      END
    END.
END.
```

15. Service process

INPUTS: Interval number, duration, node status, event definition, pointer to front of the queue, queue length.

OUTPUTS: Event definition, Length of the event queue, node status, statistical information.

BEHAVIOUR: The function of this process is to simulate the operation of a packet being serviced at a node. The PDL description of the process has been greatly simplified so that the logic involved in the process is clearly visible.

Process: Service_process

```
BEGIN
  * Decrement input queue *
  * Increment buffer pool *
  * Calculate input traffic rate (See pg 215) *
  IF ((* pkt has not been created at this node *)
    AND (* packet contains data *))
    THEN
      BEGIN
        * Calculate the hop delay *
        * Calculate incremental delay for link k,i *
        * If necessary transmit new incremental *
        * to node k. *
      END
    END.
```

```

IF (* packet contains routing information *)
THEN
  IF (* Packet contains routing information
      only *)
  THEN * Calculate delay of routing packet *
      * service the routing information *

  IF (* Packet contains data *)
  THEN
    IF (* Packet has reached it's destination *)
    THEN
      BEGIN
        * Calculate delay of packet *
        * Consume packet *
      END

    ELSE * Packet has not reached it's dest. *
      IF (* No of hops > maximum number of hops
          allowed *)
      THEN * Discard packet *
      ELSE * route data packet *
      BEGIN
        * select a routing algorithm *
        IF (* No output path found *)
        THEN * Discard packet *
        ELSE
          BEGIN
            * Calculate the departure(s) of *
            * the packet. *
            * Add departure event(s) to the *
            * event queue. *
          END
        END
      END
    END
  END
END.

```

16. Departure process

INPUTS: Network data, Node status, Event definition, Pointer to the front of the queue, Length of the event queue.

OUTPUTS: Event definition, Length of the event queue, statistical information.

BEHAVIOUR: The process simulates the departure of a packet from a node.

```

Process: Departure process
BEGIN
  * Decrement No. of pkts in the output queue *
  * Calculate time of arrival at the next node *
  * Create arrival event *
  * Add event to the event queue *
END.

```

17. Process events

INPUTS: Duration, Network data, Pointer to front of the queue, Status of all nodes, Length of the event queue.

OUTPUTS: Statistical information.

BEHAVIOUR: The basic behaviour of this process is to sequentially extract events from the front of the event queue and processed until all the events in the event queue have been processed or the maximum time allowed for the simulation has been exceeded.

Process: Process_events

```
  BEGIN
    REPEAT
      * Extract event from the event queue *
      * Decrement length of event queue *
      CASE event OF
        arrival: Call: Arrival_process
        service: Call: Service_process
        departure: Call: Departure_process
      END CASE
    UNTIL ((Length of event queue = 0) OR
           (Current time > 2xduration))
  END.
```

18. Calculate results

INPUTS: Statistical results

OUTPUTS: Statistical results

BEHAVIOUR: The behaviour of the process is to calculate average delays, costs etc. that are required as output to the program. A listing of all the statistical calculations performed in the program is given in Appendix C.

The processes described above are the main procedures required by the simulation program in general. The following processes described below are specific to the particular routing algorithm employed in the program.

19. Calculate incremental delays

As a packet arrives at a node i from node k the incremental delays are calculated. The incremental delays are estimated using an algorithm developed by A. Segall [39]. This process was developed independently of the rest of the program and was added to the program once it had been fully tested.

INPUTS: Arrival time of the packet A_m and the departure time of the packet D_m , new busy period.

OUTPUTS: $dD/df_{m,i}$, New busy period.

BEHAVIOUR: The function of this process is to calculate the incremental delays $dD/df_{m,i}$. Every $\max_m$ packets or when $dD/df_{m,i}$ changes beyond a certain tolerance the value of $dD/df_{m,i}$ is transmitted to node k .

The following procedure is used to calculate $dD/df_{m,i}$.

Let A_m be the arrival time of packet m on link (i,k) ,
 D_m be the departure time of packet m on link (i,k) ,
 m,n represent packets arriving on link (i,k) in chronological order, that is packet $m-1$ arrives before packet m and packet m arrives before packet $m+1$ etc.,

Process: Calculate incremental delays:

```

BEGIN
  IF (* Beginning of new busy period *)
  THEN * Let  $m = 0$ ,  $d_{m-1} = 0$ ,  $d_{m-2} = 0$ . *

  * Increment value of  $m$  *
  *  $C_{m-1} := 0$  *
  * Increment  $N_i$ , the current number of packets *
  * arriving on link  $(i,k)$  during the busy *
  * period. *

  FOR (*  $l := 1$  TO  $m$  *) DO
  BEGIN
    *  $C_m^n = 0$  for  $m < n$  *
    *  $C_m^n = D_n - A_m$  for  $m = n$  *
    *  $C_{m+1}^n = D_n - \max\{A_{n-1}, D_{n-1}\}$  *
    * where  $D_0 = 0$  *
    *  $C_m^n = \min(C_{m-1}^n, D_{m-1} - A_m)$  *
    * For  $m > n + 1$  *
  END

```

```

Dm-2 := Dm-1; Dm-1 := Dm
dD/dfNi(2) := [(dD/dfNi(2))*(Ni - 1)) + Ctree]/Ni
* Calculate change in dD/dfNi(2) *
IF ( (* Change in dD/dfNi(2) > Tolerance *)
    OR (* m > maximum m allowed *)
    THEN * Start a new busy *
END.

```

20. Calculation of blocked paths

In order to prevent packets from looping around the network it is necessary to define, at each node, a set of paths on which a packet destined to a particular node may not be sent, since if the packet is sent along the path then the packet may loop indefinitely around the network. The strategy to determine the set of blocked paths is based on R G Gallager's algorithm [11] which is given in Part I, Appendix C.

INPUTS: Number of nodes, current node number, destination node number, routing information at node i.

OUTPUTS: Routing information at node i.

BEHAVIOUR: Calculation of blocked links for each destination. The PDL description given below also indicates the protocol used for determining the blocked path set for each destination.

```

Process: Calculate blocked path:
BEGIN
  FOR * each neighbouring node (m) * DO
    BEGIN
      * Check for improper paths *
      IF (*  $\theta_{im(2)} > 0$ 
          and  $iD/\delta r_{m(2)} \leq iD/\delta r_{N(2)}$ 
          and  $\theta_{im(2)} \geq \beta[D_{im}(f_{im}) + iD_{m(2)} -$ 
               $iD/\delta r_{i(2)})/L_{i(2)} *$ 
          THEN * Tag routing packet *
          IF (* Any received routing packet
              was tagged *)
            THEN * Tag routing packet *
          END {for}
    END.

```

21. Calculation of traffic

The routing algorithm used requires the estimation of the incoming traffic at node i destined for node j .

INPUTS: The current time, the maximum number of packets between updates, the time of the last update and the number of packets serviced since the last update.

OUTPUTS: The new estimation of traffic $t_{i,j}$.

BEHAVIOUR: The purpose of the procedure is to estimate the current traffic flow at node i destined for node j . The traffic flow is incrementally averaged until the number of packets averaged is equal to the maximum number of packets allowed per average and then a new average is calculated.

```
Process: Calculate traffic:
BEGIN
  * Increment number of packets serviced *
   $t_{i,j} = \frac{\text{Number of packets serviced}}{\text{Current time} - \text{last update time}}$ 

  IF (* Number of packets serviced = maximum
      number of packets between updates *)
  THEN
    BEGIN
      Number of packets serviced := 0
      Last update time           := current time
    END
  END.
```

22. Calculate new $3D/3r_{i,j}$

The marginal delay with respect to the traffic generated at node i destined for node j is calculated. The value is then transmitted to all the neighbouring nodes k for which $3r_{i,k} > 0$ and k not equal to j .

INPUTS: Number of nodes in the network, current node number, destination node number and the current routing information at node i .

OUTPUTS: The new value of $3D/3r_{i,j}$.

BEHAVIOUR: The new value of $\delta D/\delta r_{ik(s)}$ is calculated using the following formula:

```

 $\delta D/\delta r_{ik(s)} := \tau \theta_{ik(s)} [dD_{ik}/df_{ik} + \delta D/\delta r_{kn(s)}]$ 
Process: Calculate new  $\delta D/\delta r_{ik(s)}$ 
BEGIN
   $\delta D/\delta r_{ik(s)} := 0$ 
  FOR k := 1 TO * Number of nodes in network * DO
    IF (* Capacity of link (i,k) > 0 *)
      THEN
         $\delta D/\delta r_{ik(s)} := \delta D/\delta r_{ik(s)} + \theta_{ik(s)} [dD_{ik}/df_{ik} + \delta D/\delta r_{kn(s)}]$ 
      END.

```

23. Update routing data

The function of this procedure is to implement R G Gallager's routing algorithm.

INPUTS: Number of nodes in the network, the current node number i , the destination node number j and the routing information of the current node i .

OUTPUTS: New values for $\theta_{ik(s)}$.

BEHAVIOUR: The new values for the output probabilities for each output path at node i and destination node j are calculated.

```

Process: Update routing data
BEGIN
  * Calculate  $\tau := \min[dD_{ik}/df_{ik} + \delta D/\delta r_{kn(s)}]$  *
  * for  $n$  not an element of  $B_{ik(s)}$ . *
  FOR k := 1 TO * Number of nodes in the network * DO
    IF (*  $k$  an element of  $B_{ik(s)}$  *)
      THEN  $\theta_{ik(s)} := 0$  and  $\delta_{ik(s)} := 0$ 
      ELSE
        BEGIN
          *  $\alpha_{ik(j)} := dD_{ik}/df_{ik} + \delta D/\delta r_{kn(s)} - \tau$  *
          *  $\beta_{ik(s)} := \min[\theta_{ik(s)}, \alpha_{ik(s)}/t_{ik(s)}]$  *
        END
      END.
  FOR k := 1 TO * Number of nodes * DO
    IF ( $k \neq k_{min}$ )
      THEN *  $\theta_{ik(s)} := \theta_{ik(s)} - \beta_{ik(s)}$  *
      ELSE *  $\theta_{ik(s)} := \theta_{ik(s)} + \tau \beta_{ik(s)}$  *
    END.
END.

```

24. Service routing data

When routing information is received from a neighbouring node it must be serviced and the routing tables of the node updated.

INPUTS: The packet containing the routing information, the status of the node, the current pointer to the front of the event queue and the length of the event queue.

OUTPUTS: The status of the node including the routing tables and the length of the event queue.

BEHAVIOUR: When a node receives routing information from a neighbouring node it is first accepted and then the node checks to see if all the appropriate routing information has been received from all the neighbouring nodes before the routing tables at the node are updated. The routing information exchanged between the nodes is the marginal delays dD/d_{ik} and $\delta D/\delta_{ik}(s)$.

Process: Service routing data:

```
BEGIN
  * Consume routing data received *
  * For each destination node in the network *
  FOR j := 1 TO * Number of nodes * DO
    IF (* Current node i <> j *) DO
      BEGIN
        * Check if all dD/dfs have been received *
        * from all the neighbouring nodes for *
        * which  $\delta_{ik}(s) > 0$ . *
        IF (* All dD/dfs have been received *) DO
          BEGIN
            Call: Calculate new  $\delta D/\delta_{ik}(s)$ 
            Call: Transmit new  $\delta D/\delta_{ik}(s)$ 
            Call: Update routing data
          END
        END
      END
    END
  END
```

25. Transmit new $\delta D/\delta_{ik}(s)$

When a new value of $\delta D/\delta_{ik}(s)$ is calculated at node i it is necessary to transmit this information to all the neighbouring nodes of node i.

INPUTS: Present time, new value of $\delta D/\delta t_{i,j,k}$, i, j, k , node status, pointer to front of the event queue and the length of the event queue.

OUTPUTS: New event(s) in the event queue.

BEHAVIOUR: The new calculated value of $\delta D/\delta t_{i,j,k}$ is transmitted to all neighbouring nodes in the network except for neighbours k and j and those in the blocked path set.

```
Process: Transmit new  $\delta D/\delta t_{i,j,k}$ 
BEGIN
  FOR  $m := 1$  TO * Number of nodes * DO
    IF ( (* Capacity of link  $(i,m) > 0$ 
      AND (*  $m \neq k$  *)
      AND (*  $m \neq j$  *)
      AND (*  $m$  not an element of  $B_{i,j}$  *) )
    THEN Call: Transmit routing data
  END.
```

26. Transmit new $dD/df_{i,k}$

When a new value of $dD/df_{i,k}$ calculated at node i it may be necessary to transmit this information to neighbouring node k .

INPUTS: Present time, new value of $dD/df_{i,k}$, i, j, k , node status, pointer to front of the event queue and the length of the event queue.

OUTPUTS: New event(s) in the event queue.

BEHAVIOUR: The new calculated value of $dD/df_{i,k}$ is transmitted to all neighbouring node k .

```
Process: Transmit new  $dD/df_{i,k}$ 
BEGIN
  IF (* Capacity of link  $(i,k) > 0$  *)
  THEN Call: Transmit routing data
END.
```

27. Transmit routing data

The function of the transmit routing data procedure is to create a packet containing the new routing information. The event of the packet departing from node i is inserted into the event queue. An important modification that has been made to this procedure is that routing packets are piggy backed along with other data packets if possible. Piggy backing is implemented by checking the output queue of the node, to see if it contains any data packets. If the output queue contains one or more data packets the routing information is piggy backed to the first data packet in the queue.

INPUTS: Present time, dD/df_{ik} , dD/dr_{ik} , i, j, k , Output path number, node status, pointer to the front of the event queue and the length of the event queue.

OUTPUTS: New event in the event queue representing the packet departing from node i .

BEHAVIOUR: The routing information is inserted into a packet and routed along the correct output path. If only the value of dD/df_{ik} is transmitted then dD/dr_{ik} is set to zero.

Process: Transmit routing information

```
BEGIN
  * Search output queue for data packet *
  IF (* data packet found *)
    THEN * Add routing data to data packet *
  ELSE
    BEGIN
      * Create new routing packet *
      * Insert departure into the event queue *
      * Increment event queue length *
    END
  END.
```

The assumption made in this process is that pure data packets, packets containing data and routing information as well as pure routing data packets are all made up of the same number of bits.

27. Transmit routing data

The function of the transmit routing data procedure is to create a packet containing the new routing information. The event of the packet departing from node i is inserted into the event queue. An important modification that has been made to this procedure is that routing packets are piggy backed along with other data packets if possible. Piggy backing is implemented by checking the output queue of the node, to see if it contains any data packets. If the output queue contains one or more data packets the routing information is piggy backed to the first data packet in the queue.

INPUTS: Present time, dD/df_{ik} , $dD/dr_{ik(s)}$, i , j , k , Output path number, node status, pointer to the front of the event queue and the length of the event queue.

OUTPUTS: New event in the event queue representing the packet departing from node i .

BEHAVIOUR: The routing information is inserted into a packet and routed along the correct output path. If only the value of dD/df_{ik} is transmitted then $dD/dr_{ik(s)}$ is set to zero.

Process: Transmit routing information

```
BEGIN
  * Search output queue for data packet *
  IF (* data packet found *)
    THEN * Add routing data to data packet *
  ELSE
    BEGIN
      * Create new routing packet *
      * Insert departure into the event queue *
      * Increment event queue length *
    END
  END.
```

The assumption made in this process is that pure data packets, packets containing data and routing information as well as pure routing data packets are all made up of the same number of bits.

28. Find the sink tree

This procedure forms part of the updated loop elimination process. The loop elimination process is used to find a set of blocked paths in the network such that packets are guaranteed not to loop in the network.

INPUTS: Network topology, source node i and destination node j .

OUTPUTS: Sink tree.

BEHAVIOUR: The process determines the sink tree.

Process: Find_sink_tree:

```
BEGIN
  * Find all output paths to the source node *
  FOR  $k := 1$  TO number of nodes DO
    IF (( $k \neq i$ ) AND (capacity of link ( $i,k$ ) > 0))
      THEN sink_tree( $i,k$ ) := connected;

  * Find output paths for all nodes except node *
  *  $i$  and node  $j$ . *
  FOR  $m := 1$  TO number of nodes [ ]
    IF (( $m \neq i$ ) AND ( $m \neq j$ ))
      THEN FOR  $k := 1$  TO Number of nodes DO
        IF (( $m \neq k$ )
          AND (sink_tree( $k,m$ ) = Not connected)
          AND (Capacity link( $m,k$ ) > 0))
          THEN sink_tree( $m,k$ ) := connected
      END.
END.
```

29. Prune dead ends

This process also forms part of the loop elimination process.

INPUTS: The newly calculated sink tree and destination node j .

OUTPUTS: Updated sink tree.

BEHAVIOUR: A dead end is defined as a node which only contains incoming paths, other than the destination node. A packet entering a dead end node k on its path to destination node j will not be able to be transmitted from node k and the packet will be blocked. Dead end nodes are removed by converting the incoming links to outgoing links. This may result in other nodes becoming dead end nodes. Therefore the process is repeated until no dead end nodes exist in the network for destination node j .

Process: Prune_dead_ends:

```
BEGIN
REPEAT
  FOR node_m := 1 TO Number nodes DO
    BEGIN
      * Check if only output paths to node *
      * m. (search for dead end). *
      * If node m is a dead end then update *
      * sink tree so that node m only has *
      * output paths. *
    END
  UNTIL (* No more dead ends found *)
END.
```

30. Loop elimination

This process eliminates all the loops in the network by defining a set of possible paths to each destination node in the network. The paths are defined from a sink tree for each destination which is guaranteed to be loop free.

INPUTS: Network topology.

OUTPUTS: The set of blocked paths for each destination node j.

BEHAVIOUR: A blocked path set for each destination node in the network is determined for each node in the network. The blocked path set guarantees loop freedom.

Process: Eliminate_loops:

```
BEGIN
FOR destination := 1 TO number of nodes DO
  BEGIN
    * select a source node i. *
    * Find sink tree for source node i and *
    * destination node j. *
    * Remove dead ends. *
    * Determine blocked path set from sink *
    * tree. *
  END
END.
```

31. Output procedure

This procedure calculates long term averages such as the average packet delay, percentage use of the network and other useful information used to determine the performance of the network. The results are then sent to an output file at the end of the simulation.

INPUTS: Statistical information gathered during the simulation.

OUTPUTS: Long term averages and important results.

BEHAVIOUR: The long term averages and results from the simulation are calculated and sent to an output file.

Process: Output procedure:

BEGIN

- * Perform calculations to determine long term *
- * averages, see Appendix C for a list of the *
- * calculations performed. *
- * Send results to an output file. *

END.

3.4 Conclusion

A description of all the important procedures in the simulation program has been provided. The description given above provides a brief illustration of the functions and purpose of each of the procedures in the program.

In order to provide a modifiable package the program has been subdivided into numerous procedures or processes. Each process performs a well defined function on a well defined set of resources or variables. Using the simulation technique employed in the program, the program may be modified to simulate other routing algorithms or network protocols.

4 CONCLUSION

Part II of the report has been entirely devoted to the design, implementation and use of the network routing simulation program. A guide on the use of the package has been provided as well an explanation of the protocols and formulae used in the program. This part of the report provides a detail analysis of the simulation package.

The program has been designed using as few assumptions as necessary. The program may be extended to simulate other routing algorithms and protocols if necessary. The network routing simulation package is capable of simulating any type of network topology consisting of up to 20 nodes. The network routing simulation program is able to provide accurate results in a moderately short time interval. The simulation program has a close one-to-one correspondence to the functions of a real network. For this reason the program provides the user with a greater understanding of the basic principles involved in analysing digital networks. One can conclude that the simulation program is an effective method of analysing a complex digital network.

APPENDICES

APPENDIX A

PROGRAM FLOW CHARTS

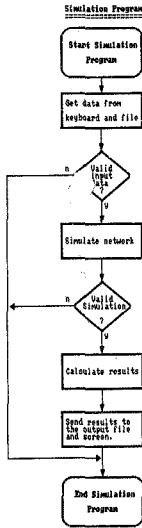


Figure A1 Flow chart of the simulation program

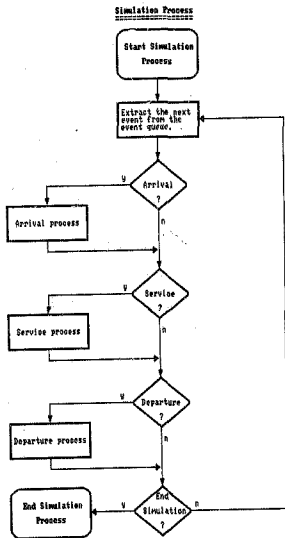


Figure A2 Flow chart of the simulation process

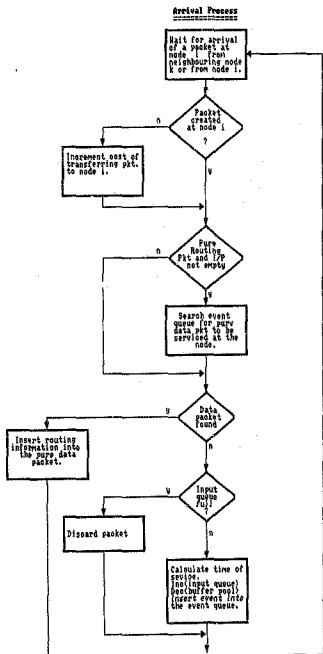


Figure B3 Flow chart of the arrival Process at a node

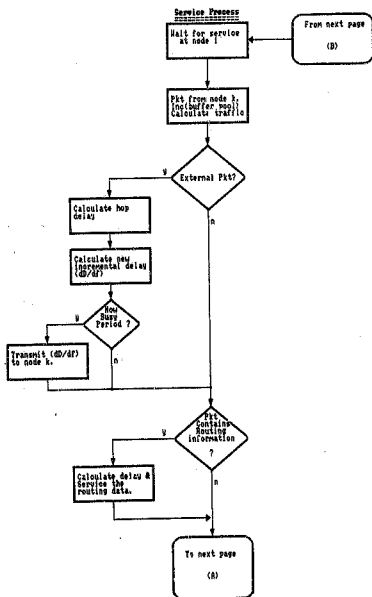


Figure A4 Flow chart of the service process at a node

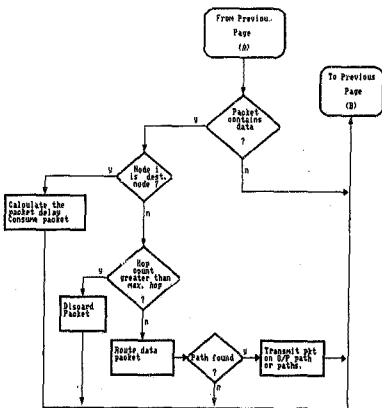


Figure A4 Flow chart of the service process (continued)

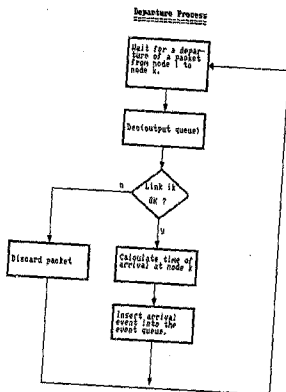


Figure 85 flow chart of the departure process at a node.

APPENDIX B

LIST OF PROGRAM VARIABLES

APPENDIX B

Data dictionary

A list and explanation of the constants and variables used in the program is provided. A understanding of the functions of the various variables in the program is essential to anyone wishing to modify the network routing simulation program.

B.1 Constants

The constants defined in the program are listed.

<u>Name</u>	<u>Type</u>	<u>Description</u>
Maxn	Integer	Maximum number of nodes allowed in the network.
Infinity	Integer	The largest time allowed in the simulation program before an infinite time is assumed.
No_of_intervals	Integer	The number of intervals the simulation program is subdivided into.
Max_q_length	Integer	The maximum length the event queue is permitted to have.
Max_intervals	Integer	The maximum number of intervals the simulation may be divided into.

The following 'constants' are used to determine the performance of the minimum delay routing algorithm and may be entered by the user at the beginning of the simulation if desired.

Max_trf_pkts	Integer	The number of packets the traffic at a node is averaged over.
--------------	---------	---

APPENDIX B

Alpha	Real	Determines the rate of convergence of the adaptive routing algorithm.
Max_n	Integer	The maximum number of packets that may be transmitted before the routing variables are updated.
Inc_delay_Tol	Real	The maximum tolerance allowed between marginal delay calculations before the routing variables are updated.

B.2 Types

All the various data types, such as records, used in the simulation program are listed.

<u>Name</u>	<u>Type</u>	<u>Description</u>
Node_ID	1..Maxn	All the possible node ID numbers possible in the network.
Intervals	1..Max_intervals	A list of all the possible interval numbers during a simulation.
Vector	Array	A vector array consisting of integer variables.
Rvector	Array	A vector array consisting of real variables.
Matrix	Array	A two dimensional array consisting of real variables.
MatrixA	Array	A three dimensional array consisting of integer variables.

APPENDIX B

MatrixB	Array	A three dimensional array consisting of real variables.
Intats_vector	Array	A (1 x Max_interval) array consisting of integer variables.
Rstats_vector	Array	A (1 x Max_interval) array consisting of real variables.
ID_sentence	Array	A (1 x 80) array of characters.

<u>Networks</u>	Record	This is a record which describes the network.
Network_ID_name	ID_sentence	Contains a description of the network.
Capacities	Matrix	Contains the capacity of all the links in the network and the network topology.
Cost	Matrix	Contains the cost of transferring a message across a link in the network.
Expected_traffic	MatrixA	The expected traffic for all source destination pairs in the network for each time interval during the simulation.
bp_packet	Integer	The number of bits contained in a packet.

<u>Incr_delays_data</u>	Record	Contains the data required to calculate the value of $dB/df_{k(s)}$ at any node in the network.
db_df_ki	Real	The current value of $dB/df_{k(s)}$.

APPENDIX B

Old_dD_df	Real	The previous value of $dD/df_{k,i(s)}$.
m	Integer	The current number of packets.
New_busy_period	Boolean	Determines if the new value of $dD/df_{k,i(s)}$ must be transferred to node k.
Ni_tot	Integer	The total number of packets between busy periods.
Dm_1,Dm_2	Real	Variables required by the algorithm.
Cm_1	Array	Previous calculated values of C_k required by the routing algorithm.
Inc_data	Incr_delays_data	The variables for dD/df_k for each destination node in the network.
Link_set	Set	The set of all possible links in the network.
<u>dD_dr_info</u>	Record	Record containing marginal delay $dD/dr_{k,i(s)}$ information.
dD_dr	real	The current value of $dD/dr_{k,i(s)}$.
Tag	boolean	Set to true if a blocked path via k to node j is detected.
<u>ADPT_info</u>	Record	A list of all the variables required by the adaptive routing algorithm.
Inc_delays	Inc_data	Record for storing the incremental delays at a node.
dD_df_ik	Array	A vector array for storing the value dD/df_{ik} for each destination node j.

APPENDIX B

dD_dr_data	Array	A two dimensional array of dD_dr_info for storing the values of $dD/dr_{k(i,j)}$.
B_i	Array	A vector array containing the set of blocked paths for each destination node j .
R_data_Rx	Array	A vector array containing the set of neighbours k for which the value of dD/dr has been received for a particular destination node j .
dD_df_Rx	Array	A vector array containing the set of neighbours k for which valid values of $dD/df_{k,i}$ have been received.
$OK_to_proceed$	Array	A vector array indicating, for each destination node j , if sufficient routing information has been received for the routing algorithm to be updated.

<u>Traffic status</u>	Record	A record containing the necessary data to calculate the value of $t_{k(i,j)}$ at node i .
No_of_pkts	Integer	The number of packets received between updates of $t_{k(i,j)}$.
Update_time	Real	The time which an update of $t_{k(i,j)}$ is referred to.
Total_traffic	Real	The current value of $t_{k(i,j)}$.

APPENDIX B

Traffic_info	array	A vector array of traffic_status and contains the traffic information at node i for each destination node j.
--------------	-------	--

<u>Routing_info</u>	Record	A record containing all the necessary routing information at a node including the routing information for the least cost and adaptive algorithms.
Number_of_nodes	Node_ID	Represents the number of nodes, that a node assumes is present in the network.
Link_capacities	Matrix	The link capacities as assumed by the node.
Link_costs	Matrix	The cost of the links.
Shortest_path_node	Vector	A vector of the output path number of the shortest path to a destination j in the network.
Output_prob.	Matrix	The values of $\theta_{ik(i)}$ at node i.
Phi	Matrix	The values of $\theta_{ik(i)}$ at node i.
Update_required	Boolean	This value is set true if the adaptive routing algorithm is required to be updated.
Traffic_data	Traffic-info	Contains the current values of $t_{i(j)}$ at node i.
ADPT_data	ADPT_info	Contains the data required by the adaptive routing algorithm.

APPENDIX B

<u>Node</u>	Record	The variable describing the status of a node i in the network.
Total_buffer_pool	Integer	The maximum number of buffers available in the input buffer pool.
Current_buffer_po.	Integer	The current number of buffers remaining in the input buffer pool.
Input_Q_length	Vector	The length of each input queue.
Output_Q_length	Vector	The length of each output queue.
Max_output_Q_len.	Integer	The maximum number of buffers available for each output queue.
Single_input_Q_l.	Integer	The length of the input queue at node i for packets generated by node i .
Bits_per_packet	Integer	The number of bits per packet.
Service_rate	Real	The service rate of the node.
Routing_data	Routing_info	Contains all the routing information present at node i .
All_nodes	Array	An array of node, and describes all the nodes in the network.
Packet_info	Set	Set of data, control or data and control and is used to describe whether a packet contains only data, routing information or both data and routing information.

APPENDIX B

Algorithm	Set	Least cost, flooding or adaptive. Describes which routing algorithm is to be simulated.
<u>General packet</u>	Record	Describes the basic contents of a packet in the network.
Time_created	Real	Describes the time the packet was originally created and is used to determine the delay of the packet.
Source_address	Node_ID	The number of the node where the packet was created.
Destination_addr.	0..maxn	The number of the destination node, 0 represent broadcasting.
Previous_node	0..maxn	The number of the node from which the packet has just been received. 0 represent no previous node.
Previous_serv_time	Real	The time the packet was previously serviced, and is used in calculating the hop delay.
Hop_count	Integer	The number of nodes the packet has passed.
Control_data	Pkt_info	Determines if the packet is a pure data or pure routing packet or a data packet with appended routing information.
dD_df	Real	The value of dD/df_{ik} to be transferred if any.

APPENDIX B

dDt_dr **dD_dr_info** The value of $dD/dr_{i,j}$ to be transferred if any.

Routing_algorithm **Algorithm** The name of the routing algorithm to be used to route the packet. This allows each individual packet to be routed using a separate routing algorithm.

Event_kind	Set	Arrival, service or departure, describes type of event.
Link	Pointer	Pointer to an event.
Event_info	Record	Describes an event in the event queue.
Next_event	Link	Pointer to the next event in the event queue.
Sentinel	Boolean	Set to TRUE if the event is the last event in the event queue, otherwise it is set to FALSE.
Time	Real	The time at which the event occurs.
Node_No	Node_ID	The node number at which the event occurs.
Output_node_No	Node_ID	If the event is a departure this stipulates to which node the packet is to be sent.
Name	Event_kind	The name or type of event.
Packet	General_packet	The definition of the packet associated with the event.

APPENDIX B

Cost	Real	The total cost of transferring the packet across the network, used to calculate the total cost of running the network.
Previous_arr_time	Real	The time of the previous arrival at a node, and is used to calculate queuing time.
Previous_dept_time	Real	The previous departure time of the packet and is used to calculate the transmission delay of the packet.

Error_status	-1..1	Determines type of error depending on value.
<u>OP_path_definition</u>	Record	When a packet is to be routed the destination address is sent to the routing algorithm and this record is used to define the path or set of paths the packet is to be routed on to reach that destination.
No_of_paths	Node_ID	The number of paths the packet is to be transmitted on.
List_of_paths	Vector	A list of the output path numbers k, on which the packet is to be transmitted on.
Path_found	Boolean	True if a valid path was found by the routing algorithm, else it is set to false.

APPENDIX B

<u>State info</u>	Record	A record containing all the necessary data required to calculate the results from the program.
Selected_algo.	Algorithm	The type of routing algorithm used in the simulation.

The following data are all integer values (of the type Intats_vector) recorded at each time interval during the simulation.

Total_packets_blocked	The total number of packets blocked.
Messages_transmitted	The number of successful packets.
Total_exp_traffic	The total expected traffic.
Total_gen_traffic	The total generated traffic.
Number of hops	Total number of hops of all the packets.
Interval_pkts_Rx	The total number of packets received referred to interval generated.
Interval_R_pkts_Rx	The total number of routing packets received referred to interval generated.

The following data are all real values (of the type Rstats_vector) recorded at each time interval during the simulation.

Percentage_R_pkts_Rx	The percentage of routing packets transferred.
Total_message_delay	Total data packet delay.
Interval_pkt_delays	Average delay/packet referred to interval generated.

APPENDIX B

Interval_R_pkt_delays	Average delay/routing packet referred to interval generated.
Total_interval_delays	Average delay/packet of data & routing packets.
Average_message_delay	The average delay/packet.
Interval_time	The simulated time at which the interval commenced.
Total_cost	The cost of running the network for that interval.
Hop_delay	The average delay/hop.
Average_percentage_use	The average percentage use of the links in the network.

The data structures defined below are required for the additional data used to calculate the final results of the simulation program.

Packets_per_link	MatrixA	The number of packets transferred over each link for each interval.
Percentage_use	MatrixB	The percentage use of each link in the network.
Total_pkts_gener.	Integer	The total number of packets generated for the entire simulation.
Total_pkts_Tx	Integer	Total number of successful packets.
No_of_pkts_block.	Integer	The total number of packets blocked for the entire simulation.

APPENDIX B

Total_pkts_discd.	Integer	The total number of packets discarded in the network, due to the failure of the routing algorithm to find an output link.
Percent_pkts_bl.	Integer	The percentage of the total number of packets blocked in the network.
Max_delay	Real	The maximum delay experienced by any packet in the network.
End_time	Real	The actual total simulated time.
Overall_msg_delay	Real	The total average packet delay.
Total_service_time	Real	The percentage time spent by the packets in the input queue and server of a node.
Total_dept_time	Real	The percentage time spent by the packets in the output queues of the nodes.
Total_tx_time	Real	The percentage time spent by the packets in the links of the network.
Overall_cost	Real	The total cost of running the network.
Delay_per_pkt_gen.	Real	The total packet delay per packet generated, including blocked packets.
Total_hop_delay	Real	The total average delay per hop.

Filename	Array	An array of characters forming a string.
=====		

APPENDIX B

B.3 Variables

The actual variables used in the program are listed and described.

<u>Name</u>	<u>Type</u>	<u>Description</u>
N	Node_ID	The total number of nodes in the network.
Network	Networks	The specification of the network.
Event	Event_in.	The description of an event in the network.
Front, back	Link	Pointers to the front and back of the event queue.
Seed	Real	A seed for the random number generator.
Nodes	All_nodes	A description of all the nodes in the network.
Total_duration	Real	The total duration of the simulation.
Event_Q_length	Integer	The current length of the event queue during a simulation.
Data_pkts_in_Q	Integer	The total number of data packets currently in the vent queue.
Sim_start_time	Time	The actual real time the simulation started.
Sim_end_time	Time	The actual real time the simulation ended.
Init_start_time	Time	The actual real time the initialization of the program started.

APPENDIX B

Num_of_sec	Integer	The actual duration of a simulation (real time).
No_of_periods	Intervals	The total number of intervals for the entire simulation.
OP_filename	filename	The name of the output file.

=====

The most important constants and variables defined in the program have been defined. Each separate procedure in the program contains additional local variables, which are not listed above. The functions of these variables may easily be identified by studying the algorithm in which they are used. The list of variables has been provided to give the interested reader a better understanding of the implementation and functions of the simulation program.

APPENDIX C

RAM CALCULATIONS AND RESULTS

APPENDIX C

C Calculation of simulation results

In this appendix the calculations and definitions used to obtain the results from the simulation program are listed. It is important to analyse the formulas used in order to verify the accuracy of the results obtained from the simulation program and to make valid deductions from the results.

Average hop delay: The average hop delay represents the average time taken for a packet to be transmitted from node i to node k , and includes the output queuing time at node i as well as the input queuing time and service time at node k . The average hop delay is the total hop delay divided by the total number of hops during any time interval.

Average packet delay: The average packet delay for a particular time interval is calculated as the total packet delay for that time interval divided by the total number of packets successfully reaching their destinations during that time interval.

Average percentage use of all links: The average percentage use of all the links in the network is the sum of the percentage use of each link in the network divided by the total number of links in the network.

Messages Transmitted: The total number of messages transmitted during an interval is defined as the number of packets successfully reaching their destinations during that interval.

APPENDIX C

Number of hops: The number of hops experienced by a packet travelling from source node i to destination node j is incremented each time the packet is serviced at a node except the time it is initially serviced at node i . The total number of hops is the sum of the individual hop counts of all the packets in the network. The average number of hops experienced by a packet is then the total number of hops divided by the total number of packets transferred during the time interval.

Packet delay referred to original interval: This represents the total delay of all the data packets, created during the same time interval a , successfully reaching their destination nodes. The average delay is defined as the total packet delay, of the packets created during time interval a , divided by the number of packets received that were generated during the time interval a .

Packets received referred to original interval: This represents total number of data packets created during time interval a , successfully reaching their destination nodes.

Packets transmitted per link: The number of packets transmitted over link (i,k) during a time interval is determined as the number of packets arriving at node k during that time interval and includes both data and routing packets.

Percentage number of routing packets: This is defined as the total number of routing packets transmitted multiplied by 100 divided by the sum of the number of routing and data packets transmitted.

APPENDIX C

Percentage use of link (i,k) : The average percentage use of link (i,k) during a time interval is defined as the number of packets transferred across the link during that time interval multiplied by the number of bits per packet and divided by the capacity of link (i,k) in bits per second.

Routing packet delay referred to original interval: This represents the total delay of all the routing packets, created during time interval a , successfully reaching their destination nodes. The average delay is defined as the total routing packet delay referred to original interval divided by the number of routing packets received referred to the same time interval.

Routing packets received referred to original interval: This represents total number of routing packets created during time interval a , successfully reaching their destination nodes.

Total cost: The total cost during a time interval is the sum of the cost for a packet to be transmitted over a link in the network during that time interval and is calculated each time a packet arrives at a node. The total cost of transferring a packet across the network is then the sum of transferring the packet across the individual links from the source to the destination node. The cost of transferring blocked packets is also calculated and the average cost per packet is then the total cost divided by the number of packets successfully reaching their destinations. It is assumed that blocked packets increase the cost per successfully transmitted packet. This assumption is valid if one is considering the overall long term cost of running the network.

APPENDIX C

Total packet delay: The total packet delay is defined as the time the packet is serviced and consumed at the destination node minus the time the packet was created. The total packet delay represents the delay of all the data packets and excludes packets containing only routing information.

APPENDIX D

PROGRAM LISTING

D.1 Main program listing

APPENDIX D

PROGRAM Network_simulator;
(*****)

```
(*****)
(*
(* Network Routing Simulation Program
(* *****)
(*
(* Name: 'NRSP.PAS'
(*
(* Description: The function of the program is to simulate
(* the routing of packets or messages around a
(* a digital network. The package will be used
(* to determine the performance of the network
(* subject to a particular routing algorithm
(* and various traffic loads.
(*
(* Versions 4.1
(* This version is currently being updated to
(* implement an adaptive routing algorithm,
(* based on R Ballage's adaptive algorithm,
(*
(* Authors: Marc Leon Francois
(*
(* Date: October 1987
(*
(* Inputs: Input File and Keyboard
(*
(* Outputs: Output file and screen
(*
(* -----
(* Modifications:
(* Author: (Name)
(* Date: (Date)
(* Description: (Description of modification(s))
(*
(* *****)

(*(* These symbols indicate writeln statements that are *)
(* used in developing and debugging the program. These *)
(* writeln statements may be deleted if necessary. *)

INCLUDE 'sys/ins/user.ins.pas'; (* System calls for obtaining *)
INCLUDE 'sys/ins/cal.ins.pas'; (* the system time *)
INCLUDE 'sys/ins/rout.ins.pas';
```

APPENDIX D

```

CONST nmax      = 10;    { Maximum number of nodes allowed }
initially      = 1000;
No_of_intervals = 20;    { Time intervals during simulation }
DeltaT         = 0.001;  { Time increments for simulation }
Max_0_length   = 30000;  { Maximum number of events allowed }
Max_intervals  = No_of_intervals + 4;
(Max_trf_pnts  = 10);    { Max No of pnts between updates }

{ Constants required for the second derivative routing algorithm }
(-----)
(Alpha         = 1.0);    { Constant reqd for new algorithm }
(Max_m        = 50);    { The max. No. of pnts/busy period }
(Loc_delay_Tol = 0.030); { Change in Loc delay tolerance }
(-----)

TYPE
Node_ID      = 1..nmax;    { All possible node ID's }
Intervals    = 1..Max_intervals;
Vector       = ARRAY[Node_ID];
Rvector      = ARRAY[Node_ID];
Matrix       = ARRAY[Node_ID,Node_ID];
Matrixd      = ARRAY[Node_ID,Node_ID,1..No_of_intervals];
Matrixs      = ARRAY[Node_ID,Node_ID,1..No_of_intervals];
Isbats_vector = ARRAY[1..Max_intervals];
Rotats_vector = ARRAY[1..Max_intervals];
ID_Sentenc   = ARRAY[1..50];
Path_info    = (Output_path, Not_connected);
MatrixC      = ARRAY[Node_ID,Node_ID];

Networks     = RECORD      { Network definition }
    Network_ID_name      : ID_Sentenc;
    Number_of_nodes      : Node_ID;
    Capacities           : Matrix;
    cost                 : Matrix;
    expected_traffic      : Matrixd;
    by_packet             : INTEGER;
    Sink_tree            : MatrixC;
END;

{ Variables required for the second derivative routing algorithm }
(-----)
Incr_delays_data = RECORD      { Incremental delays data }
    d0_of_k1             : REAL; { Calculated values of d0_of_k1 }
    d1g_d0_gf            : REAL; { for neighbouring nodes. }
    n                     : INTEGER;
    New_busy_period       : BOOLEAN;
    NI_tot                : INTEGER(32);
    Sa_1, Tw_2           : REAL; { max_m not 100 }
    Qn_1                  : ARRAY[1..300] OF REAL;
END;

```

APPENDIX D

```

Bits_per_packet : INTEGER;
Service_rate    : REAL;
Routing_data    : Routing_info;
END;

All_nodes = ADDSET(Node_ID) OF node;

Packet_info = (data, control, data_and_ctrl);
Algorithms = ((flooding, SP_or_IC, New_algo);
              ( Routing algorithm required:-
                < a) Flooding
                < b) Shortest path/least cost
                < c) Adaptive algorithm
              );

General_packet = RECORD
  ( Packet definition )
  Time_created      : REAL;
  Source_address    : Node_ID;
  Destination_address : 0..max; ( 0-> Broadcasting )
  Previous_node     : 0..max; ( 0-> no previous node )
  Previous_service_time : REAL; ( Pr. end of service )
  Hop_count        : INTEGER;
  Control_data      : Packet_info;
  dt_of            : REAL;
  cdt_of           : 0;
  j_cdt_of         : Node_ID;
  k_previous_node   : Node_ID;
  Routing_algorithm : Alg;
END;

Event_kind = (arrival, service, departure);
Link = "Event_info";
Event_info = RECORD
  ( Event definition )
  Next_event : Link;
  Sestinal   : BOOLEAN;
  Time       : REAL;
  Mode_No    : Mode_ID;
  Output_node_No : Mode_ID;
  Name       : Event_kind;
  Packet     : General_packet;
  Cost       : REAL;
  Previous_arrival_time : REAL;
  Previous_depart_time   : REAL;
END;

Error_status = -1..1;

OP_path_defn = RECORD
  ( Output paths definition )
  No_of_paths : Mode_ID;
  List_of_paths : Vector;
  Path_found   : BOOLEAN;
  No_link      : BOOLEAN;
END;

```

APPENDIX D

```

Stats_info = RECORD   ( Stats data )
    Selected_algorithm : Algorithm;
    Total_packets_blocked : tstats_vector;
    Messages_transmitted : tstats_vector;
    Total_gen_traffic : tstats_vector;
    Number_of_hops : tstats_vector;
    Interval_pkts_Rx : tstats_vector;
    Interval_R_pkts_Rx : tstats_vector;
    Percentage_R_pkts_Rx : tstats_vector;
    Total_message_delays : tstats_vector;
    Interval_pkt_delays : tstats_vector;
    Interval_R_pkt_delays : tstats_vector;
    Total_interval_delays : tstats_vector;
    Average_message_delay : tstats_vector;
    Interval_time : tstats_vector;
    Total_cost : tstats_vector;
    Hop_delay : tstats_vector;
    Average_percentage_use : tstats_vector;
    Packets_per_link : MatrixR;
    Percentage_use : MatrixR;
    Total_pkts_generated : INTEGER;
    Total_pkts_tx : INTEGER;
    No_of_pkts_blocked : INTEGER;
    Total_messages : INTEGER;
    Total_pkts_discarded : INTEGER;
    Pkts_lost_due_to_no_link : INTEGER;
    Percentage_pkts_blocked : REAL;
    Max_delay : REAL;
    End_time : REAL;
    Overall_average_delay : REAL;
    Total_service_time : REAL;
    Total_dsnh_time : REAL;
    Total_tx_time : REAL;
    Overall_cost : REAL;
    Delay_per_pkt_gen : REAL;
    Total_hop_delay : REAL;
    Delay_distribution : tstats_vector;
END;

```

```

Filename = ABRNF(1..30) OF CHAN;

```

```

VAR
  N : Node_ID;   ( Number of nodes in network )
  Network : Network; ( Network description )
  Event : Event_Info; ( Event description )
  Front, Back : Link; ( Pointers to event queue )
  Status : Error_Status; ( No error if 0 )
                               ( Error if 1 )
                               ( Warning if -1 )
  Seed : REAL; ( Global seed for random number generator )
  Nodes : All_nodes; ( Status description of all nodes )

```

APPENDIX D

```

Total_duration : REAL;      ( Total duration of the simulation )
Event_q_length : INTEGER32; ( The current length of the event queue )
Data_pkts_in_q : INTEGER32; ( The number of data pkts in event queue )
Stats_data      : Stats_info; ( Performance evaluating variables )
Sim_start_time  : time_t;     ( Used to calculate the actual duration )
Sim_end_time    : time_t;     ( time_tClock_t of the simulation in seconds )
Init_start_time : time_t;     (
Min_of_sec      : INTEGER;    ( See: Apollo Domain: Prging with general )
Max_of_sec      : INTEGER;    ( system function calls. )
Cntr_of_sec     : INTEGER;
Sub_check       : BOOLEAN;
No_of_periods   : Interval;   ( Total No of intervals for entire simul. )
Choice          : CHAR;
OP_filename     : filename;
ALPHA           : REAL;       ( These are usually declared as constants )
MAX_n           : INTEGER;
INC_DELAY_TOL   : REAL;
MAX_TPS_PETS    : REAL;

```

```

FUNCTION rand(REAL);
(=====)

```

```

(#####)
(=
(= This function produces a 'random' number R, where
(= 0 < R < 1.0 and R is uniformly distributed.
(=
(= INPUTS: Variable: Seed (Global variable)
(= OUTPUTS: Variable: Rand (Pseudo random number)
(=
(#####)

```

```

BEGIN
  seed := 125.0 * Isced + 1.01;
  seed := seed - 0.192.0 * TRUNC(seed/0.192);
  rand := Isced + 0.51/0.192;
END;

```

```

PROCEDURE INC (VAR I: INTEGER);
(=====)
(#####)
(=
(= This procedure is simply used to increment an integer
(= variable.
(= INPUTS: Variable: Integer (I)
(= OUTPUTS: Variable: Integer (I + 1)
(=
(#####)

```

```

BEGIN
  I := I + 1;
END;

```

APPENDIX D

```

PROCEDURE DEC (VAR I : INTEGER);
(*****)

(*****)
(=)
(= This procedure is simply used to decrement an integer =)
(= variable. =)
(= INPUTS: Variables: Integer (x) =)
(= OUTPUTS: Variables: Integer (x - 1) =)
(=)
(*****)

BEGIN
  I := I - 1;
END;

( Include the procedure required to initialise the special routing data )
(*****)

$INCLUDE 'initaspt.pas';

\ Include procedures required to set up the blocked path set )
(*****)

$INCLUDE 'block.pas';

PROCEDURE initialise_gsla (VAR Network_defn : Networks;
                          VAR Node_defn : All_nodes;
                          VAR Stats_defn : Stats_info);
(*****)

(*****)
(=)
(= This procedure initialises all the data. =)
(=)
(= INPUTS: Variables: Expected traffic: =)
(= Node status: Routing data and =)
(= queue lengths. =)
(= Statistics data: =)
(= OUTPUTS: Variables: Expected traffic =)
(= Node status: Routing data and =)
(= queue lengths. =)
(= Statistics data: =)
(=)
(*****)

VAR x,y,z : INTEGER;

```

APPENDIX D

BEGIN

```

( Program heading )
(-----)
writeln; writeln;
writeln('NETWORK ROUTING SIMULATION PROGRAM');
writeln('-----'); writeln;
writeln('      By M. L. Francois'); writeln; writeln;
writeln('Initialising data'); writeln('-----');

( Initialise expected traffic for all source destination pairs )
(-----)
FOR x := 1 TO maxn DO
  FOR y := 1 TO maxn DO
    FOR z := 1 TO No_of_intervals DO
      BEGIN
        Network_defn.expected_traffic(x,y,z) := 0;
        Stats_defn.Packets_per_link(x,y,z) := 0;
        Stats_defn.Percentage_used(x,y,z) := 0;
      END;
    END;

( Initialise current q lengths/OP prob/Blocked paths for all nodes )
(-----)
FOR x := 1 TO maxn DO
  BEGIN
    Mode_defn(x).Single_input_q_length := 0;
    FOR y := 1 TO maxn DO
      WITH Mode_defn(x) DO
        BEGIN
          Input_q_length(y) := 0;
          Output_q_length(y) := 0;
          Routing_data.Traffic_data(y).No_of_pkts
            Routing_data.Traffic_data(y).Total_traffic := 0;
          Routing_data.Traffic_data(y).Update_time := 0;
          Routing_data.Traffic_data(y).Tn_1 := 0;
          Routing_data.Traffic_data(y).Tn_2 := 0;
          Routing_data.Adpt_data.B(x,y) := C1;
          Routing_data.Adpt_data.B_data_Ref(y) := C3;
        END;
      END;
    END;
  END;
  BEGIN
    Output_probabilities(y,z) := 0;
    Phi(y,z) := 0;
    Adpt_data.ad_data(y,z).d0_dr
      END;
    END;
  END;
  writeln;

```

APPENDIX D

```
( Initialise all output variables to determine performance of the network )
```

```
{-----}
```

```
WITH stats_defn DO
BEGIN
  FOR x := 1 TO Max_intervals DO
  BEGIN
    Interval_time(x) := 0;
    Total_pkt_traffic(x) := 0;
    Total_packets_blocked(x) := 0;
    Messages_transmitted(x) := 0;
    Total_message_delay(x) := 0;
    Average_message_delay(x) := 0;
    Total_cost(x) := 0;
    Total_gen_traffic(x) := 0;
    Number_of_hops(x) := 0;
    Hop_delay(x) := 0;
    Interval_pkts_Rx(x) := 0;
    Interval_R_pkts_Rx(x) := 0;
    Interval_pkt_delay(x) := 0;
    Interval_R_pkt_delay(x) := 0;
    Percentage_R_pkts_Rx(x) := 0;
    Total_interval_delay(x) := 0;
    Average_percentage_util(x) := 0;
    Delay_distribution(x) := 0;
  END;

  Max_delay := 0; Total_tx_time := 0;
  Total_service_time := 0; Total_pkt_block := 0;
  Total_pkts_tx := 0; No_of_pkts_blocked := 0;
  Overall_cost := 0; Total_messages := 0;
  Total_pkts_discarded := 0; Overall_message_delay := 0;
  Total_hop_delay := 0; Pkts_lost_due_to_no_link := 0;
END;
```

```
PROCEDURE Get_data_from_file (VMH N:Node_ID; VAR Network_defn:Networks;
  VAR Node_defn :All_nodes;
  VAR Status :Error_status;
  {-----})
```

```
(-----)
(*
(* This procedure reads in the specifications of the *)
(* From an input file. The format of the input file is *)
(* specified in the user's manual of the program. *)
(*
(* INPUTS: Keyboard: filename *)
(* OUTPUTS: Variables: Number of nodes, Link capacities *)
(* Link costs, Expected traffic, *)
(* Node Service rates, buffer pool *)
(* size and output buffer size. *)
(-----)
```

APPENDIX D

```

VAR filename      : ARRAY(1..30) OF CHAR;
F                 : TEXT;
StatusA          : INTEGER32;
Finished         : BOOLEAN;
x,y,Temp_source,Temp_dest : INTEGER;

BEGIN
  writeln('Data input from file'); writeln;
  write('Enter name of input file (and path): (Default "INPUT.DAT") ');
  readln(Filename);
  IF (Filename = '') THEN filename := 'INPUT.DAT'; { Default file }
  writeln;
  OPEN(F,filename,'OLD',statusA);
  IF (statusA = 0)
  THEN reset(F)
  ELSE
    BEGIN
      writeln('ERROR!! File not found'); Status := 1; RETURN;
    END;

  { Read in Network ID }
  (-----)
  readln(F,Network_defn.Network_ID_name); readln(F);
  writeln(Network_defn.Network_ID_name); writeln;

  { Read in number of nodes }
  (-----)
  readln(F); readln(F,N); readln(F);
  writeln('Number of nodes in network = ',N); writeln;
  Network_defn.Number_of_nodes := N;

  { Read in bits per packet }
  (-----)
  readln(F); readln(F,Network_defn.bp_packet); readln(F);
  writeln('Bits per packet = ',Network_defn.bp_packet);
  writeln;

  { Read in Link capacities }
  (-----)
  readln(F); writeln('Link capacities');
  FOR y := 1 TO N DO
    BEGIN
      FOR x := 1 TO N DO
        BEGIN
          read(F,Network_defn.capacities(x,y));
          write(Network_defn.capacities(x,y):12:4);
        END;
        readln(F); writeln;
      END;
    END;
  readln(F); writeln;

```

APPENDIX D

```

( Read in Link costs )
(-----)
readln(f); writeln('Link costs');
FOR y := 1 TO N DO
  BEGIN
    FOR x := 1 TO N DO
      BEGIN
        read(f, Network_defn.cost(x,y));
        IF (x = y) THEN network_defn.cost(x,y) := infinity;
        write(Network_defn.cost(x,y):2);
      END;
    readln(f); writeln;
  END;
readln(f); writeln;

( Read in expected traffic for source destination pairs )
(-----)
(finished := FALSE; readln(f); writeln('Expected traffic');
REPEAT
  read(f, Temp_source);
  IF (Temp_source = 0)
  THEN finished := TRUE
  ELSE
    BEGIN
      read(f, Temp_dest); write(Temp_source:4, Temp_dest:4);
      FOR z := 1 TO No_of_intervals DO
        BEGIN
          read(f, Network_defn.expected_traffic(Temp_source, Temp_dest, x));
          write(Network_defn.expected_traffic(Temp_source, Temp_dest, x):8);
        END;
      END;
      readln(f); writeln;
    UNTIL finished;

( Read in service rate and buffer size for each node )
(-----)
readln(f); readln(f);
FOR x := 1 TO N DO
  BEGIN
    read(f, y); IF (y < 0) THEN N := 0; < Error in input file >
    read(f, Node_defn(x).service_rate);
    read(f, Node_defn(x).Total_buffer_pool_length);
    readln(f, Node_defn(x).Max_output_Q_length);
    write('Node ', x:2, ' Service rate = ', Node_defn(x).service_rate:2);
    write(' I/P Buffer = ', Node_defn(x).Total_buffer_pool_length:5);
    write(' O/P Buffer = ', Node_defn(x).Max_output_Q_length:5);
    Node_defn(x).Current_buffer_pool_length
      := Node_defn(x).Total_buffer_pool_length;
  END;

```

APPENDIX 3

```

( Check for errors in the input file )
(-----)
IF (N = 0)
THEN
  BEGIN
    writeLn; writeLn('ERROR') Incorrect input file (format!); writeLn;
    status := 1; RETURN;
  END
ELSE status := 0;

CLOSE(F);
END;

```

```

PROCEDURE Get_data_from_user (Var Duration : REAL; VAR Stats : Stats_info;
                              VAR Result_file : filename);
(-----)

```

```

(=====)
(*
(* This procedure simply collects the required data from *)
(* the user. *)
(* *)
(* INPUTS: Keyboard *)
(* OUTPUTS: Variables: Simulation duration *)
(* *)
(=====)

```

```

VAR ans : INTEGER;

```

```

BEGIN
  writeLn Duration := infinity + 1;
  writeLn('Time increments currently set at ',Dr:cats(2:8));

  ( Enter duration of the experiment )
  (-----)
  WHILE (Duration > infinity) DO
    BEGIN
      write('Enter duration of simulation (0 to 1000): ');
      readLn(Duration);
    END;
  writeLn;

```

```

( Select routing algorithm )
(-----)
writeLn('Select routing algorithm...');
writeLn(' 1. Shortest path/least cost');
writeLn(' 2. Flooding ');
writeLn(' 3. Adaptive algorithm');
writeLn;

```

APPENDIX D

```

REPEAT
  write('Enter choice : '); readln(ans);
UNTIL (ans IN ('1','3'));

CASE ans OF
  1: Stats.Selected_algorithm := 'SP_or_LC';
  2: Stats.Selected_algorithm := 'Flooding';
  3: Stats.Selected_algorithm := 'New_algo';
END;

( Enter seed for random number generator )
(-----)
write('Enter a seed for random number generators : '); readln(seed);
write('ok');

( This is only inserted for testing purposes take this out later )
(-----)
IF (Stats.Selected_algorithm = 'New_algo')
THEN
  BEGIN
    write('Enter alpha: '); readln(alpha);
    write('Enter maximum number of messages between updates: '); readln(Max_M);
    write('Enter incremental delay tolerances: '); readln(Inc_Delay_Tol);
    write('Enter number of packets between traffic updates: '); readln(Max_Trf_Pkts);
  END;

( Get the output file name from the user )
(-----)
write('Enter name of output file (and path): (Default "RESULT.DAT") ');
readln(Result_file);
IF (Result_file = '') THEN Result_file := 'RESULT.DAT'; ( Default file )
writeln;

END;

PROCEDURE Shortest_path (Link_costs : Matrix; N,s,t : Node_ID;
                        VAR path : Node_ID);
(-----)
(*****)
(*)
(*) This procedure calculates the shortest path from any (*)
(*) source node (s, any destination node). Note that only (*)
(*) the next node on the route to a particular destination (*)
(*) is required, and hence the ID number of the next node (*)
(*) is the only output to the procedure. (*)
(*) INPUTS: Matrix: A matrix (NxN) consisting of the (*)
(*) cost of each link in the network. (*)
(*) Variables: The number of nodes in the network (*)
(*) The source node (s) (*)
(*) The destination node (t) (*)
(*) OUTPUTS: Variables: Path, the next node along the (*)
(*) the least cost path. (*)
(*****)

```

APPENDIX D

```

TYPE Lab      = (para, tent);
Node_Label = RECORD
    predecessor : Node_ID;
    length      : REAL;
    labl       : Lab;
END;
Graph_state = ARRAY[1..maxn] OF Node_Label;

VAR state : graph_state;
    i, k : Node_ID;
    min : REAL;

BEGIN
    FOR i := 1 TO N DO
        WITH state[i] DO
            BEGIN
                predecessor := 0; length := infinity; labl := tent;
            END;
            state[i].length := 0; state[i].labl := para; k := i;

        REPEAT
            FOR j := 1 TO N DO
                IF (Link_costs[i, j] < 0) AND (state[j].labl = tent)
                THEN
                    IF (state[i].length + Link_costs[i, j] < state[j].length)
                    THEN
                        BEGIN
                            state[i].predecessor := j;
                            state[i].length := state[j].length + Link_costs[i, j];
                        END;
                    END;

                    min := infinity; k := 0;
                    FOR l := 1 TO N DO
                        IF (state[l].labl = tent) AND (state[l].length < min)
                        THEN
                            BEGIN
                                min := state[l].length;
                                k := l;
                            END;
                        END;

                    IF k = 0 THEN no path found;

                    IF (k < 0) THEN state[k].labl := para ELSE k := 0;

                UNTIL (k = 0);

                k := 0;
                IF (k < 0) THEN path := state[i].predecessor ELSE path := 0;

                writeln('Shortest path from node ', i, ' to node ', k, ' = ', path);
            END;
        END;
    END;

```

APPENDIX D

```

PROCEDURE Initialise_node_data (N : Mode_ID) Network_defn : Networks;
                                VAR Node_data : All_nodes;
                                (*****);

    (*****);
    (
    *)
    *) Each node is initialised with the current status of *)
    *) of the network, and the shortest path to all nodes, *)
    *) from all nodes is calculated. *)
    *)
    *) INPUTS: Variables: The number of nodes in the network *)
    *) The status of the network *)
    *) OUTPUTS: Variables: The status of nodes in the network *)
    *)
    (*****);

    VAR count,x,y,OP_path_No,Best_node_No : Mode_ID;

BEGIN
  FOR count := 1 TO N DO
    BEGIN
      writeln(writeln('Initialising node ',count));
      Node_defn.counts.kits_per_packet := Network_defn.kits_per_packet;
      Node_defn.counts.routing_data.Number_of_nodes := N;
      Init_routing_data(N,Node_defn.counts.routing_data,Node_defn);

      ( Initialise link capacities and costs at each node )
      (-----)
      FOR x := 1 TO N DO
        FOR y := 1 TO N DO
          BEGIN
            Node_defn.counts.routing_data.Link_capacities(x,y)
              := Network_defn.capacities(x,y);
            Node_defn.counts.routing_data.Link_costs(x,y)
              := Network_defn.costs(x,y);
          END;
        END;
      END;

      ( Initialise the routing data of each node using least cost algorithm )
      (-----)
      FOR Best_node_No := 1 TO N DO
        BEGIN
          IF (count < Best_node_No)
          THEN
            BEGIN
              Shortest_path(Node_defn.counts.routing_data.Link_costs,N,
                count,Best_node_No,OP_path_No);
              Node_defn.counts.routing_data.shortest_path_node(Best_node_No)
                := OP_path_No;
            END;
          END;
        END;
      END;
    END;
  END;

```

APPENDIX D

```

VAR Temp_store : Event_info;

BEGIN
  IF (Insert = FALSE) =
  THEN writeln('ERROR!!') Event cannot be inserted into the event queue.'
  ELSE
    IF (Q_length >= Max_Q_length)
    THEN writeln('ERROR!!') Event queue full'
    ELSE
      BEGIN
        New_arrival^.next_event := Temp^.next_event;
        Temp_store := Temp;
        Temp := New_arrival;
        New_arrival := Temp_store;
        Temp^.next_event := New_arrival;
        Q_length := Q_length + 1;
      END;
      [insert := false;
    END;

PROCEDURE List_q (Front : Link; Q_length : INTEGER);
{=====}

{
  (=====)
  (x)
  (x) This procedure prints out all events in the event
  (x) list and can be used for debugging purposes.
  (x)
  (x) INPUTS: Pointer: Pointer to the front of Queue
  (x) OUTPUTS: Screens: Events in queue in chronological
  (x) order.
  (x)
  (=====)
}

VAR Temp : LINK;
    Count : INTEGER;

BEGIN
  Count := 1; Temp := Front; writeln;
  writeln('Displaying queue in chronological order');
  writeln('-----');
  writeln('Length of event queue = ', Q_length);
  WHILE (Temp^.sentinel <> TRUE) DO
    BEGIN
      write(count); write(' '); Temp^.Time; write(' '); Temp^.Name; write(' '); Temp^.Node_no;
      writeln(' '); Temp^.Percent_Control_data;
      count := count + 1; Temp := Temp^.next_event;
    END;
    writeln;
    writeln('All events displayed. ');
  END;

```

APPENDIX D

```

PROCEDURE Update_an_event (duration,time,E_time      : REAL;
                           Node_Number,Source_No,
                           Dest_No,No_of_Previous_node : Node_ID;
                           Hop_No                      : INTEGER;
                           Ctrl_data                  : packet_info;
                           Arr_service_dept            : Event_kind;
                           Cost_of_Is                  : REAL;
                           First_dept_time             : REAL;
                           Required_algorithm           : Algorithm;
                           VAR New_event_data           : Event_info);

```

```

(=====)
(*
*)
*) This procedure sets up an event which is to be added
*) to the event list.
*)
*)
*) INPUTS: Variables: Time of the event
*) Node number
*) Source node number
*) Destination node number
*) Previous node number (0 if none)
*) Hop count
*) Ctrl data (data of ctrl packet)
*) Event kind (arrival,service,dept)
*)
*) OUTPUTS: Variables: Event
*)
*)
*) (=====)

```

```

BEGIN
  WITH New_event_data DO
    BEGIN (Enter event data )
      Sequential := FALSE; Time := E_time;
      Name := Arr_service_dept; Node_No := Node_number;
      Cost := Cost_of_Is;
    WITH packet DO
      BEGIN (Enter packet data )
        Source_address := Source_No;
        Destination_address := Dest_No;
        Previous_node := No_of_previous_nodes;
        Hop_count := Hop_No;
        Control_data := Ctrl_data;
        Previous_dept_time := First_dept_time;
        Routing_algorithm := Required_algorithm;
      END;
    END;
  END;
END;

```

APPENDIX D

```

PROCEDURE Enter_events (N : Mode_ID; Duration : REAL;
VAR Q_front, Q_back : Link;
VAR Length_of_Q : INTEGER32;
VAR Q_ptr_in_Q : INTEGER32;
VAR Stats : Stats_info;
[*****])

(*****
*)
*) This procedure allows the user to enter one or more *)
*) events manually into the event queue. *)
*)
*) INPUTS: Constants: Delta T *)
*)          Number of Intervals (10) *)
*)          Variables: Number of nodes *)
*)                    Duration of the simulation *)
*) OUTPUTS: Pointers: Pointers to the front and back *)
*)              of the event queue. *)
*)          Variables: The length of the event queue. *)
(*****)

VAR Source_node, Destination_node : Mode_ID;
Current_clock : REAL;
Insert : BOOLEAN;
New_event_data : Event_info;
New_event, Temp_marker : Link;
Int_No : INTEGER;
Answer : CHAR;

BEGIN
  writeln;
  writeln('Event entry process');
  writeln('-----');

  REPEAT
    ( Get time of event from the user )
    (-----)
    REPEAT
      writeln; write('Enter time of event: '); readln(Current_clock);
      IF (Current_clock > Duration)
      THEN writeln('Time entered must be within the duration');
      UNTIL (Current_clock <= Duration);

    ( Set source and destination node from the user )
    (-----)
    REPEAT
      writeln;
      write('Enter the source node ID: '); readln(Source_node);
      write('Enter the destination node ID: '); readln(Destination_node);
      IF ((Source_node > N) OR (Destination_node > N))
      THEN writeln('Nodes entered do not exist');
      UNTIL ((Source_node <= N) AND (Destination_node <= N));
  
```

APPENDIX D

```

( Insert event into the event queue )
(-----)
Int_No := TRUNC((Current_clock * No_of_intervals)/Duration) + 1;
INC(Stats.Total_gen_traffic[Int_No]);

Update_an_event(Duration,Current_clock,Source_node,Source_node,
Destination_Node,0,0,0,0,0,0,Current_clock,
Stats.Selected_algorithm,New_event_data);
New_event_data.Packets_Time_created := Current_clock;
Find_insert_pos(Current_clock,0,front,0,back,Temp_marker,Insert);
NEW(New_event);
New_event := New_event_data;
Insert_event(Temp_marker,New_event,length_of_Q,0_pks_in_Q,Insert);

writein; write('Enter another event (y/n) ? '); readln(Answer);

UNTIL (NOT(Answer ENE('Y','y')));

Stats.Total_pks_generated := Length_of_Q;

END;

( Include procedures required for Second derivative routing algorithm )
(=====)

$INCLUDE 'Transmit.pas';
$INCLUDE 'adapt.pas';

PROCEDURE Random_arrivals (N : Mode_ID; Duration : REAL;
Traffic_expected : NatrxnA;
VAR Q_front, Q_back : Lnat;
VAR Length_of_Q : INTEGER32;
VAR Q_pks_in_Q : INTEGER32;
VAR Stats : Stats_info);
(=====)

(=====)
(
( This procedure generates random arrivals to nodes )
( for each source destination pair, with the average )
( packets rate given by the expected traffic. )
(
(
( INPUTS: Constants: Delta T )
(          Number of intervals (IO) )
( Variables: Number of nodes )
(          Duration of the simulation )
(          Expected traffic )
( OUTPUTS: Pointers: Pointers to the front and back )
(          of the event queue. )
(          Variable: The length of the event queue. )
(
(=====)

```

APPENDIX D

```

VAR Source_node, Destination_node, X, Y : INTEGER;
    Current_clock, Probability, Random_No : REAL;
    msg_no : INTEGER;
    Insert : BOOLEAN;
    New_event_data : Event_info;
    New_event, Trap_marker : Link;

BEGIN
    writelng; writelng; writeln('Generating random arrivals');
    writeln('-----');
    FOR Source_node := 1 TO N DO
        BEGIN
            (sz) writeln(Source_node);
            FOR Destination_node := 1 TO N DO
                IF Destination_node <> Source_node
                THEN
                    BEGIN
                        BEGIN
                            Current_clock := 0.0;

                            REPEAT
                                X := TRUNC(((Current_clock * No_of_intervals)/Duration) + 1);
                                Probability :=
                                    Traffic_expected(source_node,destination_node,X)/Delta;
                                Random_No := Rand;

                                IF Random_No < Probability
                                THEN
                                    BEGIN
                                        INQ(Stats.Total_gen_traffic(Y));
                                        Q_pkts_in_Q := Q_pkts_in_Q + 1;
                                        Update_an_event(Duration,Current_clock,Source_node,Source_node,
                                            Destination_node,0,0,data,arrival,0,Current_clock,
                                            Stats.Selected_algorithm,New_event_data);
                                        New_event_data.Packet_Type_created := Current_clock;
                                        Find_insert_pos(Current_clock,0,front,Q_back,Trap_marker,insert);
                                        NEW(New_event);
                                        New_event := New_event_data;
                                        Insert_event(Trap_marker,New_event,Length_of_Q,Q_pkts_in_Q,insert);
                                    END;
                                    Current_clock := Current_clock + Delta;
                                UNTIL (Current_clock >= Duration);

                                ( Update total expected traffic for the time interval )
                                (-----)
                                FOR Y := 1 TO No_of_intervals DO Stats.Total_exp_traffic(Y) :=
                                    Stats.Total_exp_traffic(Y) +
                                    Traffic_expected(Source_node,Destination_node,Y);
                            END;
                            writelng;
                            Stats.Total_pkts_generated := Length_of_Q;
                        END;
                    END;
                END;
            END;
        END;
    END;

```

APPENDIX D

```

PROCEDURE Arrival_process (int_no : Intervals;
                           VAR Event_defn : Event_info;
                           VAR Q_front,Q_back : Link;
                           VAR Event_R_length : INTEGER32;
                           VAR Q_pkts_in_Q : INTEGER32;
                           VAR Node_status : Node;
                           VAR Stats_defn : Stats_info);
(=====)

(=====)
(1)
(2) This procedure simulates the arrival of a packet at a
(3) node.
(4) In this version the packet is discarded if the input
(5) queue is full, otherwise the service time is
(6) calculated.
(7)
(8)
(9) INPUTS: Event description
(10) Status of node at which arrival occurs
(11) Pointers: Pointers to the front and back of
(12) the event queue.
(13) Variables: The length of the event queue.
(14) Performance variables
(15) OUTPUTS: Updated event description
(16) Updated node status
(17) Pointers: To front and back of the event Q.
(18) Variables: The length of the event queue.
(19) Performance variables
(=====)

VAR Current_node_ID : Node_ID;
    Time_of_service : REAL;
    New_event_ptr, Temp_ptr : Link;
    Insert, Piggy_back_used : BOOLEAN;
    Finished : BOOLEAN;
    No_in_queue : INTEGER;
    Temporary_ptr : LINK;

BEGIN
    Current_node_ID := Event_defn.Node_No;
    Event_defn.Previous_arrival_time := Event_defn.Time;
    Piggy_back_used := FALSE;
    Finished := FALSE;

    IF (int_no <= No_of_intervals)
    THEN INC(Stats_defn.Packets_per_Link(Event_defn.Packet.Previous_node,Event_defn.Node_no,int_No));

```

APPENDIX D

```

( Check if packet has not been generated within the node )
(-----)
IF (Event_defn.Packet.Previous_node < 0)
THEN
BEGIN
    Event_defn.cost := Event_defn.cost
    + Node_status.Routing_data;
    Link_costs(Event_defn.Packet.Previous_node, Event_defn.Node_No);
    IF (Event_defn.Packet.Control_data = Data) THEN INC(Stats_defn.Total_pkts_Tx);
END;

( Is packet arriving a pure routing packet ? and input queue > 0 )
(-----)
IF (Event_defn.Packet.Control_data = Control)
THEN
    IF (Node_status.Total_buffer_pool_length > Node_status.Current_buffer_pool_length)
    THEN
        BEGIN
            Search_input_0(Current_node_ID, Event_defn.Packet.Previous_node,
                0, front, temporary_ptr, Piggy_back_used);
            IF (Piggy_back_used = TRUE)
            THEN
                BEGIN
                    Finished := TRUE;
                    Update_data_ptr(Temporary_ptr, Event_defn.Packet.R_previous_node,
                        Event_defn.Packet.dl_of, Event_defn.Packet.dl_of_dr,
                        Event_defn.Packet.l_dlc_dr);
                END;
            END;
        END;

( If routing packet is not piggy backed )
(-----)
IF (Finished = FALSE)
THEN
    BEGIN
        ( Check if input queue is not full )
        (-----)
        IF (Node_status.Current_buffer_pool_length = 0)
        THEN
            BEGIN
                IF (Event_defn.Packet.Control_data = Data)
                THEN
                    BEGIN
                        B_pkts_in_0 := B_pkts_in_0 - 1;
                        Stats_defn.Total_packets_blocked[Int_no] :=
                            Stats_defn.Total_packets_blocked[Int_no] + 1;
                        Stats_defn.Total_cost[Int_no] := Stats_defn.Total_cost[Int_no]
                            + Event_defn.Cost;
                    END;
                END;
            END;
        END;
    END;

```

APPENDIX D

```

ELSE
  BEGIN
    No_in_queue := (Node_status.Total_buffer_pool_length -
      Node_status.Current_buffer_pool_length);

    { Calculate the time of service }
    {-----}
    { Time of service is the time the packet LEAVES the server... }
    { Time of service = Current time + queuing delay + a small delay }
    { + Time taken to be served. }
    { The time taken to be served is currently set to be a constant }
    { but may be modified later to include varying server times. }
    {-----}
    IF (Node_status.service_rate = 0) THEN Time_of_service := Infinity
    ELSE
      BEGIN
        Time_of_service := Event_defn.Time +
          (No_in_queue/Node_status.service_rate) +
          (1 + DeltaT) * (1/Node_status.service_rate);
        IF (Event_defn.Packet.Control_data = Data)
          THEN Stats_defn.Total_service_time := Stats_defn.Total_service_time
            + (Time_of_service - Event_defn.Time);
      END;

    { Increment input Q and decrement buffer pool }
    {-----}
    IF (Event_defn.Packet.previous_pnode = 0)
      THEN INC(Node_status.Q_inq_in_pnode_length);
    ELSE INC(Node_status.Input_Q_length(Event_defn.Packet.Previous_pnode));

    DEC(Node_status.Current_buffer_pool_length);
    INC(No_in_queue);

    { Update next event }
    {-----}
    Event_defn.Time := Time_of_service;
    Event_defn.Name := Service;

    { Insert new event into event queue }
    {-----}
    NEW(New_event_ptr);
    New_event_ptr := Event_defn;
    Find_insert_pos((Time_of_service, 0, front_Q_back, Temp_ptr, Insert));
    Insert_event(Temp_ptr, New_event_ptr, Event_Q_length, 0, pnode_in_Q, Insert);
  END;
END;

```

APPENDIX D

```

PROCEDURE Adaptive_algo (No_of_nodes : Node_Id; Event_defn : Event_Info)
    Node_Status : NODE;    VAR OP_Path : OP_Path_Defn;
    (*****)
```

```

    (*****)
    (**)
    (**) The procedure determines the output link a packet is to
    (**) be sent on by using an adaptive routing algo.
    (**) algorithm.
    (**)
    (**)
    (**) INPUTS: Variables: Event; Defines packet, dest No etc.
    (**) Node_Status: Defines routing path
    (**) for the particular node.
    (**) OUTPUTS: Variables: OP_Path; Defines the output link
    (**) The packet is to be sent on.
    (**)
    (*****)
```

```

VAR Random_No : REAL;
    Phi_Total : REAL;
    Finished : BOOLEAN;
    Dest_Node_No : Node_Id;
    Temp_Path_No : Node_Id;
    Old_Path_No : Node_Id;
    OP_Prob : REAL;
```

```

BEGIN
    (sx) WriteLn(' Adaptive algorithm');
    Temp_Path_No := 0; Random_No := RAND;
    Phi_Total := 0; Finished := FALSE;
    Dest_Node_No := Event_defn.Packet.Destination_address;
    OP_Path_No_Link := False;

    ( Select appropriate output path )
    (-----)
    REPEAT
        Temp_Path_No := Temp_Path_No + 1;
        OP_Prob := Node_Status.Routing_data.
            Prob(Temp_Path_No, Dest_Node_No);
        IF OP_Prob < 0.1
        THEN
            BEGIN
                Phi_Total := Phi_Total + OP_Prob;
                IF (Random_No < Phi_Total) THEN Finished := TRUE;
            END;
        UNTIL ((Finished) OR (Temp_Path_No = No_of_nodes));
```

APPENDIX D

```

{ Check if packet is not sent back on the same link }
(-----)
IF ((Finished = TRUE) AND (Temp_path_No = Event_defn.Packet.Previous_node))
THEN
  BEGIN
    Finished := FALSE; Old_path_No := Temp_path_No; Temp_path_No := 0;

    REPEAT
      Temp_path_No := Temp_path_No + 1;
      IF ((Temp_path_No < Old_path_No)
        AND (NOT(Temp_path_No IN (Node_status.Routing_data.Adpt_data.Bit(Dest_Node))))))
      THEN finished := TRUE;
    UNTIL ((finished) OR (Temp_path_No = No_of_nodes));
  END;

{ check if a path was found }
(-----)
IF ((finished = FALSE)
THEN
  BEGIN
    writeLn('Packet discarded at node ', Event_defn.Node_No, ' destination ', Dest_node_No);
    OP_path.Path_found := FALSE;
    OP_path.No_Link := TRUE;
  END
ELSE
  BEGIN
    OP_path.No_of_paths := 1;
    OP_path.List_of_paths[] := Temp_path_No;
    OP_path.Path_found := TRUE;
  END;
END;

END;

PROCEDURE Short_path (Event_defn : Event_Info; Node_status : NORE;
                     VAR OP_path : OP_path_defn);
(=====)

((=====))
(( ))
(( The procedure determines the output link a packet is ))
(( to be sent on by using the shortest path or least ))
(( routing algorithm. ))
(( ))
(( INPUTS: Variables: Event: Defines packet, dest No etc. ))
(( Node status: Defines routing path ))
(( for the particular node. ))
(( OUTPUTS: Variables: OP_path defines the output link ))
(( The packet is to be sent on. ))
(( ))
((=====))

```

APPENDIX D

```

VAR Finished      : BOOLEAN;
    Dest_node_no  : NODE_ID;
    OP_prob       : REAL;
    Temp_path_No  : INTEGER;

BEGIN
  writeln(' Least cost algorithm');
  Temp_path_No := 0; Finished := FALSE;
  Dest_node_No := Event_defn.Packet.Destination_address;
  OP_path_No_Link := False;

  { Select appropriate output path }
  {-----}
  REPEAT
    Temp_path_No := Temp_path_No + 1;
    OP_prob := Node_status.Routing_data.
      Output_probabil::link(Temp_path_No, Dest_node_No);
    IF (OP_prob = 1) THEN Finished := TRUE;
  UNTIL ((Finished) OR (Temp_Path_No = max));

  { check if a path was found }
  {-----}
  IF ((Finished = FALSE)
  THEN
    BEGIN
      OP_path_Path_found := FALSE;
      OP_path_No_Link := TRUE;
    END
  ELSE
    BEGIN
      OP_path_No_of_paths := 1;
      OP_path_List_of_paths := Temp_path_No;
      OP_path_Path_found := TRUE;
    END;
  END;

END;

PROCEDURE Flood_pkt (Event_defn : Event_info; Node_status : NODE;
  VAR OP_path : OP_path_defn);
  {-----}

  {-----}
  { This routing algorithm selects the output paths }
  { according to a flooding routing algorithm. }
  { }
  { }
  { NOTE: Currently this algorithm does not support }
  { broadcasting as this would result in numerous }
  { looping packets. A protocol must be first be }
  { implemented to remove looping packets. }
  { }

```

APPENDIX D

```

Inc_data      = ARRAY(Node_ID) OF Inc_delays_data;
Link_net      = SET OF Node_ID;
All_link_sets = ARRAY(Node_ID) OF Link_set;

Dg_dr_info    = RECORD
    Dg_dr      : REAL;
    Tag        : BOOLEAN;
END;

Adpt_info     = RECORD
    Inc_data    : Inc_data;
    cS_of_ik    : ARRAY(Node_ID) OF REAL;
    cS_dr_data  : ARRAY(Node_ID, Node_ID) OF Dg_dr_info;
    BI          : All_link_sets;
    S_data_Rx   : All_link_sets;
    cS_ofs_Rx   : Link_set;
    All_cS_ofs_received : BOOLEAN;
    delta       : Matrix; ( Via path, Best node )
    OK_to_proceed : ARRAY(Node_ID) OF BOOLEAN;
END;

-----)

Traffic_status = RECORD
    No_of_pkts : INTEGER;
    Update_time : REAL;
    Total_traffic : REAL;
    In_1        : REAL;
    To_2        : REAL;
END;

Traffic_Info = ARRAY(Node_ID) OF Traffic_status;

Routing_Info = RECORD ( Routing variables )
    Number_of_nodes : Node_ID;
    Link_capacity    : Matrix; ( Links C1j i --> j )
    Link_costs       : Matrix; ( Links C1j i --> j )
    Shortest_path_node : Vector;
    Output_probabilities : Matrix; ( OP_path No., Best. )
    Pts              : Matrix; ( OP_path No., Best. )
    Update_required   : BOOLEAN;
    Traffic_data      : Traffic_info;
    Adpt_data         : Adpt_info;
END;

Node = RECORD ( Node definition )
    Total_buffer_pool_length,
    Current_buffer_pool_length : INTEGER;
    Input_W_length, ( Each input link has it's own queue )
    Output_W_length : Vector;
    Max_output_W_length : INTEGER;
    Single_input_W_length : INTEGER; ( Input W length for )
                                     ( traffic generated )
                                     ( within the node. )

```


240

APPENDIX D

```

( Calculate the new input traffic for the destination )
(-----)
Calc_traffic(Event_defn.Time,Node_status.Dlts_per_packet,
             Node_status.Routing_data.Traffic_data(Event_defn.Packet.Destination_address));

( Check if packet has not been created in this node then calculate hop delay )
(-----)
IF ((Event_defn.Packet.Previous_node < 0) AND (Event_defn.Packet.Control_data = Data))
THEN
  BEGIN
    State_defn.Hop_delay(Int_No) := State_defn.Hop_delay(Int_no)
                                + (Event_defn.Time
                                  - Event_defn.Previous_dept_time)
    INC(State_defn.Number_of_hops(Int_No));
  END;

( If packet has not been created at this node then update incremental delay (tl) )
(-----)
IF ((Event_defn.Packet.Previous_node < 0) AND (State_defn.Selected_algorithm = New_algo)
AND (Event_defn.Packet.Control_data <> Control))
THEN
  BEGIN
    Link_dept_time := Event_defn.Time - ((Node_status.Service_rate)/
    Calc_inc_delay(Event_defn.Packet.Previous_service_time,Link_dept_time,
                  Node_status.Routing_data.Adpt_data.Inc_delays(Event_defn.Packet.Previous_node));
    IF (Node_status.Routing_data.Adpt_data.Inc_delays(Event_defn.Packet.Previous_node)
    THEN
      Transmit_ID_of(Event_defn.Time,
                     Duration,
                     Node_status.Routing_data.Adpt_data,
                     Inc_delays(Event_defn.Packet.Previous_node),ID_of_hl,
                     Event_defn.Packet.Previous_node,
                     Event_defn.Node_No,
                     Node_status,
                     0_front, 0_back,
                     Event_R_length,
                     R_pkts_in_0);
    END;

( Check if packet contains routing information )
(-----)
IF ((Event_defn.Packet.Control_data = Control)
OR (Event_defn.Packet.Control_data = Data_and_ctrl))
AND (State_defn.Selected_algorithm = New_algo)
THEN
  BEGIN
    IF ((Event_defn.Packet.Control_data = Control)
OR (Event_defn.Packet.Control_data = Data_and_ctrl))
    THEN

```

APPENDIX D

```

BEGIN
  Stats_defn.Total_cost(int_no) := Stats_defn.Total_cost(int_no) + Event_defn.cost;

  { Calculate interval delays of routing packet }
  {-----}
  Temp_int_no := TRUNC(((Event_defn.Packet.Time_created +
                        No_of_intervals/duration) + 1);
  Stats_defn.Interval_R_pkt_delays(Temp_int_no) :=
  Stats_defn.Interval_R_pkt_delays(Temp_int_no) + (Event_defn.Time -
  Event_defn.Packet.Time_created);
  INC(Stats_defn.Interval_R_pkts_rcv(Temp_int_no));
END;

{ Service routing data }
{-----}
Service_routing_data(Event_defn.Duration,Node_status,
                     Q_front,Q_back,Event_defn.Length,R_pkts_in_Q);
IF (Event_defn.Packet.Control_data = data_and_ctrl)
THEN Event_defn.Packet.Control_data := data;
END;

{ Check if packet contains data }
{-----}
IF ((Event_defn.Packet.Control_data = data) OR (Event_defn.Packet.Control_data = data_and_ctrl))
THEN
  { Check if data has reached it's destination }
  {-----}
  IF (Event_defn.Packet.Destination_address = Event_defn.Node_no)
  THEN
    BEGIN
      write('Packet reached destination and is consumed');
      R_pkts_in_Q := R_pkts_in_Q - 1;
      WITH Stats_defn DO
        BEGIN
          { Calculate the message delay }
          {-----}
          Current_pkt_delay := Event_defn.Time - Event_defn.Packet.Time_created;
          Total_message_delays(int_no) := Total_message_delays(int_no)
          + Current_pkt_delay;
          INC(Messages_transmitted(int_no));
          Total_cost(int_no) := Total_cost(int_no) + Event_defn.Cost;

          { Calculate the delay distribution }
          {-----}
          INC(delay_distribution(i) + TRUNC(Current_pkt_delay * 101));
          IF (Current_pkt_delay > Max_delay) THEN Max_delay := Current_pkt_delay;
        END;
      END;
    END;
  END;
END;

```

APPENDIX D

```

( Calculate the delay with respect to the original interval )
(-----)
Temp_int_No := TRUNC(((Event_defn.Packet.Time_created +
  No_of_intervals/Duration) + 1);
Interval_pkt_delays(Temp_int_No) :=
Interval_pkt_delays(Temp_int_No) + (Event_defn.Time -
  Event_defn.Packet.Time_created);
INC(Interval_pkts_Rd(Temp_int_No));
END; (with)
END (if pkt reached it's destination)

( If data packet has not reached it's destination )
(-----)
ELSE

( If packet has not reached it's destination, then check the hop count )
(-----)
IF (Event_defn.packet.Hop_count >= Node_status.Routing_data.Number_of_nodes)
THEN
  BEGIN
    INC(Stats_defn.Total_pkts_discarded);
    IF (Event_defn.Packet.Control_data = data) THEN D_pkts_in_0 := D_pkts_in_0 - 1;
  END
ELSE

( Route data packet )
(-----)
  BEGIN
    Select_routing_algorithm(Node_status.Routing_data.Number_of_nodes,
      Event_defn.Node_status,Temp_OP_path);

( If no output path found )
(-----)
    IF (Temp_OP_path.No_link = TRUE)
    THEN
      BEGIN
        writeln(' No output path found, packet discarded');
        INC(Stats_defn.Pkts_not_dug_to_no_link);
      END
    ELSE
      BEGIN

( Calculate departure(s) of the packet )
(-----)
        Present_time := Event_defn.Time;
        FOR count := 1 TO Temp_OP_path.No_of_paths DO
          BEGIN
            IF (count > 1) THEN D_pkts_in_0 := D_pkts_in_0 + 1;
            OP_0 := Temp_OP_path.List_of_paths(count);
          END
        END
      END
    END
  END
END

```

APPENDIX D

```

Generate_departure(OP,Q,Present_time,Int_no,Duration,
                  Event_defn,Node_status,No_dept,
                  Q_front,Q_back,Event_Q_length,
                  Q_pkts_in_Q,Stats_defn);

(* If a departure is possible then update event queue *)
IF (No_dept = FALSE)
THEN
BEGIN
  Stats_defn.Total_dept_time := Stats_defn.Total_dept_time
    + (Event_defn.time - Present_time);
  Departure_time := Event_defn.time;
  MEN(New_event_ptr);
  New_ptr := Event_defn;
  IF (Q = ) THEN Event_defn.Packet.Previous_node := 0;
  Find_insert_pos(Departure_time,Q_front,Q_back,Temp_event_ptr,
  Insert_event(Temp_event_ptr,New_event_ptr,
    Event_Q_length,Q_pkts_in_Q,Insert));
END; (* IF No_dept *)
END; (*FOR count*)
END; (*ELSE*)
END; (*ELSE*)

PROCEDURE Departure_process (Network_data : Networks;
                             Int_No : Intervals;
                             VAR Node_status : Nodes;
                             VAR Event_defn : Event_Info;
                             VAR Q_front,Q_back : Link;
                             VAR Event_Q_length : INTEGER32;
                             VAR Q_pkts_in_Q : INTEGER32;
                             VAR Stats_defn : Stats_Info);
(*=====*)
(* This procedure simulates the departure of a packet to *)
(* a neighbouring node. The arrival time to the next *)
(* node is calculated and the event is inserted into the *)
(* event queue. *)
(* INPUTS: Variables: The status of the departure node. *)
(* The event description. *)
(* Pointers: Pointers to the front and back of *)
(* of the event list. *)
(* OUTPUTS: Variables: The updated event on the event Q. *)
(* The updated node status. *)
(* The event queue length. *)
(* Pointers: Pointers to the front and back of *)
(* of the event list. *)
(*=====*)

```

APPENDIX D

```

VAR New_arrival_time : REAL;
New_event_ptr      : LINK;
Temp_event_ptr     : LINK;
Insert              : BOOLEAN;

BEGIN
  DEC(Node_status.Output_0_length(Event_defn.Output_node_No));

  IF (Network_data.
    Capacities(Event_defn.Node_No,Event_defn.Output_node_No) = 0)
  THEN
    BEGIN
      IF (Event_defn.Packet.Control_data = data)
      THEN
        BEGIN
          P_pkts_in_0 := P_pkts_in_0 + 1;
          Stats_defn.Total_packets_blocked(Int_No) := Stats_defn.Total_packets_blocked(Int_No) + 1;
        END;
      END;
    ELSE
      BEGIN
        ( Calculate the arrival time due to the transmission delay )
        (-----)
        New_arrival_time := Event_defn.Time + (2 * DeltaT);
        (-----)
        IF (Event_defn.Packet.Control_data = data)
        THEN Stats_defn.Total_Tx_time := Stats_defn.Total_Tx_time
          + (New_arrival_time - Event_defn.Time);

        WITH event_defn DO
          BEGIN
            Time := New_arrival_time;
            Name := arrival;
            Node_No := Event_defn.Output_node_No;
          END;
          NEW(New_event_ptr);
          New_event_ptr^ := Event_defn;
          Find_insert_pos(New_arrival_time, @front_0, @back, Temp_event_ptr, Insert);
          Insert_event(Temp_event_ptr, New_event_ptr, Event_0_length, @pkts_in_0, Insert);
        END;
      END;
    END;

  PROCEDURE Process_events (Duration : REAL;
    Network_data : Network;
    VAR Front_0_0, Back_0_0 : LINK;
    VAR Length_of_event_0 : INTEGER32;
    VAR @pkts_in_0 : INTEGER32;
    VAR Status_of_nodes : All_nodes;
    VAR Stats_defn : Stats_info;
    VAR Tot_no_of_intervals : Interval);
    (*****
  
```

APPENDIX D

```

(#####)
(1)
(2) This procedure removes the events from the front of
(3) the queue, and control is passed to the appropriate
(4) process depending on the particular event extracted
(5) from the queue. Since the events are inserted into
(6) the queue in chronological order, the events are
(7) extracted from the queue in chronological order.
(8)
(9)
(10) INPUTS: Pointers: The pointers to the front and back
(11) of the event queue.
(12) Variables: The length of the event queue
(13) The status of all the nodes in the
(14) network.
(15) OUTPUTS: Screen: Appropriate messages are displayed
(16) on the screen.
(17) Variables: The length of the event queue
(18) The status of all the nodes in the
(19) network.
(20)
(#####)

VAR Current_event : Event_info;
    Terminate_proc : BOOLEAN;
    Termination_reason : 0..4;
    Ptr_to_old_event : Link;
    Interval_No : Integers;
    Old_event_No : Integers;

BEGIN
    writeln; writeln('Simulation begins');
    writeln('-----'); writeln;

    Terminate_proc := FALSE; Old_event_No := 0; Termination_reason := 0;

    WHILE ((Front_of_Q.Sentinel = FALSE) AND (Terminate_proc = FALSE)) DO
    BEGIN
        { Extract next event from the event queue }
        {
            Current_event := Front_of_Q; { Extract event data from Q }
            Ptr_to_old_event := Front_of_Q;
            Front_of_Q := Front_of_Q.next_event; { Remove event from Q }
            DISPOSE(Ptr_to_old_event);
            Length_of_event_Q := Length_of_event_Q - 1;
        }
    END

```

APPENDIX D

```

( Calculate interval number )
{-----}
Interval_No := TRUNC(((Current_event.Time + No_of_intervals*Duration) / I);
IF (Interval_No > Old_int_No)
THEN
BEGIN
  Old_int_No := Interval_No;
  Stats_Defn.Interval_Time[Interval_No] := Current_event.Time;
  writeln('-----');
  writeln('Time = ', Current_event.Time);
END;

( Process the event )
{-----}
CASE Current_event.name OF
  arrival : BEGIN
    Arrival_process(Interval_No,
                    Current_event,
                    Front_of_Q,
                    Back_of_Q,
                    Length_of_event_Q,
                    D_pkts_in_Q,
                    Status_of_nodes[Current_event.Node_no],
                    Stats_defn);
  END;
  service : BEGIN
    Service_process(Interval_No,
                   Duration,
                   Current_event,
                   Front_of_Q,
                   Back_of_Q,
                   Length_of_event_Q,
                   D_pkts_in_Q,
                   Status_of_nodes[Current_event.Node_no],
                   Stats_defn);
  END;
  departure : BEGIN
    Departure_process(Network_data,
                     Interval_No,
                     Status_of_nodes[Current_event.Node_no],
                     Current_event,
                     Front_of_Q,
                     Back_of_Q,
                     Length_of_event_Q,
                     D_pkts_in_Q,
                     Stats_defn);
  END;
END; { Case }

```

APPENDIX D

```

IF (D_pkts_in_3 <= 0) THEN Termination_reason := 2;
IF (Current_event.time > 2 * duration) THEN Termination_reason := 2;
IF Termination_reason < 0 THEN Termination_reason := TRUE;
END;

Tot_no_of_intervals := Interval_No;
writeln;
IF (Termination_reason)
THEN
BEGIN
CASE Termination_reason OF
1: writeln('Simulation ended: Time exceeds infinity');
2: writeln('Simulation ended: All data packets transferred');
3: writeln('Simulation ended: Duration exceeds four times expected duration');
END; (case)
END; ((if Termination_reason))
Stats.defn.End_time := Current_event.time;
writeln; writeln('End of simulation'); writeln;
list_n(Front_of_3, Length_of_event_3);
END;

PROCEDURE Calc_results (N : MODE_ID; Network_defn : Networks;
No_of_ints : Intervals; VAR Stats : State_info);
(*****)

(#####)
( )
( )
( ) This procedure calculates the required output results )
( ) for the simulation. )
( ) )
( ) )
( ) INPUTS: Variables: The number of intervals in the )
( ) simulation. )
( ) The results from the simulation. )
( ) )
( ) OUTPUTS: Variables: The updated results from the sim. )
( ) )
(#####)

VAR I : Intervals;
v, y : Mode_ID;
Tot_msgs_tx : INTEGER;
Tot_no_of_hops : INTEGER;
No_of_links : INTEGER;
Interval_duration : REAL;
Tot_msg_delay : REAL;
Tot_hop_delay : REAL;
Total_time : REAL;
Total_use : REAL;

BEGIN
Tot_msgs_tx := 0; Tot_no_of_hops := 0;
Tot_hop_delay := 0; Tot_msg_delay := 0;

```

APPENDIX D

```

( Calculation of costs and delays )
(-----)
FOR i := 1 TO No_of_stats DO
  WITH stats DO
    BEGIN
      Overall_cost := Overall_cost + Total_cost[i];
      IF (Messages_transmitted[i] < 0)
      THEN
        BEGIN
          Total_cost[i] := (Total_cost[i]/Messages_transmitted[i]);
          Average_message_delay[i] := (Total_message_delays[i]/Messages_transmitted[i]);
        END;

      ( Calculation of routing algorithm statistics )
      (-----)
      IF (Selected_algorithm = New_algo)
      THEN
        IF ((Interval_pkts_Rx[i] < 0) OR (Interval_R_pkts_Rx[i] < 0))
        THEN Total_interval_delays[i] := ((Interval_pkt_delays[i] + Interval_R_pkt_delays[i])/
          (Interval_pkts_Rx[i] + Interval_R_pkts_Rx[i]));

        IF (Interval_pkts_Rx[i] < 0)
        THEN Interval_pkt_delays[i] := Interval_pkt_delays[i]/Interval_pkts_Rx[i];

        IF (Interval_R_pkts_Rx[i] < 0)
        THEN Interval_R_pkt_delays[i] := Interval_R_pkt_delays[i]/Interval_R_pkts_Rx[i];

        IF ((Interval_pkts_Rx[i] < 0) OR (Interval_R_pkts_Rx[i] < 0))
        THEN Percentage_R_pkts_Rx[i] := ((100 * Interval_R_pkt_Rx[i]) /
          (Interval_R_pkts_Rx[i] + Interval_pkts_Rx[i]));

      ( Calculation of general results )
      (-----)
      Tot_msg_Rx := Tot_msg_Rx + Messages_transmitted[i];
      Tot_no_of_hops := Tot_no_of_hops + Number_of_hops[i];
      Tot_msg_delay := Tot_msg_delay + Total_message_delays[i];
      Tot_hop_delay := Tot_hop_delay + Hop_delay[i];

      IF (Number_of_hops[i] < 0)
      THEN Hop_delay[i] := Hop_delay[i]/Number_of_hops[i];

      No_of_pkts_blocked := No_of_pkts_blocked + Total_packets_blocked[i];
    END; ( with )

  ( Calculation of overall average results )
  (-----)
  WITH Stats DO
    BEGIN
      IF (Tot_msg_Rx < 0)
      THEN Overall_message_delay := Tot_msg_delay/Tot_msg_Rx;
    END;

```

APPENDIX D

```

IF (CP_path_No < 0)
THEN
  BEGIN
    Mode_defn(count).Routing_data.
      Output_probabilities(CP_path_No, Dest_node_No) := 1;
    Mode_defn(count).Routing_data.
      Pn(CP_path_No, Dest_node_No) := f;
  END;
END; (IF count)
ELSE Mode_defn(count).Routing_data.
  shortest_path_node(Dest_node_No) := 0;

END; (IF Dest_node_No)

( Initialise the blocked paths set )
(-----)
FOR x := 1 TO N DO
  IF (Network_defn.capacity(count, x) = 0)
  THEN
    FOR y := 1 TO N DO
      Mode_defn(count).Routing_data.adpt_data.B[y]
        := Mode_defn(count).Routing_data.adpt_data.B[y] + (x);
    END; (FOR count)
  END;

```

```

PROCEDURE Initialise_event_Q (VAR Front_of_Q, back_of_Q : link;
  VAR Qlength : INTEGER32;
  VAR Q_pnts_in_Q : INTEGER32);
(-----)

```

```

(*****)
(
  ( An empty event queue is created and sentinel (dummy) )
  ( event is inserted into the queue. )
  (
  ( INPUTS: Pointers: Front (pointer to front of queue) )
  ( Back (pointer to back of queue) )
  ( OUTPUTS: Queue: An empty event queue )
  ( Variable: Queue_length (Set to zero) )
  (
  (=====)

```

```

BEGIN
  NEW(Front_of_Q);
  Front_of_Q.sentinel := TRUE;
  Back_of_Q := Front_of_Q;
  Q_pnts_in_Q := 0;
  Qlength := 0;
END;

```

APPENDIX E

```
PROCEDURE Find_Insert_posn Insert_time : REAL;
                                Front_of_B : Back_of_B : LINK;
                                VAR Temp : LINK;
                                VAR Insert : BOOLEAN;
(=====)

(=) (=)
(=) (=)
(=) This procedure finds the correct chronological order (=)
(=) in the event queue where a new event is to be (=)
(=) inserted. (=)
(=) (=)
(=) INPUTS: Pointers: The pointers to the front and back (=)
(=) of the list (=)
(=) Variable: The time at which the event occurs (=)
(=) OUTPUTS: Pointer: The pointer to the insert point in (=)
(=) the queue (=)
(=) Boolean: Set to true if an insert position (=)
(=) is found, indicating the event may (=)
(=) be inserted into the queue. (=)
(=====)

BEGIN
Temp := Front_of_B;
WHILE ((Temp^.nextlink = FALSE) AND (Temp^.time < Insert_time))
DO Temp := Temp^.next link;
Insert := TRUE;
END;
```

```
PROCEDURE Insert_event (VAR Temp, New_arrival : LINK;
                        VAR B_ptrk_in_B : INTERGER;
                        VAR Insert : BOOLEAN);
(=====)

(=) (=)
(=) (=)
(=) This procedure inserts an event into the event list. (=)
(=) The event is inserted before the temporary pointer. (=)
(=) Using a temporary storage variable it is not (=)
(=) necessary to traverse through the whole list (=)
(=) or to insert the event into the list. (=)
(=) (=)
(=) (=)
(=) INPUTS: Pointers: The pointer indicating the position (=)
(=) where the event is to be inserted. (=)
(=) The pointer to the new event to be (=)
(=) inserted into the queue. (=)
(=) OUTPUTS: Variable: The length of the event queue. (=)
(=) Boolean: Set to false after the insertion of (=)
(=) the event. (=)
(=====)
```

APPENDIX D

```

(***      Initialise event queue      ***)
(-----)
    Initialise_event_queue(Front,Back,Event_0_length,Data_pkts_in_0);

(***      Generate random arrivals or get arrivals from user    ***)
(-----)
    writeln; write('Enter events manually (y/n) ? ');
    readln(Choice);
    IF (Choice IN 'y','Y')
    THEN Enter_events(N,Total_Duration,Front,Back,Event_0_length,
                    Data_pkts_in_0,Stats_data)
    ELSE Random_arrivals(N,Total_duration,Network.Expected_traffic,
                    Front,Back,Event_0_length,Data_pkts_in_0,Stats_data);

(***      Initialise nodes      ***)
(-----)
    Initialise_Node_data(N,Network,Nodes);

(***      Find and eliminate loops in the network      ***)
(-----)
    IF (Stats_data.selected_algorithm = New_Algo)
    THEN
        BEGIN
            Eliminate_loops(Network);
            Find_all_BUs(Network,Nodes);
            List_blocked_paths(Network,Nodes);
        END;

(***      Set start time of simulation      ***)
(-----)
    cal_get_local_time(t_start_time);

(***      Process events and perform simulation      ***)
(-----)
    Process_events(Total_Duration,Network,Front,Back,
                    Event_0_length,Data_pkts_in_0,Nodes,Stats_data,No_of_periods);
    cal_get_local_time(t_end_time);

(***      Calculate results and send results to file      ***)
(-----)
    Calc_results(N,Network,No_of_periods,Stats_data);
    Send_OP_to_file(N,Stats_data,No_of_periods,Total_duration,
                    Network.Network_ID_name,OP_filename);

```

APPENDIX D

```

(***      Get end time of simulation and calc. duration      ***)
(-----)

sub_clock := cal_sub_clock(Sim_end_time, Sim_start_time);
Annu_of_sec := cal_sclock_to_sec(Sim_end_time);
sub_clock := cal_sub_clock(Sim_start_time, init_start_time);
Innu_of_sec := cal_sclock_to_sec(Sim_start_time);
Cnuu_of_sec := Annu_of_sec + Innu_of_sec;
writeln;
writeln('Duration of initialisation = ', Annu_of_sec, ' seconds');
writeln('Duration of simulation      = ', Innu_of_sec, ' seconds');
writeln('Total duration              = ', Cnuu_of_sec, ' seconds');
writeln;
writeln('=====');
writeln;

END. ( Simulation program )
(*****)

```

D.2 Routine to initialise the routing data

APPENDIX D

```

(=====)
(*)
(*) Second derivative Routing Algorithm (*)
(*) ===== (*)
(*) (*) (*)
(*) Name: 'JMITAdpt.pas' (*)
(*) (*) (*)
(*) Description: The function of the module is to initialise (*)
(*) the data required by the second derivative (*)
(*) routing algorithm. (*)
(*) (*) (*)
(*) Version: 1.0 (*)
(*) (*) (*)
(*) Author: Marc Leon Francois (*)
(*) (*) (*)
(*) Date: November 1987 (*)
(*) (*) (*)
(-----)
(*) Modifications (*)
(*) Author: (Name) (*)
(*) Date: (Date) (*)
(*) Description: (Description of modification(s)) (*)
(*) (*) (*)
(=====)

```

```

PROCEDURE Init_routing_data (N : Node_12);
    VAR Temp_Adpt_Info : Adpt_Info1;
(=====)

(=====)
(*)
(*) The function of this procedure is to initialise the (*)
(*) the routing data at each node. This procedure is (*)
(*) called from the 'Init_node_data' procedure in the (*)
(*) the main prog. This procedure is therefore executed (*)
(*) for each node in the network. (*)
(*) (*) (*)
(*) INPUTS: Constant: The number of nodes in the network. (*)
(*) Variables: The Adpt routing data for the node. (*)
(*) OUTPUT: Variables: The Adpt routing data for the node. (*)
(*) (*) (*)
(=====)

VAR i, j, k : Node_12;

```

APPENDIX D

```

BEGIN
  writeo('initialising data')
  WITH Temp_Adpt_Info DO
    BEGIN
      ob_ofs_Rx := 0;
      All_ob_ofs_Received := FALSE;
      FOR i := 1 TO N DO
        WITH Inc_delays(i) DO
          BEGIN
            ob_ofs_xi := 0;
            ob_ofs_of := 0;
            New_busy_period := TRUE;
            xi_tot := 0;
          END;
        END;

      FOR j := 1 TO N DO
        FOR k := 1 TO N DO
          BEGIN
            delta(j,k) := 0;
            ds_dr_data(j,k).ob_dr := 0;
            ob_dr_data(j,k).Toq := FALSE;
          END; (for j,k)

          FOR j := 1 TO N DO ok_to_proceed(j) := FALSE;
          FOR j := 1 TO N DO ob_of_xi(j) := 0;
          FOR j := 1 TO N DO ob_data_Rx(j) := 0;
        END; (with temp_adpt)
      END;
    END;
  END;

```

APPENDIX D

D.3 Procedures required for the minimum delay routing algorithm

```

(-----)
(*
(* Second derivative Routing Algorithm
(* -----)
(*
(* Name: 'Adapt.pas'
(*
(* Description: The function of the module is to simulate
(* the second derivative routing algorithm.
(* The module basically consists of the
(* operations that can be performed on the
(* routing data.
(* (List the operations below)
(*
(* Version: 2.0
(* This version calculates the incremental
(* delays.
(*
(* Author: Marc Leon Francois
(*
(* Date: November 1987/January 1988
(*
(* -----)
(* Modifications
(* Author: (Name)
(* Date: (Date)
(* Description: (Description of modification(s))
(*
(-----)

```

```

FUNCTION Max (X,Y : REAL) : REAL;
(-----)

```

```

(-----)
(*
(* The function of this process is to simply find the
(* maximum between two real numbers x and y.
(*
(* INPUTS: Variables X,Y two real numbers
(* OUTPUTS: Variable Max(X,Y)
(*
(-----)

```

```

BEGIN
  IF (X > Y) THEN Max := X ELSE Max := Y;
END;

```

APPENDIX D

```

FUNCTION Min (X,Y : REAL) : REAL;
(=====)

(=====)
*)
*)
*) The function of this process is to simply find the *)
*) minimum between two real numbers x and y. *)
*)
*)
*) INPUTS: Variables: X,Y Two real numbers *)
*) OUTPUTS: Variables: Min(X,Y) *)
*)
(=====)

BEGIN
  IF (X < Y) THEN Min := X ELSE Min := Y;
END;

PROCEDURE Calc_Inc_Delay (In, Out : REAL;
  VAR Delay_Info : Inc_Delays_Data);
(=====)

(=====)
*)
*) The function of this procedure is to calculate the *)
*) incremental delay along a path, to a particular dest- *)
*) ination for a particular node. The delay is updated *)
*) each time a message leaves a node. *)
*)
*)
*) INPUTS: Variables: The arrival and departure time of *)
*) the packet. *)
*) OUTPUTS: Variables: The new incremental delay. *)
*)
(=====)

VAR I : INTEGER;
    Ctot : REAL;
    CIn : REAL;
    delta_Id_of : REAL;

BEGIN
  WITH Delay_Info DO
    BEGIN
      IF (New_busy_period = TRUE)
      THEN
        BEGIN
          n := 0; New_busy_period := FALSE; In_1 := 0; In_2 := 0;
          END;
          n := n + 1; Ctot := 0; NI_tot := NI_tot + 1;
        END;
      END;
    END;
  END;

```

APPENDIX D

```

FOR i := 1 TO n DO
  BEGIN
    IF (i = n) THEN Cln := Cn - An;
    IF (i = n - 1) THEN Cln := Cn_1 - Bn(Cn, Cn_1);
    IF (i < n - 1) THEN Cln := Min(Cn, Cn_1 - An);
    IF (Cln < 0) THEN Cln := 0;
    Ctot := Ctot + Cln;
    Cn[i] := Cln;
  END;

  Cn_2 := Cn_1; Cn_1 := Cn;
  d0_d0_xi := ((d0_d0_xi * (N1_tot - 1)) + Ctot) / N1_tot;

  Delta_d0_d0 := d0_d0_xi - Old_d0_d0;
  IF (ABS(Delta_d0_d0) > Inc_delay_Tot) THEN New_busy_period := TRUE;
  IF (n > max_n) THEN New_busy_period := TRUE;
  IF (New_busy_period = TRUE) THEN Old_d0_d0 := d0_d0_xi;
END;
END;

PROCEDURE All_R_data_Rn (Event_defn : Event_Info;
                        : Node_ID;
                        VAR Node_Status : Node );
(*=====*)
(* The function of this procedure is to check if all of *)
(* the necessary routing data has been received from the *)
(* neighbouring nodes. The routing algorithm may only be *)
(* updated once all this data has been received by the *)
(* node. *)
(* *)
(* INPUTS: Event_defn: The event defining the packet *)
(*          Just received and containing the routing in- *)
(*          formation. *)
(*          Node_Status: The status of the node which *)
(*          just received the routing information. *)
(* *)
(* OUTPUTS: Node_Status: The status of the node which *)
(*          just received the routing information. *)
(*          (In particular the variable defining if all *)
(*          the necessary data has been received by the *)
(*          node. *)
(* *)
(*=====*)
VAR i : Node_ID;

```

APPENDIX D

BEGIN

(Check if d0_fr has been received from all neighbouring nodes)

(-----)

IF (Node_status.Routing_data.Adpt_data.OK_to_proceed[j] = FALSE)

THEN

BEGIN

Node_status.Routing_data.Adpt_data.OK_to_proceed[j] := TRUE;

FOR k := 1 TO Node_status.Routing_data.Number_of_nodes DO

IF ((Node_status.Routing_data.Link_capacity(Event_defr.Node_No,k) > 0)

AND (k < j))

AND (Node_status.Routing_data.Ph[k,j] > 0)

AND NOT ((Node_status.Routing_data.Adpt_data.R_data_Rt[j]))

THEN Node_status.Routing_data.Adpt_data.OK_to_proceed[j] := FALSE;

END; (if)

END;

PROCEDURE Adj_d0_fr_Rt (Current_ : Node_ID;
Event_defr : Event_Info;
VAR Node_status : Node)
(-----)

(*****)

(*)

(*) The function of this procedure is to check if all of

(*) the necessary d0_frs have been received from the

(*) neighbouring nodes. The routing algorithm may only be

(*) updated once all this data has been received by the

(*) node.

(*)

(*) INPUTS: Event_defr: The event defining the packet

(*) just received and containing the routing in-

(*) formation.

(*)

(*) Node_status: The status of the node which

(*) just received the routing information.

(*)

(*) OUTPUTS: Node_status: The status of the node which

(*) just received the routing information.

(*)

(*) (In particular the variables defining if all

(*) the necessary data has been received by the

(*) node.)

(*)

(*****)

VAR k : Node_ID;

BEGIN

APPENDIX D

```

( Check if  $\phi_D$  has been received from all neighbouring nodes )
(-----)
Node_status.Routing_data.Adpt_data.All_dls_Received := FALSE;
IF (Node_status.Routing_data.Adpt_data.All_dls_Received = FALSE)
THEN
  BEGIN
    Node_status.Routing_data.All_dls_Received := TRUE;
    FOR  $k := 1$  TO Node_status.Routing_data.Number_of_nodes DO
      IF (Node_status.Routing_data.Link_capacities(Event_defs.Node_No,k) > 0)
      AND (Node_status.Routing_data.Phill(current_j) > 0.0)
      AND (NOT (X (Node_status.Routing_data.Adpt_data. $\phi_D$ _R $_k$ )))
      THEN Node_status.Routing_data.Adpt_data.All_dls_Received = FALSE;
    END; ( if )
  END;

```

```

PROCEDURE Calc_blocked_paths (N, i, j : Mode_ID;
                             VAR R_data : Routing_Info);
(*****)

```

```

(=====)
(
  ( The function of this procedure is to calculate the
  ( paths for which Phi is lagproper. That is the paths
  ( for which Phi say not be increased from zero.
  (
  ( INPUTS: Variables: N, the number of nodes in the
  ( network as known by node i.
  ( i, the current node number.
  ( j, the destination node number.
  (
  ( Routing_data: Phi[i,k()]: Phi values
  (
  (  $\phi_D$ _dr[k()]:
  ( T[i,j] Traffic for node i
  (
  ( OUTPUTS: Routing_data:  $\phi_D$ _dr[i,j]: Tag
  (
  (=====)

```

```

VAR n : INTEGER;
Tag_data : REAL;
Tag_pdt : BOOLEAN;

```

```

BEGIN
  Tag_pdt := FALSE;
  FOR n := 1 TO N DO
    BEGIN
      ( Check for lagproper paths )
      (-----)
      IF (R_data.Phi(n,j) > 0)
      AND (R_data.Adpt_data. $\phi_D$ _dr_data(n,j). $\phi_D$ _dr
      R_data.Adpt_data. $\phi_D$ _dr_data(n,j). $\phi_D$ _dr
      AND (R_data.Adpt_data. $\phi_D$ _dr_data(n,j). $\phi_D$ _dr > 0))
      THEN

```

APPENDIX D

```

(* INPUTS: Variables: Event; Defines packet, dest No etc. *)
(*      Mode status: Defines routing path *)
(*      for the particular node. *)
(* OUTPUTS: Variables: OP_path: defines the output link *)
(*      The packet is to be sent on. *)
(*      *)
(=====)

VAR p,d : Node_ID;

BEGIN
  writeln(' Flooding algorithm');
  IF (Event_defn.packet.destination_address = 0)
  THEN writeln(' Broadcast packet');

  p := Event_defn.Node_No; (*Current node number *)
  OP_path.No_of_paths := 0;  OP_path.Path_found := FALSE;
  OP_path.No_link := False;

  (* Flood packet on all output links of the node *)
  (=====)
  FOR d := 1 TO Mode_status.Routing_data.Number_of_nodes DO
    IF ((p < d) AND (Mode_status.Routing_data.Link_capacity(p,d) < 0)
        AND (d < Event_defn.Packet.Previous_node))
    THEN
      BEGIN
        INC(OP_path.No_of_paths);
        OP_path.List_of_paths(OP_path.No_of_paths) := d;
      END;

  IF (OP_path.No_of_paths < 0) THEN OP_path.Path_found := TRUE;

END;

PROCEDURE Select_routing_algorithm (No_of_nodes : Node_ID;
                                   Event_defn : Event_info;
                                   Mode_status : Mode;
                                   VAR OP_path : OP_path_defn);
(=====)

(=====)
(*
(* The function of this procedure is to correctly choose
(* the appropriate routing algorithm to route the packet
(* currently being serviced. This procedure allows
(* numerous algorithms to be selected if desired. For
(* example it may be necessary to route certain packets
(* using a flooding algorithm, and others using a least
(* cost algorithm. Each message may then insist on a
(* particular routing algorithm if desired. *)

```

APPENDIX D

```

(* INPUTS: Variable: Event: This variable is required *)
(* since it contains the info. about *)
(* The packet to be routed. *)
(* Variable: Mode status: Contains info. about *)
(* the particular node. *)
(* OUTPUTS: Variable: OP_path: This defines the output *)
(* link(s) the packet has to be trans *)
(* mitted. *)
(*****)

BEGIN
CASE Event_defn.packet.Routing_algorithm OF
  SP_or_LG : Short_path(Event_defn,Mode_status,OP_path);
  Flooding : Flood_pkt(Event_defn,Mode_status,OP_path);
  Now_also : Adaptive_algo(Nb_of_nodes,Event_defn,Mode_status,OP_path);
END;
END;

PROCEDURE Generate_departure (OP_Q_No : Node_ID;
                             Present_time : REAL;
                             Period_No : Interval;
                             Duration : REAL;
                             VNR_Event_defn : Event_info;
                             VNR_Mode_status : Mode;
                             VNR_No_departure : BOOLEAN;
                             VNR_Front_of_Q_Back_of_Q : Link;
                             VNR_Length_of_Q : INTEGER32;
                             VNR_Q_ptr_in_Q : INTEGER32;
                             VNR_Status_defn : Status_info);
(*****)

(* ***** *)
(* This procedure simulates the queuing of a packet onto *)
(* the appropriate output queue as determined by the *)
(* the routing algorithm. The departure time is calc- *)
(* ulated. The incremental delays for the routing algo- *)
(*  are updated, and if necessary the appropriate updated *)
(* routing data is transferred to the neighbouring nodes. *)
(* ***** *)
(* INPUTS: Variables: The Q/P queue number via which *)
(* the packet must be transmitted. *)
(* The interval number. *)
(* The duration of the simulation. *)
(* Pointers: To the front and back of the event *)
(* queue. *)
(* Variables: The length of the event queue. *)
(* OUTPUTS: Variables: The definition of the next arrival *)
(* event. *)
(* The status of the current node. *)

```

APPENDIX D

```

(* Pointers: To the front and back of the event *)
(* queue. *)
(* Variable: The length of the event queue. *)
(* Boolean: True if no departure is generated *)
(* Variables: The statistics variables. *)
(*****)

VAR Dept_time : REAL;
    Current_node_No : INTEGER;
    Pkts_per_sec : REAL;
    j : Node_Id;

BEGIN
    Current_node_No := Event_defn.Node_No; No_departure := TRUE;
    IF (Node_status.Output_Q_length(OP_Q_No) = Node_status.Max_output_Q_length)
    THEN
        BEGIN
            INC(State_defn.Total_packets_blocked(Period_No));
            IF (Event_defn.Packet.Control_data = data) THEN O_pkts_in_Q := O_pkts_in_Q + 1;
        END
    ELSE
        BEGIN
            No_departure := FALSE;
            Pkts_per_sec := 1000 * (Router.capacity_in_Kbps)
            Node_status.Coupling_data.Link_capacity := (Current_node_No, OP_Q_No) /
            Node_status.Bits_per_packet;

            { Calculate the departure time }
            { ----- }
            { Now calculate the departure = Current time + (Capacity of the }
            { of the link x the number of packets in the queue) + a small }
            { delay. }
            { ----- }

            Dept_time := (Present_time) +
                (Node_status.Output_Q_length(OP_Q_No) / Pkts_per_sec)
                + (1 * Delta_t);
            INC(Node_status.Output_Q_length(OP_Q_No)); { Increment OP_Q }

            { Update next event }
            { ----- }
            WITH Event_defn DO
                BEGIN
                    Time := Dept_time;
                    Name := Departure;
                    Output_Node_No := OP_Q_No;
                    Previous_dept_time := Dept_time;
                    Packet.Previous_node := Current_node_No;
                    Packet.Previous_service_time := Present_time;
                    Packet.Node_count := Packet.Node_count + 1;
                END;
            END; { else }
        END; { Procedure }

```

APPENDIX D

```

( Check if blocked path set has to be updated )
(-----)
IF (NOT(new_k IN Mode_status.Routing_data.Adpt_data.BI[Current_j]) AND
  (Mode_status.Routing_data.Adpt_data.d0_dr_data[new_k].Current_j.Tag = TRUE))
  THEN Mode_status.Routing_data.Adpt_data.BI[Current_j] :=
    Mode_status.Routing_data.Adpt_data.BI[Current_j] + {new_k};
END;

END;

PROCEDURE Reset_received_data (N : Mode_ID); VAR Temp_Adpt_data : Adpt_info;
(=====)

(=====)
(*
  (* After using the received routing data to update the *)
  (* routing tables the flags are reset and the node waits *)
  (* for additional data to be received. *)
  (*
  (* INPUTS: N : Number of node in the network. *)
  (* OUTPUTS: Adpt_data : Routing info containing flags. *)
  (*
  (=====)

VAR x : Mode_ID;

BEGIN
  WITH Temp_Adpt_data DO
    BEGIN
      All_d0_data_received := FALSE;
      d0_data_Rx := [];

      FOR x := 1 TO N DO
        BEGIN
          OK_to_proceed(x) := FALSE;
          (d0_data_H(x) := []);
        END;
      END;
    END;
  END;

END;

PROCEDURE Service_Routing_data (Event_defn : Event_info;
  Duration : REAL;
  VAR Mode_Status : Mode;
  VAR Q_front, Q_back : Link;
  VAR Event_Q_length : INTEGER32;
  VAR Q_size_in_Q : INTEGER32);
(=====)

```

APPENDIX D

```

(*****)
(*
(* The function of this procedure is to perform the app- *)
(* appropriate functions when a node receives a packet con- *)
(* taining routing information. *)
(*
(* INPUTS: Event_def: The event and packet containing *)
(*           the routing data. *)
(*          Duration: The duration of the simulation. *)
(*          Node_status: The status and routing info of *)
(*                      the node. *)
(* OUTPUTS: Node_status: The updated routing tables. *)
(*          Pointer: Pointer to front and back of the *)
(*                event queue. *)
(*          Q length: The length of the event queue. *)
(*
(*****)

VAR j : Node_Id;

BEGIN

  ( Accept the new routing data )
  (-----)
  Receive_routing_data(Event_def,Node_status);

  ( For each destination node j )
  (-----)
  FOR j := 1 TO Node_status.Routing_data.Number_of_nodes DO

    ( If j <> i )
    (-----)
    IF (j <> Event_def.Node_No)
    THEN
      BEGIN

        ( Check if all d0/d1s have been received )
        (-----)
        All_d0_d1s_Rx(Event_def,Node_status);

        ( If all d0/d1s have been received then )
        (-----)
        IF (Node_status.Routing_data.Rept_data.All_d0_d1s_received)
        THEN
          BEGIN

            ( Check if all d0/d1s have been received )
            (-----)
            All_d_data_Rx(Event_def,j,Node_status);
          
```

APPENDIX D

```

( IF all the d0/ds have been received )
{-----}
IF (Node_status.Routing_data.Adpt_data.R( to_procmt[] ) = TRUE)
THEN
BEGIN
  Calculate_new_d0_d0(Node_status.Routing_data.Number_of_nodes,
    j,Event_defn.Node_no,
    Node_status.Routing_data);
  Transmitt_new_d0_d0(Event_defn.Time,
    Operation,
    Node_status.Routing_data,
    Adpt_data.d0_d0_data(Event_defn.Node_No,j),
    j,
    Event_defn.Node_No,
    Event_defn.Packet.N_Previous_node,
    Node_status,
    Q_front,Q_back,
    Event_Q_length,
    Q_size_in_Q);
  Update_Routing_data(Node_status.Routing_data,Node_status,
    Event_defn.Node_No, j,
    Node_status.Routing_data);
  Reset_received_data(Node_status.Routing_data.Number_of_nodes,
    Node_status.Routing_data.Adpt_data);
END; (if all d0/ds)
END; (if i <> j)
END; (procedure)

```

{-----}

APPENDIX D

D.4 Procedures required for the elimination of loops

```
(*****)
(*
(* Name: 'Block.Pas'
(*
(* Description: This program is used to determine any loops
(* in the network and to determine a valid
(* blocked path set.
(*
(* Version: 1.0
(*
(* Author: Marc Leon Francois
(*
(* Date: 28 March 1988
(*
(*
(*
(*****)
```

```
PROCEDURE Find_j (Network_defn : KNetwork; j : Node_ID;
VAR i : node_ID);
[*****]
```

```
(*****)
(*
(* This procedure is to find a suitable source node for
(* the particular destination node.
(*
(* INPUTS: Constants: Network definition,
(* Constants: j the destination node
(* OUTPUTS: Variable: i the source node.
(*
(*
(*****)
```

```
BEGIN
  (* Find source node numerically furthest from j *)
  (-----)
  (* i := Network_defn.Number_of_nodes - j + 1 *)

  (* Use node i or node N as source node *)
  (-----)
  (* IF i = 1 *)
  (* THEN i := Network_defn.Number_of_nodes *)
  (* ELSE i := 1 *)

  (* Use node N as source node or node i *)
  (-----)
  (* IF j = Network_defn.Number_of_nodes *)
  (* THEN i := 1 *)
  (* ELSE i := Network_defn.Number_of_nodes *)
```

END;

APPENDIX D

```
PROCEDURE Find_sink_tree (i,j : Mode_ID) VAR Network_defn : Networks);
  (*****)
```

```
  (*****
  (i This procedure is to find the sink tree from a pair)
  (j source destination pair.
  (k
  (l INPUTS: Constant: Network definition,
  (m Constant: i the source node and
  (n j the destination node.
  (o OUTPUTS: Variables: The matrix defining the sink tree
  (p defined in network_defn.
  (q
  (r
  (s
  (t
  (*****)
```

```
  VAR k, m : Mode_ID;
```

```
  BEGIN
```

```
    { Relativize the sink tree }
    (-----)
    FOR k := 1 TO Network_defn.Number_of_nodes DO
      FOR m := 1 TO Network_defn.Number_of_nodes DO
        Network_defn.Sink_tree(k,m) := Not_connected;
```

```
    { Find all output paths for the source node }
    (-----)
    FOR k := 1 TO Network_defn.Number_of_nodes DO
      IF ((k < j) AND (Network_defn.Capacities(i,k) > 0))
      THEN Network_defn.Sink_tree(i,k) := Output_path;
```

```
    { Find output path for all nodes k, k(i) and k(j) }
    (-----)
    FOR m := 1 TO Network_defn.Number_of_nodes DO
      IF ((m < i) AND (m < j))
      THEN
        FOR k := 1 TO Network_defn.Number_of_nodes DO
          IF ((k < i) AND (Network_defn.Sink_tree(k,m) = Not_connected)
          AND (Network_defn.Capacities(m,k) > 0))
          THEN Network_defn.Sink_tree(m,k) := Output_path;
```

```
  END;
```

Author Francois Marc Leon

Name of thesis Routing Strategies In Digital Networks. 1988

PUBLISHER:

University of the Witwatersrand, Johannesburg

©2013

LEGAL NOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed, or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page) for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.