(4)

RADC-TR-89-279
Final Technical Report
December 1989

AD-A218 829

ISSUES IN GRADIENT-BASED ADAPTIVE AND DISTRIBUTED SCHEMES FOR DYNAMIC NETWORK MANAGEMENT

University of Massachusetts

Christos G. Cassandras, James F. Kurose, Don Towsley

MAR 08 1990
E D

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

90 03 08 053

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-89-279 has been reviewed and is approved for publication.

APPROVED:

JOHN M. COLOMBI, 1Lt, USAF
Project Engineer

APPROVED:

JOHN A. GRANIERO
Technical Director
Directorate of Communications

FOR THE COMMANDER:

JAMES W. HYDE, III.
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (DCLD) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notice on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-89-279
6a. NAME OF PERFORMING ORGANIZATION University of Massachusetts 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (DCLD)
6c. ADDRESS (City, State, and ZIP Code) Amherst MA 01003	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) DCLD	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-88-D-0027
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 33126F; PROJECT NO. 2022; TASK NO. 01; WORK UNIT ACCESSION NO. P4

11. TITLE (Include Security Classification)
ISSUES IN GRADIENT-BASED ADAPTIVE AND DISTRIBUTED SCHEMES FOR DYNAMIC NETWORK MANAGEMENT

12. PERSONAL AUTHOR(S)
Christos G. Cassandras, James F. Kurose, Don Towsley

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Apr 88 TO Apr 89	December 1989	176

16. SUPPLEMENTARY NOTATION
Prime contractor for this effort is Syracuse University.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Routing Algorithms Flow Control Queueing Systems
25	05		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report details work which proceeded in three primary areas: (1) Algorithm performance, (2) flow control of real-time traffic and (3) scheduling mechanisms for real-time traffic. Algorithm performance was made between Gallagher's distributed datagram algorithm, the CODEC virtual circuit, and the Arpanet datagram (primarily single path) algorithms. Results on the comparative mean-delay performance study includes the ability of gradient-based algorithms to out-perform their shortest path counterparts, especially under moderate to heavy traffic.

A key research issue was the use of Perturbation Analysis to obtain gradient estimates on-line without imposing any queueing modeling assumptions on the network.

The approach on real-time traffic control research was toward developing algorithms to

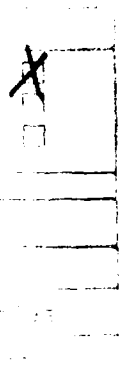
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL John M. Colombi, 1Lt, USAF	22b. TELEPHONE (Include Area Code) (315) 330-7751 22c. OFFICE SYMBOL RADC (DCLD)

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

reduce packet loss (end-to-end) due to deadline constraints.

It was shown that it is advantageous to apply flow control that deliberately rejects or discards portions of the incoming traffic. Various policies that were examined included traffic rejection as a function of queue length and LIFO Service disciplines. Results, simulation, analysis and mathematical proofs are included.



A-1

TABLE OF CONTENTS

1. INTRODUCTION.

This report summarizes the work we have performed over the past 12 months with two primary objectives in mind:

1. Compare the properties of existing and proposed routing algorithms for communication networks

2. Investigate network management problems arising in a multiple traffic class environment, specifically in the presence of real-time traffic

In what follows, we discuss these objectives in section 1.1. We then identify the basic issues related to routing algorithms in section 1.2, so as to set the stage for our comparative study of such algorithms. In section 1.3, we present the issues involved in networks required to handle mutiple traffic classes (e.g. data, voice, video). Finally, in section 1.4, we describe the organization of this report.

1.1. Project Objectives.

To meet the first objective above, we have considered the following three representative routing algorithms: (1) the CODEX algorithm [1],[2], implemented at CODEX Corp. and commercially available, (2) Gallager's distributed algorithm, proposed [3] and widely referenced in the technical literature but not implemented to date, and (3) the well-known ARPANET algorithm [4], currently in operation. The choice of these algorithms is largely motivated by their diverse nature and operating conditions, as will be further discussed below. This report identifies the essential characteristics and critical issues in these algorithms through experimention under a variety of traffic and network operating conditions. Details are presented in sections 2, 3, and 4.

A critical issue that has emerged from our comparative study is that of performing efficient *gradient estimation*, an essential function in the process of selecting routes in a network. The interpretation of a "gradient" in this context is the *sensitivity* of a predefined performance measure (typically, the average packet delay) with respect to various adjustable routing variables; it is precisely this information which is systematically processed to implement routing adjustments in an adaptive sense. Our emphasis has been on obtaining this information *on-line* (using actual network data as available in practice), rather than relying on analytical expressions based on often restrictive assumptions. Details of this ongoing research are provided in section 5.

The second objective of our work has led to the formulation and solution of some fundamental problems arising when real-time traffic is present in a network. Direct extensions of existing

routing algorithms (including the three algorithms considered in this report) are not possible without a thorough understanding of the characteristics of such traffic. Our results are reported in section 6.

1.2. Basic Issues in Routing Algorithms.

This section is devoted to a description of the basic issues related to routing algorithms. Routing is one of the principal functions of the Network Layer in the OSI reference model, and primarily deals with deciding the path (i.e. a sequence of links) that a packet originating at one node of a network takes in order to reach its destination. This decision is traditionally based on the following network-wide objective: the minimization of the *mean delay* incurred by all packets served by the network. This consists of queueing delays at intermediate nodes, as well as transmission and propagation delays on the links traversed.

In what follows, we briefly discuss the main features used in differentiating routing algorithms:

- packet switching schemes: *Datagram* (DG) vs. *Virtual Circuits* (VC)
- information dissemination: *Centralized* vs. *Distributed* algorithms
- route selection: *shortest path* vs. *gradient-based* algorithms

1.2.1. Datagram vs. Virtual Circuit Packet Switching.

In a datagram packet switching scheme, every packet follows a path from its source to its destination *independent* of the paths taken by other packets. On the other hand, in a virtual circuit environment packets are constrained to follow fixed paths which are established during the *call setup* phase.

Thus, the main operational concerns in datagram routing are:

- At every node, determine the outgoing link on which an arriving packet is to be transmitted. This decision is normally read from a nodal routing table.

- Update all nodal routing tables periodically, in order to accommodate any changes in the network traffic conditions.

In contrast, the concerns of a virtual circuit routing algorithm are :

- Determine the best path (sequence of links) for a new virtual circuit

- Reroute existing virtual circuits periodically, in order to accommodate any changes in the network traffic conditions.

The ARPANET algorithm is based on a datagram packet switching scheme. The same is true for Gallager's algorithm, although the route selection mechanism is different (see section 1.3 below). Finally, the CODEX algorithm employs virtual circuits, except that control packets are still treated as datagram-based traffic.

1.2.2. Centralized vs. Distributed Routing Algorithms.

In centralized routing, a Routing Control Center (RCC) collects feedback information from all the nodes in the network and, after processing, sends back all control information required to implement routing decisions. On the other hand, in distributed routing every node gathers information from all other nodes in the network, either directly or indirectly, and makes local routing decisions based on this acquired information. The obvious drawback of a centralized scheme is that a failure at the RCC may disable the entire network. Thus, a key motivation for using Gallager's algorithm is precisely its distributed nature.

Clearly, the exact process through which information is disseminated in a network is of great importance. For the purposes of our comparative study, however, it is assumed that an appropriate mechanism is in place which provides all nodes with all information required to implement routing decisions.

1.2.3. Shortest Path vs. Gradient-Based Routing Algorithms.

Routing decisions are invariably based on routing tables stored at network nodes. Most datagram networks (e.g. ARPANET [4] and TYMNET [5]) employ *shortest path algorithms* to find the best paths between all *source-destination* pairs. Each link in the network is assigned a "cost" or "length", which is specified as a function of the link capacity and the link traffic rate (in packets/sec or bits/sec). Once all the links have been assigned such lengths, shortest paths may be computed for all source-destination pairs using several standard algorithms (to be discussed later). The nodal routing tables are created/updated according to the shortest paths determined.

On the other hand, there have been several approaches based on the formulation and solution of a constrained optimization problem [3],[6],[7],[8]. The resulting *gradient-based* algorithms attempt to optimize a network-wide performance measure by adjusting routing probabilities for all outgoing links at each node. In this approach, routing probabilities are periodically updated; with each such update, nodal routing tables are updated. Gallager's algorithm belongs to this class of datagram routing algorithms.

In the case of virtual circuit switching schemes, there have also been efforts to develop optimal decentralized routing policies [9],[10],[11]. But the CODEX algorithm is based exclusively on shortest paths. However , it should be noted that the "costs" assigned to links are different from those used in shortest path datagram routing.

Shortest path algorithms are generally conceptually simple; however they are heuristic, and do not guarantee optimality of performance. Moreover, they are known to be highly susceptible to instabilities manifested in the form of oscillatory behavior in the observed mean delay [12],[13].

Gradient-based algorithms are more sophisticated and can achieve optimal network-wide performance under stationary traffic conditions [3]. However, they too are susceptible to instabilities. Moreover, successful implementation of these algorithms relies on the development of efficient methods for estimating the required *gradient* information.

The gradient information involved in optimal routing algorithms consists of all individual mean link delay derivatives with respect to the corresponding link flow rates. One approach to estimating these derivatives is based on modeling each link as a queueing system (for example, the well-known M/M/1 queue) for which an *analytical* expression of the mean delay is available. Hence, the required derivatives can also be evaluated analytically, provided the link flow rate is known. The disadvantages of this approach are (a) it is hard to verify that a given link model corresponds to actual link behavior, and (b) since in practice link flow rates are not known in advance, they have to be estimated.

A second approach is to directly estimate the derivatives *on line* by using techniques such as Perturbation Analysis (PA) techniques [14],[15]. This method is not limited by most restrictive assumptions regarding the underlying network model. From an implementation standpoint, PA-based derivative estimators are marginally more complicated than the estimators otherwise required for link flow rates.

In the context of the comparative study of routing algorithms, this report examines the operational characteristics of both shortest-path-based and gradient-based routing algorithms under different traffic conditions. The issue of gradient estimation is further investigated in section 5.

1.3. Network Management in Multiclass Traffic Environments.

The current trend is to design networks with the ability to handle multiple traffic classes, such as data, voice, and video. Direct extensions of existing routing algorithms are not possible in such a setting, primarily due to the drastically different nature of some of these traffic classes. Of particular interest is the case of *real-time traffic*: a packet in this class is characterized by a deadline,

and the packet is considered lost or useless if it does not reach its destination by that deadline. Thus, instead of considering the *mean packet delay* as our performance measure, for real-time traffic we wish to minimize *the fraction of packets violating their deadlines.*

As we further discuss in section 6 of this report, there are several issues that need to be addressed in the context of this environment. It is particularly interesting that conventional packet handling approaches may no longer be desirable; it can be shown, for instance, that the First-In-First-Out (FIFO) service discipline is sometimes the worst possible policy for real-time traffic, with the Last-In-First-Out (LIFO) being the best.

1.4. Report Organization.

The contents of this report may be outlined as follows.

- **Section 2:** the basic operations of the three algorithms, the CODEX algorithm, Gallager's gradient-based algorithm, and the ARPANET algorithm, are described. This discussion is limited to those aspects which are important to our comparative study.
- **Section 3:** the experimental setup for our comparative study is described in detail. This includes the network configuration and the traffic environments considered, as well as some of the implementation details.
- **Section 4:** the results of the experiments for different parametric and operational variations are presented and discussed in detail.
- **Section 5:** the problem of gradient estimation in gradient-based algorithms is addressed. Here, we compare approcahes based on analytical approximations to on-line techniques relying on direct data collection and processing.
- **Section 6:** we consider the problem of routing in networks with real-time traffic, identify some key issues, and address several related problems.
- **Section 7:** we present the main conclusions of our study, and outline our ongoing work and some future research directions.

2. THE ALGORITHMS.

In this section, we briefly describe the basic operation of the three algorithms to be compared.

2.1. CODEX ROUTING ALGORITHM.

A detailed description of the CODEX routing algorithm along with its implementation methodology can be found in [1] and [2]. We consider here some of the important issues from a performance study point of view. The algorithm, in essence, has two processes running concurrently :

- Single path Datagram routing (DG) for network control traffic
- Virtual Circuit routing (VC) for network user traffic

2.1.1. The Datagram (DG) Routing Algorithm.

Datagram routing is meant for control packets and other system-related traffic. This generally constitutes a small fraction of the total traffic on the network.

The first function of this algorithm is the assignment of a "length" for every link in the network. The *length* of a link (i,j) is denoted by l_{ij}, and is defined to be the *mean delay* experienced by packets traversing the link. It is assumed that a link behaves like an M/M/1 queueing system (specifically: packets are placed in the link following a Poisson process, and packet lengths are treated as random variables characterized by an exponential distribution). The analytical expression for the mean packet delay (or "length") in such a queueing system is

$$l_{ij} = \frac{f_{ij}}{C_{ij} - f_{ij}} + f_{ij}\, d_{ij}$$

where C_{ij} is the link capacity (in bits per sec), f_{ij} is the mean flow rate (in bits per sec), and d_{ij} is the propagation delay on the link (in sec). Note that C_{ij} and d_{ij} are given link parameters, but f_{ij} must be estimated. This is done by defining an *observation interval* over which the average flow rate is measured. At the end of such an interval, node i obtains estimates of the flow rates on all its outgoing links. Once an estimate of f_{ij} is obtained, l_{ij} is computed from the expression above.

The second function performed, following link length assignment, is the determination of shortest paths (sequences of links from source to destination) for every pair of network nodes. Based on these paths, each node maintains a *routing table* where each entry corresponds to an outgoing link leading to a destination node on the shortest path. The table contains such entries for all possible destination nodes from the node concerned. At the end of an observation interval,

when new flow estimates are obtained, link lengths are updated, and hence new shortest paths are computed. These are used to update the corresponding entries in the nodal routing tables.

It should be noted that this datagram routing scheme relies on the existence of a broadcast-type algorithm [1], where every node broadcasts all relevant information, i.e. the estimates of flow rates on each of its outgoing links, to every other node along a minimum spanning tree rooted at itself. It is clear that the algorithm is asynchronous in nature due to the fact that it takes some time for information to travel from one node to another. As a result, some nodes will be updating their routing tables earlier than others. There is no mechanism in the CODEX algorithm to prevent the formation of loops caused by inconsistency in routing updates resulting from this asynchronism. Thus, the algorithm does not guarantee the delivery of control packets within bounded time delays. However, packets can be lost only in the event of a node/link failure.

2.1.2. The Virtual Circuit (VC) Routing Algorithm.

The CODEX virtual circuit routing algorithm, which is much more sophisticated than its datagram counterpart above, is used to route the user traffic on the network, which forms the bulk of the total traffic. As discussed earlier, a broadcast-type algorithm is assumed, through which every node informs every other node about any traffic changes on a link or any topological change due to node/link failure.

Traffic between a source-destination pair may be carried on multiple virtual circuits depending on the number of sessions set up between the two nodes. A *session* is set up whenever a node wishes to communicate with another node in the network, and is terminated after both nodes have finished their traffic transactions. Two paths are associated with each session, one for the outgoing traffic and the other for the incoming traffic, and they are completely independent. The algorithm is *centralized* in the sense that a single node, in particular the destination node for each VC, is responsible for all routing and rerouting decisions.

A question naturally arises as to how the destination node can make routing decisions at the time a VC is set up. This is not a problem because every session is associated with one forward and one reverse path: first, the destination node establishes the reverse path; then, the source node can establish the forward path. The details of this process can be found in [1].

The VC routing is based on a shortest path algorithm; however, the link length assignment process is completely different from the one used for datagram routing (see previous section). Moreover, the algorithm is executed at the destination node only at the time of setting up, and at time instants when a VC is selected for rerouting. The shortest paths in this case are computed only

for a particular source-destination pair under consideration; this is in contrast to the datagram algorithm where this is done for all node pairs.

Finally, the CODEX virtual circuit routing algorithm allows different classes of traffic to be routed differently according to assigned priorities. In fact, each VC is assigned a number reflecting the priority of the traffic class it carries. Flow rates on links are estimated separately for virtual circuits with different priority numbers. As shown below, these priority numbers are incorporated into the link lengths.

There are two aspects to the VC routing algorithm: the first establishes a VC upon initiation of a session, and the second allows rerouting of VC's to periodically occur in order to account for changes in the network operating conditions. These two aspects are considered in the subsections that follow.

2.1.2.1. Routing Policies for VC Setup.

In contrast to the DG routing algorithm, the VC routing algorithm distinguishes between a link "length" and a link "cost". Thus, the first function of this algorithm is the assignment of a "cost" for every link in the network. The *cost* of a link (i,j) is denoted by D_{ij}, and is defined to be a weighted sum of *mean delays* experienced by packets of different priorities traversing the link. As before, an M/M/1 queueing model is assumed for all links. The precise analytical expression used for D_{ij} is

$$D_{ij} = \left(\sum_{k=1}^{M} p_k F_{ij}^k \right) \left\{ \frac{1}{C_{ij} - \sum_{k=1}^{M} F_{ij}^k} + d_{ij} \right\}$$

where C_{ij} is the link capacity (in bits per sec), F_{ij}^k is the mean flow rate (in bits per sec) for a VC with priority k, d_{ij} is the propagation delay on the link (in sec), and p_k is a positive weighting factor representing the priority, with k=1,...,M. As in the DG routing algorithm, C_{ij} and d_{ij} are given link parameters, but F_{ij}^k must be estimated. This is again done by defining an *observation interval* over which the average flow rate is measured. At the end of such an interval, node i obtains estimates of the flow rates for all priorities on all its outgoing links. Once an estimate of F_{ij}^k is obtained for all k, D_{ij} is computed from the expression above.

The second function of the algorithm is to assign link "lengths" *whenever a new VC is introduced only*. This is done as follows. Suppose a new VC with estimated traffic rate Δ (in bits per sec) and priority k is to be set up. We assume that Δ is available to the destination node. Then, the *length* of link (i,j) is the estimated incremental link cost given by

$$l_{ij} = D_{ij}(F_{ij}^1, \ldots\ldots, F_{ij}^k + \Delta, \ldots\ldots, F_{ij}^M) - D_{ij}(F_{ij}^1, \ldots\ldots, F_{ij}^M)$$

The first term in the above expression corresponds to the *projected* link cost (as a result of the new VC) and the second term to the *current* cost of the link.

The final function upon setup is performed by the destination node, which computes a shortest path from the source to itself using these link lengths. The computation is done by Pape's version of the Bellman-Moore algorithm [16]; this is quite efficient as well as easy to implement. An outline of the algorithm is presented in Appendix I, with an example to illustrate its operation. A coded version in the form of a subroutine (SPATH) can be found in Appendix II. The complete path resulting from this algorithm is stored at the destination and is sent with a control packet to the source node while establishing the path. As this packet propagates, the intermediate nodes become aware of their upstream and downstream neighbors and update their routing tables accordingly.

2.1.2.2. Rerouting Policies for Active VC's.

To make the routing adaptive to changing network conditions, it is essential to have some mechanism for periodically rerouting some or all VC's. Rerouting may occur due to traffic changes or topology changes (link/node failure). In either case the procedure is similar.

As already mentioned, the CODEX algorithm uses an observation interval over which link traffic rates are estimated. At the end of every observation interval, a fraction of the active VC's is chosen for rerouting. The selection is done randomly, by simply assigning each VC a certain probability; we shall refer to it as the *Rerouting Probability*.

If a VC is selected for rerouting, it is viewed as a new VC. Thus, the procedure described in the previous subsection is invoked: first, link costs are computed for all links based on the traffic conditions that would result if this VC were removed; then, link lengths are evaluated by treating the VC as a newly arriving one. It is clear that this procedure requires an estimate of the traffic rate on the virtual circuit, in addition to the latest link flow updates (a link flow typically consists of traffic due to several VC's using this link). The VC traffic rate is obtained by the destination node by monitoring the traffic flow on the virtual circuit over its entire active period (from the time it was established to the rerouting instant).

As an example, consider a candidate VC with an estimated flow rate Δ and priority k. So, if link (i,j) is being currently used by this VC, its length after removal of the VC would be,

$$l_{ij} = D_{ij}(F_{ij}^1, \ldots\ldots, F_{ij}^k, \ldots\ldots, F_{ij}^M) - D_{ij}(F_{ij}^1, \ldots\ldots, F_{ij}^k - \Delta, \ldots\ldots, F_{ij}^M)$$

On the other hand, if link (i,j) is not currently being used by the VC under consideration, its link length is

$$l_{ij} = D_{ij}(F_{ij}^1, \ldots\ldots, F_{ij}^k + \Delta, \ldots\ldots, F_{ij}^M) - D_{ij}(F_{ij}^1, \ldots\ldots, F_{ij}^k, \ldots\ldots, F_{ij}^M)$$

Using these lengths, a new shortest path is computed for the VC. If the resulting cost is lower than the current one, then the VC is rerouted along the new shortest path. Details of this implementation can be found in [2].

2.1.3. Algorithm Parameters.

Based on the discussion above, it is clear that there are two critical aspects which affect this algorithm's performance:

- the choice of observation interval: too short results in poor traffic rate estimates, too long delays possible performance improvements
- the choice of rerouting probabilities for VC's: too small limits adaptivity, too large may cause overloading of some links and lead to oscillations in performance

There is generally no well-defined way of choosing these parameters; the range of suitable values may vary from network to network.

2.2. GALLAGER'S ALGORITHM.

In this section, we briefly describe the operation and main features of Gallager's distributed routing algorithm, which is designed for datagram networks. A detailed account of the algorithm can be found in [3].

This algorithm is applied independently at each node to update the routing tables according to the latest information acquired from neighboring nodes only. This is in contrast to the broadcasting required in the CODEX algorithm. The algorithm has been shown to minimize the mean packet delay under stationary traffic conditions [3], and is expected to perform well under *quasistatic*

conditions; quasistatic conditions are present when traffic rates change sufficiently slowly to enable the algorithm to adapt.

The critical steps involved in Gallager's algorithm may be summarized as follows (details are omitted, but may be found in [3]):

1. Over an *observation interval.*, each node estimates:
   - the traffic rate on each outgoing link
   - the sensitivity (*gradient*) of the mean packet delay on each outgoing link with respect to the traffic rate through that link - this is also referred to as the link *marginal delay*
2. At the end of an observation interval, each node:
   - identifies the "best" outgoing link, i.e. the link with the smallest marginal delay (gradient estimate)
   - decreases the traffic rate into all other outgoing links
   - adds the total traffic rate removed from these links into the "best" link

   where the amount of traffic rate adjustments is regulated by a *step size* parameter
3. The process continues until equilibrium is reached, i.e. all ougoing links have the same marginal delays.

2.2.1. Algorithm Parameters.

In theory, Gallager's algorithm achieves the minimum possible network-wide mean packet delay, provided step sizes are chosen sufficiently small. In practice, there are three critical aspects which determine the algorithm's performance:

- the choice of observation interval: too short results in poor estimates, too long delays improvements
- the choice of step size: too small results in slow convergence, too large may lead to instabilities
- the method of gradient estimation used

Clearly, the first two choices are the analogs of the observation interval and rerouting probabilities required in the CODEX algorithm. The issue of gradient estimation is readily resolved if one is willing to assume that links behave like M/M/1 queueing systems, as is done in the CODEX algorithm; then, an analytical expression for a link's marginal delay may be used. However, one of

our findings is that the behavior of the algorithm is much more sensitive to estimate errors based on this assumption (see section 5 and [19]), and that on-line gradient estimation techniques may be preferable.

2.3. ARPANET ALGORITHM.

The original ARPANET algorithm is one of the oldest routing algorithms to be implemented on a real-life network. It is an adaptive and distributed type of algorithm designed for datagram networks. Its success generated a great deal of research in this area. However, in the course of time, some of its major flaws - such as high susceptibility to oscillations - became apparent, and this led to its eventual replacement by a new and improved version of the algorithm. In this report, we shall be concerned with this new version only, and discuss some of the features relevant from our performance study point of view. A detailed account can be found in [4].

There is an important distinction between the datagram routing performed by the ARPANET algorithm and that used in Gallager's algorithm. The main difference lies in the number of paths available between any source-destination pair. Whereas in Gallager's algorithm a packet between a node pair may take any of several available paths depending on the routing assignments at each node to the outgoing links, in the ARPANET algorithm it is constrained to the single best path available between this pair of nodes. Therefore, some times this type of an algorithm is referred to as a *single path* datagram.

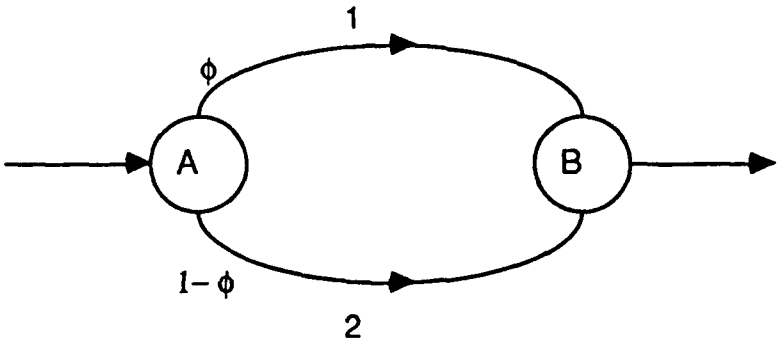


Figure 1: *A Simple 2-link Network*

For example, in the simple 2-link network shown in **Figure 1**, Gallager's algorithm would route a packet from A on link 1 with probability ϕ and on link 2 with probability $(1-\phi)$. But the ARPANET algorithm would route the packet on either link 1 or link 2 depending on which has a

lower mean delay based on the most recent estimates. Subsequent packets would also be routed on the same link until the next update instant, when the other link may have a smaller mean delay.

The path selection is done by employing a shortest path algorithm with "lengths" assigned to links (similar to the lengths defined in section 2.1.1). Some of the important steps in the operation of the basic ARPANET algorithm are outlined below.

1. Each node estimates the mean packet delay on each outgoing link over an *observation interval*. Individual packet delays are measured by time stamping the packets. This delay includes the queueing delay, as well as the transmission and propagation delay on the links.

2. If the current mean delay differs from the last update by more than a certain *threshold*, only then is it passed on to the other nodes in the network.

3. Mean delay updates are transmitted to all other nodes by employing a form of flooding (for details, see [4]).

4. At the end of an observation interval, each node uses a shortest path algorithm to find the minimum delay path between this node and all other nodes. ARPANET employs a version of Dijkstra's algorithm [17] (some important modifications have been made so that changes in topology or traffic conditions require only some incremental calculation rather than a complete recalculation of all the shortest paths). Based on the new shortest paths, the node updates its routing table accordingly.

It is worthwhile to note at this point that the basic philosophy of operation of the ARPANET algorithm makes it prone to oscillations in the observed performance. The problem of oscillations, though not as acute as in the original algorithm, does exist in the new algorithm. The policy of diverting *all* the flows to a currently less congested region of the network without any regulatory mechanism (like the step size in Gallager's algorithm and rerouting probabilities in the CODEX algorithm) causes this area to be heavily congested at the next update time. Conversely, the region which is currently congested becomes lightly loaded at the next update, thus making it attractive for routing. This shifting back and forth of traffic flows gives rise to oscillations. This is very apparent in the example of **Figure 1** where all the traffic would be continuously switching between link 1 and link 2. These oscillations may be damped by adding a *bias factor* to the link length which may reduce the sensitivity of the algorithm to network congestion.

2.3.1. Algorithm Parameters.

As already mentioned, the ARPANET has no equivalent to the step size parameter in Gallager's algorithm and the rerouting probabilities in the CODEX algorithm, in order to regulate the tendency for instabilities. The main parameter here is the observation interval, which is set to 10 seconds. In

addition, a threshold of 64 msec is used to trigger reporting a new mean delay estimate in step 2 above. This threshold decreases at the rate of 12.8 msec with every observation interval (10 sec). Thus, one update is guaranteed every 6 observation intervals. After every update reporting, the threshold is set back to 64 msec. This mechanism ensures reporting small changes slowly and large changes more rapidly.

3. EXPERIMENTAL SETUP

In this section the experimental setup used to carry out the simulations of the chosen routing algorithms is described. Most of the experiments were conducted under similar network conditions as those presented in [18] and [19], where Gallager's distributed routing algorithm employing PA techniques was considered. This was done to enable us to compare results in that work with some of our results here. However, some additional experiments investigating the effect of network size on the mean delay performance of the routing algorithms were also performed using a different setup; this will be discussed in section 4.5.

3.1. Network Configuration.

A 6-node network, similar to that used in [19], was used for our experimentation. This is shown in **Figure 2**, where link capacities are shown in Kbps. It can be seen that the link capacities were chosen to vary in the range from 9.6 Kbps to 72 Kbps. Also note that some of the links are not bidirectional, and some bidirectional links are not fully duplex.

3.2. Traffic Arrival Characteristics.

We consider a traffic environment with 19 traffic-generating source-destination pairs. The packet interarrival time distributions are arbitrarily chosen from a set of exponential and uniform distributions. A list of these distributions is presented in **Table 1**, where $EX(\alpha)$ denotes an exponential distribution with mean α msec and $UN(\alpha,\beta)$ denotes a uniform distribution over the interval $[\alpha,\beta]$ msec. The same traffic interarrival distributions were used in the performance study reported in [19] as well.

In the case of datagram algorithms, each of the packets is treated independently. But from a virtual circuit routing point of view, there are 19 virtual circuits corresponding to the 19 traffic generating source-destination pairs. Alhough in the CODEX algorithm a communication session between two nodes is expected to have two associated virtual circuits (one for each direction), in our experiments each session was limited to a single virtual circuit (i.e. the existence of a VC between node A and node B does not imply that there is a VC from B to A). This, however, causes no loss of generality for our performance study.

3.3. Packet Characteristics.

Two types of packets generated: 80% are long (1 Kbit) and 20% are short (128 bits). Long packets may be viewed as user data packets, while short ones represent system or control traffic. This distinction does not affect the handling of packets in the datagram routing setting (ARPANET and Gallager's Algorithms). However, as already described in section 2.1, the CODEX algorithm routes short packets on a datagram basis and long packets through virtual circuits.

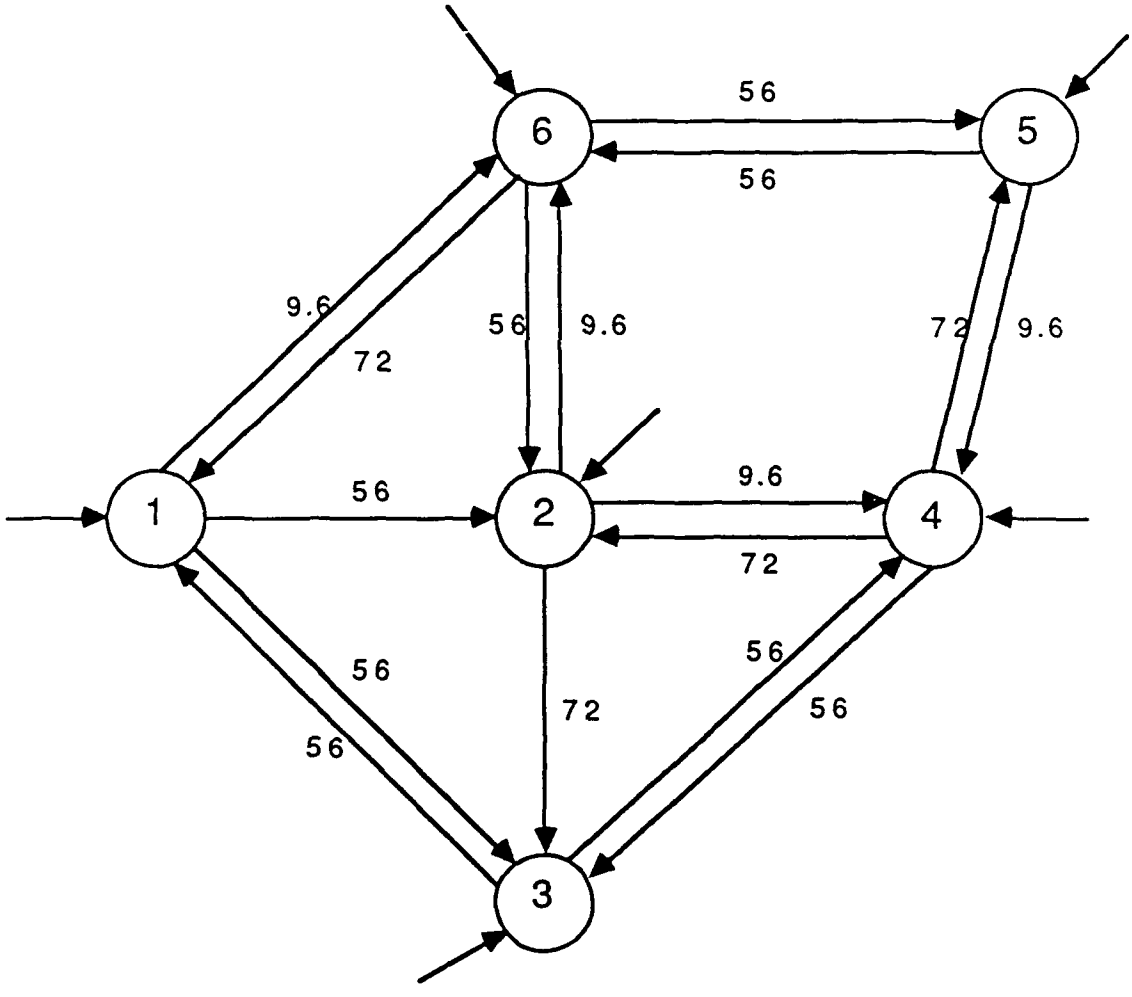


Figure 2: *Network Used for Performance Study*

SOURCE-DESTINATION PAIR	INTERARRIVAL TIME DISTRIBUTION
1-3	EX(60)
1-4	EX(200)
1-5	EX(300)
1-6	EX(300)
2-1	UN(80,100)
2-5	UN(80,100)
2-6	UN(60,80)
3-1	EX(70)
3-2	EX(300)
3-5	EX(300)
4-1	EX(200)
4-3	EX(300)
4-6	EX(90)
5-1	EX(90)
5-2	EX(90)
5-3	EX(90)
6-1	UN(60,80)
6-2	UN(80,100)
6-4	EX(70)

TABLE 1. *List of External Traffic Distributions Between Node Pairs*

3.4. Traffic Generation.

The traffic generation process may be considered in several ways while experimenting on routing algorithms. The simplest way is to model traffic arriving at the network as a *stationary* stochastic process. This means that all traffic streams are switched ON at t = 0, and remain active for the entire simulation run length. This type of traffic generation has been used in performance studies such as [19]. It allows one to study the *transient* (convergence properties, ability to clear congestion) and *steady state* behavior of routing algorithms, but it does pose certain practical problems, described below:

1. If the initial routing assignments are not properly chosen, some of the links may be assigned packet flows beyond their capacities, thus making them unstable. Under these conditions, most

of the theory based on modeling links as stable queueing systems becomes invalid, and the behavior of an algorithm may be unpredictable. This issue is addressed in [3], where it is assumed that a flow control mechanism is present, which ensures that no link flow ever exceeds the corresponding link capacity.

2. The method of stationary traffic inputs is not suitable for a performane study of the CODEX algorithm. Being a VC-based algorithm, its slow rerouting mechanism is not designed to clear heavy traffic congestions that would arise due to the simultaneous activation of several virtual circuits upon initialization. This congestion buildup may in fact take an extraordinarily long time to be cleared, so that steady state conditions can be established. Therefore, this would be an unrealistic model of the actual CODEX algorithm operation, and is likely to lead to wrong conclusions (for example: it may appear that large value of rerouting probabilities improve the adaptivity of the CODEX algorithm; as we shall see in the next section, this is incorrect). In contrast, datagram routing algorithms such as Gallager's assume stationary traffic inputs over reasonably long periods, and may not be able to adapt to very frequent variations in network traffic resulting from switching virtual circuits ON and OFF.

To accommodate the above considerations, we adopted the following scheme:

> VC's are gradually switched ON at time instants offeset by random intervals, assumed to be exponentially distributed. Once a VC is ON, it remains active for the entire run length. From a datagram standpoint, the arrival of VC's constitutes random traffic surges, and the transient response of the datagram algorithms to these surges can be observed.

The only transient phenomena in this traffic generation model are the arrivals of new VC's. To introduce more fluctuations into the traffic input process, in some experiments an active VC is turned OFF after generating a geometrically distributed number of packets. Upon termination, a VC may also be reactivated after an exponentially distributed time interval.

Some other implementation-related points worth mentioning are:

- For simplicity, the priorities of all virtual circuits have been assumed to be the same.
- Even though in practice the ARPANET uses a modified version of Dijkstra's shortest path algorithm, and the CODEX algorithm uses a modified version of Moore's shortest path algorithm, we have used in our experiments only one implementation of Moore's algorithm (see APPENDIX I). Since we assume no overhead due to processing

(estimation, shortest path computations, routing table updating) or to communication of control packets, this simplification does not affect our comparative performance study.

3.5. Algorithm Software.

The code for our implementation of the CODEX and ARPANET algorithms can be found in APPENDIX II and III respectively. The code used for Gallager's algorithm was taken from [19] and suitably modified for our experimental requirements. All software was implemented in the SIMAN simulation language using FORTRAN discrete event models. Depending on the parameters and operating environments of interest, several variations of the software in these Appendices were actually implemented in the course of this work.

4. COMPARATIVE PERFORMANCE RESULTS.

In this section, we present the results of our comparative performance study based on the experimental setup described in section 3. There are two important network performance measures to consider in comparing routing algorithms.The first is the network *throughput*, which is a measure of the quantity of service, and the second is the *mean delay* experienced by packets, which is a measure of the quality of service. The latter has been commonly used as a basis for comparing routing algorithms [19],[20]. The mean delay can be defined as the average amount of time a packet takes from the time of its arrival to the network until it reaches its destination node. This delay consists of queueing delays at intermediate nodes, as well as transmission and propagation delays on the links traversed.

The performance comparison of the three algorithms (CODEX, Gallager's, and ARPANET) reported here is on the basis of the mean delay performance measure. For comparison purposes, each algorithm is applied to the same network under similar traffic conditions.

In section 2, the critical parameters associated with each algorithm were identified. A major part of our work has focused on investigating the effect of these parameters on the behavior of the algorithms. We reiterate here what these parameters are:

- an *observation interval* is used by all three algorithms to collect information from the network before making routing adjustments
- a *traffic adjustment* parameter is used in the CODEX and Gallager's algorithms at the end of every observation interval (no such parameter is used in the ARPANET). Traffic adjustments are accomplished in the CODEX algorithm through the VC rerouting probabilities, and in Gallager's algorithm through the step size, a scaling factor for transfering packet flows from one link to another.

Our results in this section are organized as follows.

- In section 4.1, we examine the effect of the observation interval on the mean delay obtained when each of the three algorithms is used.
- In section 4.2, we study the effect of the traffic adjustment parameters employed in the CODEX and Gallager's algorithms. The adaptivity of these two algorithms to varying traffic conditions are also examined in the context of their mean delay behavior.
- In section 4.3, we tune the parameters of the CODEX and Gallager's algorithms as well as possible, and compare their mean delay performances under these conditions.
- In section 4.4, we study the mean delay performance of the standard ARPANET algorithm, and compare it to that of the CODEX and Gallager's algorithms.

- In section 4.5, we investigate the effect of packet resequencing, manifested in datagram networks as additional end-to-end delay. This allows us to make truely fair comparisons between Gallager's algorithm and the CODEX algorithm.
- In section 4.6, we study the effect of network size on the behavior of the algorithms.
- In section 4.7, we summarize our results and conclusions regarding the comparative advantages and disadvantages of the three routing algorithms.

Apart from the algorithm parameters, the performance also depends on the implementation methodology of some critical operations such as the gradient estimation process required in Gallager's algorithm. This issue will be separately examined in section 5.

4.1. Effect of Observation Interval on Mean Delay Performance.

As already described in section 2, the operation of all three algorithms requires some sort of feedback from the network. The CODEX algorithm requires estimates of individual link flows and of traffic rates on each of the active virtual circuits. Gallager's algorithm requires estimates of the sensitivities of individual link delays with respect to corresponding flow rates. The ARPANET algorithm measures average mean delays over all links, which are then used in shortest path computations. In all cases, estimation of the expectation of a random variable is involved. This estimation is normally done over an interval of time which we refer to as the *observation interval.*

Assuming stationary distributions for the random variables (typically, mean link delays) whose means have to be estimated, a large number of samples (infinite in the limit) is required for the estimation to be sufficiently accurate (strictly speaking: to guarantee almost sure convergence of the sample mean to the true expectation). This implies that the observation interval has to be sufficiently long. In practice, however, the traffic input may not be stationary. Under these conditions, the speed of response of a routing algorithm as traffic conditions change is important, and the algorithm may respond faster if a short observation interval is used. The speed of response can be measured in terms of the time the algorithm takes to converge to its new steady state value from the moment a change in traffic occurs. A proper choice of observation interval length must take this factor into consideration.

Figure 3 shows the effect of the observation interval on the mean delay performance of the CODEX algorithm. In this experiment all virtual circuits were initially activated and kept active for the entire simulation run duration. A rerouting probability of 0.6 is considered here, which is large enough to bring the network out of the initial congestion in approximately 10 iterations. The observation interval was then varied from 6 sec, which corresponds to successful departure of

approximately 1000 packets over the network, to 56 sec, corresponding to 10000 packet departures approximately. The mean packet delay in msecs is plotted over time in secs. We can see that the network starts with an extremely large mean delay due to the initial congestion, and eventually clears out. This transient behavior is not of critical importance here, since in practice rarely would a virtual circuit network see this much traffic applied at the same instant.

The behavior of the algorithm after clearing the initial congestion is of importance to our study. It can be seen that a small observation interval of 6 sec produces a highly oscillatory mean delay profile. This is attributed to the large amount of noise in the estimation of the link flows, resulting in erroneous reroutings, as well as the large variance of the mean network delay itself. Clearly, as the observation interval is increased to 30 seconds and then to 56 seconds, the oscillations subside considerably, although the speed of convergence is relatively slow.

The effect of the observation interval on the mean delay performance of the ARPANET algorithm is presented in **Figure 4**. Contrary to our observations for the CODEX algorithm above, here the oscillations decrease with decreasing observation interval. This behavior is explained by the fact that the inherent oscillations in the ARPANET are so significant that the effect of any mean delay estimation errors becomes negligible in comparison. The ARPANET algorithm is heuristic in nature and there is no guarantee that it can ever find a fair allocation of traffic over all the links in the network. This problem is exacerbated under moderate to high load conditions; in such circumstances there would always be some links which are heavily congested while some others are lightly loaded. The switching of traffic back and forth between these two sets of links gives rise to the oscillations observed. By taking a larger observation interval we only allow the congestion to build up before switching to the other set of links where again the congestion builds up over the interval. Thus, we get oscillations of higher amplitude as compared to the case of a smaller observation interval. However, this problem is not as serious under very light loads, which is typically the situation in the ARPA network.

The effect of the observation interval on the performance of Gallager's algorithm was examined in [19]. The observations there were similar to those applicable to the CODEX algorithm, i.e. a longer observation interval produces mean delay results with smaller variance. For the sake of completeness, some results have been reproduced in **Figure 5**. The observation interval corresponding to 10000 packets/iteration is equivalent to 56 sec approximately. Other observation intervals can similarly be obtained by considering the fact that they would be proportional to the number of packets/iteration. A reasonably small step size of 0.01 E -05 was used to ensure convergence of the algorithm in this case.

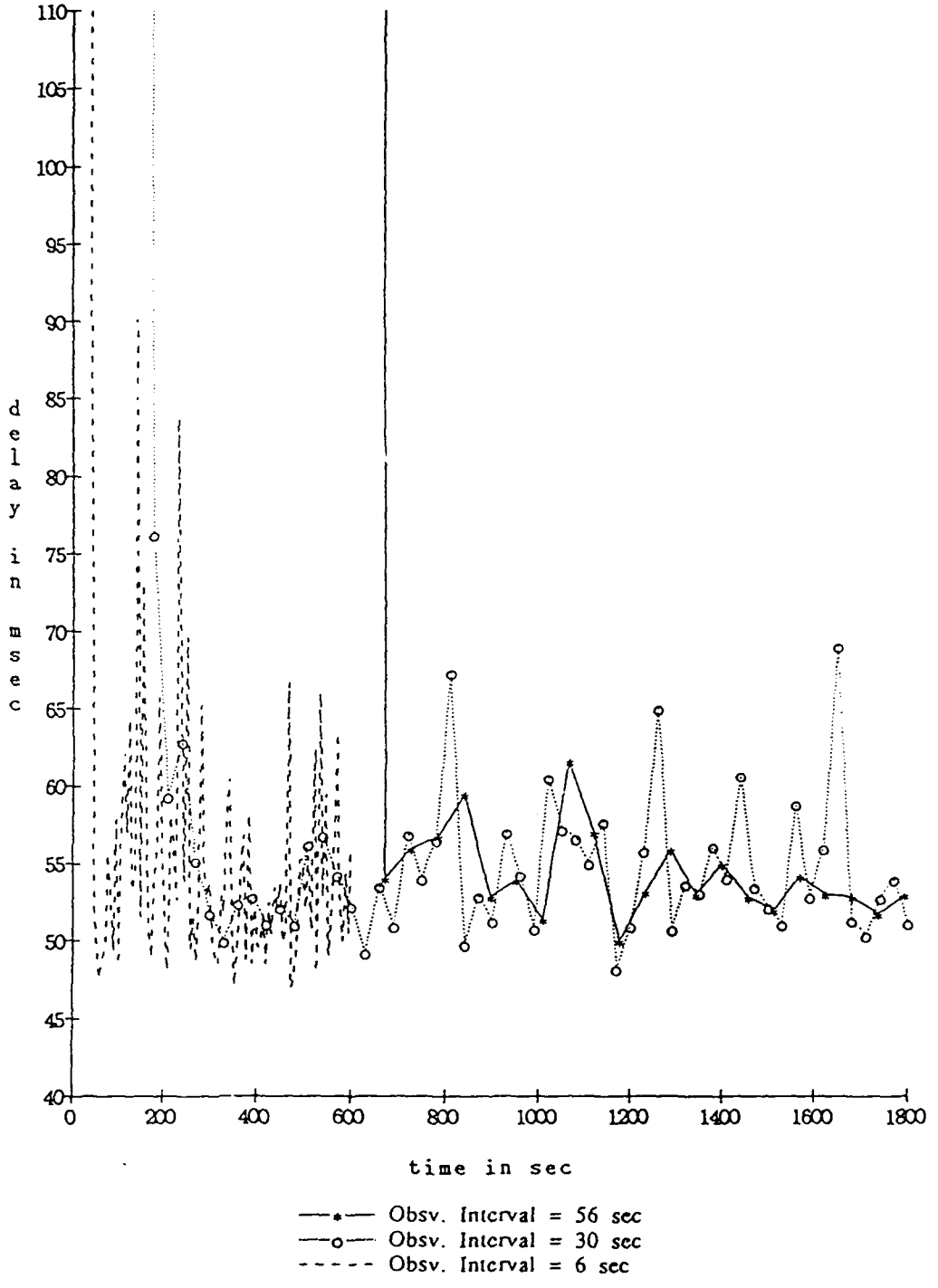


Figure 3 : **Effect of the Observation Interval on the Mean Delay Performance of the CODEX Algorithm**
(6 Node Network, All Traffic Streams Activated at the beginning, Reroute Prob. = 0.6)

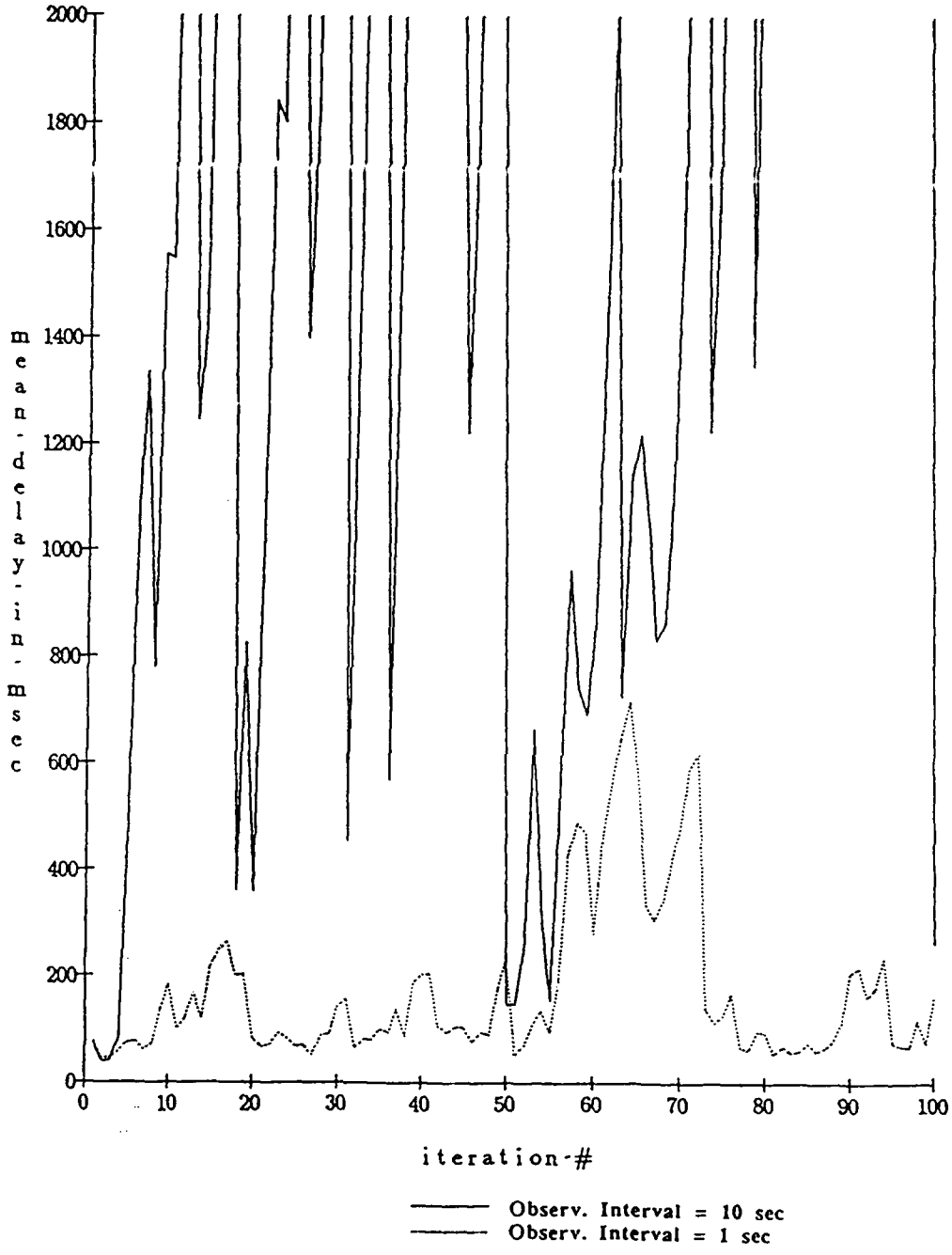


Figure 4 : Effect of the Observation Interval on the Mean Delay Performance of the ARPANET Algorithm
(6 Node Network, Traffic Streams Activated gradually)

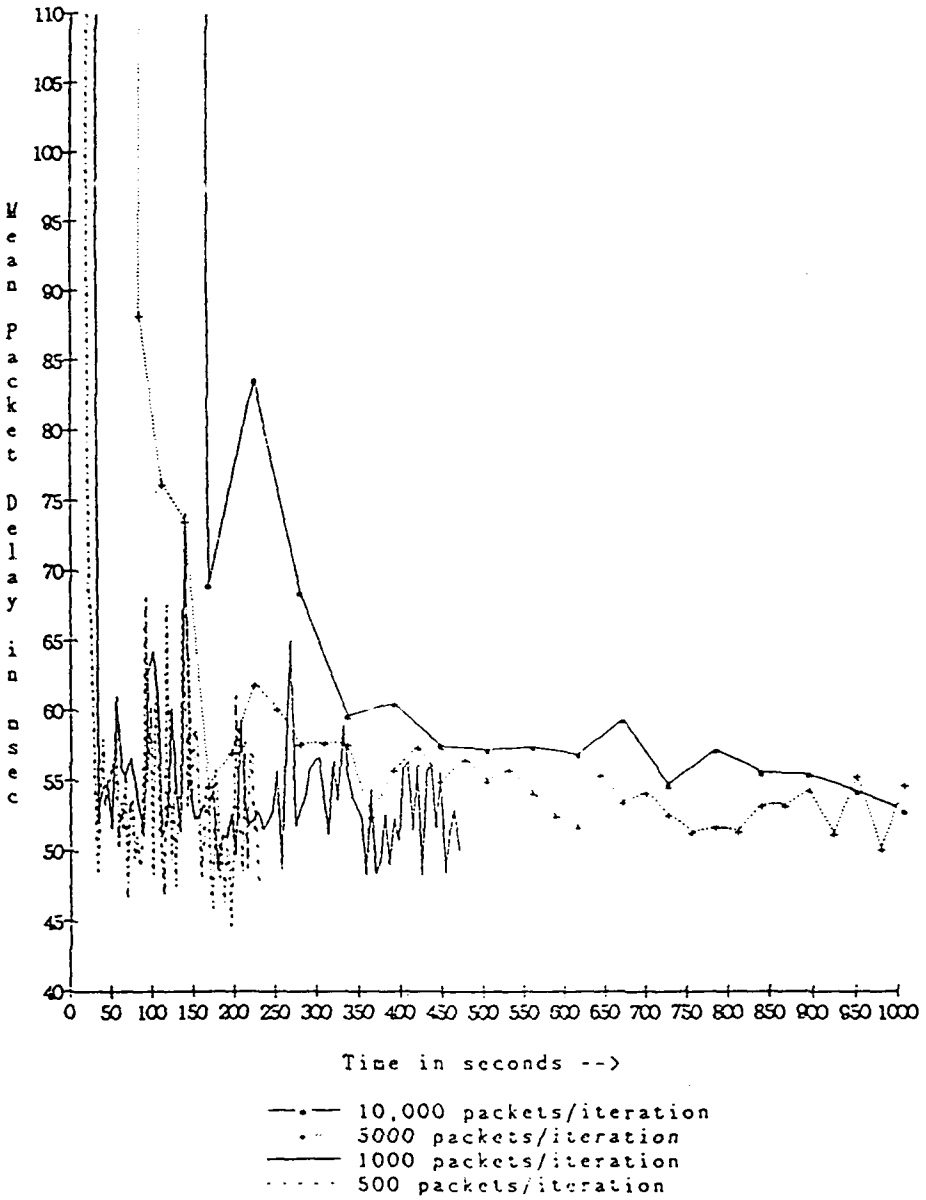


Figure 5 : Effect of the Observation Interval on the Mean Delay Performance of Gallager's Algorithm
(6 Node Network, Traffic Streams Activated at the beginning
step size = 0.01 E - 05)

4.2. Effect of Traffic Adjustment Parameters on Mean Delay.

As discussed in Section 2, the CODEX and Gallager's routing algorithms incorporate regulatory mechanisms to control the rate at which traffic adjustments are done at each update instant. This allows them to adapt to traffic changes and to limit undesirable oscillations. Unfortunately, ARPANET does not have any such control over flow adjustments and is thus highly susceptible to oscillations.

The traffic adjustment mechanism appears in the form of VC *rerouting probabilities* in the CODEX algorithm, and as a *step size* in Gallager's algorithm, which scales the amount of outgoing traffic flow at a node transfered from "bad" links to a "good" link.

- in the CODEX algorithms, the rerouting probability determines the expected number of virtual circuits which may be considered for rerouting at the end of an observation interval. As was pointed out earlier, a large rerouting probability causes too many VC's to be rerouted simultaneously leading to oscillations, while a small value reduces the adaptivity of the algorithm to traffic changes.
- In Gallager's algorithm, a small value of the step size causes a slow but steady convergence to the optimal mean delay, whereas a large value causes the algorithm to oscillate about the optimum.

The choice of these parameters is therefore very important from an implementation point of view. This choice may also depend on other factors, such as the network size and loading conditions. Gallager's algorithm was shown [3] to require an extremely small value of step size to guarantee convergence; but in practice, one has to choose a suitable value through experimentation on the specific network under consideration. Similarly, in the case of the CODEX algorithm, one must resort to such experimental means, since there are no theoretical guidelines for the choice of rerouting probabilities.

The experimental results depicting the effect of VC rerouting probabilities on the mean packet delay in the CODEX algorithm are presented in **Figure 6.** In this experiment, we activate the virtual circuits gradually, each with an exponentially distributed offset period. Once activated, a virtual circuit remains active for the entire duration of simulation. This was done to avoid the initial congestion which may be irrecoverable in the case of a small rerouting probability. The observation interval was chosen to be 56 sec, which is a reasonably long period.

It is interesting to note that even when the traffic is stationary after a sufficiently long period from the start of the simulation, one needs a non-zero value of rerouting probability to obtain a relatively oscillation-free mean delay. The oscillatory behavior for the case of zero rerouting probability can be attributed to the variance of the estimated mean delay, which forces the

determination of shortest paths to constantly change. This can be improved either by using a much larger observation interval or by choosing a suitable value of rerouting probability. This is illustrated by the choice of a rerouting probability of 0.4 in **Figure 6.** From an implementation stanpoint, the latter is more practical.

However, a very large value of rerouting probability results in extremely large oscillations, as seen for the value of 0.8 in **Figure 6**. In fact, over a certain period of time, the mean delay was too large to be plotted on the same scale. These large oscillations are due to the uncoordinated rerouting of too many virtual circuits at the same time.

Similar experiments were performed for different values of step sizes for Gallager's algorithm [19]. The results are shown in **Figure 7**. The same trend was observed as in the case of rerouting probabilities for the CODEX algorithm, i.e. a small step size results in a slow but smooth convergence to the optimal mean delay value, whereas a large value causes rapid convergence at the expense of oscillations at steady state. Moreover, an extremely small value of the step size is unable to bring the network out of an initial congestion.

4.3. Comparison of Mean Delay Performance of the CODEX and Gallager's Algorithms with Tuned Parameters.

In this section, we compare the transient and steady state mean delay performances of the CODEX and Gallager's algorithms under suitably tuned parameters. The experiments are conducted for two different traffic environments :

1. *Long-run stationary traffic* environment: the traffic generating source-destination pairs are gradually activated with exponentially distributed random offset periods. Thus, the traffic is initially not stationary, but becomes stationary after a sufficiently long period of time.
2. *Quasistatic traffic* environment: the traffic generating source-destination pairs are activated and deactivated at random instants of time as described in section 3.

For both algorithms, a reasonably large observation interval of 56 sec was chosen. The rerouting probability and step size were tuned (by trial and error) to obtain the best possible achievable performance in terms of the smoothness of the mean delay convergence profile. Accordingly, a rerouting probability of 0.6 in the CODEX algorithm, and a step size of 0.01 E - 05 in Gallager's algorithm were chosen respectively.

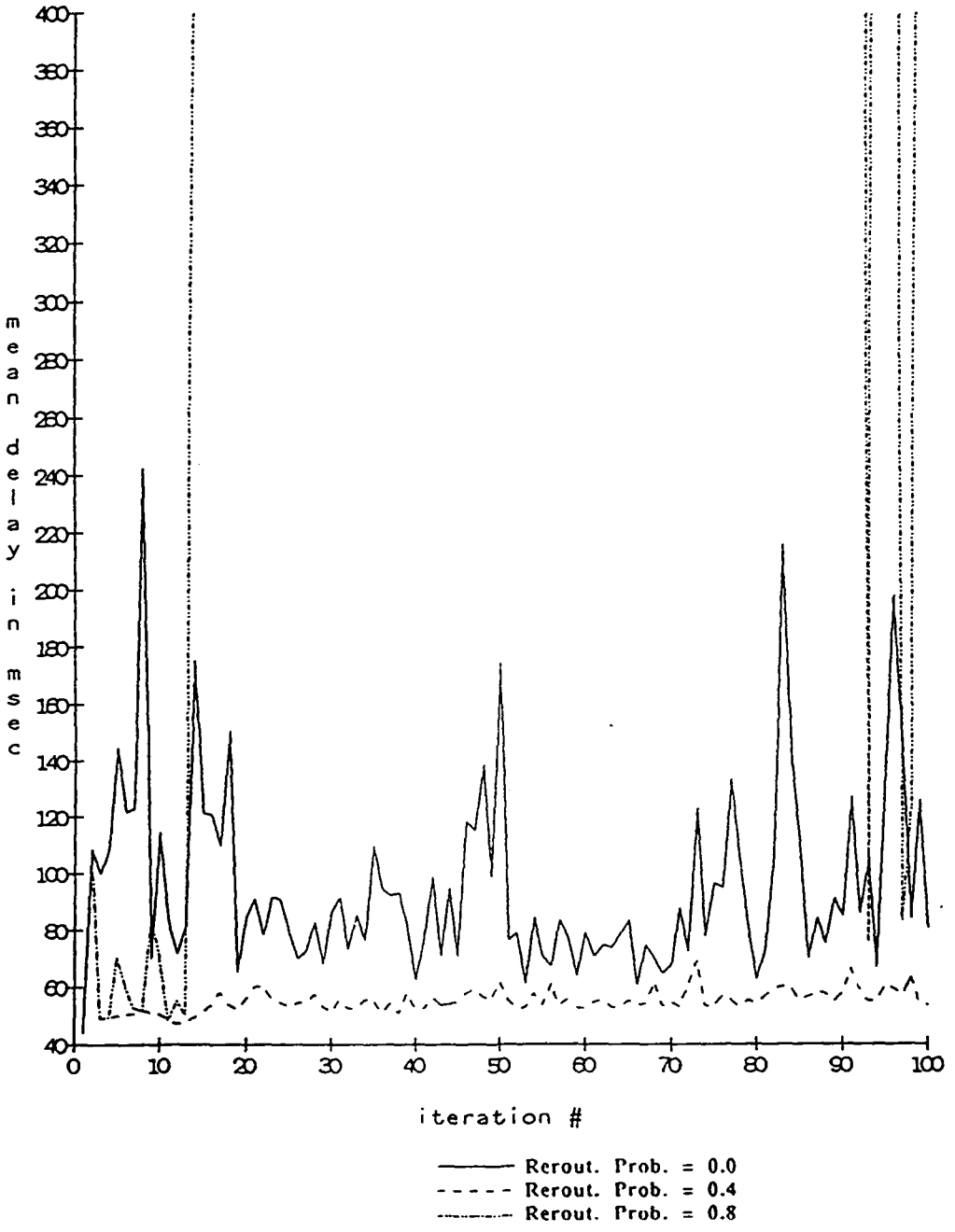


Figure 6 : **Effect of Rerouting Probabilities on the Mean Delay Performance of the CODEX Algorithm**
(6 Node Network, Traffic Streams Activated gradually, Observ. Interval = 56 sec)

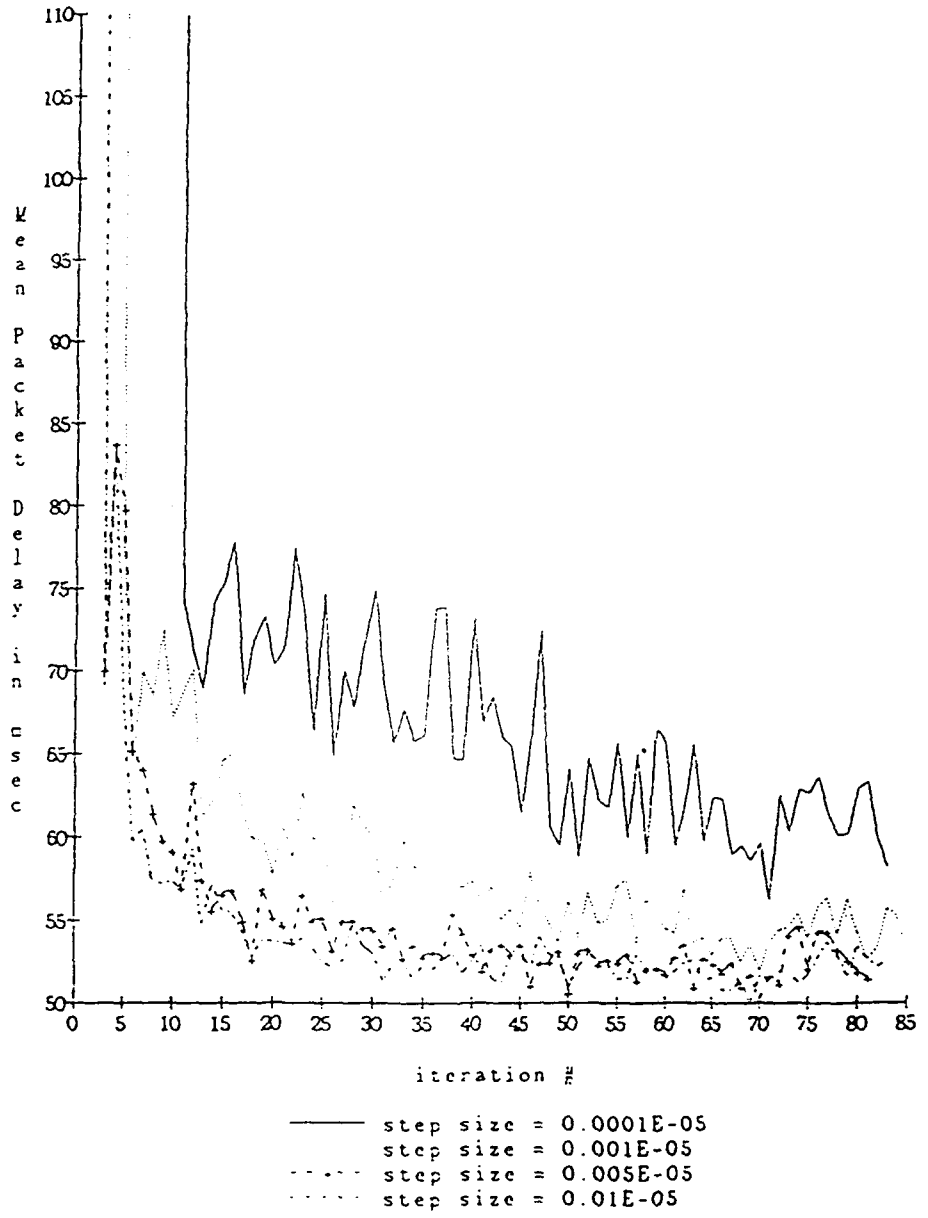


Figure 7 : Effect of Step Sizes on the Mean Delay Performance of Gallager's Algorithm
(6 Node Network, All Traffic Streams Activated at the beginning,
Observ. Interval = 10000 packets (approx. 56 sec))

4.3.1. Long-run Stationary Traffic Environment.

The results for this case are presented in **Figure 8**. The mean network delays for the algorithms are plotted against time measured as the number of iterations implemented (each iteration corresponding to a 56 sec observation interval).

4.3.1.1. Transient Behavior.

It can be seen in **Figure 8** that in the first few iterations, Gallager's algorithm tends to be more oscillatory compared to the CODEX algorithm (for iterations less than 10), and slower to converge (up to iteration 20). This is because the CODEX algorithm is better suited to handle new source-destination sessions by finding the paths of least incremental delays for the new virtual circuits. On the other hand, Gallager's algorithm adapts slowly to the increased traffic due to the small step size typically used. Clearly, one could tune the step size *for a transient period only*, so as to achieve a behavior similar to that of the CODEX algorithm.

4.3.1.2. Steady State Behavior.

Once the initial transients have died out and stationary traffic conditions have been established, Gallager's algorithm converges to the optimal mean delay value, while the CODEX algorithm displays continuously oscillating behavior. One can see that the nominal mean delay is nearly 5% higher in the case of the CODEX algorithm.

This difference in mean delay at steady state is due to the fact that Gallager's algorithm is designed to actually converge to the *optimal* mean delay under stationary traffic inputs (which is the case here after all the traffic generating pairs have been activated). In contrast, the CODEX algorithm is heuristic in nature and does not guarantee optimal performance.

The difference in the degree of oscillations for the two algorithms is attributed to the fundamental difference in the method of traffic adjustments. At any link, Gallager's algorithm can shift a fraction of the current traffic on the link taking *any desired value* between 0% to 100% (depending on the estimated derivatives and the step size); the CODEX algorithm is only allowed to do so *in units of virtual circuits*. Therefore, the amount of traffic shifted depends on the traffic rates of the virtual circuits being rerouted. If the cumulative traffic rate of the VC's being rerouted is significant, their shifting can give rise to considerable oscillations.

4.3.1.3. Further observations on Steady-State Oscillations in the CODEX Algorithm.

The fact that rerouting of virtual circuits is indeed the reason for the observed oscillatory behavior of the CODEX algorithm is corroborated in the results of the following experiments. Now, instead

of routing short packets as datagrams and long packets as virtual circuits, all the packets are routed exclusively through virtual circuits. This effectively *increases* the traffic rates of individual virtual circuits. It can be seen from **Figure 9** that in this case the oscillations are substantially larger (since there are larger traffic rates on virtual circuits).

Taking an opposite approach, each of the virtual circuits is divided into 20 parts, and each component VC is routed independently. This effectively *reduces* the traffic rate of each virtual circuit by a factor of 20, and increases their total number by a factor of 20. As expected, the oscillations in the mean delay were considerably reduced as shown in **Figure 10**. Of course, the rerouting probability had to be adjusted to a much smaller value (0.025) compared to its original value of 0.6. This brings out the fact that the CODEX algorithm performs well in traffic conditions where there is *a large number of virtual circuits each contributing a small amount of traffic* to the total network traffic.

4.3.2. Quasistatic Traffic Environment.

In this case, every traffic generating source-destination pair (a) is activated, (b) remains *active* until it generates a geometrically distributed random number of packets (with a mean of 2000), and (c) is deactivated and remains *inactive* for an exponentially distributed random period (with a mean of 4 minutes). The active period obviously depend on the traffic generation rates of the individual VC's. The results are presented in **Figure 11**, and provide additional insight on the adaptivity of the algorithms to more rapidly changing traffic conditions.

Even though Gallager's algorithm performed better than the CODEX algorithm under stationary traffic conditions as shown earlier, it is clear from **Figure 11** that it has poorer adaptivity to fast-changing traffic conditions. The sharp peaks observed in the mean delay response of Gallager's algorithm correspond to arrivals of new VC's. As was noted in the previous subsection, the CODEX algorithm is suited to adapt better to arriving VC's through its explicit VC setup routing mechanism, whereas Gallager's algorithm tries to adapt to the situation rather slowly. This suggests that in an environment expected to feature such abrupt traffic changes, a combination of approaches is desirable: an initial setup procedure based on shortest path algorithms, combined with a gradient-based scheme seeking to optimize the steady-state performance of the network.

Finally, note the nominal mean delays observed in **Figure 11** are lower than those seen in previous cases. This is simply due to the fact that the overall network load is now lower, since traffic generating source-destination pairs are periodically inactive.

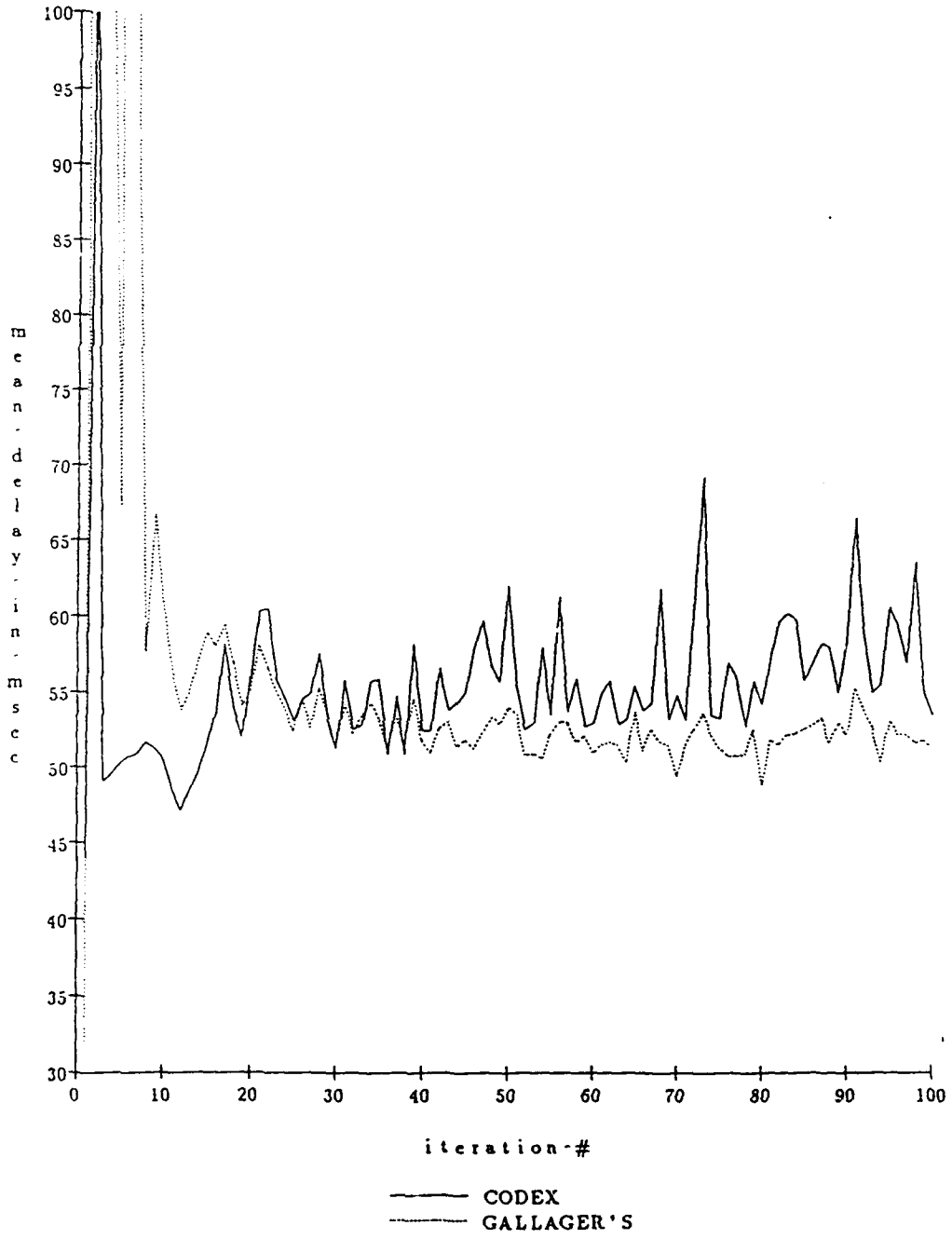


Figure 8 : **Comparison of the Mean Delay Performances of the CODEX and Gallager's Algorithms.**
(6 Node Network, Traffic Streams Activated gradually,
Observ. Interval = 56 sec, Step Size for Gallager's Alg. = 0.01 E - 05
Rerout. Prob. for the CODEX Alg. = 0.6)

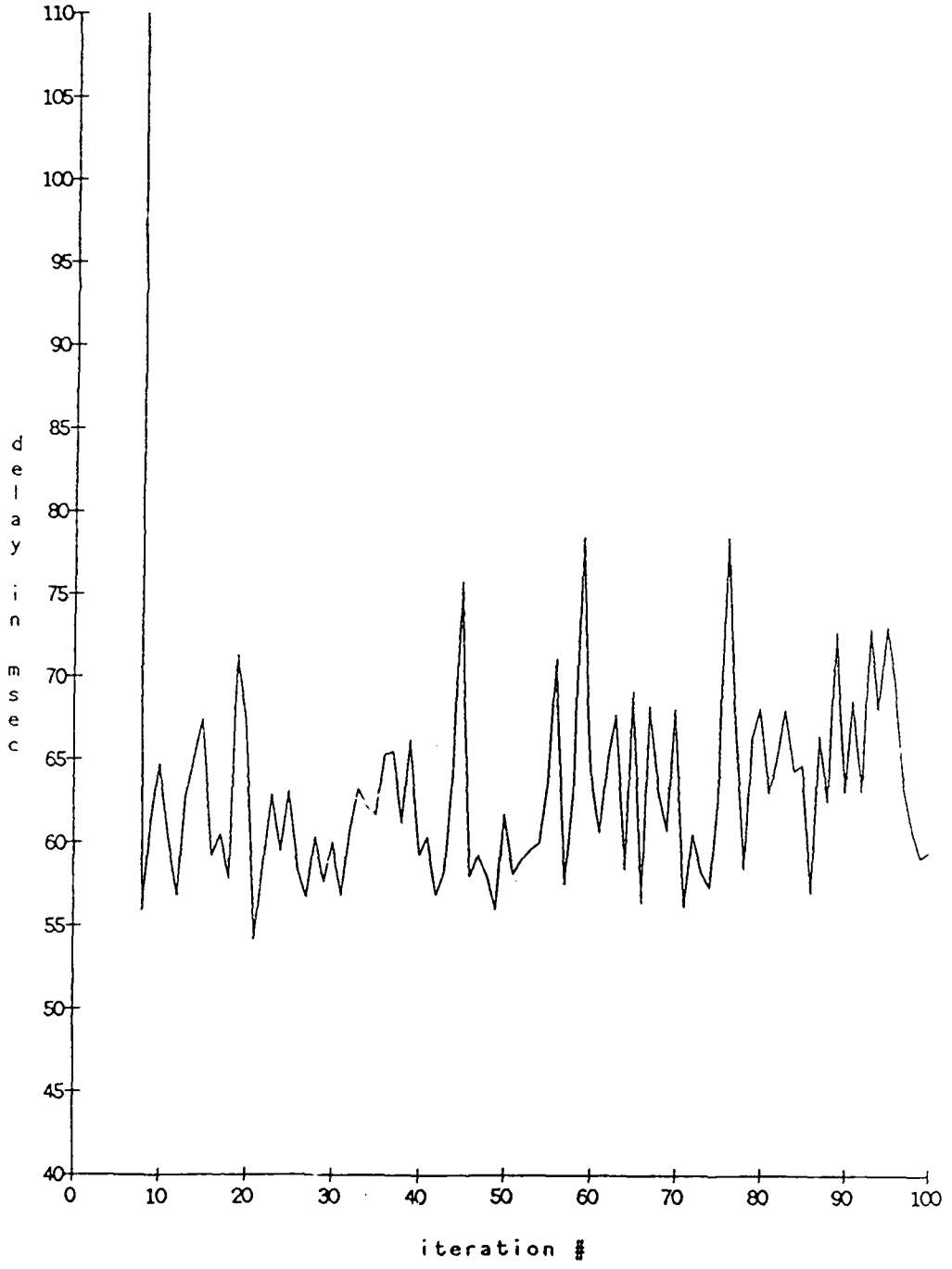


Figure 9 : Effect of Increasing the Traffic Rates of Virtual Circuits in the CODEX Algorithm
(6 Node Network, Traffic Streams Activated gradually,
Observ. Interval = 56 sec, Rerout. Prob. = 0.6)

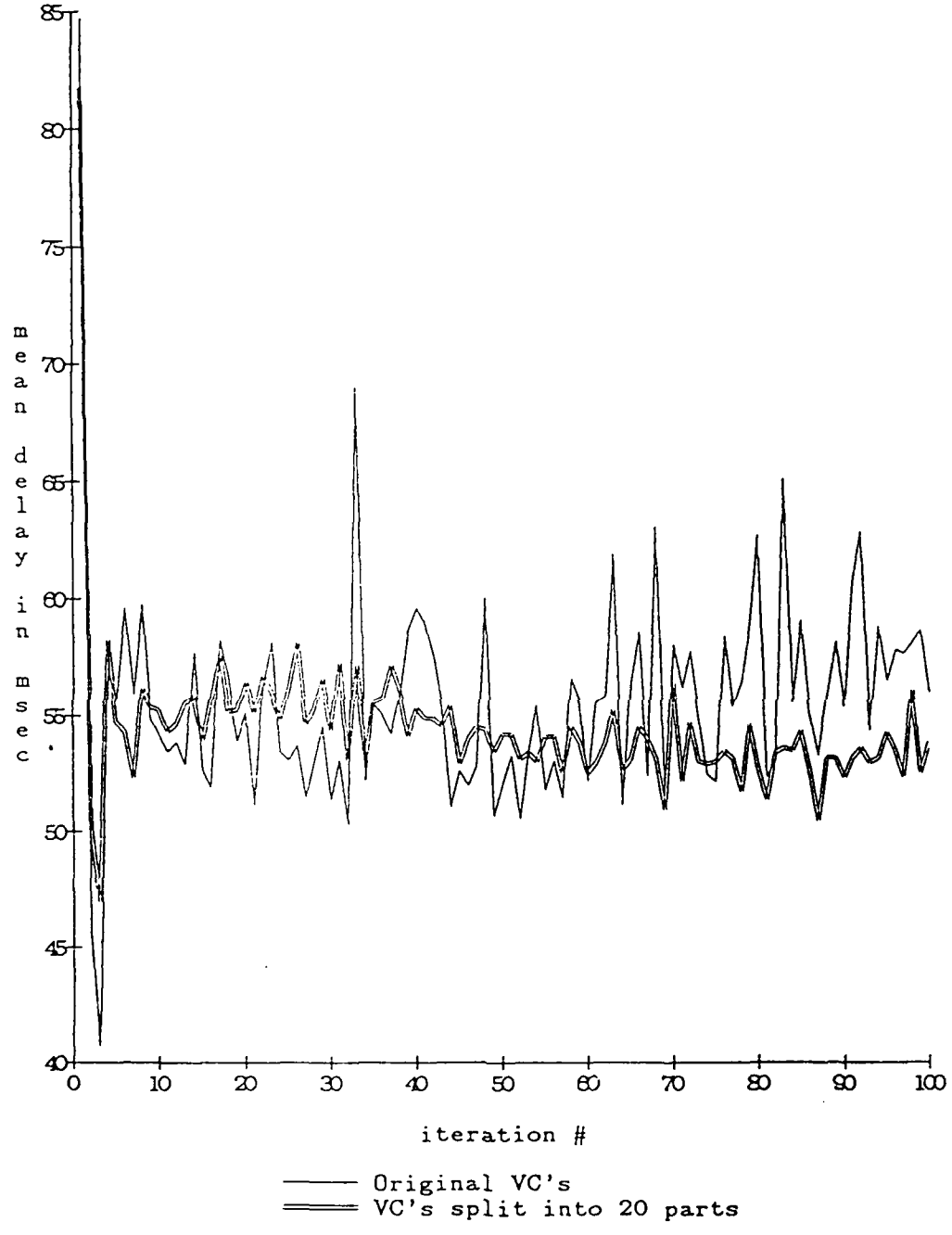


Figure 10 : Effect of Decreasing the Traffic Rates of Virtual Circuits in the *CODEX Algorithm*

(6 Node Network, Traffic Streams Activated gradually, Observ. Interval = 56 sec, Rerout. Prob. = 0.6)

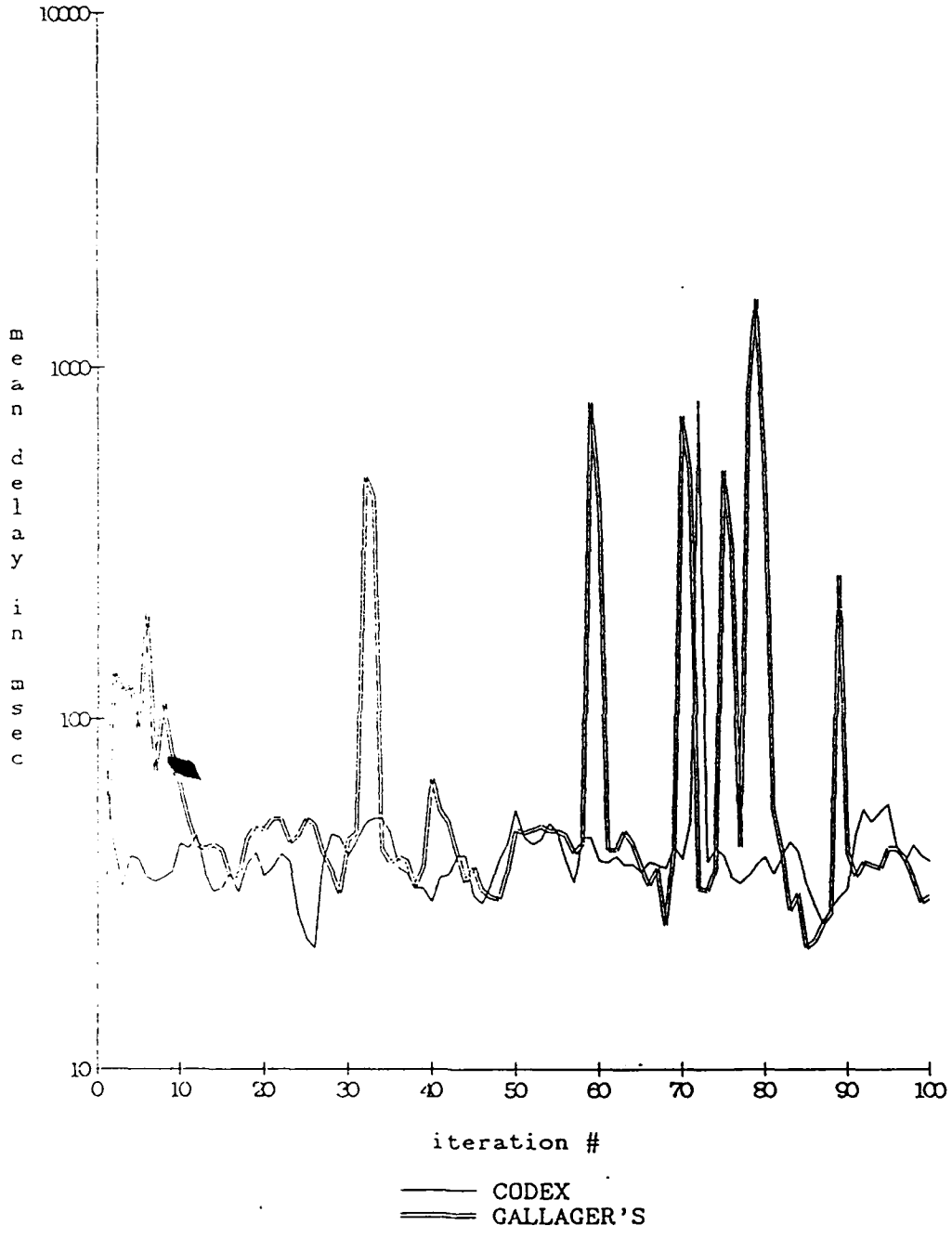


Figure 11 : Comparison of Mean Delay Performances of the CODEX and Gallager's Algorithms under Quasistatic Traffic
(6 Node Network, Traffic Streams Switched ON/OFF,
Observ. Interval = 56 sec, Step Size for Gallager's Alg. = 0.01 E - 05
Rerout. Prob. for the CODEX Alg. = 0.6)

4.4. Mean Delay Performance of the Standard ARPANET Algorithm.

In this section, the mean delay results of applying the ARPANET algorithm to the network of **Figure 2** are compared to those of the CODEX and Gallager's algorithms under similar traffic conditions. The implementation details follow the description of the algorithm in Section 2.3, i.e. an observation interval of 10 sec and the staggered threshold mechanism for reporting changes in link delays were used. The threshold mechanism avoids frequent reporting of small changes in the link delay which could cause large shifts in traffic from one region to another in the network. This mechanism tends to limit oscillations, while ensuring that at least one update takes place over several successive observation intervals.

As already mentioned, a bias factor equal to the transmission time of a packet is also used to update the link lengths. This bias factor is added to the link length and acts as a damping factor for the oscillations. For instance, it ensures that any link which was completely idle over the observation interval is not assigned a zero link length.

The results are reported in **Figure 12.** It can be seen that ARPANET displays extremely large high-frequency oscillations. Moreover, the value of the nominal mean delay is much higher compared to the CODEX and Gallager's algorithms. The main reasons for this behavior were discussed in earlier sections: the basic operating philosophy of the ARPANET algorithm causes a far from optimal performance and high susceptibility to oscillations. However, we were able to identify a number of factors which may be contributing to the oscillatory performance:

1. Traffic intensity
2. Nonuniformity of link capacities
3. Size of the network

In the remainder of this section, we examine the first two factors. To investigate the performance of the algorithm in large networks, substantial modifications of our experimental setup were required; a discussion of the modified setup and our results are presented in section 4.5.

4.4.1. Effect of Traffic Intensity on Oscillations in ARPANET.

In reality, the ARPANET algorithm may not be performing as poorly as it appears in **Figure 12.** One of the reasons may be that it is designed to operate in environments with a very light traffic load on the network. Under light load condition, the congestion level on various links is not sufficiently high to

- make some links appear attractive compared to others; as a result, there are less traffic shifts
- result in extremely large delays causing high amplitude oscillations

To verify the effect of light load on the observed oscillations, the traffic intensities of each of the 19 source-destination pairs in our experiment were reduced by 50% and then by 20%. Thus, the overall traffic intensity was reduced by the same factors as well.

The mean delay behavior of the ARPANET algorithm under these light load conditions is presented in **Figure 13**. The results are plotted on a log scale to accommodate the large range of variation. It can be seen that a 50% load reduction results in a drastic improvement in performance, both in terms of nominal mean delay and the extent of oscillations. A further reduction to 20% of the original load does not substantially affect the oscillation characteristics (as expected, however, the mean delay is reduced).

4.4.2. Effect of Nonuniformity of Link Capacities on Oscillations in ARPANET.

It was speculated that the nonuniform link capacities in our network (ranging from 9.6 Kbps to 72 Kbps) could be contributing to the observed oscillations in the mean delay. This is in fact the case, and reveals *a key limitation of algorithms which do not make use of mean delay sensitivity information.*

Considering the simple 2-link network of **Figure 1**, a little thought can validate the above speculation. The ARPANET algorithm takes into account only the absolute mean delays on the two links, without considering the delay sensitivities with respect to their corresponding traffic rate. As a result, *two links having the same absolute delay are treated the same, whereas they may obviously have critically different sensitivities depending on their respective capacities.*

In a large network, however, it is difficult to examine the effect of nonuniform capacities in an isolated manner: any change in link capacity causes changes in the overall loading conditions, which makes it very hard to achieve similar loading conditions for two different configurations. Nonetheless, we conducted some experiments which do illustarte the effect of nonuniform link capacities as speculated above. These results are presented in **Figure 14.**

First, all link capacities were made uniform by assigning them a single capacity of 56 Kbps. The oscillations were significantly reduced. The mean delay was also reduced because the loading conditions were substantially lighter and no congestion built up over each observation interval.

Next, we sought another uniform link capacity in the range of 9.6 Kbps to 72 Kbps which would produce a higher nominal mean delay than the original one. A capacity of 40 Kbps was found to meet this requirement. The results, as seen in **Figure 14**, show considerably less oscillations with a mean delay which is higher by a factor of 10.

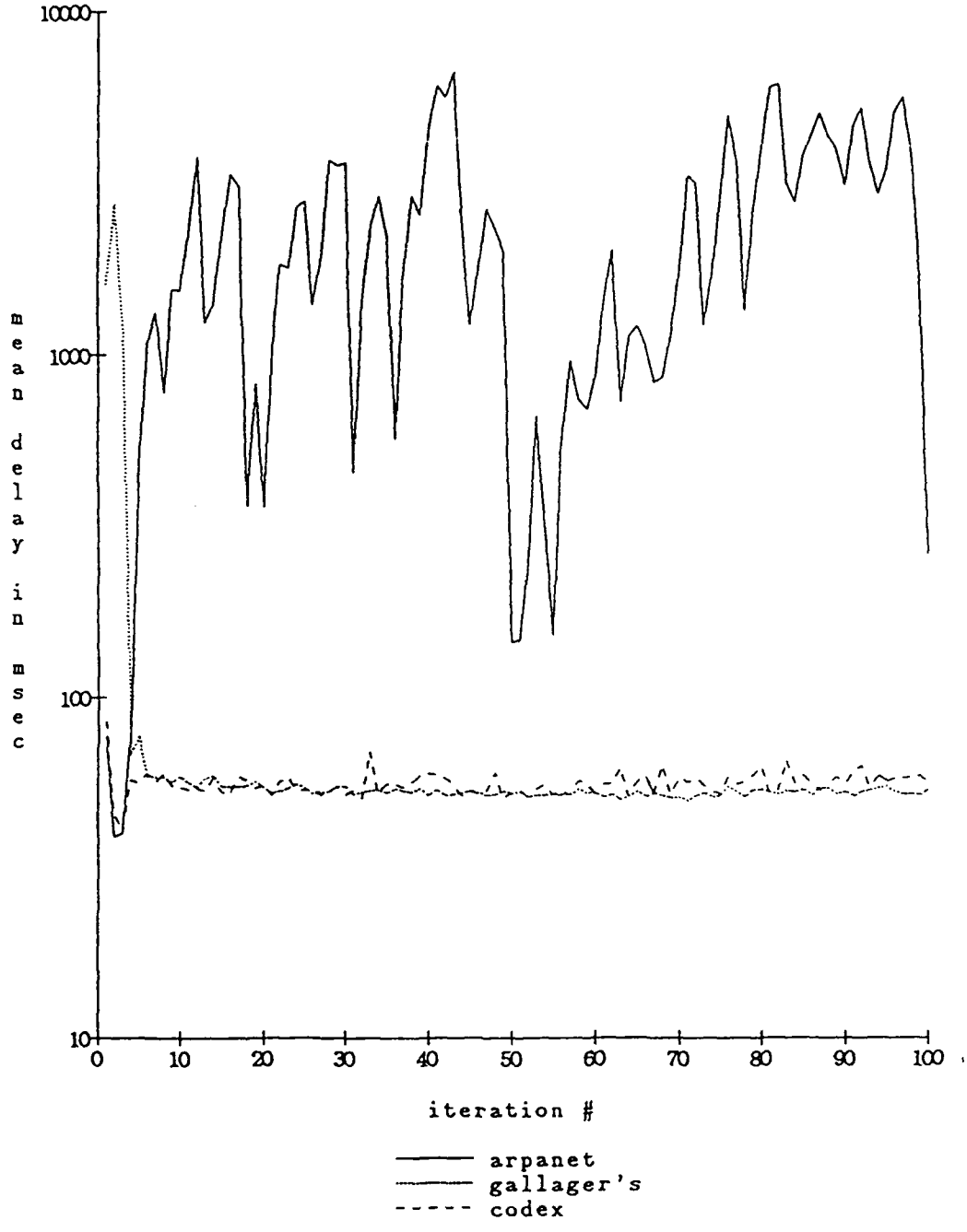


Figure 12 : Comparison of Mean Delay Performances of the CODEX Gallager's and the ARPANET Algorithm
(6 Node Network, Traffic Streams Activated Gradually,
CODEX : Observ. Interval = 56 sec, Rerout. Prob. = 0.6
Gallager's : Observ, Interval = 56 sec, Step Size = 0.01 E - 05
ARPANET : Observ. Interval = 10 scc (staandard alg.))

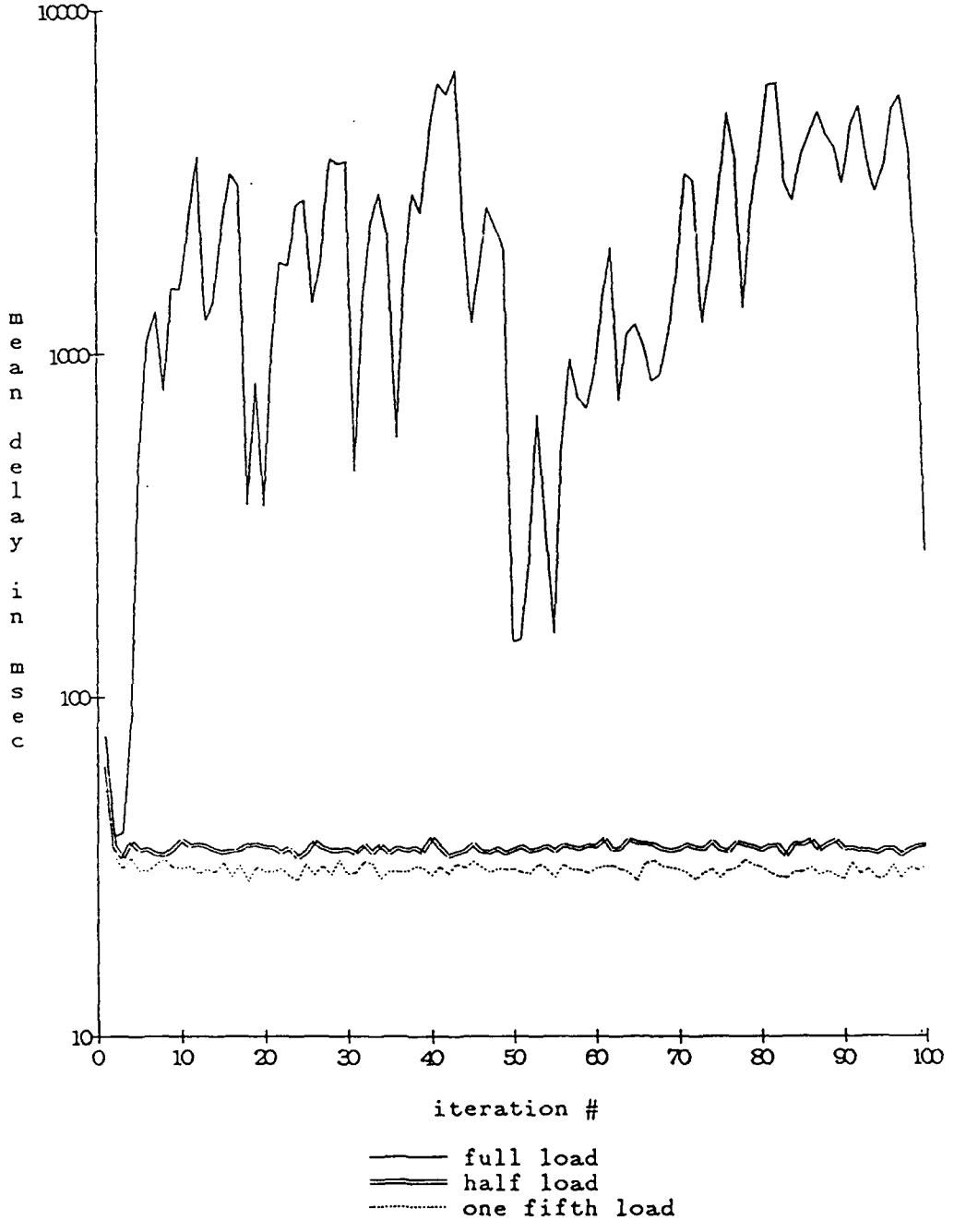


Figure 13 : Mean Delay Performance of the ARPANET Algorithm Under Light Traffic Conditions
(6 Node Network, Traffic Streams Activated Gradually
Observ. Interval = 10 sec)

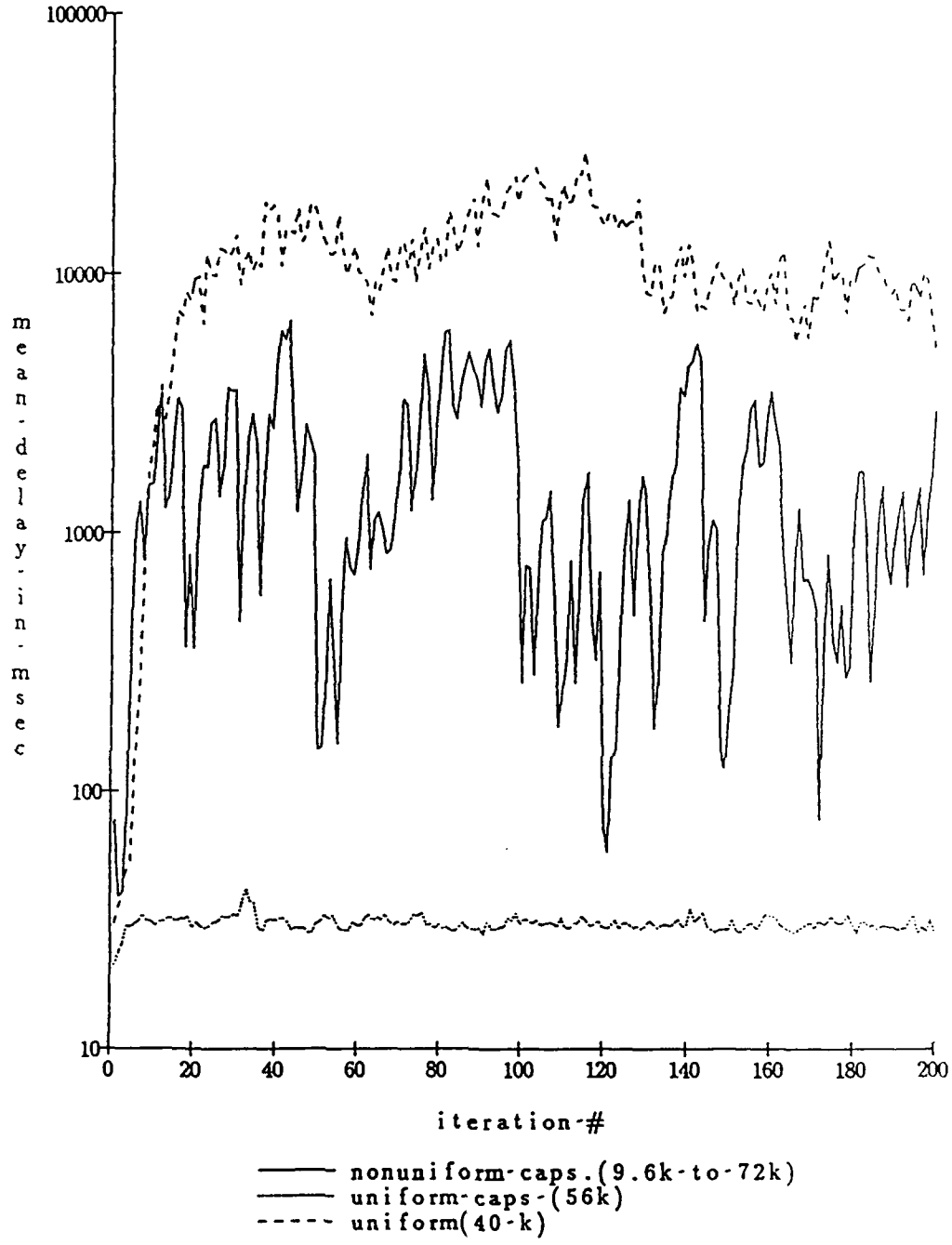


Figure 14 : **Effect of Nonuniformity of Link Capacities on Mean Delay Performance of the ARPANET Algorithm**
(6 Node Network, Traffic Streams Activated Gradually
Observ. Interval = 10 sec)

4.5. Effect of Packet Resequencing on Mean End-to-End Delay in Datagram Networks.

One advantage of using virtual circuits over datagrams is the sequential delivery of packets at the destination. In the case of datagram networks, the packets have to be resequenced at the destination nodes before being passed on to the host. A simple way of accomplishing this is to assign a sequence number to each of the packets for a source-destination pair. At any time, the destination node is expecting a packet with a particular sequence number. Any arriving packet with a sequence number less than this is stored in a buffer. When the expected packet arrives, all the packets forming a sequence are removed from the buffer and passed on to the host. Thus, "resequencing delays" are the queueing delays experienced by packets stored in this buffer.

For a fair comparison of mean delay performance one must consider the end-to-end delay, which includes the resequencing delay, in the case of datagram routing. It should be noted that even though the CODEX algorithm is a VC-type algorithm, resequencing delays do occur when a VC is rerouted: when a new path is established, there are still some packets propagating on the old path; consequently, the destination node must be able to do resequencing for a brief period of time, until the old path has been completely drained of all the packets. One would, however, expect that this resequencing delay is very small when averaged over all the packets in an observation interval.

Experiments were performed incorporating the resequencing mechanism described above for Gallager's algorithm. The comparative results showing the mean end-to-end delay performance of Gallager's algorithm with and without resequencing are shown in **Figure 15**. It can be seen that the increase in the nominal delay is very small, amounting to approximately 5% of the nominal mean delay. This mean end-to-end delay performance of Gallager's algorithm with resequencing is compared with that of the CODEX algorithm in **Figure 16**. It is clear that under *stationary traffic conditions* Gallager's algorithm outperforms the CODEX algorithm with respect to both nominal mean delay and steady state oscillations. Thus, resequencing delays have little effect on the relative performances of the two algorithms. However, the situation may be different for a larger network where resequencing delays may become significant due to a larger expected number of hops.

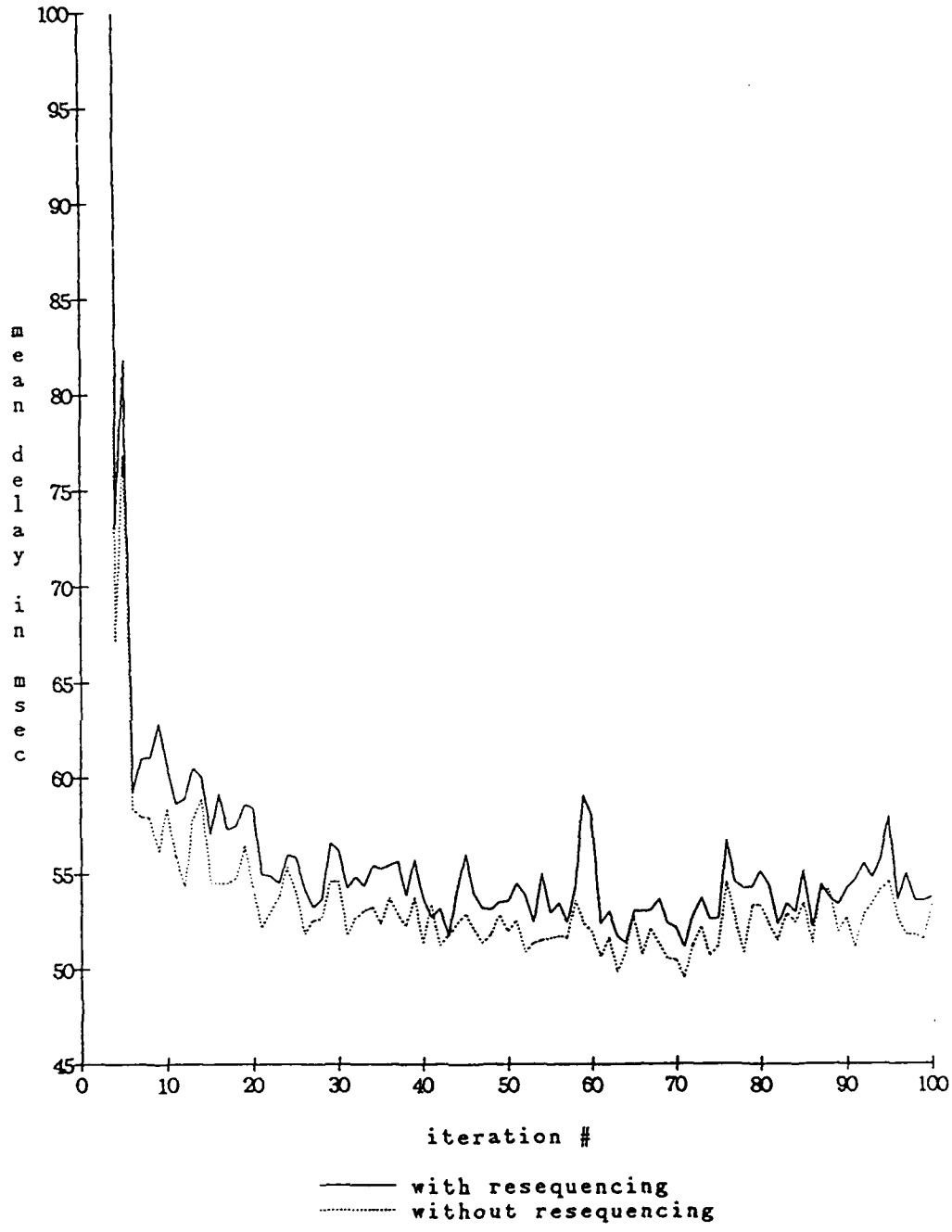


Figure 15 : Effect of Resequencing on the End-to-end Delay in Gallager's Algorithm
(6 Node Network, Traffic Streams Activated Gradually
Observ, Interval = 56 scc, Step Size = 0.01 E - 05)

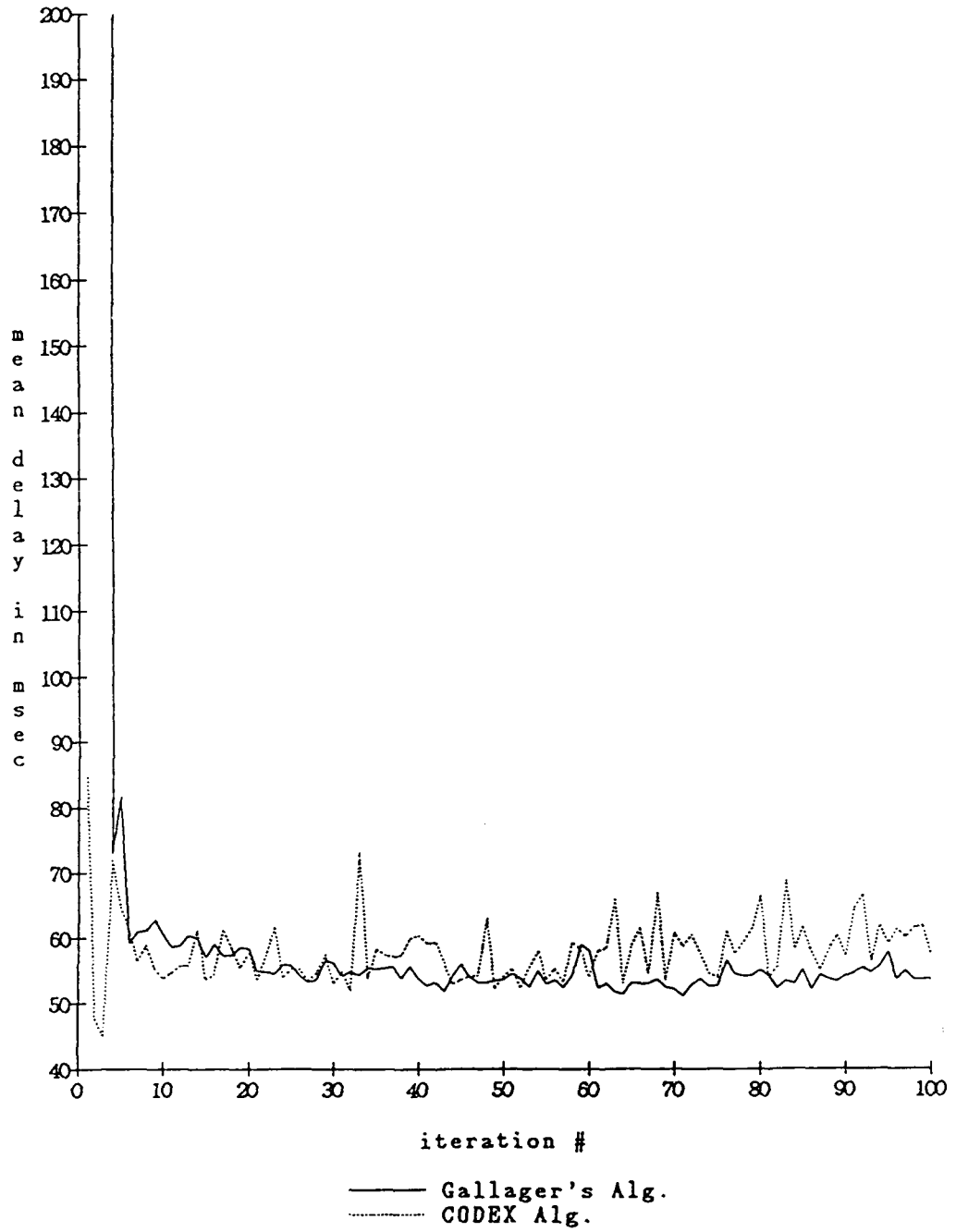


Figure 16 : Comparison of Mean Delay Performance of the CODEX Algorithm to that of Gallager's Algorithm with Resequencing
(6 Node Network, Traffic Streams Activated Gradually
CODEX : Observ. Interval = 56 sec, Rerout. Prob. = 0.6
Gallager's : Observ, Interval = 56 sec, Step Size = 0.01 E - 05)

4.6. Effect of Network Size on Routing Algorithm Performance.

So far, all our experiments were conducted on the 6-node, 16-link network of **Figure 2**. This is a small network in comparison to most networks in practice. Unfortunately, experimenting on a very large network is prohibitively expensive. Thus, it is a challenging task to identify adequately large networks which, in conjunction with a proper design of experiments, can bring out the essential characteristics of a routing algorithm.

As already pointed out in previous sections, there is sufficient reason to believe that some of the features we have identified for some of the algorithms may be quantitatively different in large networks. For example, the sensitivity of the mean delay to rerouting probabilities in the CODEX algorithm is expected to be smaller in a large network with a large number of virtual circuits each with a very small traffic rate. Similarly, the effect of packet resequencing on end-to-end delays in datagram networks may become more apparent in a large network. Some of these issues were studied for the CODEX algorithm using a large network setup which is described below.

4.6.1. Experimental Setup.

The experimental setup used here is briefly described in the next few subsections.

4.6.1.1. Network Configuration.

The large network shown in **Figure 17** was set up. This is the same network used in [21] for a study of finite buffered nodes in store and forward networks. It has 19 nodes and 33 full-duplex links with capacities ranging from 9.6 Kbps to 50 Kbps. The network configuration is such that it appears as if two smaller networks have been connected with a few high speed (50 Kbps compared to 9.6 Kbps and 19.2 Kbps) links.

4.6.1.2. Traffic Arrival Characteristics.

It was assumed that every node generates traffic for every other node in the network with an interarrival distribution chosen from six such different distributions as shown in **Table 2**. The parameters of the distributions are specified in msec. Thus, we have 342 different source-destination pairs, each one of which generates long and short packets (see section 4.6.1.3 below).

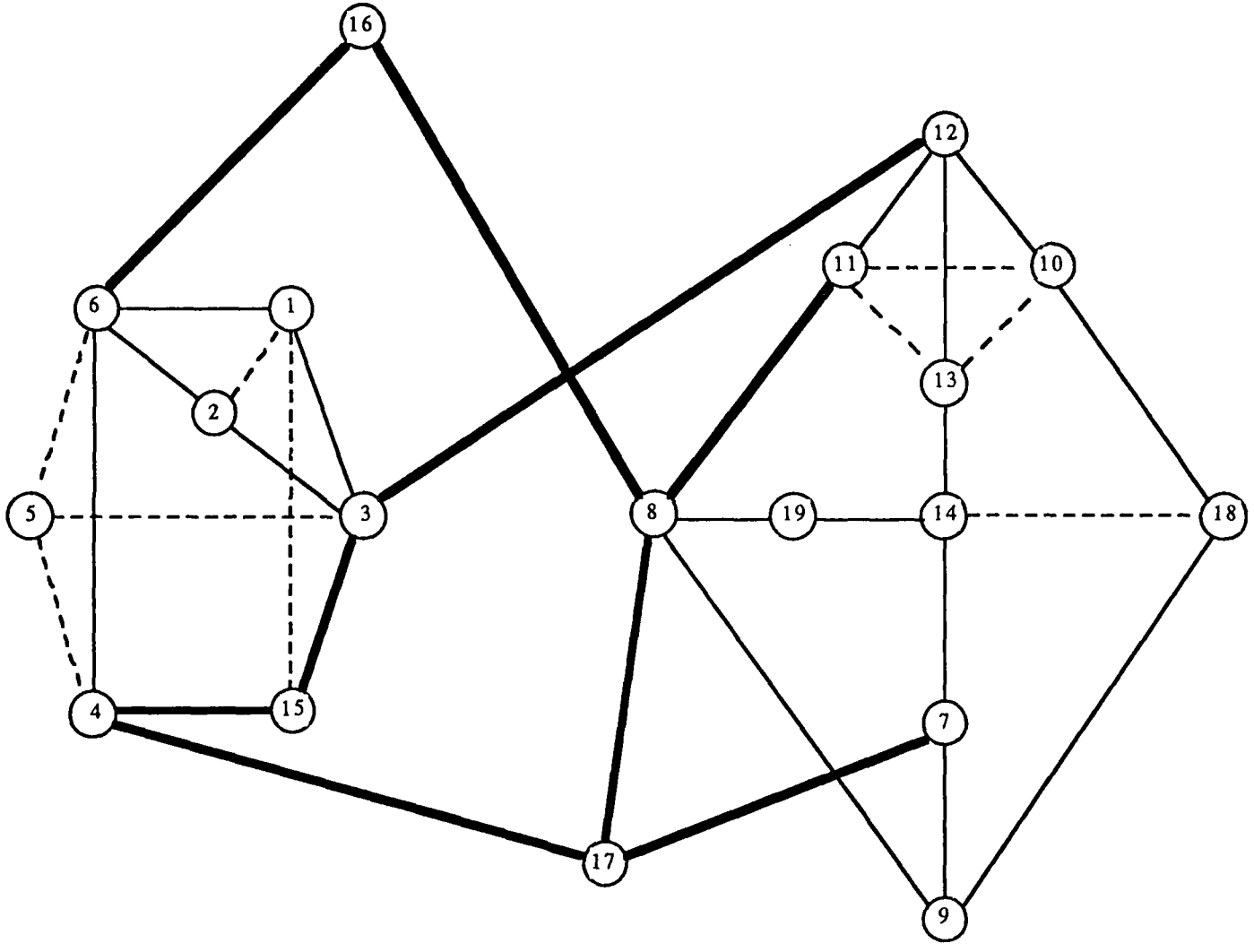


Figure 17 : The Large Network Configuration
(19 Nodes, 33 Full-Duplex Links)

DISTRIBUTION No.	DISTRIBUTION
1	EX(600)
2	EX(2000)
3	EX(3000)
4	UN(800,1000)
5	UN(600,800)
6	EX(900)

TABLE 2. *List of Distributions Used in Large Network*

4.6.1.3. Packet Characteristics.

As in the case of our original 6-node network, there are two types of packets generated: 80% are long (1 Kbit) and 20% are short (128 bits). As before, the CODEX algorithm routes short packets on a datagram basis and long packets through virtual circuits.

4.6.2. Effect of Rerouting on Mean Delay Performance of the CODEX Algorithm.

The first experiment that we report on here is intended to study the effect of VC rerouting on the mean delay of the CODEX algorithm applied to the above network. The 342 virtual circuits are gradually activated with exponentially distributed random offset times. Once activated, a VC remains active for the entire duration of the simulation. An observation interval of 56 sec (which corresponds to the departure of over 10000 packets) is chosen as in the case of the small network. The rerouting probabilities are varied from 0 to 0.5.

The mean delay results are presented in **Figure 18.** It can be seen that the nominal mean delay is relatively less sensitive to rerouting probabilities over the range considered, as compared to the case of the smaller network. Particularly at the lower range of rerouting probability values this insensitivity is more apparent. However, this does not imply that one can do away with rerouting altogether. This is verified in **Figure 19,** where one can clearly see the reduction in nominal mean delay (nearly 10%) achieved by choosing to reroute even with a small rerouting probability of 0.05. But any further increase in rerouting probability (as seen for the case of 0.2) does not improve the steady state mean delay (while certainly incuring additional processing overhead). When trading off between rerouting and processing overhead, it should be pointed out that the algorithm is better able to adapt to incoming virtual circuits in the initial 20 to 30 iterations of the algorithm for a larger value of rerouting probability.

Finally, note that steady state oscillations in the mean delay are insignificant (less than 5%). This is consistent with our previous observation and explanation in the context of the small network, i.e. a large number of VC's, each with a very small traffic rate, tends to reduce oscillations.

4.6.3. Mean Delay Performance of the CODEX Algorithm Under Quasistatic Traffic Conditions.

The second experiment on the large network examines the response of the CODEX algorithm to frequently changing traffic. Similar to the case of the smaller network, traffic generating source-destination pairs were switched ON and OFF at random instants of time. The active traffic generation periods of source-destination pairs were defined so that a geometrically distributed number of packets with a mean of 500 is processed. The idle periods for source-destination pairs were chosen from an exponential distribution with mean 2.5 min. This creates an environment where input traffic is rapidly changing.

Our results are shown in **Figure 20** (for the large network) and **Figure 21** (for the small network). It is apparent that the large network is better able to adapt to the traffic fluctuations than the small network is. There are no sharp peaks observed in the mean delay profile as was the case in the smaller network. Instead, the network adapts to traffic variations rather smoothly. This is once again attributed to the fact that the traffic rates of individual virtual circuits are very small compared to the overall network traffic.

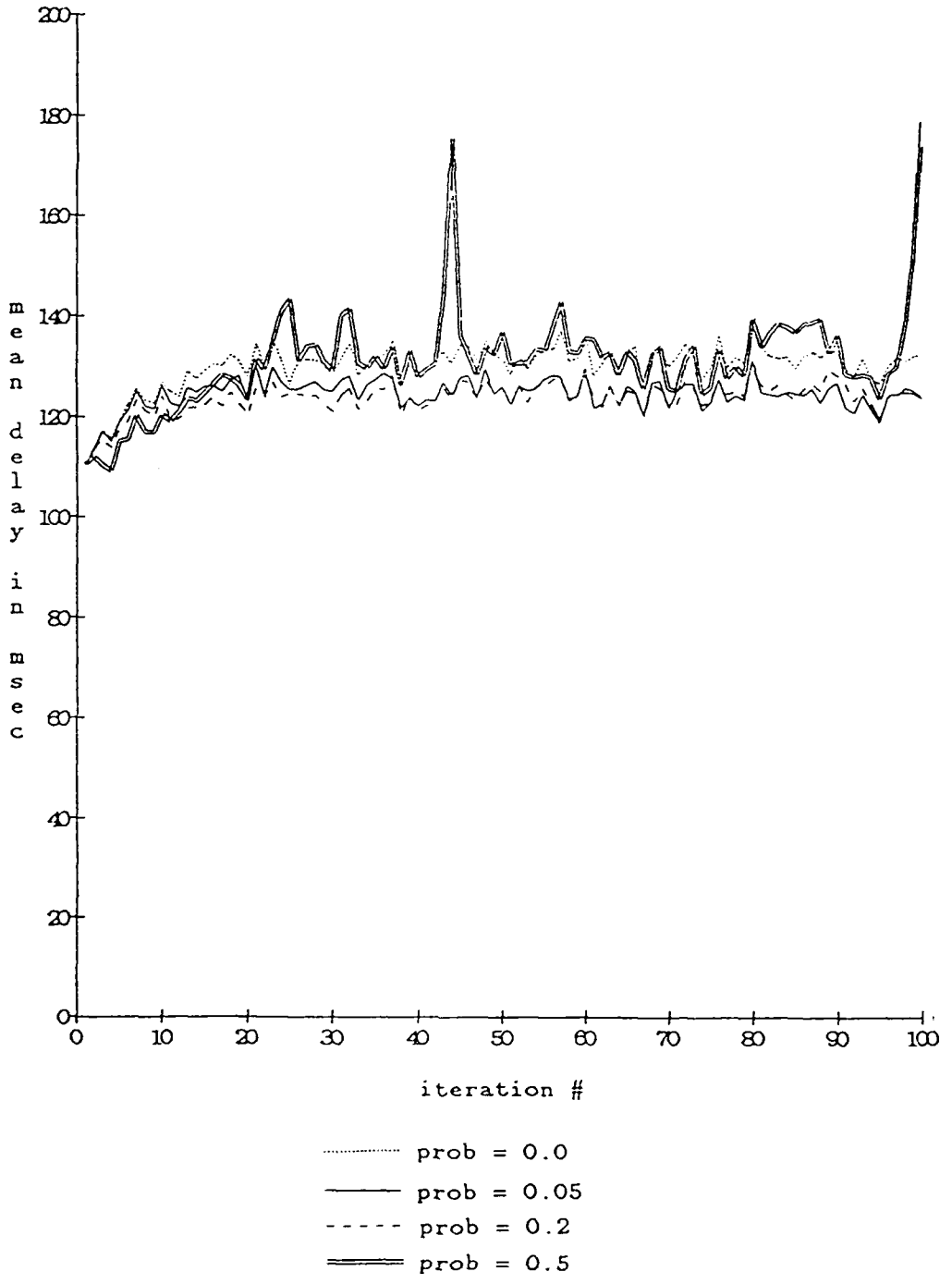


Figure 18 : Sensitivity of the Mean Delay Performance of the CODEX Algorithm to the rerouting of VC's in a Large Network
(19 Node Network, Traffic Streams Activated Gradually
Observ. Interval = 56 sec)

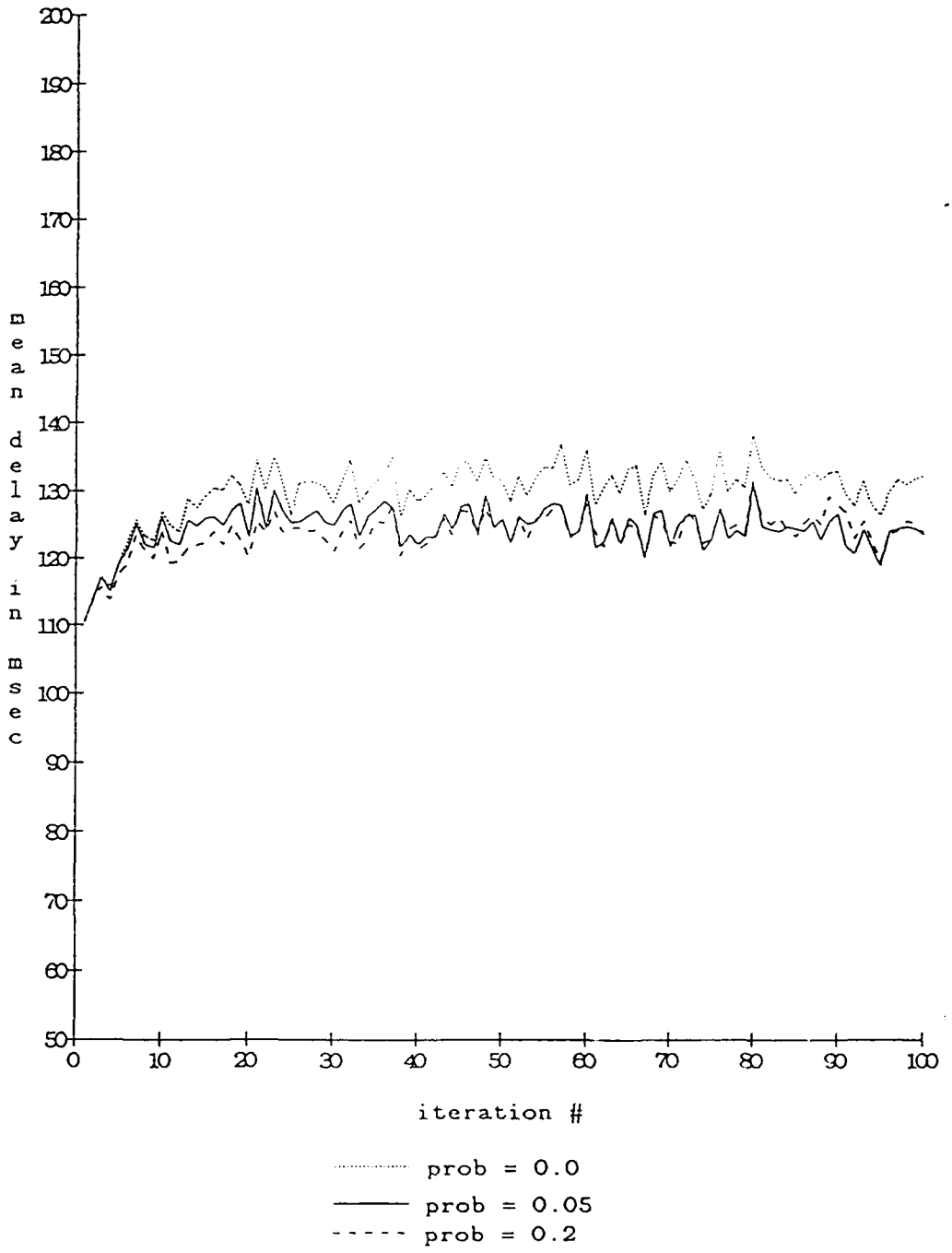


Figure 19 : Effect of Rerouting on the Mean Delay Performance of the CODEX Algorithm in a Large Network
(19 Node Network, Traffic Streams Activated Gradually
Observ. Interval = 56 sec)

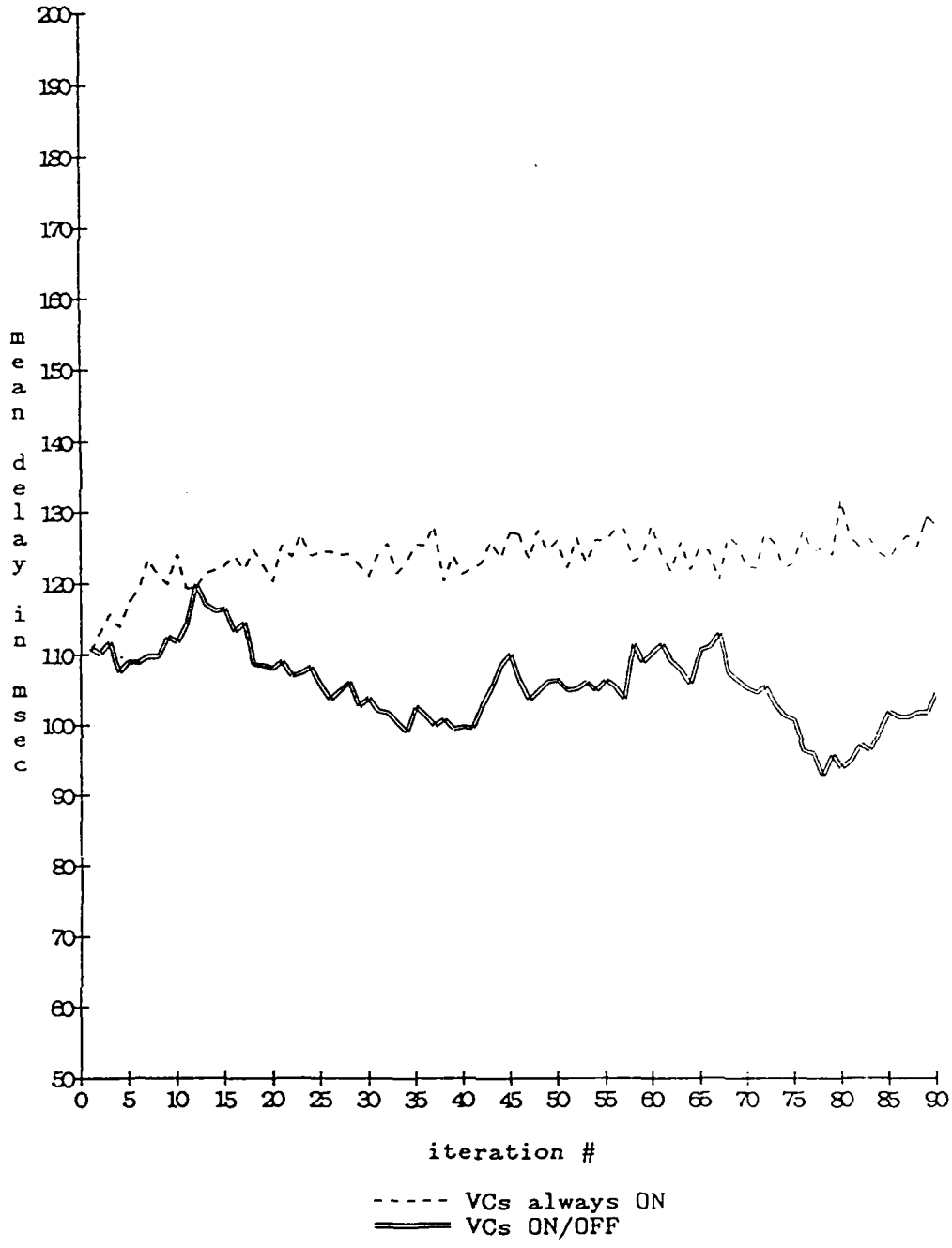


Figure 20 : Mean Delay Performance of the CODEX Algorithm in Quasistatic Traffic Conditions in a Large Network
(19 Node Network, Traffic Streams Switched ON/OFF
Observ. Interval = 56 sec)

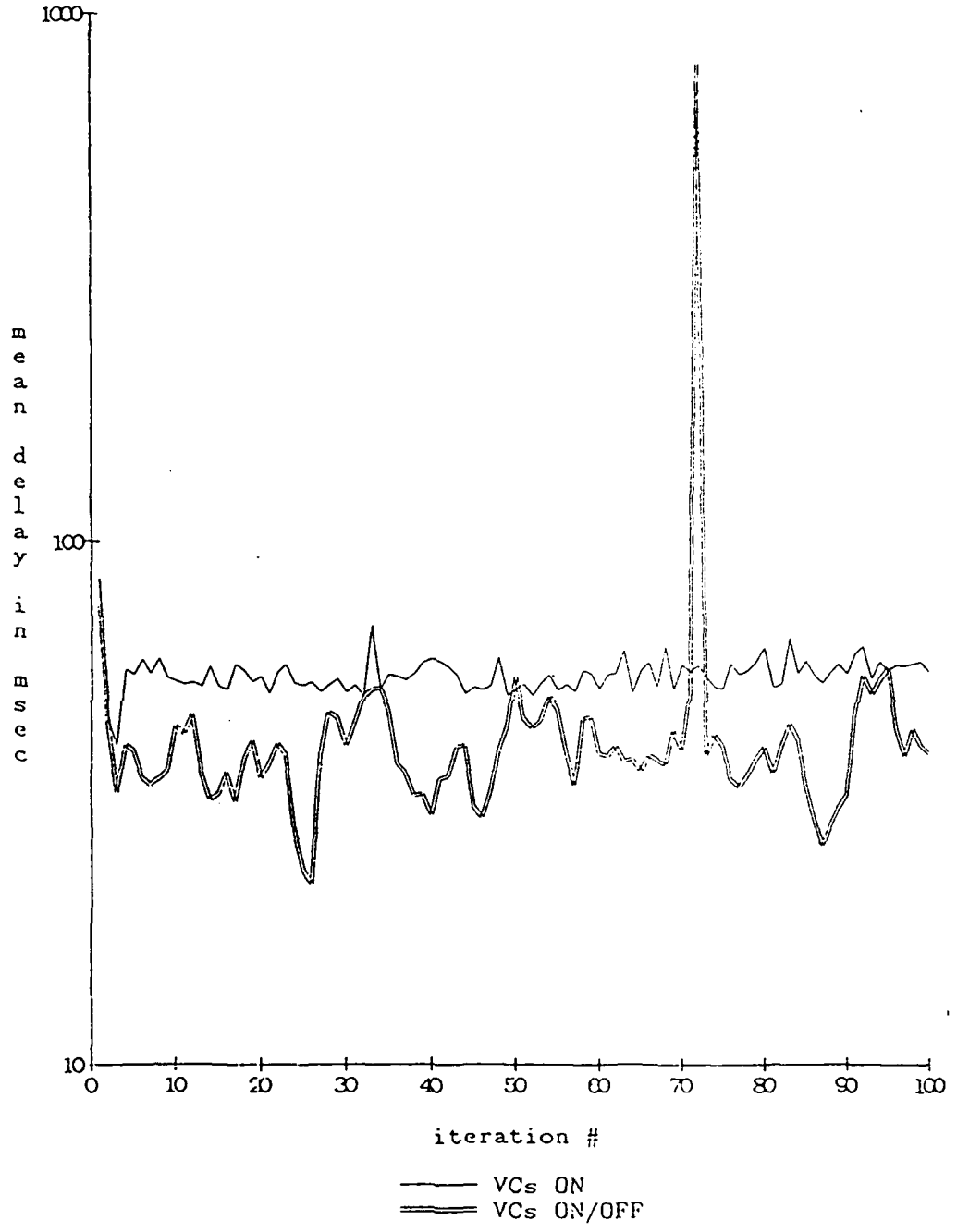


Figure 21 : Mean Delay Performance of the CODEX Algorithm in Quasistatic Traffic Conditions in the Small Network
(6 Node Network, Traffic Streams Switched ON/OFF
Observ. Interval = 56 sec)

4.7. Summary of the Results and Conclusions.

In this section, we summarize all the results presented above. Our results may be categorized as follows:

- Results that depict the behavior of the routing algorithms under parametric variations
- Results that depict the behavior under various traffic conditions (long-run stationary vs quasistatic)
- Results that study the effect of operational and environmental conditions, such as resequencing in datagram networks and the size of a network.

4.7.1. Parametric Variations.

With respect to the *observation interval*, it was seen that both the CODEX and Gallager's algorithms behave similarly: a longer interval produces fewer steady state oscillations compared to a shorter interval. This is due to increased accuracy in the required estimates (link traffic rates, link delay derivatives) in the case of a longer interval. As far as the transient performance is concerned, the convergence of the algorithms is faster under a small observation interval. This was seen from the rates at which the initial congestion was cleared in the experiments.

However, in the case of the ARPANET algorithm oscillations tend to increase with longer observation interval. Here, estimation accuracy is not crucial compared to the congestion buildup over a long uncontrolled observation period; thus, it is more important to make frequent traffic adjustments to prevent such buildup.

Regarding the effect of *traffic adjustment* parameters (i.e. the step size in Gallager's algorithm and the rerouting probability in the CODEX algorithm), our findings suggest that a small value of the corresponding parameter reduces the adaptivity of the algorithm. This was manifested in the case of Gallager's algorithm as a slower speed of convergence and in the case of the CODEX algorithm as larger oscillations.

On the other hand, a large value of the traffic adjustment parameter causes steady state oscillations in the mean delay behavior of both the algorithms. In the case of Gallager's algorithm this is due to the well known fact that a large optimization step parameter causes oscillations around the optimal point. But in the case of the CODEX algorithm the oscillations are caused by uncoordinated simultaneous rerouting of several VC's (it is again noted that the ARPANET algorithm does not have any such traffic adjustment parameter).

One important conclusion is that a dynamic adjustment of these algorithm parameters allows the network manager the flexibility to trade off between fast convergence and limited oscillatory behavior at steady state. The precise mechanisms for implementing such dynamic adjustments depends on the algorithm used and the characteristics of the network itself.

4.7.2. Traffic Variations.

When the algorithm parameters were tuned for our particular network environment, a direct comparison of the CODEX and Gallager's algorithms revealed certain interesting differences. *The CODEX algorithm produces slightly higher (about 5 % in our experiment) stationary mean delay, and it tends to be more oscillatory*. The higher mean delay is due to the suboptimal performance of the CODEX algorithm, while the oscillations are due to the mechanism of rerouting virtual circuits.

The mean delay behavior of Gallager's and the CODEX algorithms were studied under quasistatic traffic conditions, where traffic sessions were started and terminated at random instants of time. The results indicated that *the CODEX algorithm has a better adaptivity to such fluctuating traffic conditions*. This is inherent in the routing mechanism of the algorithm, where every new VC is routed on a path of least incremental delay. On the other hand, if a small step size is used in order to ensure the convergence of Gallager's algorithm such adaptivity to traffic fluctuations is not possible. Again, however, this suggests the need for dynamically adjusting the step size parameter.

The mean delay performance of the standard ARPANET algorithm was compared with that of Gallager's and the CODEX algorithms. This comparison indicated that *the ARPANET produces a far from optimal mean delay. Moreover, it displays large-amplitude oscillations*. This behavior is inherent in the shortest path algorithm used in ARPANET. However, we found that the oscillatios are limited (a) when traffic conditions are lighter, and (b) when link capacities are uniform.

4.7.3. Operational and Environmental Variations.

Regarding the effect of *resequencing* at destination nodes on the end-to-end mean delay in datagram networks, our results indicate this effect is negligible (less than 10%) compared to the mean delay without resequencing. Even after considering resequencing delays, the mean end-to-end delay in the case of Gallager's algorithm was found to be lower than the mean delay in the case of the CODEX algorithm.

Our experiments on a large 19-node network with 342 VC's indicate that increased network size reduces the sensitivity of the CODEX algorithm to VC rerouting and to rapid fluctuations in

traffic conditions. This is because the traffic contribution of an individual VC switiched ON or OFF is much smaller in a large setting.

5. GRADIENT ESTIMATION IN ROUTING ALGORITHMS.

As pointed out earlier, the method of gradient estimation is a critical issue in gradient-based routing algorithms such as Gallager's. This algorithm requires the derivatives of individual link delays with respect to the corresponding packet flow rates. One may either use a direct approach of modeling each link as a queueing system and then analytically evaluating the derivative, or resort to on-line methods such as the Perturbation Analysis (PA) techniques[14],[15] or the Likelihood Ratio approach. In the following discussion, we shall focus our attention on comparing analytical techniques to PA.

Our motivation comes from observations made in a previous study [19], where it was found that using a simple M/M/1 link model for derivative estimation in Gallager's algorithm results in a nominally higher mean delay compared to that obtained when PA-based derivative estimationwas used. Moreover, large steady-state oscillations were observed in the former case. The network under consideration in this case was the 6-node network of **Figure 2**. The complexity of the network configuration makes it hard to gain immediate insight into the above phenomena. Thus, in this section we consider a simple 2-node, 2-link model on which Gallager's algorithm is applied using M/M/1 and PA methods of derivative estimation respectively. The resulting observations and the ensuing discussions provide some understanding of the effects of derivative estimation methodologies on the overall mean delay performance of a gradient-based routing algorithm.

5.1. Two Approaches to Gradient Estimation.

In this section we review the common analytical method and PA-based on-line methods for gradient estimation.

5.1.1 The Analytical Method.

The analytical method is based on an *assumed* queueing model for a link. Usually, one assumes packets enter a link forming a Poisson process. One then uses the M/M/1 model for a link if the packet lengths are variable (and assumed to be drawn from an exponential distribution) or the M/D/1 model if the packet length is known to be fixed. In either case, the underlying idea is that an analytical expression for the delay on a link (i,j) is available as a function of the link flow rate λ_{ij}:

$D_{ij} = f(\lambda_{ij}, C_{ij})$ where f is a known function and C_{ij} is the link capacity

We evaluate the derivative of this expression at the estimated value of the flow rate, and hence obtain the desired estimate of the gradient :

$$\hat{D}_{ij}'(\lambda_{ij}) = \frac{\partial f}{\partial \lambda_{ij}} \Bigg|_{\lambda = \hat{\lambda}_{ij}}$$, where $\hat{\lambda}_{ij}$ is the estimated flow rate on the link (i,j)

As an example, in case of the M/M/1 model one obtains the formula

$$\hat{D}_{ij}'(\lambda_{ij}) = \frac{C_{ij}}{(C_{ij} - \hat{\lambda}_{ij})^2}$$

where C_{ij} is the capacity of link (i,j).

A simple way of estimating the flow rate is to monitor the arrival process until N packets have arrived. If the time between the arrival instants of the (i-1)th and ith packet is denoted as A_i, then

$$\hat{\lambda}_{ij} = N \cdot \left(\sum_i A_i \right)^{-1}$$

where $\sum_i A_i = T$, an observation interval over which the N packets arrive. Note that this is the estimated flow rate in packets per unit time; instead, one can readily use the cumulative number of bits to obtain the flow rate in bits per unit time.

5.1.2. On-Line Methods Based on Perturbation Analysis (PA).

As opposed to the above analytical approach, which is based on an assumed model for links, PA techniques [14],[15] do not require explicit knowledge of such models. Instead, they are on-line methods for estimating gradients, using only actual data obtained during normal operation. Specifically, one monitors arrivals and departures of packets on a particular link.

There are two PA techniques suitable for estimating link delay derivatives. The first, Infinitesimal Perturbation Analysis (IPA), is an efficient asymptotically unbiased estimator. The second, Finite Perturbation Analysis (FPA), may provide small biases, but it does not require any link flow rate information.

5.1.2.1. The IPA Estimator.

Suppose N packets were transmitted over link (i,j) during an observation interval which consisted of M busy periods. The number of packets transmitted over the mth busy period is denoted by n_m. Then the IPA estimate of the gradient of mean delay with respect to the average flow rate on the link is given by

$$\hat{D}_{ij}'(\lambda_{ij}) = \frac{1}{N} \cdot \sum_{m=1}^{M} \sum_{i=1}^{n_m} \sum_{j=2}^{i} (A_i / \theta)$$

where A_i is the interarrival time between the (i-1)th and ith packets, and $\theta = 1 / \lambda_{ij}$; λ_{ij} needs to be estimated as was the case with the M/M/1 formula in section 5.1.1.

It is worthwhile comparing this estimator with the M/M/1 approach above from an *implementation standpoint*. In both cases one needs to estimate λ_{ij}, the link flow rate. In the M/M/1 case, an arithmetic computation is required to obtain the gradient estimate. Through IPA, the gradient estimate is directly obtained as a simple accumulator of interarrival times. Note that the second sum above is reset to zero at the beginning of every busy period. Thus, IPA requires no computation other than the accumulation of time intervals (one timer and minimal storage for the accumulator).

Finally, note that the estimate of the link flow, λ_{ij}, appears squared in the denominator of the M/M/1 formula. This implies that gradient estimates obtained through that formula may be *particularly sensitive to small estimate errors*. This could be a serious problem in the case where only small observation periods are available.

5.1.2.2. The FPA Estimator.

FPA is based on hypothetically perturbing the parameter of interest, which is the mean interarrival time in our case, and then tracking the effect of propagating this perturbation on the performance measure under consideration. Let R be the nominal mean link delay and θ the nominal mean interarrival time. Let $\Delta\theta$ be the perturbation in the parameter θ which results in a change of ΔR in the nominal mean delay. The computation of ΔR involves keeping track of the perturbed sample path and a few simple additions and subtractions at the departure of every packet [19]. This gives ΔR for each of the departing packet which is averaged over all the departures. The desired estimate of the gradient is given by

$$\hat{D}_{ij}'(\lambda_{ij}) = R - \frac{\Delta R}{\Delta\theta/\theta}$$

An appealing feature of this estimator is that it does not need an estimate of θ. On the other hand, the choice of a proper $\Delta\theta$ is critical. Moreover, the mean square error characteristics are poorer compared to IPA. In practice, the estimates obtained through IPA and FPA show little difference.

5.2. Gradient Estimation Case Study.

In this section we describe our ongoing work based on a simple network model. The objective is to study the effects of the method of derivative estimation on the mean delay behavior of Gallager's algorithm. First, we present the model under consideration followed by some of our critical findings to date.

5.2.1. Model Description.

The simple 2-node, 2 link network, used for this analysis is presented in **Figure 22**. Packets arrive at node 1 with a uniform interarrival distribution defined over the interval [15, 25] msec. Packet lengths are assumed to require processing times which are triangularly distributed over the interval [0, 40] msec. These distributions were chosen to recreate the phenomena observed in [19], and to ensure that the standard assumptions used in the M/M/1 model are not satisfied.

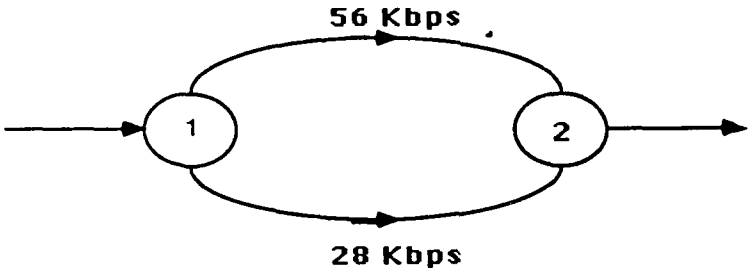


Figure 22: *Simple Network for Studying Gradient Estimation*

5.2.2. Routing Performance Comparison Using Analytical and PA-Based Gradient Estimation.

Gallager's algorithm was applied at node 1 of our network to route the arriving packets onto either link 1 or link 2. The link delay derivatives were estimated through the M/M/1 formula and PA methods respectively. The mean network delays were obtained for an observation interval corresponding to 10000 packet departures on the network and a reasonably small step size of 0.01. These results are presented in **Figure 23**, where the mean delays are plotted against the number of iterations of the algorithm.

The following observations are apparent from **Figure 23**:

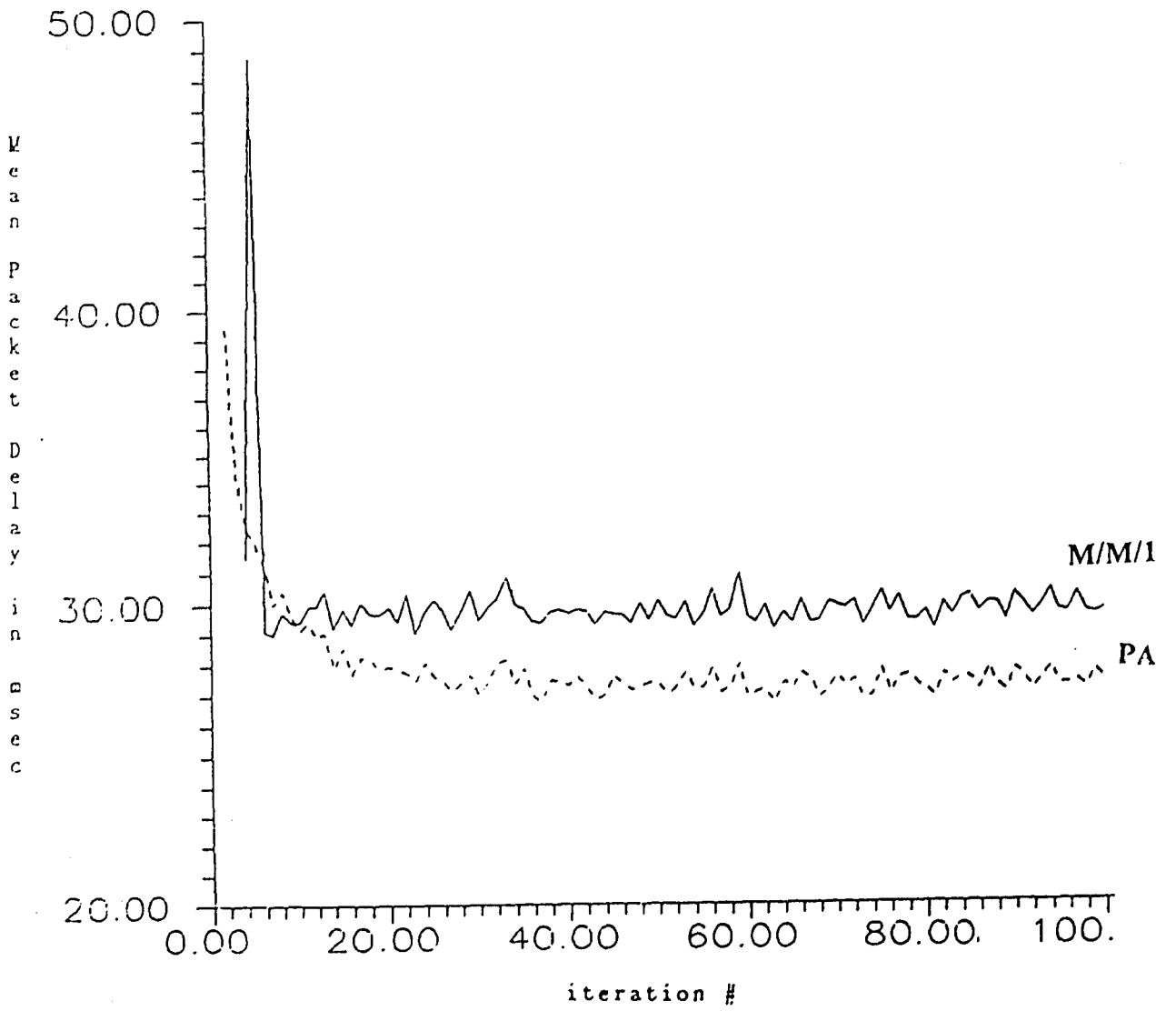


Figure 23 : Mean Delay Performance Of Gallager's Algorithm with M/M/1 and PA Derivative Estimation Methods
(2 Node Network, Uniform Interarrival Distn,
Triangular Packet Length Distn, Step Size = 0.01,
Ovserv. Interval = 200 sec (~10000 packets))

1. The M/M/1 estimation method gives *nominally higher mean delay* than the PA method.

2. Large steady state *oscillations* occur in the case of the M/M/1 estimation method.

3. The M/M/1 approximation leads to *faster* convergence, but to the *wrong* steady state value.

We provide below a framework for explaining these observations and gaining further insight into the issue of gradient estimation in routing.

Let Λ describe the external arrival process to the network and Φ the routing assignment vector for the links. Then, Gallager's algorithm is expected to optimize a *convex* cost function of the form $D(\Lambda,\Phi)$, representing the mean system delay. For a stationary traffic input, the optimization is done with respect to Φ. In the context of the two link network, Φ can be replaced by a scalar ϕ, $0 \leq \phi \leq 1$, which is the routing probability associated with link 1 (the 56 Kbps link in **Figure 22**).

If we assume M/M/1 models for the links, we are effectively optimizing a cost function $D_m(\hat{\Lambda},\phi)$, which would in general be different from the actual cost function $D_a(\Lambda,\phi)$ we wish to optimize. In other words, there are two sources of error in this approach:

1. The function $D_m(\cdot)$ is based on modeling assumptions, and generally differs from the actual one $D_a(\cdot)$.
2. $\hat{\Lambda}$ is only an estimated value of Λ, which introduces different errors from iteration to iteration.

To explain wht the M/M/1 model provides such poor performance, as shown in **Figure 23**, consider **Figure 24**, where the relative locations and shapes of the functions $D_m(\hat{\Lambda},\phi)$ and $D_a(\Lambda,\phi)$ are chosen to facilitate our discussion. These conjectured shapes and locations are found to be in agreement with some approximate constructions based on experimental data. It is particularly important to note that the shape and location of the M/M/1 cost function $D_m(\hat{\Lambda},\phi)$ are changing according to the estimated values of $\hat{\Lambda}$ from iteration to iteration, as shown by the dotted curve in **Figure 24**.

We now consider each of the three observations made above with regard to **Figure 23**:

1. Higher nominal mean delay:

A higher nominal mean delay may result under the optimal routing assignments ϕ_m^* for the M/M/1 cost function and ϕ_a^* for the actual cost function, since the two do not coincide (as shown in **Figure 24**). In this case the M/M/1 approximation method would result in a higher nominal mean delay than the actual optimal mean delay. The difference would amount to

$$| D_a(\Lambda,\phi_m^*) - D_a(\Lambda,\phi_a^*) |$$

2. Larger oscillations:

Larger steady state oscillations in the case of the M/M/1 approximation method can be attributed to two factors:

1. The *change in the location and shape* of the M/M/1 cost function as the flow estimates change
2. A larger *sensitivity* of the actual cost function D_a with respect to ϕ at the value ϕ_m^* which is optimal routing assignment for the M/M/1 cost function.

To illustrate these points, let us assume that using the M/M/1 approximation method we have reached the optimal routing assignment ϕ_m^* on the solid M/M/1 cost function in **Figure 24**. Next, a small error $\Delta\Lambda$ in the estimation of the incoming traffic rate Λ results in a shift $\Delta\phi_m^*$ in the optimal point, where

$$\Delta\phi_m^* = \left[\frac{\partial \phi_m^*}{\partial \hat{\Lambda}} \Big|_{\hat{\Lambda}=\Lambda} \right] . \Delta\Lambda$$

This change in the optimal routing assignment causes a corresponding change ΔD_a in the observed *actual* mean delay, where

$$\Delta D_a = \left[\frac{\partial D_a}{\partial \phi} \Big|_{\phi=\phi_m^*} \right] . \Delta\phi_m^*$$

Since such changes may occur with every observation interval, we see that these factors contribute to the observed oscillations in the case of the M/M/1 approximation method.

3. Faster convergence:

Faster convergence, though to a wrong value, can be attributed to a larger value of the estimated derivative in the case of the M/M/1 approximation. For a specific choice of step size in Gallager's algorithm, a larger value of derivative would cause the algorithm to converge at a faster rate. A larger value of the derivative implies a steeper cost function for the M/M/1 method as depicted in **Figure 24**.

5.3. Experimental validation

The above conjectures regarding the possible shape and location of the M/M/1 cost function with respect to the actual cost function (**Figure 24**) are corroborated by constructing *approximate* cost functions from experimental data. The M/M/1 cost function $D_m(\hat{\Lambda},\phi)$ is constructed by calculating

the system delays from the estimated flow rates $\hat{\Lambda}$ at different routing assignments ϕ and then interpolating them. The actual cost function $D_a(\Lambda,\phi)$ is constructed by interpolating the directly estimated mean system delays for different routing assignments ϕ. These are presented in **Figure 25**.

It can be seen from **Figure 25** that the M/M/1 curve is indeed steeper than the actual curve for this experimental setup. Moreover, the two optimal routing assignments for the two cost functions do not coincide. The higher delay observed in the M/M/1 approximation (~29 msec) in **Figure 23** is indeed the actual delay at the optimal routing assignment for the M/M/1 cost function.

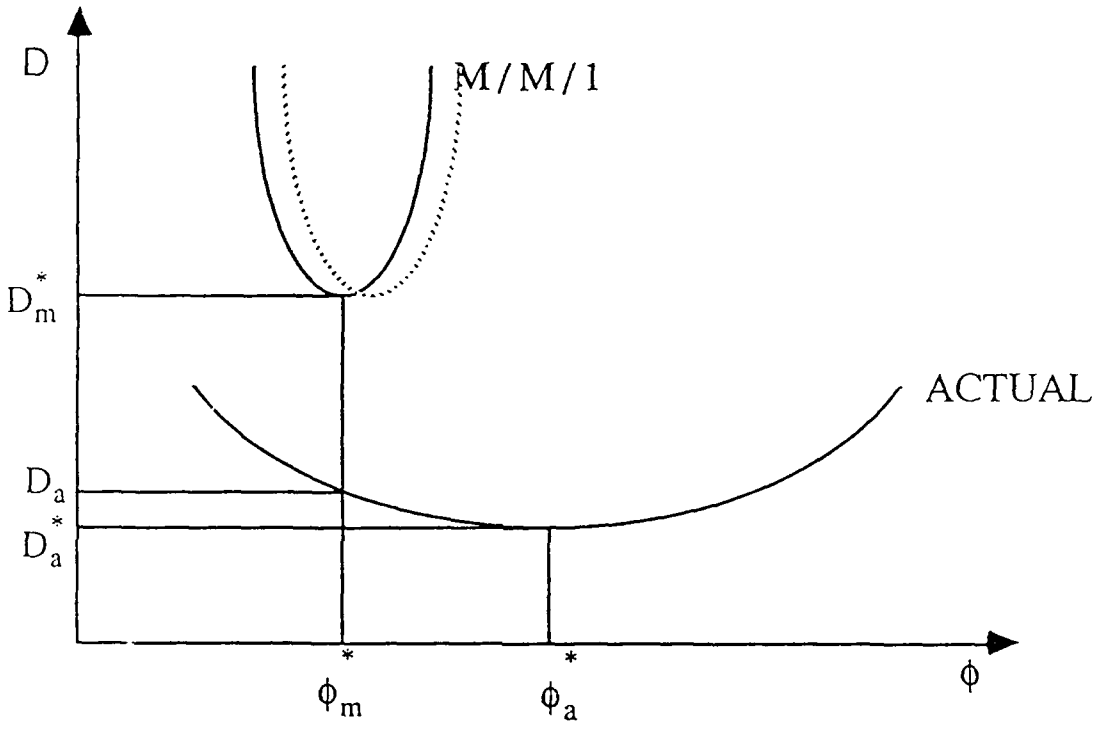


Figure 24 : **A Possible Explanation of the Mean Delay Performance of Gallager's Algorithm with M/M/1 Derivative Estimation**
(2 Node Network, Uniform Interarrival Distn,
Triangular Packet Length Distn)

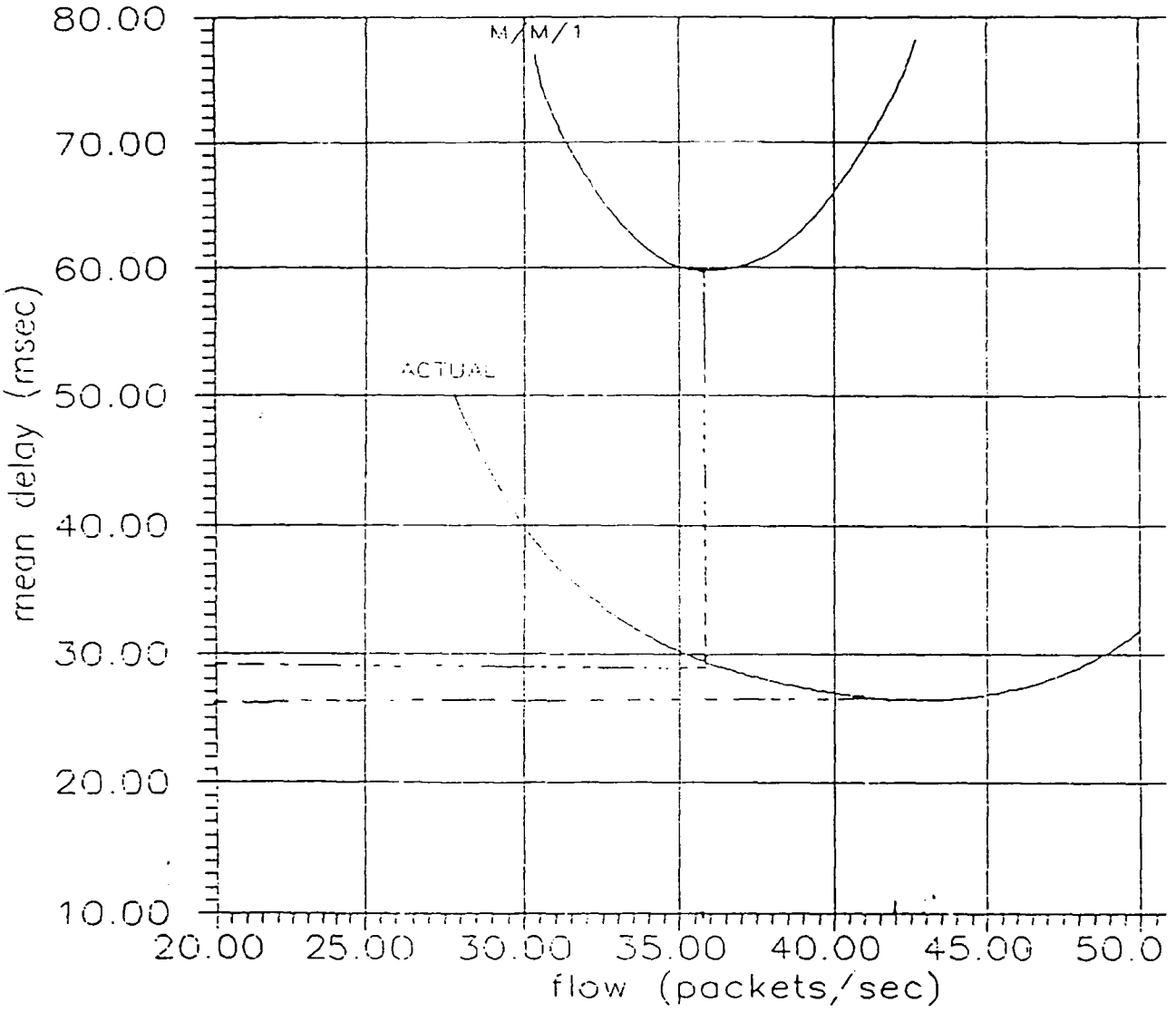


Figure 25 : Approximate M/M/1 and Actual Cost Functions Constructed From Experimental Data
(2 Node Network, Uniform Interarrival Distn,
Triangular Packet Length Distn, Step Size = 0.01,
Observ. Interval = 200 sec (~10000 packets))

6. NETWORKS WITH MULTIPLE TRAFFIC CLASSES.

When multiple traffic classes are present in a network, direct extensions of the routing algorithms we have considered in previous sections are generally not possible. First, each traffic class may seek to optimize a different performance measure. And second, additional overhead is incurred at each node which must now be able to distinguish between the various traffic classes to be processed.

Of particular interest in current network management developments is *real-time traffic*. Real-time traffic is characterized by *deadlines* within which service must be completed (e.g. [22],[23]). Thus, when a packet is transmitted, we require that its destination be reached by some given *deadline*, characteristic of the packet itself or of the traffic class to which it belongs. If this condition is not satisfied, the packet is useless and is considered lost. In such problems, the objective of interest is the minimization of the *probability of loss*, measuring *the fraction of packets that exceed their deadlines.*

In this section, we consider four aspects of network management in the presence of real-time traffic:

- In section 6.1, we address a basic flow control and routing problem for real-time traffic. This is done for a simple model consisting of several links in parallel.
- In section 6.2, we investigate the effect of selecting different performance measures for real-time traffic, possibly providing suboptimal performance, but simpler in nature than the "probability of loss" metric mentioned above.
- In section 6.3, we investigate the issue of "scheduling" real-time traffic, i.e. selecting the next packet to be transmitted at a particular link. We show that the conventional First-In-First-Out (FIFO) policy is often the worst for this type of traffic, and that the Last-In-First-Out (LIFO) policy is often the best one.
- In section 6.4, we address the problem of using simple local control at network nodes in order to minimize the end-to-end (global) probability of loss of real-time traffic.

Many of the results presented in this section have also been reported in the form of technical papers, which may be found in APPENDICES IV-VI.

6.1. Routing and Flow Control of Real-Time Traffic.

We begin by presenting a basic model for our work. Consider a stream of packets arriving to n parallel links according to an arbitrary arrival process with rate λ, as shown in **Figure 26**. The rate of transmission at link i is $\mu_i > 0$, and all links are independent. We assume that there is the

option of rejecting messages before they are assigned to any link; let λ_0 denote the corresponding flow. In addition, let λ_i be the flow to link i, for i = 1,...,n. Finally, let D be a random variable that denotes the deadline associated with a random packet.

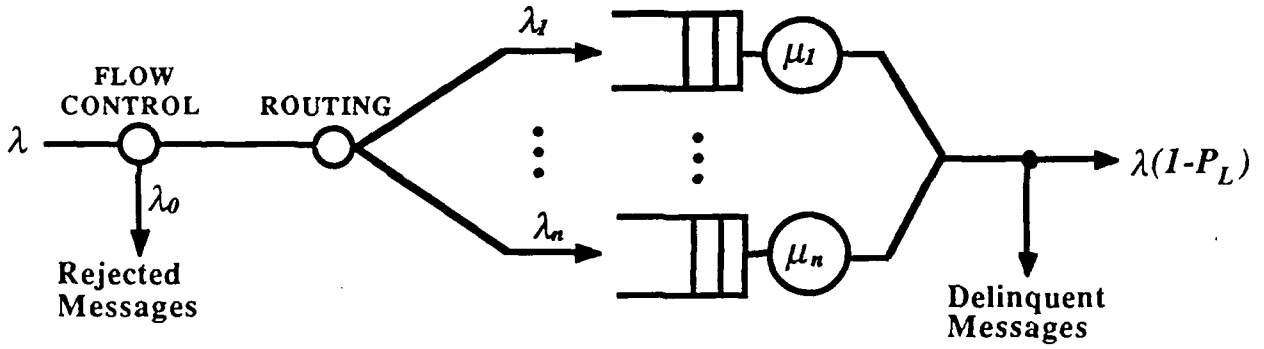


Figure 26: *Link Assignment Model with Admission Policy*

6.1.1. Problem Formulation.

We wish to determine a flow allocation $(\lambda_0,...,\lambda_n)$ so as to minimize the *probability of loss*, where a job is considered *lost* if it is either *rejected* upon arrival or *accepted* but is *delinquent* (i.e., misses its deadline). Let $P^r(\lambda_0)$ be the probability that a random packet is rejected, and let $P^d(\lambda_1,...,\lambda_n)$ be the probability that a packet is delinquent given that it is accepted. Then the total probability of loss is $P_L = [P^r + (1-P^r)\cdot P^d]$. We shall assume in what follows that a packet is delinquent if its total delay T exceeds the deadline D, i.e. $P^d = P[T > D]$. However, the analysis also applies to other definitions of delinquency, such as the event that the queueing time of a packet exceeds the deadline.

It is convenient to translate this flow allocation problem from one of minimizing the *probability of loss* to one of minimizing the *loss rate*, L, where $L = \lambda \cdot [P^r + (1-P^r)\cdot P^d]$. Clearly, for a fixed load λ, the two problems are equivalent. Note that

$$\lambda \cdot P^r = \lambda_0 , \qquad \lambda \cdot (1-P^r) \cdot P^d = \sum_{i=1}^{n} \lambda_i \cdot P_i^d(\lambda_i)$$

where $P_i^d(\lambda_i)$ is the probability that a packet is delinquent given that it is routed over link i. Thus, the performance metric can be written as

$$L(\lambda_0,\ldots,\lambda_n) = \lambda_0 + \sum_{i=1}^{n} \lambda_i \cdot P_i^d(\lambda_i)$$

Furthermore, let

$$f_i(\lambda_i) = \lambda_i \cdot P_i^d(\lambda_i) \qquad i = 1,\ldots,n)$$

$$f_0(\lambda_0) = \lambda_0$$

Then, a general statement of the optimal flow allocation problem is the following:

$$\min_{(\lambda_0,\ldots,\lambda_n)} \sum_{i=0}^{n} f_i(\lambda_i)$$

subject to

$$\sum_{i=0}^{n} \lambda_i = \lambda$$

$$\lambda_i \geq 0, \qquad i = 1,\ldots,n$$

6.1.2. Solution of the Flow Allocation Problem.

Under reasonable assumptions on the nature of the functions $P_i^d(\lambda_i)$ and $f_i(\lambda_i)$, we have been able to solve this problem and characterize the optimal flow allocation in terms of the *packet loss rate sensitivities* $h_i(\lambda_i) = \partial f_i(\lambda_i)/\partial\lambda_i$, for i=0,...,n, where the derivative is defined for $\lambda_i \in [0,\mu_i)$. Details may be found in APPENDIX IV. The main result of our analysis is contained in the following statement:

The optimal solution $(\lambda_0^*,\ldots,\lambda_n^*)$ to the above problem satisfies the following necessary and sufficient conditions:

$$h_i(\lambda_i^*) = \alpha \quad \text{if } \lambda_i^* > 0$$

$$h_i(\lambda_i^*) \geq \alpha \quad \text{if } \lambda_i^* = 0$$

such that:

$$\sum_{i \text{ s.t. } \lambda_i^*>0} h_i^{-1}(\alpha) = \lambda, \qquad \alpha > 0$$

Thus, the problem is reduced to that of determining the value of α satisfying these equations.

An important corollary of our main result is that there exists a critical load value, λ_{crit}, given by

$$\lambda_{crit} = \sum_{i=1}^{n} h_i^{-1}(1)$$

The value λ_{crit} defines the *load threshold*, beyond which rejection of packets will always improve performance. This is illustrated in **Figure 27**, where the *effective throughput*, $\lambda \cdot (1-P_L)$, is always higher with $\lambda_0 > 0$ than with $\lambda_0 = 0$ (no admission policy) as long as $\lambda > \lambda_{crit}$. That is, when the load exceeds the critical value, the optimal flow policy will reject some *positive* flow of packets.

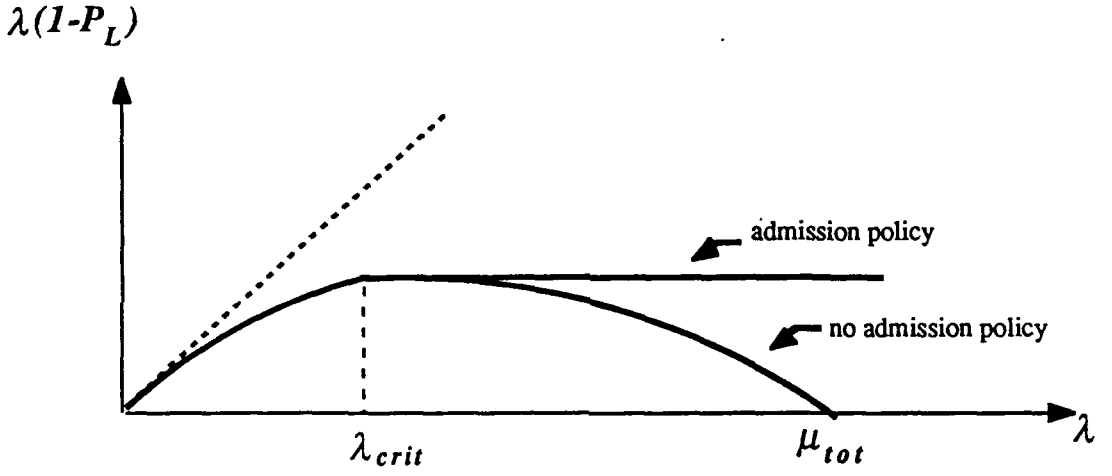


Figure 27: *Load Threshold in Optimal Admission Policy*

Some additional properties of the optimal flow allocation presented in APPENDIX IV.

6.1.3. An Efficient Algorithm for Implementing Optimal Flow Allocation.

The following algorithm can be used to obtain the solution to the problem of minimizing the packet loss rate for the system of **Figure 26**.

Algorithm:

1. Ordering of links:
   - order links so that $h_0(0) \geq h_1(0) \geq h_2(0) \geq \ldots \geq h_n(0)$.
2. Determination of the set of active links, *A*:
   - define $\lambda(m) = \Sigma_{i>m}\, g_i(h_m(0))$,
   - use binary search to find m such that $\lambda(m) \leq \lambda < \lambda(m+1)$,

- $I = \{1, \ldots, m\}$, $A = \{m+1, \ldots, n\}$.

3. Solve equations presented in previous section to obtain $(\lambda_0^*, \ldots, \lambda_n^*)$.

Details and a simple interpretation of this algorithm are provided in APPENDIX IV. It is important to note, however, that the above algorithm can only be used *if analytical expressions of the loss rate functions $f_i(\lambda_i)$ are available*. Examples of systems for which this is the case are also provided in APPENDIX IV.

6.1.4. On-Line Techniques for Implementing Optimal Flow Allocation.

It is often the case that accurate models for the links in **Figure 26** are not available, and that operating conditions frequently change. In such cases, one is limited to observing actual system data (e.g. packet arrival times at a node) and making admission and routing decisions based on that information alone. There are several algorithms suitable for such on-line control schemes, mostly dependent on estimating gradients (sensitivities) of the performance metrics with respect to the parameters of interest. This approach is similar to the techniques (e.g. Perturbation Analysis) presented in section 5.1.2 of this report. In what follows, we simply illustrate the use of such techniques in the context of the real-time traffic flow allocation problem.

If a single link is present, the critical load value identified earlier becomes $\lambda_{crit} = h_1^{-1}(1)$. Recalling the definitions of $f_i(\cdot)$ and $h_i(\cdot)$, this implies that when $\lambda = \lambda_{crit}$, the accepted flow, λ_1^c, is determined from

$$\frac{\partial}{\partial \lambda_1}[\lambda_1 P_1^d(\lambda_1)] = 1$$

which implies that

$$\lambda_1^c = [1 - P_1^d(\lambda_1)] \cdot \left(\frac{\partial P_1^d}{\partial \lambda_1} \right)^{-1}$$

with the right hand side evaluated at $\lambda = \lambda_{crit}$. If this system is directly observed, an estimate of $P_1^d(\lambda_1)$ can be obtained under any admission policy yielding a flow λ_1. Suppose that, at the same time, an estimate of the sensitivity of the delinquency probability with respect to the flow, $(\partial P_1^d / \partial \lambda_1)$, can also be obtained. Then, one would immediately be able to determine whether the load threshold point shown in **Figure 27** has been reached or not, and hence determine the optimal flow allocation. Furthermore, in an adaptive sense, flows can be adjusted on-line until the optimal allocation is found. In many cases, this gradient estimation is accomplished without knowledge of the actual load λ or the link capacity μ_1 [23].

6.2. Two Suboptimal Methods for Routing Real-Time Traffic.

In this section, we consider once again the model of **Figure 26** without the admission control part and with slightly different notation, as shown in **Figure 28.**

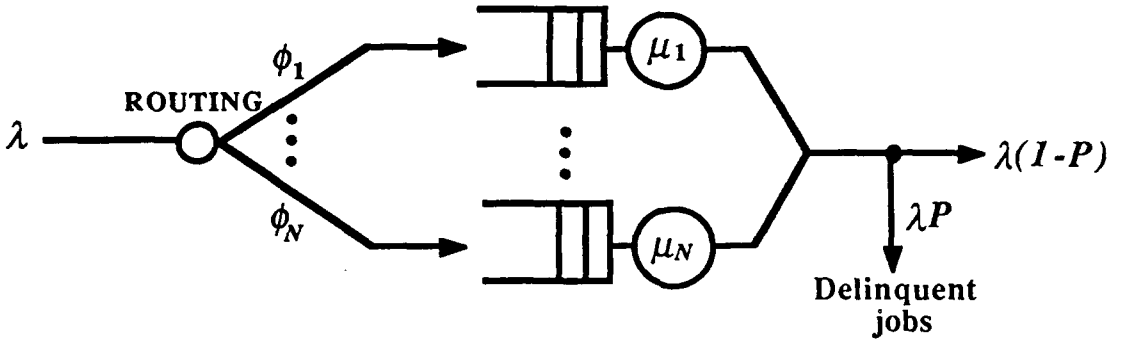


Figure 28: *Routing in N-Server Model*

Ideally, our objective is to determine a routing vector $\Phi = (\phi_1,\ldots,\phi_N)$ so as to minimize the *probability of delinquency*, where a job is considered delinquent if it misses its deadline. Let P_τ be the probability a packet is delinquent with respect to a deadline of τ, and let T be the system time of a random packet. Then we define

$$P_\tau = \Pr\{T > \tau\} = 1 - F_T(\tau)$$

where $F_T(\cdot)$ is the cumulative distribution of T. We assume that the fixed deadline τ is greater than the mean $E[T]$ that results from our choice of routing variables. Note that for a problem to be interesting, τ must be large enough to be a realistic deadline, and the assumption would hold. Clearly, P_τ depends on the routing vector Φ since the distribution of T is affected by the amount of traffic routed to each link. We will refer to $P_\tau(\Phi)$ as the *objective function.* A formal statement of the optimization problem is

$$\min_{\Phi} P_\tau(\Phi)$$

subject to

$$\sum_{i=1}^{N} \phi_i = 1,$$

$$0 \le \phi_i \le \frac{\mu_i}{\lambda}, \qquad i = 1,\ldots,N.$$

where the last two equations represent the flow conservation constraint, and the requirement that the routing variables must be non-negative and sufficiently small that the capacity of a link is not exceeded. Note that although packet flow may not exceed the capacity of a link, it may saturate the link. Let $\Phi^*(\tau)$ be the routing vector that is optimal with respect to the overall performance objective $P_\tau(\Phi)$ for deadline τ. Unfortunately, for most systems of interest, the distribution of T, $F_T(\cdot)$, is unknown and difficult to estimate, hence the objective function cannot be evaluated, and a closed-form expression for $\Phi^*(\tau)$ cannot be found. This motivates our effort to determine simpler performance measures which can still provide "near-optimal" behavior in the sense of minimizing the *probability of delinquency*.

Suppose that the first and second moments of T can be found, either by calculation or by estimation. In addition, assume that T has a finite mean and variance. Then, we define two *performance measures* that depend exclusively on the first two moments of T and not on the distribution itself: (1) the *mean system time*, which we will denote by

$$M = \mathrm{E}[T],$$

and (2) the *normalized distance of the mean system time from the deadline*,

$$D_\tau = \frac{\tau - \mathrm{E}[T]}{\sigma_T},$$

where σ_T^2 is the variance of T. Once again, M and D_τ depend on Φ, as does σ_T. Note that the performance measure M is independent of the value of the deadline, τ. Using the performance measures defined above, we can formulate two sub-optimal methods for routing jobs to the N servers:

$$\min_{\Phi} M(\Phi) \qquad \text{subject to (2) and (3),} \qquad \textbf{(Method 1)}$$

$$\max_{\Phi} D_\tau(\Phi) \qquad \text{subject to (2) and (3).} \qquad \textbf{(Method 2)}$$

Method 1 is simply the existing approach of minimizing the mean delay of packets. Method 2 is a new method of maximizing a *distance* function of the mean and variance.

We define Φ^M and $\Phi^D(\tau)$ to be the routing allocations that are optimal with respect to performance measures M and D_τ for deadline τ. Then, the issue we investigate is whether $P_\tau(\Phi^M) < P_\tau(\Phi^D(\tau))$ or vice versa, and the overall accuracy of these suboptimal methods. To do so, define the *fractional error* in method 1 as

$$\Delta^M(\tau) = \frac{P_\tau(\Phi^M) - P_\tau(\Phi^*(\tau))}{P_\tau(\Phi^*(\tau))},$$

and, similarly for method 2,

$$\Delta^D(\tau) = \frac{P_\tau(\Phi^D(\tau)) - P_\tau(\Phi^*(\tau))}{P_\tau(\Phi^*(\tau))}.$$

Clearly both $\Delta^M(\tau)$ and $\Delta^D(\tau)$ will be positive for all $\tau > 0$ since $P_\tau(\Phi^*(\tau))$ is optimal. For a given τ, we desire to use the method producing the smallest error. We have evidence that there is a critical τ above which method 2 has a smaller error, and below which method 1 has less error. In what follows, we summarize results from two simple system examples for which a complete analytical solution is available; details of the analysis may be found in APPENDIX V.

Example 1: Two Parallel M/M/1 links:

In this case, the fractional errors $\Delta^M(\tau)$ and $\Delta^D(\tau)$ defined above are plotted in **Figure 29** as a function of the deadline τ. Clearly, there is a critical deadline, τ_c, above which $\Delta^D(\tau) < \Delta^M(\tau)$ and below which $\Delta^M(\tau) < \Delta^D(\tau)$; in the figure, we observe that $\tau_c \approx 10$ sec. This example supports our conjecture above, that method 2 results in better performance when the deadline is above some critical value.

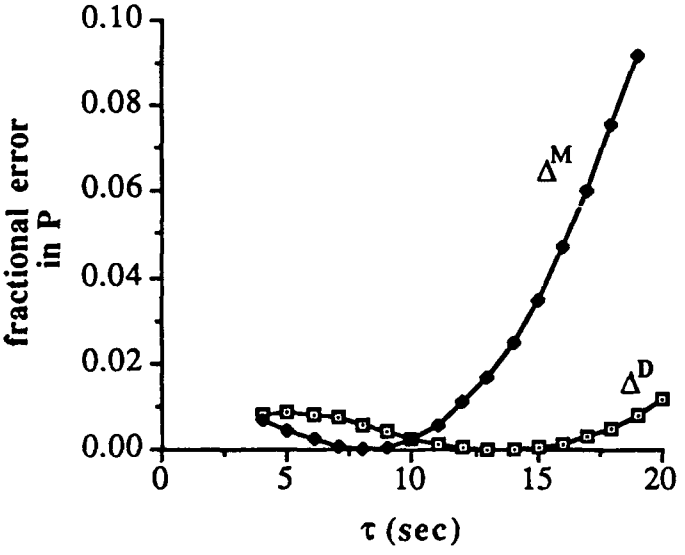


Figure 29: *Error Resulting from Sub-Optimal Methods (Example 1)*

Example 2: Three Parallel M/M/1 links:

As before, in **Figure 30**, the fractional errors, $\Delta^M(\tau)$ and $\Delta^D(\tau)$, are plotted as a function of τ. Note that these curves have the same general nature as do the curves in **Figure 29**. Once again, there exists a critical deadline, $\tau_c \approx 6$, that divides the graph into regions according to which method results in the least error.

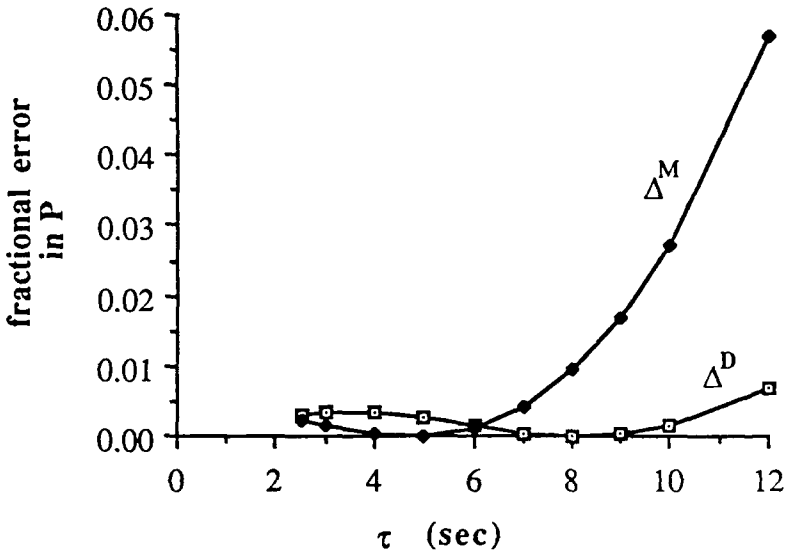


Figure 30: *Error Resulting from Sub-Optimal Methods (Example 2)*

The examples strongly suggest that Method 2 is a promising approach for implementing real-time traffic routing algorithms based on a performance measure which does not require the evaluation of the entire delay distribution of packets. Furthermore, there is substantial validation of our observations which is based on known results from probability theory. Details are provided in APPENDIX V.

6.3. Scheduling Policies for Real-Time Traffic.

In most network applications, it is taken for granted that packets should be processed on a First-In-First-Out (FIFO) basis. This, however, need not be the case for real-time traffic, where, as we have seen, the objective is to maximize the probability that a customer's system time does not exceed a given *deadline*.In fact, it can be explicitly shown that for deadlines that are i.i.d. random variables with concave cumulative distribution functions, the *Last-In-First-Out (LIFO) policy gives the highest probability of success, and FIFO the lowest, over the class of all work-conserving non-*

preemptive service disciplines that are independent of service time and deadline. This is a general result, applicable to G/G/1 link models, with extensions possible to policies that allow preemption in G/M/1 systems, deadlines imposed on queueing (rather than system) time, and multi-server queues. Details of this work can be found in APPENDIX VI (see also [26]).

We have ample experimental results corroborating the optimality of *LIFO*. **Figure 31** illustrates these results for a simple M/M/1 queueing model, where packets are characterized by exponentially distributed deadlines. Here, the probability of packets exceeding their deadlines is plotted as a function of the traffic load λ, with R denoting the packet delay and D its deadline. Clearly, the FIFO policy not only provides poor performance, but it also results in zero effective throughput in overload situations. In contrast, LIFO and LIFO-P (preemption allowed) guarantee a substantial throughput even when the system operates above capacity.

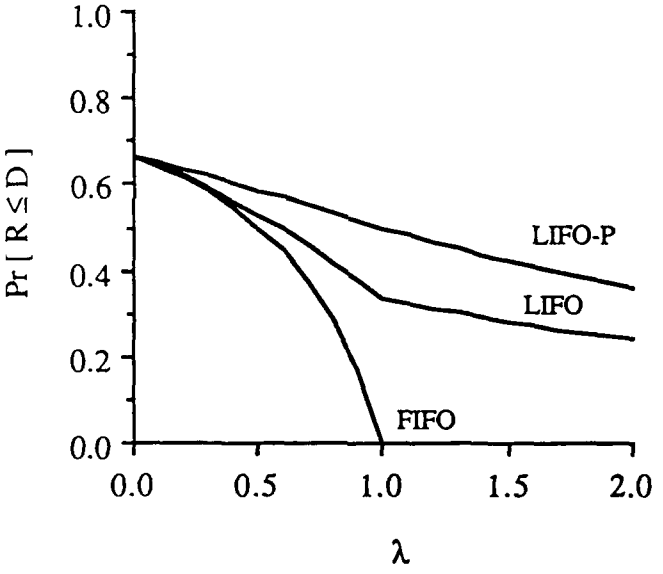


Figure 31: *Comparison of LIFO-P, LIFO, and FIFO Service Disciplines for an M/M/1 Queue with Exponential Deadlines*

A special case of interest arises when packet deadlines are all fixed. It can still be shown that there exists a critical deadline, τ_{crit}, such that LIFO is better than FIFO as long as $D < \tau_{crit}$, i.e.the LIFO policy provides better performance if the fixed deadline is sufficiently tight.

There are numerous problems and related research that emerges from these basic results. Some of the issues currently investigated, as well as possible future directions are presented in section 7.

6.4. Controlling End-to-End Packet Loss in Real-Time Applications

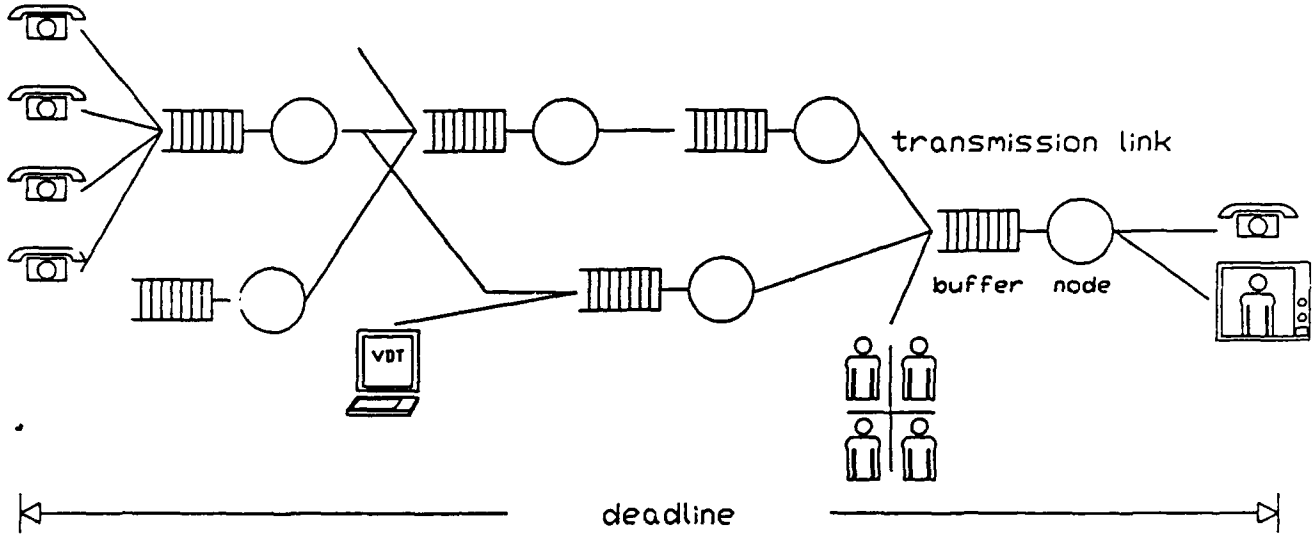


Figure 32: *A sample network*

High-speed networks of the future will have to adequately handle a range of traffic types, including real-time traffic such as voice, video and sensor data. This section summarizes research investigating some approaches that strive to minimize the fraction of packets delayed more than a given end-to-end deadline. (These late packets are referred to as delinquent and are assumed to be useless to the receiver.) The algorithm presented works as a gate at each node in the virtual circuit, removing packets that either will "use up" their alloted time until delinquency at the node or will have so little time left that their probability of making it in time to the receiver is very small. Among the many possibilities of implementing this idea, the simplest removes all packets who will spend more than a prescribed amount of time, the local deadline, in the node or in the buffer. Removing such packets eases congestion at the node itself and at all "downstream" nodes and will be shown to benefit the overall loss rate for a properly set

The section is structured as follows. First, we will evoke in Section 6.4.1. the pertinent characteristics of high-speed networks, in particular those that distinguish them from wide-area packet switched networks presently in use. The network and traffic characteristics also explain why traditional, moment-based performance measures are of limited use (Section 6.4.2.). Since we are investigating a tandem link rather than an individual queue and are also interested in percentiles rather than moments, we have to impose some limitations

on the model, detailed in Section 6.4.4. Section 6.4.5. briefly summarizes how under these assumptions the overall loss can be computed.

With systems of this complexity, simulation becomes essential in increasing the confidence in the analysis, indicating how reasonable certain simplifying assumptions indeed are. Since the simulation result represents only a random sample from the family of all possible outcomes, confidence intervals are used to assess the reliability of the performance estimates. As traditional methods face difficulties under the simulation conditions prevalent here, the method of spectral estimation, described in Section 6.4.6., was investigated and shown to offer some advantages.

Two models for individual queues treated in the literature were used to evaluate the network performance. In the first (Section 6.4.7.), only packets whose system time, that is their waiting plus service time, would fall below the local deadline are admitted to the system. In the second model (Section 6.4.8.), only a bound on the waiting time is imposed. Both models fall under the category of queues with impatient customers and have traditionally been used to model such systems as banks or post offices, where customers do not join a queue if their estimate of the time spent in the service facility exceeds their tolerance level.

Using the two node models mentioned in the last paragraph, we can show that for suitable local parameters we can avail ourselves to a substantial (factor two) performance gain by dropping packets within the network. However, since the optimal parameter value depends on the system parameters such as length of the virtual circuit and system load, Section 6.4.9. concerns itself with the determination of the optimal parameter value and presents a promising heuristic.

This part of the report concludes by summarizing the insights gained and outlining some of the issues that need further study.

6.4.1. Characteristics of Broadband Networks

While most current wide-area data networks operate at transmission speeds of between 4.8 KBits/second to 1.544 MBits/second (T1 rate), it is anticipated that future networks will offer backbone-bandwidths around 1 Gbits/second, with end-user bandwidths in excess of 140 Mbits/second. Only optical networks will be able to supply this transport capacity.

The increase in bandwidth and change in transmission media has at least four important consequences for the analysis and control of data networks. First, in high-speed networks the propagation delay dominates the transmission delay. For example, a 1000 mile transmission line incurs a delay of approximately 9 milliseconds. At a line speed of 10 kBits/second, a typical ARPAnet packet of 1000 bits occupies the transmitter for 100 ms, while at 100 MBits/second this time reduces to 100 μs. Since the packet size will not increase in proportion to the line speed (it may even be lower for some traffic types, notably voice), the packet rate also increases dramatically. Traditionally, general-purpose

computers or programmable communication controllers could easily keep up with incoming packets, reshaping and processing them as needed. At rates of 100,000 packets a second, all processing will most likely have to be restricted to whatever can be executed in hardware. Thirdly, the increased significance of propagation delays combines with the computational limitations to make centralized control and routing algorithms unattractive or impossible. By the time a master node has processed the information gathered and returned any control commands, the load at the node has probably changed drastically. Lastly, optical fibers incur fewer errors than most other media, making it possible to migrate error correction to the edges of the network.

Current networks usually carry a single type of traffic, for example voice or data; a major goal of future highspeed networks is the integration of a number of traffic types within a single switching and transmission network. The traffic types envisioned span the range from telemetry at several hundred bits per second to high-definition television at several hundred megabits per second. a dynamic range of six orders of magnitude. For our purposes, we can divide traffic into two broad-classes, depending on whether it imposes a bound on the delay, the time between start of transmission at the originating node and completion of reception and processing at the receiving node, or not.

For a number of traffic types, packets that exceed a certain delay are no longer of value. Traffic types that fall under this category will be termed *real-time* and include voice (telephony, but not voice mail), broadcast video and teleconferencing for the higher bitrates and telemetry (sensor data and control messages) on the lower end of the spectrum. Network control messages are also delay-sensitive, but are typically to be delivered as fast as possible and are often handled out-of-band or with a priority scheme. For voice, the flow of conversation and echo, caused by acoustical coupling in the handset and impedance mismatches in the analog parts of the circuit, restrict the permissible delay to about 600 ms and 30 to 40 ms. Echo cancellation circuits are used to extend the echo-imposed bound to about 300 ms [31]. Causes of delay in a typical voice circuit are summarized in **Figure 33** and can be divided into fixed, system-dependent delays, distance-proportional delays and variable delays. For most voice coding mechanism, the coding and decoding delay is fixed, while the propagation delay depends directly on the distance. The total switching delay depends on the number of intermediate nodes a circuit passes through, but that delay component remains constant through the duration of a session or call. A packet suffers variable delays by being queued waiting for resources, with the amount of delay depending on the network congestion. This last type of delay distinguishes circuit from packet switched networks and comprises our primary focus of concern.

For other real-time applications, the delay sources are the same. For teleconferencing, considerations similar to telephony apply as voice and video need to be synchronized. The focus shifts slightly in the case of broadcast video, as absolute delays are less important than delay variability, as it directly determines the necessary size of the playout buffer at the receiver. For control applications, delays severely limit the achievable bandwidth and

may make systems uncontrollable (see, for example, [27, p. 284f]). Real-time computing typically involves control applications and thus faces the same restraints, except that communication delays are replaced by computation delays and packets become tasks.

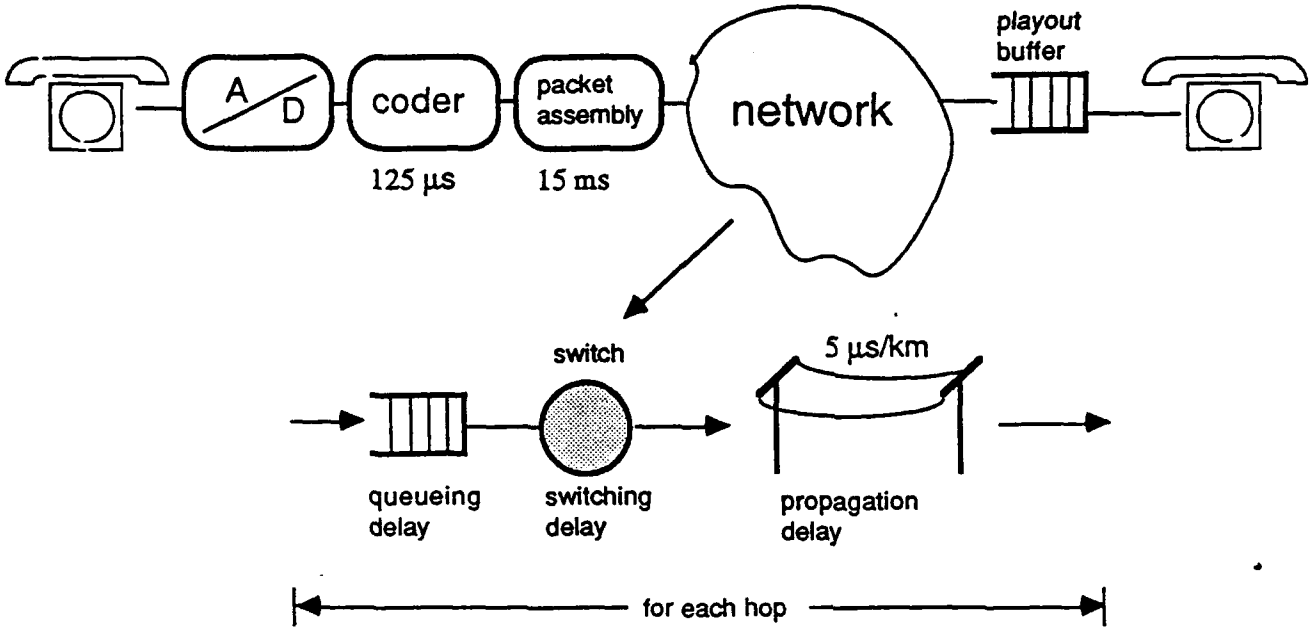


Figure 33: *Delay budget*

6.4.2. Performance Measures

As already discussed in the introduction of Section 6, most of the traditional analysis and control of data networks was concerned with determining and optimizing mean delay (or throughput), on occasion measuring higher moments such as the variance. While this performance measure may be adequate for data circuits, it has very little meaning for delay-bounded traffic. In the latter case, it does not matter exactly when a packet reaches its destination, as long as it does so within the permissible delay window. Thus, percentiles, the fraction of packets arriving within a given amount of time, become the measure of choice. Since we are typically interested in the low losses, we need to concern ourselves with the "tail" of the delay distribution rather than its central moment. We will refer to packets that miss their deadline as *delinquent*. The fraction of delinquent packets will be termed *packet loss*, while the rate of on-time packet arrivals shall be called *goodput*.

The first two moments are poor predictors of packet loss or goodput. As an example, consider Figure 35, showing the cumulative distribution functions for the exponential

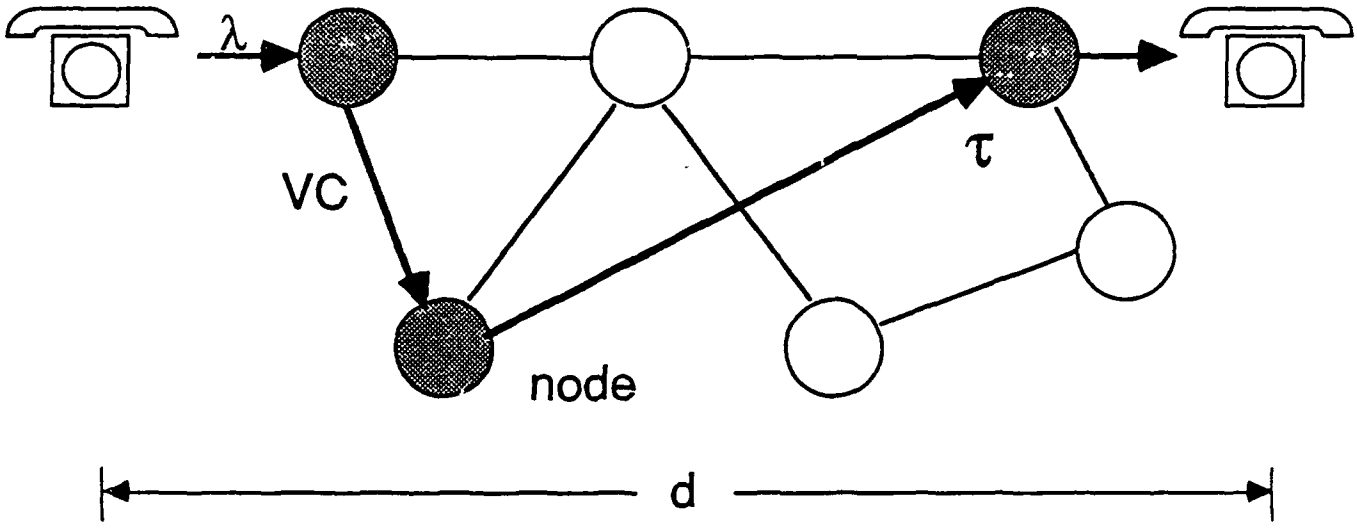


Figure 34: *A virtual circuit with an end-to-end deadline*

(dotted line) and log-normal distribution (solid line), both with mean and variance one. The figure shows that distributions with the same first two moments may have significantly different percentiles.

6.4.3. Local Queue Control

Broadly speaking, queue control policies can be divided into four classes, listed in increasing order of information used:

non-rejecting, non-deadline based These policies serve all arriving customers, but not necessarily in the order of arrival. Examples are standard, infinite-buffer queues with service first-come, first-served or last-come, first served. Within this class of policies we can further distinguish between preemptive and non-preemptive policies.

probabilistic rejection Policies of this class randomly reject packets with a fixed probability, independent of current work or the characteristics of the packet

non-rejecting, deadline based Scheduling jobs with least time until extinction first falls in this category.

rejection based on measure of virtual work Here, customers are rejected if the work queued and left in service exceeds a certain threshold. All finite-buffer queues and queues with impatient customers, such as those described in this report, fall into this category.

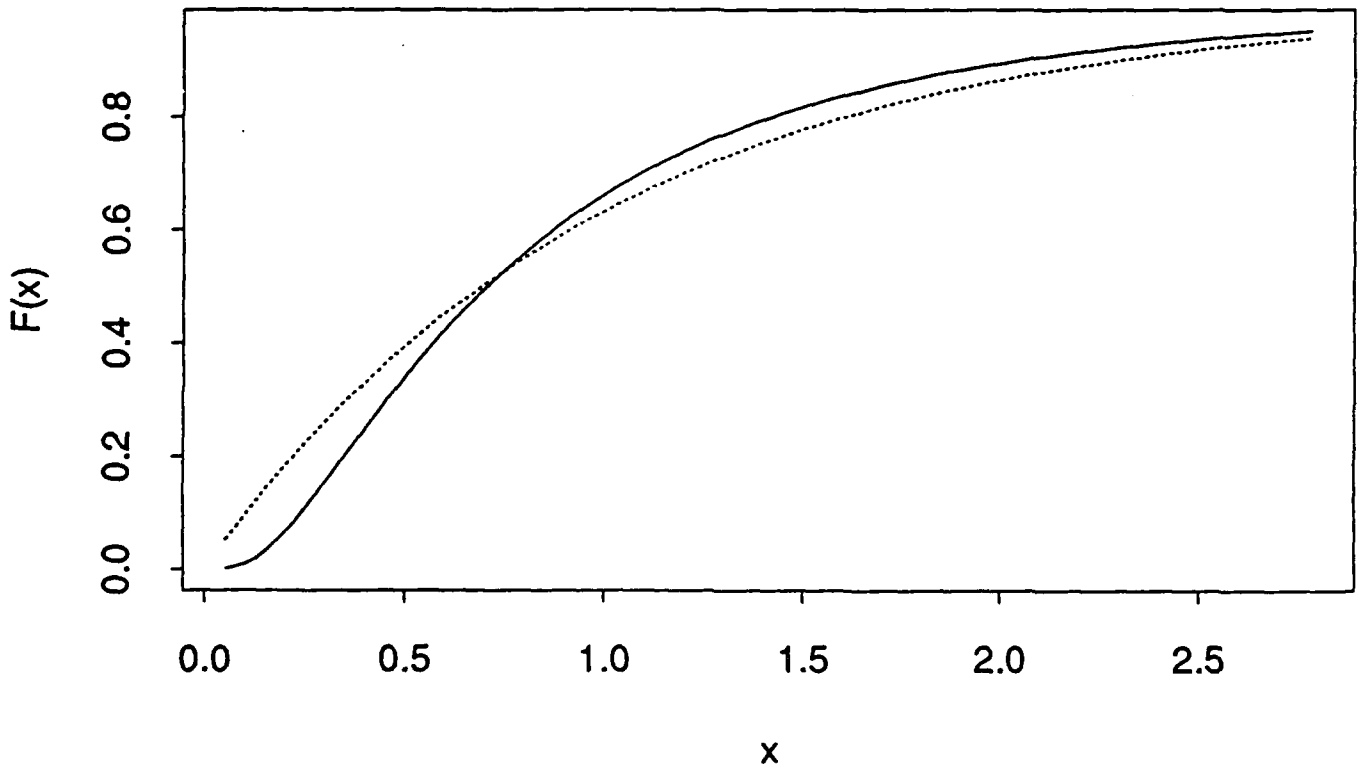


Figure 35: *Two distributions with identical first and second moment*

rejection based on customer deadline A customer that has exceeded or is close to its deadline is rejected, with the threshold possibly depending on the current state of the queue.

All rejection policies can certainly be combined with different service-order policies.

As pointed out in Section 6.4.1., the characteristics of high-speed fiber optics networks with diverse traffic impose a number of restraints on any algorithm that works with individual packets. The algorithm has to be very parsimonious in computation, distributed and should work with local information only. In addition, as with any network algorithm, the amount of information carried in the packet header should be minimal as it translates into a direct loss of usable bandwidth.

We are primarily interested in virtual-circuit based networks, where a session is established and released explicitly. Many high-bandwidth or high-incidence applications such as video and voice have a very natural concept of a session. The virtual circuit approach also reduces the per-packet processing necessary to make routing decisions and is therefore favored for high-speed, compute-bound networks. The algorithm presented, however, could be adapted to datagram-based services by including an estimate of the hop count in the packet header.

The algorithm described in this report aims to improve global performance, that is packet loss rate, by applying a local control mechanism. We assume that we are given a global deadline to be met. This deadline would be typically determined by echo and buffering considerations, as described above and would comprise the variable, that is traffic-dependent, part of the total delay budget.

We also assume that a certain packet loss is acceptable and can be compensated for. Packet loss occurs not only because of missed deadlines, but also because of transmission impairments resulting in bit errors. (If header bits are corrupted, the whole packet appears to be lost.) In real-time applications, retransmissions are not feasible and thus a certain loss cannot be avoided even without delay variations. (Forward error correction may reduce this loss, naturally.) Missing data could be ignored, as might be the case for a digital controller or estimated from previous data, as for voice and video. In the simplest of estimation algorithms, we assume that everything stays the same and reuse the last received packet.

It is clear that a packet that is going to expend its alloted end-to-end delay at a single queue in the virtual circuit stands no chance of beating the end-to-end deadline d and may as well be discarded at that node before it consumes transmission resources at the node and all nodes "down stream". We can also argue that packets that come close to spending their alloted end-to-end time in a single queue will probably miss the deadline, so that on average discarding them early will incur a gain for other packets, outweighing the single loss.

The local control mechanism is designed to be easily implementable in hardware. On arrival, a packet can easily determine how much time it is going to spend in the system as

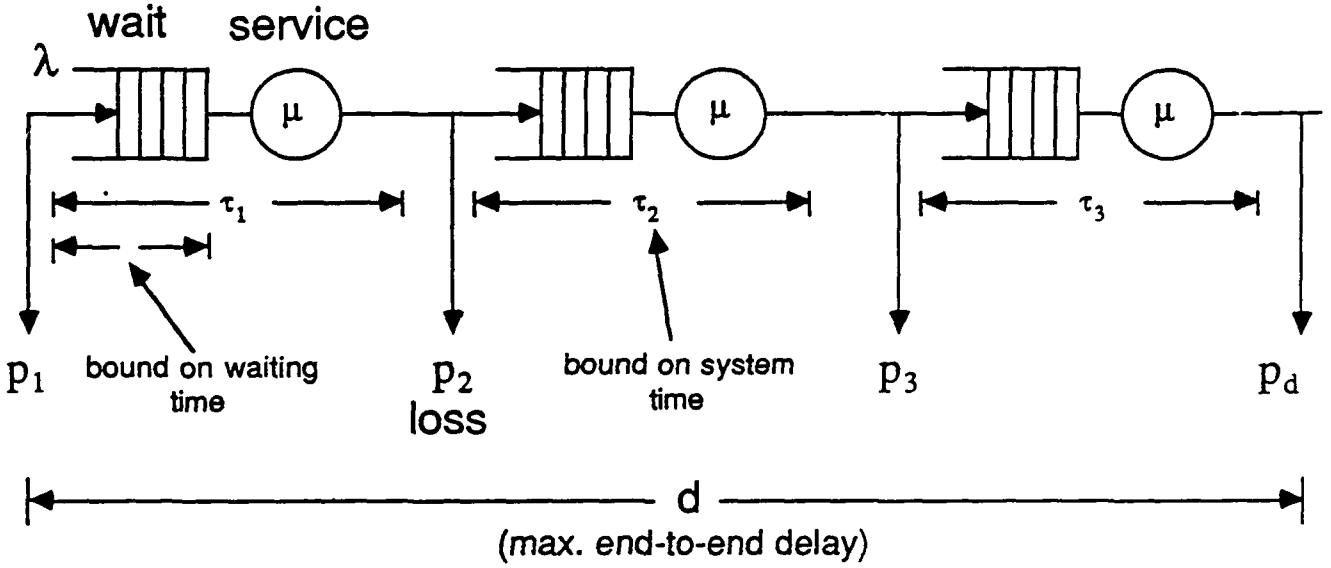


Figure 36: *The virtual circuit model*

the amount of time spent in a transmitter with buffer feeding a circuit of fixed bandwidth is directly proportional to the number of bits stored in the buffer. Thus, two possible control mechanisms are immediately apparent. First, a packet can be discarded if the waiting time, that is the time between arrival and the time the packet enters transmission, exceeds a certain fixed threshold, here labeled τ. In the second approach, we take the packet's transmission (service) time into account. In practical terms, on arrival of a packet, the control mechanism would compare the current number of bits in the buffer to a set threshold and reject the packet if the threshold is exceeded. For the second approach, the control mechanism would add the length of the arriving packet to the number of bits already in the buffer. Since the operation is based on additions and a comparison, a fast hardware implementation is feasible.

In the queueing theory literature, these models are variously refered to as queues with bounded waiting or system time, finite queues or queues with impatient customers. Queues of this type could be used to model the behavior of customers entering a bank or post office, where customers leave the server without obtaining service if their estimate of the waiting time exceeds their tolerance level. The number of bits queued that a packet sees on arrival is termed virtual work.

A number of other queueing-theoretical models could also be applied here. If we raise the granularity from the bit to the packet level, we arrive at a queue with finite customer capacity, such as the M/M/1/K queue.

6.4.4. Modeling Assumptions

Since most real queueing systems, especially networks of queues, are too complex to yield to simple closed-form solutions, we have two choices. First, we can model a real system by a system that approximately reflects the important characteristics and yields to analytical effort. Secondly, we can use bounds which hold for very general systems. Since these bounds are usually not very tight, we chose the first approach. In this section, we will summarize the assumptions that we had to make in order to use the model described and give some indication on how restrictive each assumption is. All of the assumptions are typical for analysis and also allow comparison with other studies. In general, a we model a virtual circuit as a tandem queue with the following additional assumptions:

Service time exponentially distributed The service time is drawn from an exponential distribution with mean $1/\mu$. Other possible choices will be discussed in Section 7.2.

Poisson arrivals Arrivals are assumed to follow a Poisson process, with exponentially distributed interarrival times with mean $1/\lambda$. This assumption is justified if the number of sources feeding into a single queue is large, with each contributing only a small fraction of the total traffic. While experimental evidence indicates that this distribution is a reasonable model for reality for data packet traffic and call arrivals to a telephone exchange, others traffic types, such as a number of voice sources with silence detection, are less well served by the Poisson model.

We make this assumption for both the traffic entering the virtual circuit and the internal traffic. However, even if we grant that the traffic into the first node of the virtual circuit could be represented by a Poisson process, the output process from the first queue is certainly not Poisson, since the queue may have discarded packets. On the other hand, the fraction of packets discarded at every node is typically (in the cases investigated) less than one percent and thus it can be argued that the process will be reasonably close to the input process. In addition, it would seem that the dropping mechanism would eliminate packets in periods of congestion rather than uniformly. Therefore, the actual performance should be better than that predicted by the model. In later experiments, moments were measured for the input streams of all nodes and seen to be very close to those expected for a Poisson process.

Kleinrock assumption The assumption is stated in [30, p. 322] as follows: "Each time a message is received at a node within the network, a new length is chosen independently" according to the service time distribution. This is clearly in violation of reality since packets maintain their length as they traverse the network, but it is claimed in [30] that the effect of the assumption on the performance measure has been shown to be negligible in most interesting networks.

No interfering traffic In a real network, each node is part of a large number of virtual circuits. In the analysis and simulation presented, we concentrate on a single virtual circuit and model the fact that a circuit does not get the full resources of a node by scaling the service rate appropriately (see [24]).

The last three assumptions ensure the independence of each node from the other nodes. For simulation, all assumptions can be dispensed with, however, if the Kleinrock assumption is dropped, interfering traffic has to be introduced explicitly to avoid "pipelining" effects.

6.4.5. Loss Computation

A packet is considered lost if it is either discarded by any of the M nodes it traverses or if the total time spent in the virtual circuit exceeds a set, fixed deadline d.

The probability that a packet is not dropped in any of the M (independent) nodes is

$$P[A] = \prod_{i=1}^{M} P_i[J]$$

where $P_i[J]$ denotes the probability that a packet is admitted to queue i. Note that in computing $P[i]$ the reduced traffic due to packets dropped in nodes $0, 1, \ldots, i-1$ needs to be taken into account, where $\lambda_i = P_{i-1}[J]\lambda_{i-1}$.

Thus, given that the end-to-end distribution of system time for non-discarded packets is $S(d|J)$, the total loss probability is by the law of total probabilities

$$\begin{aligned} P[L] &= (1 - S(d|A))P[A] + (1 - P[A]) \\ &= 1 - S(d)P[A] \end{aligned} \tag{1}$$

The first equation shows the contribution of dropped and late packets to the total loss.

Since each link is assumed to be independent from all others, the VC quantities of interest can be computed from the individual queue measures. The arrival probability $P[A]$ becomes the product of the admittance probabilities, and the end-to-end system time the convolution of the individual system times.

$$S(t|J) = S_1(t|J) * S_2(t|J) * \cdots * S_M(t|J)$$

In the Laplace domain, the convolution becomes multiplication, resulting in

$$S(t|J) = \mathcal{L}^{-1}\left\{\prod_{i=1}^{M} S_i^*(s|J)\right\}$$

with $\mathcal{L}^{-1}$ representing the inverse Laplace transform.

If the Laplace transform of the waiting time is given, an additional integration step needs to be introduced:

$$S(t|J) = \int_0^t \mathcal{L}^{-1} \left\{ \prod_{i=1}^{M} \frac{\mu}{s+\mu} w_i^*(s|J) \right\} dt \tag{2}$$

Similar relations hold for the waiting time.

6.4.6. Confidence Interval Computation

Simulation was used to gain confidence in the approximations and results presented. However, determining points on the system time distribution for multinode systems using simulation is difficult.. Compared to the estimation of means and variances, it is necessary to run simulations for much longer to gain valid results. As part of the investigation, we looked into methods yielding quantitative indicators for the necessary length of a simulation or, given a certain run length, its accuracy.

A simulation is nothing but a statistical experiment, whose outcome represents just one of the many possible outcomes. A confidence interval estimate is an interval centered about a point estimate for the performance measure derived in the traditional way. A valid 90% confidence should include the true (but unknown) performance measure in nine out of ten cases. Classical confidence interval estimation methods assume that the underlying experiments are independent and normally distributed.

For simulations, the literature offers a number of choices on how to arrive at such experiments. For non-terminating systems, that is, systems with no natural end point in time, independent replications and batch means are most commonly recommended. All methods assume that the system measured had reached steady-state, with all transient effects eliminated. The method of independent replications reruns the experiment a number of times (typically, between ten and fifty times), each time using a different set of random numbers to generate arrival and service instances, and then treats each run as one experiment. The method has the disadvantage that we have to eliminate the transient phase from each simulation run, wasting computation time. In addition, there is a tradeoff between running many short or a few long simulations, with effects on the width of the confidence interval and the bias of the point estimate. The method of batch means divides a single run into segments, treating each as an independent experiment. Again, the tradeoff between the number of batches and the length of an individual batch is hard to quantify. The spectral method, proposed by [29], circumvents the problem by eliminating the necessity of independent observations. The method computes an estimate of the sample variance by noting its proportional relationship to the zero point of the power spectral density. The power spectral density is estimated by the magnitude-square of the Fourier coefficients. Since the zero point of the Fourier spectrum estimates the mean rather than the variance, the non-zero Fourier spectrum is extrapolated to the zero point by fitting a

second-degree polynomial to a smoothed version of the logarithmic spectrum. The method outlined was implemented as part of the author's discrete event simulation package and tested extensively for both single queues and queueing networks. It was shown to give reliable estimates for the confidence interval. All confidence intervals appearing on graphs in this report were computed using the spectral method.

6.4.7. Model I: Bounded System Time

In the first model investigated, packets are rejected if their total time in the system, that is, the time spent waiting plus the time of their own transmission, exceeds a given local deadline τ. Some analytical results for this queue are given in [25] and [28]. As part of this work, the missing quantities for a single queue (distribution of virtual waiting time, density and distribution for system time) were found and simplified expressions for existing quantities (density of virtual waiting time, Laplace transform of virtual waiting time) derived. Even after these simplifications, the expression for the Laplace transform still contains two infinite summations which need to be truncated for practical evaluation.

The evaluation of the end-to-end loss for a single value of the parameter τ follows the outline shown in Section 6.4.5.. The computation itself requires between five and fifteen minutes of MicroVAX CPU time for each value of τ, while the simulation consumes about two hours of CPU time for one million data packets.

Two system models were investigated. Both models operate in the heavy-traffic area, with 80% utilization at the first node. Here, the average service rate is normalized to one unit of work per unit time, while the arrival rate is set to 0.8 units of work per unit time. The end-to-end deadline in each case was chosen such that without applying any local control, 5% of the packets entering the virtual circuit are delinquent.

Both cases confirm that simulation and analysis are reasonably close. The difference can at least be partially explained by a simplifying assumption affecting only the analysis, but not the simulation. In the simulation, the input stream to node i is equal to the output stream of node $i-1$, while the analysis presumes that each input stream is Poisson and independent of the others.

For the simulation, the length of the transient phase was estimated by plotting the loss probability for each packet arrival over the first 1000 packet arrivals. The loss probability was estimated by repeating the experiment one hundred times and smoothing the resulting data points. A typical result is shown in **Figure 39**. To be on the safe side, the length of the transient period was estimated from the data at 2,000 packet arrivals. The erratic behavior of the curve near the right edge is due to the decreasing window size of the smoothing algorithm.

The **Figure 38** shows that proper choice of the local deadline τ can result in a significant drop in the end-to-end loss, here from 5% to about 2.5%. As is to be expected, it is better to err on the side of caution in selecting the local deadline; the overall loss

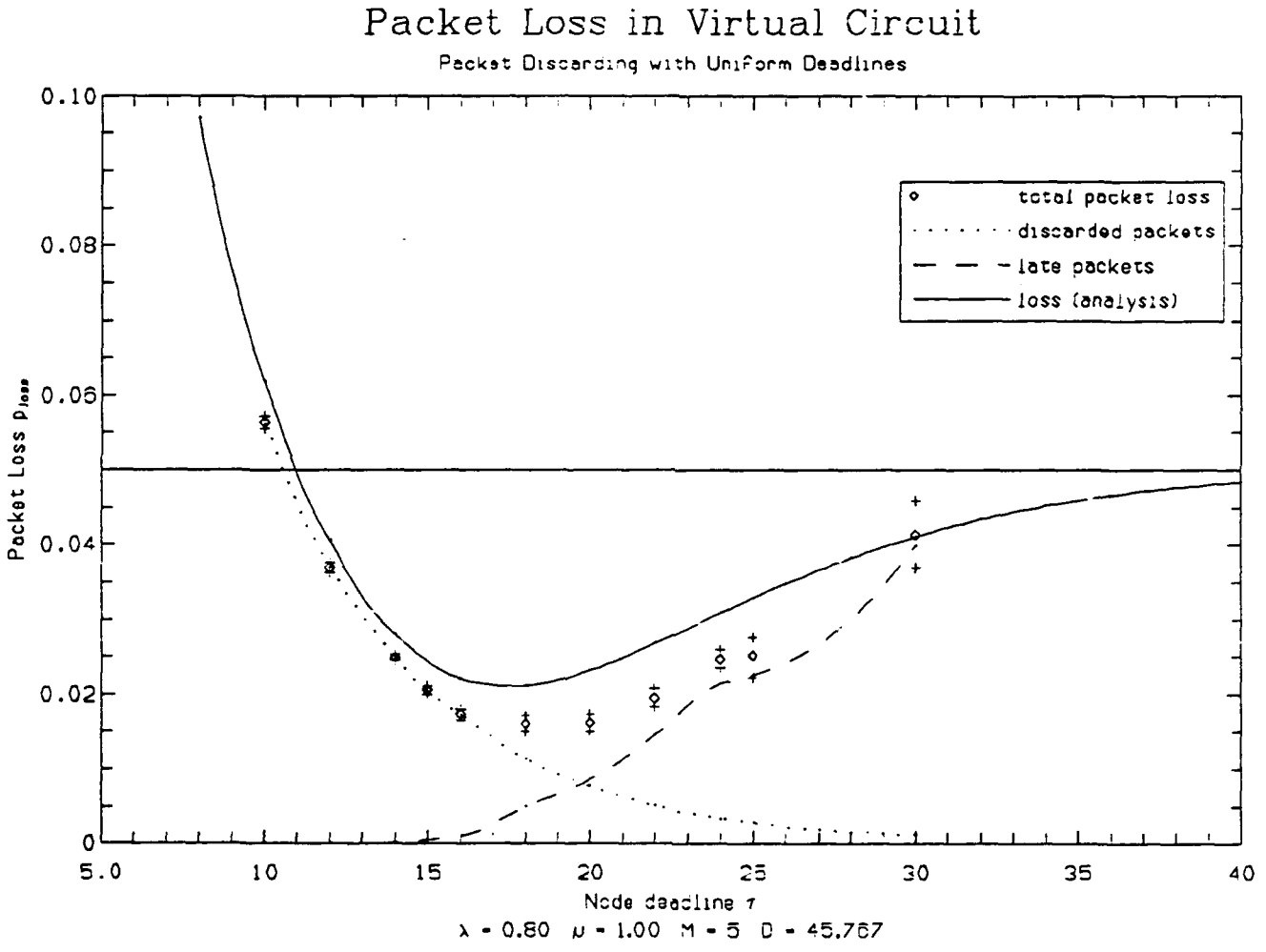


Figure 37: *Virtual circuit with five nodes*

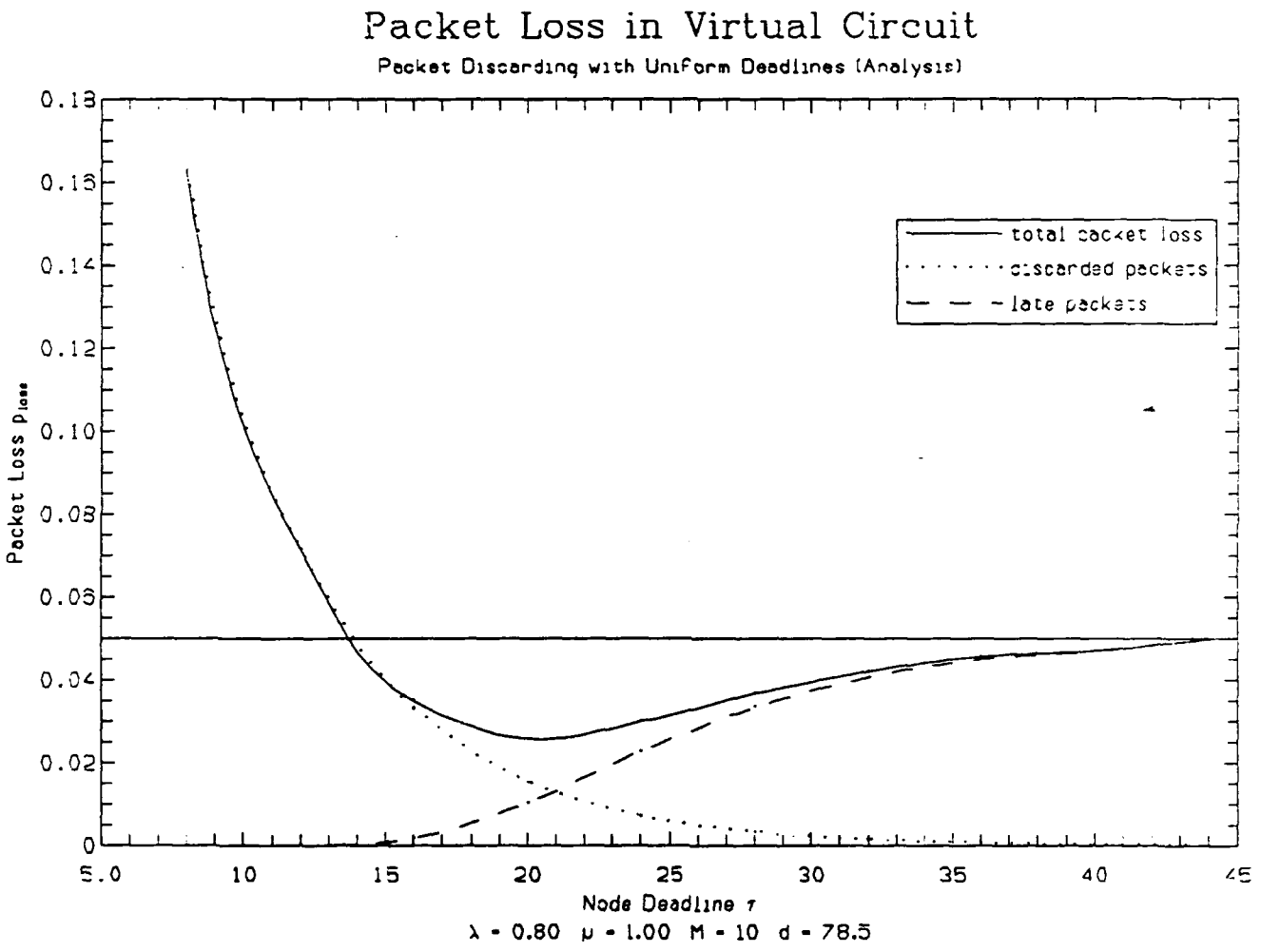


Figure 38: *Virtual circuit with ten nodes*

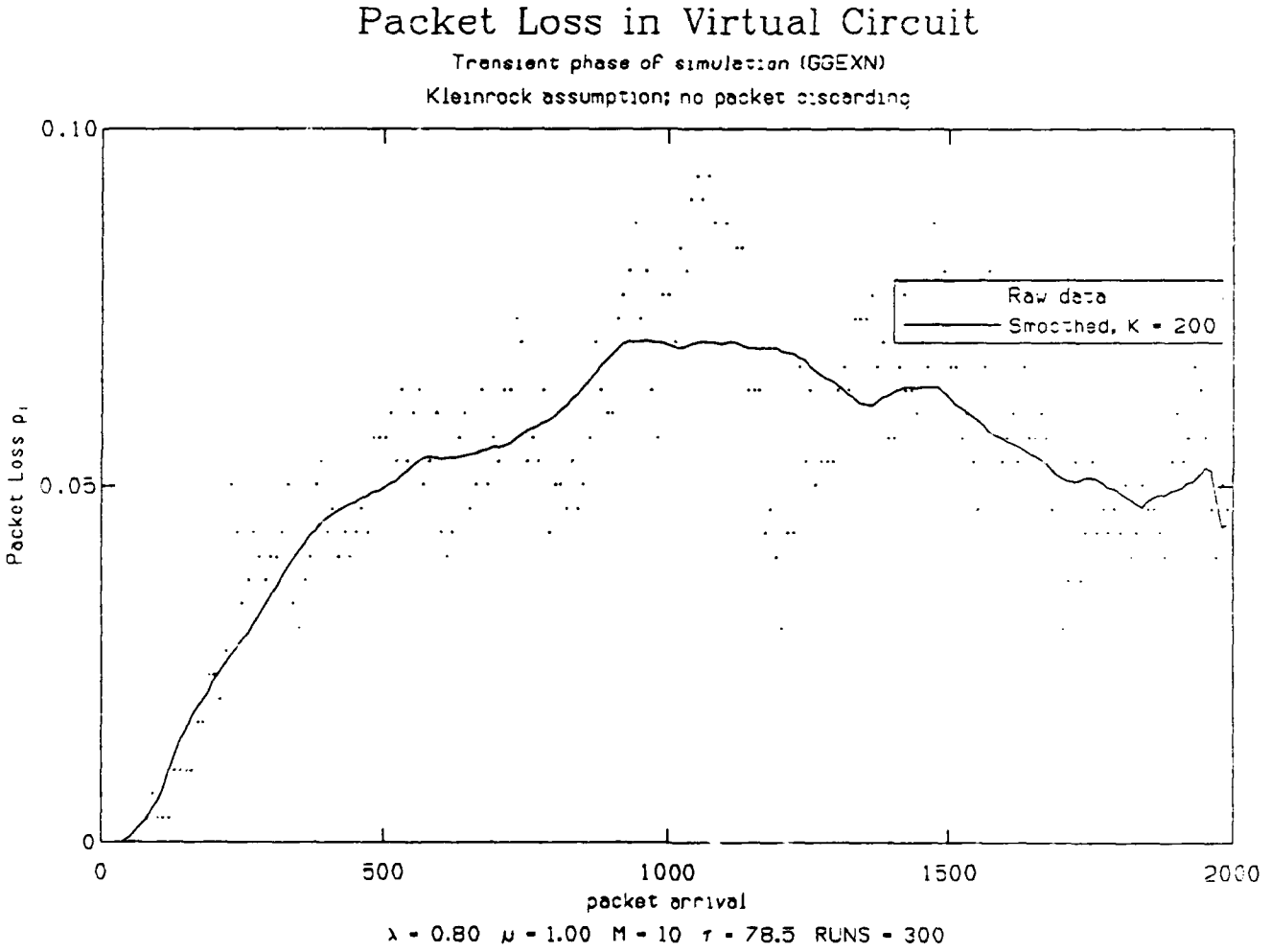


Figure 39: ***Transient phase of virtual circuit simulation***

rises steeply as the deadline is tightened. The results also show that simply dividing the end-to-end deadline, here 78.5, by the number of nodes in the circuit, ten, results in far too stringent local deadlines.

The above results are confirmed by looking at the case of five nodes, as shown in **Figure 37**. Again, the end-to-end loss can be cut in half by applying local control. As before, the best performance achieved under simulation is slightly better than that predicted by analysis. On the graph, the confidence intervals around the simulation points (shown as diamonds) are marked with plus signs and are seen to be sufficiently narrow. The graph also shows that there is no simple, proportional relationship between the length of a virtual circuit and the optimal deadline. For ten nodes, the optimal deadline falls at approximately 20, about 0.25 of the end-to-end deadline, while for five nodes the optimal deadline is 17 and the factor 0.37. However, the graphs also show that using a single local deadline of 20 would give near-optimal performance in both cases.

In order to investigate the distortion of the arrival process by the dropping mechanism, we measured the first three moments of the interarrival time of *admitted packets* and compared it to the values of an exponential distribution with the same mean. Here, the local deadline was set to $\tau = 12$, that is, more packets than optimal were being dropped in transit. It can be seen from **Table 6.4.1** that the simulation distribution has slightly lower higher moments than would hold for an exponential distribution, possibly explaining the better performance observed for the simulation as compared to the analytical prediction.

stage	1st moment	2nd moment		3rd moment	
	measured	measured	theory	measured	theory
1	1.262	3.171	3.185	11.92	12.06
2	1.272	3.207	3.236	12.07	12.35
3	1.282	3.251	3.287	12.33	12.63
4	1.290	3.299	3.328	12.58	12.89
5	1.299	3.333	3.375	12.74	13.15

Table 6.4.1: *Moments of interarrival time for five-node network*

The control mechanism in model I has a slight "philosophical" disadvantage. Since the packet's own service time is figured in the decision of whether to accept or reject it, long packets stand a slightly lower chance of being admitted than short ones. For the range of parameters investigated here, the effect is slight since the optimal local deadline is about a factor of twenty higher than the average packet service time. The second model, presented below, avoids this problem.

6.4.8. Model II: Bounded Waiting Time

The second model is a slight modification of the first, in that only the waiting time of an arriving customer is taken into account. This model was studied by [26], providing admission probability, density of the waiting time distribution and its first moment. Fortunately, the waiting time density has a closed-form Laplace transform, with no infinite summations. However, since the waiting time is a distribution with finite support (that is, it is strictly zero outside a finite interval), the transform equation is non-rational, that is, it contains an exponential term. This innocent-seeming fact has severe repercussions for the determination of optimal deadlines, as discussed in Section 6.4.9..

The results are similar to the first model and not reproduced here.

6.4.9. Uniqueness of Minimum and Optimal Local Deadline

Based on the experimental evidence and numerical examples, this section investigates two conjectures concerning the uniqueness and location of the optimal local deadline and outlines the problems facing a formal proof. Since the Laplace transform expression for model I contains an infinite summation, the product and its inverse (not explicitly determinable) are unwieldy candidates for further analysis. As mentioned, things look brighter for the second model, as a closed-form solution, in both the transform and time domain, exists. However, it can be shown that the time-domain expression for the end-to-end waiting time in a virtual circuit with M nodes contains on the order of M^3 terms of the form $t^k e^{-t+c} u(-t+c)$, with k and c differing from term to term. For $M = 10$, the exact number is 440. Additional difficulties arise since the $u(-t+c)$ in each term forces it to zero outside a certain interval.

Conjecture 1 *The function of packet loss vs. local deadline has a unique minimum.*

A sufficient condition for the uniqueness of a minimum of a function is its concavity. (A function is concave if it lies below any straight line connecting any two of its function values.) It is clear and can be shown from the derivatives (for model II) at zero and infinity that the function of packet loss with respect to local deadline is not concave. Alternatively, we could show that the first partial derivative with respect to τ has a unique zero — given the difficulties in deriving an explicit expression discussed above for the inverse of the Laplace transform a rather unlikely prospect.

Thus, we turn our attention to the more practical issue of finding the optimal τ. In the following, *late loss* refers to the fraction of packets that traverse the whole virtual circuit without being dropped, but miss their end-to-end deadline. The fraction of packets that are dropped by the control mechanism within the network is called *drop loss* for short. The *uncontrolled loss* is incurred if the network applies no local control.

Conjecture 2 *The optimal local deadline τ is approximately at the intersection of late loss and drop loss. At that point, drop loss and late loss each contribute half of the total loss.*

If this conjecture holds, an on-line control algorithm to find the optimal local deadline suggests itself. We can assume that the node at the receiving edge of the network can determine how many packets were lost and late, for example by looking at sequence counters within the packets. Thus, it could send commands back to the transit nodes to raise or lower the local deadline until the drop loss equals the late loss.

The proof of this conjecture is even more difficult than showing uniqueness. Since the relation holds only approximately, we would need general closed-form expressions parametrized on τ for the total loss as well as the loss components. Note that the conjecture does not hold for a single queue, as the optimal local deadline for a single queue can easily be shown to be the end-to-end deadline, i.e., all packets that are admitted make the end-to-end deadline; in other words, drop loss is equal to total loss.

7. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS.

This section summarizes our main conclusions based on the results and observations presented in sections 2-6, and outlines some ongoing and future research directions. Section 7.1 summarizes our conlusions pertaining to the comparative performance study and issues of gradient estimation (a more detailed account of these conclusions was already presented in section 4.7). Section 7.2 presents conclusions and open problems related to multiclass traffic networks.

7.1. Comparative Performance Study.

One of the objectives of the performance study was to compare the gradient-based routing algorithms with the shortest-path-based ones. *If traffic conditions are somewhat stationary, gradient-based algorithms outperform their shortest path counterparts.* The reason for this lies in the basic nature of their respective operations. While gradient-based algorithms lead (under suitably chosen parameters) to oscillation-free mean delay characteristics at steady state, shortest path algorithms are known to suffer from instability problems manifested in the form of oscillations. The shortest path datagram algorithms, such as the ARPANET algorithm, are prone to oscillations due to lack of any control mechanism to regulate the traffic adjustments at each update instant. On the other hand, the CODEX algorithm, which is a shortest path virtual circuit algorithm, displays oscillatory behavior due to its mechanism of rerouting of virtual circuits. However, we have found that the CODEX algorithm is less sensitive to rerouting probabilities and rapidly changing traffic conditions when applied to a large network.

On the other hand, we have found that *oscillations are considerably reduced under light load conditions.* Under these conditions, the simplicity of shortest path algorithms becomes their main attractive feature. This is probably why the ARPANET algorithm is still a favorite in light traffic datagram environments. But, under moderate to heavily loaded conditions it performs very poorly compared to gradient-based routing algorithms such as Gallager's. This poor performance is accentuated in networks characterized by links with different capacities.

Under quasistatic conditions, *where traffic changes appreciably with the initiation and termination of source-destimation sessions, a gradient-based algorithm may not be able to adapt to the changing conditions as rapidly as a shortest path algorithm.* The adaptivity of gradient-based algorithm depends on the step size chosen. There is a tradeoff involved in the choice of this parameter. While a larger value provides greater adaptivity, a smaller one ensures oscillation-free convergence to the steady state. Consequently, one has to select an appropriate value depending on

the specific requirements, leading to the strong argument for *mechanisms to dynamically adjust this step size parameter.*

Apart from the issue of steady state oscillations, the other important performance measure is the nominal mean delay achieved by the algorithm. *Gradient-based routing algorithms are designed to produce optimal mean delays, whereas shortest path algorithms are heuristic in nature with no guarantee of optimality.* This was apparent in our performance comparison study. Under moderately high load conditions, the mean delay in the ARPANET algorithm was about 20 times higher than the one obtained through Gallager's algorithm; this difference is much less severe under lightly loaded conditions. In contrast to the ARPANET algorithm, the CODEX algorithm does achieve near-optimal mean delays. This is attributed to the fact that the CODEX algorithm computes shortest paths based on incremental link delays rather than the absolute link delays as in the case of the ARPANET.

Despite their many attractive features, gradient-based routing algorithms have not yet become popular in practice. One of the reasons has been the lack of simple and efficient gradient estimation techniques. However, *the use of Perturbation Analysis techniques provides an opportunity for obtaining good gradient estimates without imposing restrictive modeling assumptions on the network environment.* Moreover, PA estimation does not impose any significant computational burden making it infeasible under real-time conditions. With the use of such techniques, the future of gradient-based routing algorithms looks promising, and future research direction could be directed at further exploring the implementation-related issues for such algorithms.

Another objective of our work was the performance comparison of virtual circuit routing and datagram routing. We have found that one must take several factors into consideration before drawing any conclusions relevant to this issue. The nature of traffic conditions in the network is one such important factor. As was mentioned earlier, under stationary traffic conditions the steady state performance of the VC-based CODEX algorithm displays oscillatory behavior. This can be attributed to the VC rerouting mechanism. In fact, any VC-based algorithm incorporating a rerouting mechanism is bound to suffer from the problem of steady state oscillations. On the other hand, it was seen that the rerouting mechanism is indispensable, since, otherwise, no adaptivity would be possible. However, we have shown that *oscillations can be limited by imposing an upper bound on the traffic rates of individual VC's.* Whether such a constraint is desirable or not is a high level issue, and a subject for further research.

Virtual circuit routing algorithms are designed to operate in *session-oriented environments.* As a result, they are highly adaptive to traffic changes caused by the initiation and termination of sessions. In contrast, in such changing traffic environments, datagram routing algorithms face

temporary congestions; the time required to clear such congestions depends on the selection of the respective algorithm parameters.

While comparing virtual circuits to datagrams, a naturally arising issue is that of *resequencing*. This is a potential drawback for datagram networks, since (a) additional processing at the end nodes is required, and (b) packets experience additional buffering delays at the end nodes. We have found this additional delay to be insignificant compared to the overall mean system delay. However, the extra processing requirement may be a more crucial factor, depending on the network under consideration.

We have also found that *differences in link capacities* play a critical role in the ARPANET algorithm. This may be a significant factor in designing routing algorithms for future networks, intended to operate in a multi-media environment.

In the context of examining different gradient estimation techniques for routing algorithms, some interesting observations were made in section 5. These observations provide insight which is currently only opening up directions for further research. As we have seen, the key issues relate to two types of errors introduced by analytical approximations: (a) traffic rate estimation errors which are magnified in commonly used expressions for mean delay derivatives, and (b) modeling errors, which force any routing algorithm one might select to higher mean delys. The effect of these errors, however, may greatly depend on the properties of different arrival and packet transmission processes, which remain to be investigated.

7.2. Multiclass Traffic Networks.

Our work thus far has focused on network management issues related to *real-time traffic*. We have shown (section 6.1) that *when the performance measure of interest is the probability of losing packets which exceed a given deadline, it is advantageous to apply flow control that deliberately rejects part of the incoming traffic.* Thus, some packets which might be lost anyway are prevented from accessing the network, a fact which contributes to increasing the fraction of the accepted packets making their deadlines. Approaches towards developing explicit algorithms to implement flow control and routing for the case of n parallel links were presented in section 6.1.3 and 6.1.4. Because of the complexity of the problem, we have also found that some simpler performance measures (such as the one presented in section 6.2) can be used to obtain adequate suboptimal performance.

Our work on flow control and routing of real-time traffic appears to be the point of departure for a variety of interesting extensions. First, it is possible that our explicit results for the M/M/1

link models may be extended to more general situations, so as to obtain analytical solutions using the simple algorithm presented in section 6.1.3. Second, of great interest is the wide applicability of stochastic gradient estimation and on-line optimization techniques that take advantage of observed system data with little or no information concerning parameters or distributions involved in an analytical model. Motivated by the results in [23], we believe that such techniques will indeed be useful in this framework, in order to develop adaptive schemes for flow control and routing in networks with real-time traffic.

Another challenging problem concerns the extension of our flow allocation problem for real-time traffic to the case of *dynamic policies*, i.e. policies taking into account some system state information (e.g. instantaneous queue lengths in making link assignments). Our preliminary results suggest that this is indeed a fruitful area for further research.

In section 6.3, we considered the issue of scheduling real-time packets. Our key result is that for G/G/1 queues where customers have deadlines drawn from a concave cumulative distribution function, *a LIFO service discipline is optimal over the set of work-conserving non-preemptive policies that are independent of service demands and deadlines*. We also show FIFO to be the worst policy. In addition, we extend the basic result to policies that allow preemption and to more general systems.

These results suggest several interesting questions pertaining to the desirable properties of the LIFO service discipline. For example, we can ask how a LIFO policy compares to a FIFO policy with state-dependent admission control. There is, in fact, evidence that LIFO can do better even in this case. Another idea is to make the same comparisons in a system where customers are removed from the queue as soon as they are detected to have exceeded their deadline. Further, we can question whether LIFO is optimal for any deadline distributions that do not satify the concavity constraints.

Regarding our work reported in section 6.4, we have presented a method of improving end-to-end loss in networks with real-time constraints using a strictly *local control policy*. The policy drops those packets at transit nodes that are perceived to have little chance of meeting their end-to-end deadline, thereby alleviating congestion. The policy is very suitable for hardware implementation, requires no central control and no exchange of control information.

While the policy investigated thus far seems promising, a number of issues remain to be resolved, most of which depend on the type of network the policy is to be applied in. It remains to be investigated how the policy performs under different traffic characteristics (interarrival and service time distribution and means), as well as for a mix of traffic types and non-uniform virtual-circuit loading.

The control algorithm for setting the local deadline also poses some interesting questions. Should τ be set locally, without feedback from the receiving end and based on estimates of the traffic and the knowledge of the VC length encoded in the packet or the VC lookup table? Or should the end node measure the magnitude of the loss components and send control messages back to the transit nodes?

Other policies such as minimum-laxity or last-in, first-out scheduling, can also improve the end-to-end loss. Does the policy described here do better by itself than other policies or can the scheduling policies be improved by selectively dropping packets? We would expect to do better by incorporating packet-specific information such as time remaining until deadline in the decision whether to accept it or not. The amount of improvement and possible implementations are still open questions.

REFERENCES

[1] P.A. Humblet, and S.R. Soloway, "Algorithms for Data Communication Networks - Part 1", Codex Corp., 1986.

[2] P.A. Humblet, S.R. Soloway, and B. Steinka, "Algorithms for Data Communication Networks - Part 2", Codex Corp., 1986.

[3] R.G. Gallager, "A Minimum Delay Routing Algorithm Using Distributed Computation", *IEEE Trans. Commun.*, COM-23, pp. 73-85,1977.

[4] J.M. McQuillan, I. Richer, and E.C. Rosen, "The New Routing Algorithm for the ARPANET", *IEEE Trans. Commun.*, COM-28, pp. 711-719,1980.

[5] L. Tymes, "Routing and Flow Control in TYMNET", *IEEE Trans. Commun.*, COM-29, pp. 392-398,1981.

[6] L. Fratta, M. Gerla, and L. Kleinrock, "The Flow Deviation Method: An Approach to Store-and-Forward Communication Network Design", *Networks*, Vol. 3, pp. 97-133, 1973.

[7] E.M. Gafni, "Convergence of a Routing Algorithm", MS Thesis, Univ. of Illinois, Dept. of Electrical Engineering, 1979.

[8] D.P. Bertsekas, E.M. Gafni and R.G. Gallager, "Second Derivative Algorithms for Minimum Delay Distributed Routing in Networks", *IEEE Trans. Commun.*, COM-32, pp. 911-919,1984.

[9] A.A. Economides, P.A. Ioannou and J.A. Silvester, "Decentralized Adaptive Routing for Virtual Circuit Networks Using Stochastic Learning Automata", *Proceedings of IEEE INFOCOM' 88*, New Orleans, March 1988.

[10] W.K. Tsai, "Optimal Quasi-static Routing for Virtual Circuit Networks Subjected to Stochastic Inputs", Ph.D. Thesis, Dept. of Elect. and Comp. Sc., M.I.T., 1986.

[11] W.K. Tsai, J.N. Tsitsiklis, and D.P. Bertsekas, "Some issues in Distributed Asynchronous Routing in Virtual Circuit Data Networks", *Proceedings of 25th IEEE Conf. on Decision and Control,* Athens, Greece, 1986.

[12] D.P. Berstsekas, "Dynamic Models of Shortest Path Routing Algorithms for Communication Networks with Multiple Destinations", *Proc. 1979 IEEE Conf Decision and Contr*, Ft. Lauderdale, FL, pp. 127-133.

[13] D.P. Berstsekas, "Dynamic Behaviour of Shortest Path Routing Algorithms for Communication Networks", *IEEE Trans Auto Contr.*, AC-27, 1982, pp. 60-74.

[14] Y.C. Ho, and C.G. Cassandras, "A New Approach to the Analysis of Discrete Event Dynamic Systems", *Automatica*, 19, pp. 149-167, 1983.

[15] Y.C. Ho, X. Cao, and C.G. Cassandras, "Infinitesimal and Finite Perturbation Analysis for Queueing Networks", *Automatica*, 19, pp. 439-445, 1983.

[16] U. Pape, "Implementation and Efficiency of Moore - Algorithms for the shortest Route Problem", *Mathematical Programming*, Vol. 7, pp. 212-222, 1974.

[17] E. Dijkstra, "A Note on Two Problems in Connection with Graphs", *Numer. Math.*, Vol. 1, pp. 269-271, 1959.

[18] C.G. Cassandras, M.V. Abidi, D.F. Towsley, "Distributed Routing With On-line Marginal Delay Estimation", to appear in *IEEE Trans. Commun.*, 1989.

[19] M.V. Abidi, "Use of Perturbation Analysis Techniques for Optimal Routing in Computer Networks", M.S. Thesis, Univ. of Massachusetts, Amherst, Dept. of Electrical and Computer Engineering, 1987.

[20] D.A. Reed, C. Kim, "Packet Routing Algorithms for Integrated Switching Networks", *Proc. ACM SIGMETRICS Conf.*, May 1987, pp. 7-15.

[21] S.S. Lam, "Store-and-Forward Buffer Requirements in a Packet Switching Network", *IEEE Trans. Commun.*, Vol. COM-24, No. 4, April 1974, pp. 394-403.

[22] J.F. Kurose, and R. Chipalkatti, "Load Sharing in Soft Real-Time Distributed Computer Systems", *IEEE Trans. on Computers*, C-36, 8, pp. 993-1000, 1987.

[23] C.G. Cassandras, S.G. Strickland, and J-I. Lee, "Discrete Event Systems with Real-Time Constraints: Modeling and Sensitivity Analysis", *Proc. 27th Conf. Decision and Control,* pp. 220-225, December 1988.

[24] A. Bhargava, J.F. Kurose, D. Towsley, and G.VanLeemputt, "Performance Comparison of Error Control Schemes in High Speed Computer Comunication Networks", *Proceedings of the 1988 IEEE INFOCOM,* New Orleans, April 1988.

[25] J. W. Cohen, "Single Server Queues with Restricted Accessibility", *Journal of Engineering Mathematics,* 3(4):265-284, October 1969.

[26] B.T. Doshi, and H. Heffes, "Overload Performance of Several Processor Queueing Disciplines for the M/M/1 queue", *IEEE Transactions on Communications,* COM-34(6):538-546, June 1986.

[27] G.F. Franklin, J.D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*. Addison Wesley, Reading, Massachusetts, 1987.

[28] B. Gavish, and P.J. Schweitzer, "The Markovian Queue with Bounded Waiting Time", *Management Science,* 23(12):1349-1357, August 1977.

[29] P. Heidelberger and P.D. Welch, "A Spectral Method for Confidence Interval Generation and Run Length Control in Simulations", *Communications of the ACM,* 24(4):233-245, April 1981.

[30] L. Kleinrock, *Queueing Systems - Computer Applications*, Volume 2, Wiley-Interscience, New York, 1976.

[31] R.P. Singh, and S. Singhal, "Voice Transmission over Mixed Packet- and Circuit-Switched Networks", *Proceedings of the 1988 IEEE INFOCOM*, pp. 126-134 (2A.3), New Orleans, April 1988.

APPENDIX I

PAPE'S VERSION OF MOORE'S SHORTEST PATH ALGORITHM

There are numerous shortest path algorithms available in the literature. Pape's version of the Moore algorithm [], which is employed in the CODEX routing scheme, is one of the most efficient ones. In this appendix we present an outline of Moore's algorithm followed by Pape's implementation methodology. A coded version of the algorithm appears as the subroutine *spath* in our implementation given in Appendix II.

Moore's Algorithm

Consider a network with set of nodes X and directed arcs (i,j) having distances d_{ij}. We want to find the shortest paths from a node to all other nodes. This is done by starting with a set of nodes containing only the source node i and then applying the following algorithm at each step until this set becomes a null set (depleted of all elements). Let S_v denote this set in the vth step and D^v_{ik} the shortest distance from node i to node k obtained at step v.

The initial conditions are :

$$S_0 = \{i\} ;$$

$$D^0_{ik} = \infty, \quad k \neq i ;$$

$$D^0_{ii} = 0 ;$$

The algorithm is applied for steps $v = 1, 2, \ldots$ and terminates if $S_v = \{\}$.

At step v :

$$D^v_{ik} = \min \{ D^{v-1}_{ik}, D^{v-1}_{il} + d_{lk} : l \in S_{v-1} \}, k \in S_v ;$$

$$S_v = \{ k : k \in S_v, D^v_{ik} \neq D^{v-1}_{ik} \} ;$$

Implementation

The above algorithm can be implemented in various ways. The data structures used for this purpose to a great extent determines the efficiency. The following implementation in the form of a label correction algorithm [] using a linked list has been found to be the most efficient for different types of graph structures [].

We start with the linked list containing the source node only and initialize the shortest paths to D(j) = 0 for j = i, and D(j) = a very large value for any other value of j. While the list is non-empty, we remove the node at the head of the list and examine all its outgoing links. Let us call this node h. If there exists a link (h,k) such that

D(k) > D(h) + length (h,k), then we make the assignment
D(k) = D(h) + length (h,k)

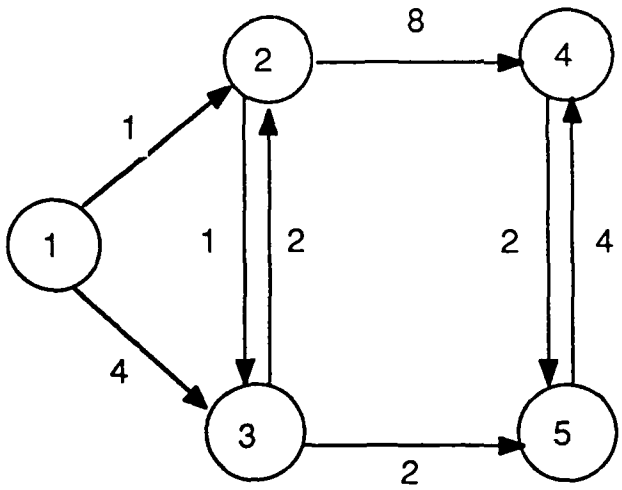
Now if k has been on the list before but is currently not there, it is put at the head of the list. On the other hand, if it has never been on the list, it is put at the tail of the list.

We keep track of the existence of the node in the list by realizing the list in the form of an array (one entry for every node). A node entry contains 0 if it has never been on the list, -1 if it was on the list but not currently there or a positive number indicating the next node in the list . For the last node on the list, the entry is a very large number.

While the above algorithm is executed, we go on updating the table at every node which contains the next node on the shortest path from itself to every other node in the network.

Example

To make the operation of the above implementation clear, we present an illustrative example. Consider the network shown below with the link length indicated adjacent to the links.



We want to find the shortest paths from node 1 to all other nodes. As far as routing is concerned, we should just know the next node to which node 1 may forward a packet depending on the destination node. So we present below, the node list, the routing assignments at node 1 as mentioned above, and the distances to other nodes from node 1 as the result of each iteration of the algorithm.

iteration # 0 iteration # 1

list : 1

dest	dist	next node
2	inf	-
3	inf	-
4	inf	-
5	inf	-

list : 2, 3

dest	dist	next node	
2	1	2	
	3	4	3
	4	inf	-
	5	inf	-

iteration # 2

list : 3, 4

dest	dist	next node
2	1	2
3	2	2
4	9	2
5	inf	-

iteration # 3

list : 4, 5

dest	dist	next node
2	1	2
3	2	2
4	9	2
5	4	2

iteration # 4

list : 5

dest	dist	next node
2	1	2
3	2	2
4	9	2
5	4	2

iteration # 5

list : 4

dest	dist	next node
2	1	2
3	2	2
4	8	2
5	4	2

iteration # 6

list : NIL

dest	dist	next node
2	1	2
3	2	2
4	8	2
5	4	2

Hence the algorithm terminates with the above results in iteration # 6, i.e. to go to any node from node 1 a packet has to be forwarded to node 2.

APPENDIX II

SIMULATION CODE FOR THE CODEX ALGORITHM

APPENDIX II

The SIMAN files of the coded CODEX algorithm are presented here. Though several different modifications were made to obtain the results under different experimental conditions, the codes presented below bring out the essential features of the algorithm.

MODEL FILE

```
BEGIN;
          STATION,42-47;                          FOR REROUT. SCHED
ISPLIT    BRANCH,2:
             ALWAYS,INTR:                         GO FOR REROUTING
             ALWAYS,CYCLE;                        WAIT FOR NEXT TURN
CYCLE     DELAY:CO(9):NEXT(ISPLIT);               DELAY BY OBSERV INT.
INTR      EVENT:2:DISPOSE;                        EXEC REROUT PROCEDURE
;
          STATION,1-19;                           APPROP. SD PAIR
PROD      EVENT:4;                                CREATE A PACKET FOR VC
;
SPLIT     BRANCH,2:
             ALWAYS,SYSTEM:
             ALWAYS,GEN;                          ONE TO SYS,ONE TO GEN
GEN       DELAY:ED(M):NEXT(PROD);                 EXP INTERARRIVAL TIME
;
SYSTEM    ASSIGN:A(3)=TF(7,A(1)):MARK(2);         ASSIGN THE SOURCE
          ASSIGN:A(4)=TF(8,A(1));                 ASSIGN THE DESTN
          ASSIGN:A(5)=DP(8,1);                    TYPE(LONG/SHORT)
          ASSIGN:X(3)=A(3)+19;                    STN TO ENTER NET
          ASSIGN:A(7)=A(3);                       LAST NODE VISITED
;
          ROUTE:0.0,X(3);                         ENTERS AT SOURCE
          STATION,20-25;                          EACH NODE IS A STATION
          BRANCH,1:
             IF, (A(4)+19).EQ.M, REACHED:
             ELSE,PROCEED;                        REACHED DEST OR GO ON ?
;
REACHED   EVENT:3;
          TALLY:17,INT(2):DISPOSE;                ADD PACK DEL & DISPOSE
;
PROCEED   EVENT:1;                                DET FORWARDING LINK
          ROUTE:0.0,J+25;                         SEND TO CORRECT LINK
          STATION,26-41:MARK(6);                  EACH LINK IS A STATION
          QUEUE,M-25;                             WAIT FOR IDLE LINK
          SEIZE:LINK(M-25);                       GRAB THE LINK
          ASSIGN:X(4)=M-25;                       ASSIGN LINK NO
          ASSIGN:X(6)=TF(9,X(4));                 ASSIGN LINK CAPACITY
          DELAY:X(6)*TF(10,A(5));                 PRODUCE LINK DELAY
          RELEASE:LINK(M-25);                     RELEASE THE LINK
          BRANCH,1:
```

```
            IF, A(5) .EQ. 1, SHT:
            ELSE, LNG;                                  CHECK PACKET SIZE LNG
SHT         COUNT:M-25:NEXT(CONT);                      INC SHORT COUNT
LNG         COUNT:M-9:NEXT(CONT);                       INC LONG COUNT
CONT        TALLY:M-25,INT(6);                          TALLY LINK DELAY
            ASSIGN:J=A(7)+19;                           ASSIGN NEXT NODE
            ROUTE:0.0,J;                                SEND TO THE NODE
;
END;
```

EXPERIMENTAL FILE

```
BEGIN;
PROJECT,CODEX,B.P. MOHANTY,7/6/88;
DISCRETE,6000,8,16,47;
RESOURCES:1-16,LINK;

ARRIVALS:1,STATION(1),0.0,1,1:2,STATION(2),0.0,1,2:
         3,STATION(3),0.0,1,3:4,STATION(4),0.0,1,4:
         5,STATION(5),60000.0,1,5:6,STATION(6),60000.0,1,6:
         7,STATION(7),60000.0,1,7:8,STATION(8),60000.0,1,8:
         9,STATION(9),120000.0,1,9:10,STATION(10),120000.0,1,10:
         11,STATION(11),120000.0,1,11:12,STATION(12),120000.0,1,12:
         13,STATION(13),180000.0,1,13:14,STATION(14),180000.0,1,14:
         15,STATION(15),180000.0,1,15:16,STATION(16),180000.0,1,16:
         17,STATION(17),240000.0,1,17:18,STATION(18),240000.0,1,18:
         19,STATION(19),240000.0,1,19:20,STATION(42),0.0,1,20:
         21,STATION(43),9000.0,1,21:22,STATION(44),18000.0,1,22:
         23,STATION(45),27000.0,1,23:24,STATION(46),36000.0,1,24:
         25,STATION(47),45000.0,1,25;

DISTRIBUTIONS:1,EX(1,1):2,EX(2,2):3,EX(3,3):4,EX(3,3):5,UN(4,4):
              6,UN(4,4):7,UN(5,5):8,EX(6,6):9,EX(3,3):10,EX(3,3):
              11,EX(2,2):12,EX(3,3):13,EX(7,7):14,EX(7,7):15,EX(7,7):
              16,EX(7,7):17,UN(5,5):18,UN(4,4):19,EX(6,6);

PARAMETERS:1,60.0:2,200.0:3,300.0:4,80,100:5,60,80:6,70.0:7,90.0:
           8,0.2,1,1.0,2:9,56000:10,250000:11,2000:12,0.05,1,0.1,2,
           0.15,3,0.20,4,0.25,5,0.3,6,0.35,7,0.4,8,0.45,9,0.5,10,
           0.55,11,0.6,12,0.65,13,0.7,14,0.75,15,0.8,16,0.85,17,
           0.9,18,0.95,19,1.0,20;

TABLES:1,1,1,0,1,2,0,0,3:
       2,1,1,0,0,4,5,0,6:
       3,1,1,7,0,0,8,0,0:
       4,1,1,0,9,10,0,11,0:
       5,1,1,0,0,0,12,0,13:
       6,1,1,14,15,0,0,16,0:
       7,1,1,1,1,1,1,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6:
       8,1,1,3,4,5,6,1,5,6,1,2,5,1,3,6,1,2,3,1,2,4:
       9,1,1,1,1,5.83,0.78,5.83,5.83,1,1,0.78,1,0.78,5.83,1,0.78,1,1:
       10,1,1,2.29,18.29:
       11,1,1,0,5,8,11,14,17:
       12,1,1,0,0,9,0,15,18:
       13,1,1,1,0,0,12,16,0:
       14,1,1,2,0,0,0,0,19:
       15,1,1,3,6,10,0,0,0:
       16,1,1,4,7,0,13,0,0:
       17,1,1,60,200,300,300,90,90,70,70,300,300,200,300,90,90,90,90,
       70,90,70;
```

```
TALLIES:1,LNK 1 TO 2 DEL:2,LNK 1 TO 3 DEL:3,LNK 1 TO 6 DEL:
        4,LNK 2 TO 3 DEL:5,LNK 2 TO 4 DEL:6,LNK 2 TO 6 DEL:
        7,LNK 3 TO 1 DEL:8,LNK 3 TO 4 DEL:9,LNK 4 TO 2 DEL:
        10,LNK 4 TO 3 DEL:11,LNK 4 TO 5 DEL:12,LNK 5 TO 4 DEL:
        13,LNK 5 TO 6 DEL:14,LNK 6 TO 1 DEL:15,LNK 6 TO 2 DEL:
        16,LNK 6 TO 5 DEL:17,NETWORK DELAY;

COUNTERS:1,LNK 1 TO 2 STRAF:2,LNK 1 TO 3 STRAF:3,LNK 1 TO 6 STRAF:
         4,LNK 2 TO 3 STRAF:5,LNK 2 TO 4 STRAF:6,LNK 2 TO 6 STRAF:
         7,LNK 3 TO 1 STRAF:8,LNK 3 TO 4 STRAF:9,LNK 4 TO 2 STRAF:
         10,LNK 4 TO 3 STRAF:11,LNK 4 TO 5 STRAF:12,LNK 5 TO 4 STRAF:
         13,LNK 5 TO 6 STRAF:14,LNK 6 TO 1 STRAF:15,LNK 6 TO 2 STRAF:
         16,LNK 6 TO 5 STRAF:17,LNK 1 TO 2 LTRAF:18,LNK 1 TO 3 LTRAF:
         19,LNK 1 TO 6 LTRAF:20,LNK 2 TO 3 LTRAF:21,LNK 2 TO 4 LTRAF:
         22,LNK 2 TO 6 LTRAF:23,LNK 3 TO 1 LTRAF:24,LNK 3 TO 4 LTRAF:
         25,LNK 4 TO 2 LTRAF:26,LNK 4 TO 3 LTRAF:27,LNK 4 TO 5 LTRAF:
         28,LNK 5 TO 4 LTRAF:29,LNK 5 TO 6 LTRAF:30,LNK 6 TO 1 LTRAF:
         31,LNK 6 TO 2 LTRAF:32,LNK 6 TO 5 LTRAF;

REPLICATE,100,0,56000,NO;
END;
```

DISCRETE EVENT MODEL FILE

```
C A must in any discrete event model file; initializes every run

      SUBROUTINE PRIME

C Declaration of common variables, both SIMAN as well as user defined

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,
     *           TFIN,J,NRUN

      COMMON/NETVAR/nodes,ivcs,link(10,10),idfor(10,10),ivfor(400,10),
     *              dellnk(10,10),rsp(10,10),rlp(10,10),edel(10,10),
     *              ivcflg(400),vcst(400),rscnt(400),rlcnt(400),npt

      DIMENSION rlen(10,10),ifor(10,10),dsrt(10,10)

      IF (NRUN .EQ. 1) THEN

C init last update time X(1), rerouting prob X(7), no. of VC components
npt

         X(1) = 0.0
         X(7) = 0.05
         npt = 20

C open files to store results of interest

         OPEN(UNIT=9,FILE='result.dat',STATUS='UNKNOWN')
         REWIND(9)
         WRITE(9,*) '           NRUN      AVG.DEL',X(7)
         CLOSE(9)

         OPEN(UNIT=10,FILE='topo.dat',STATUS='OLD')
         READ(10,*) nodes
         READ(10,*) ivcs
         DO 10, i = 1,nodes
            READ(10,*) (link(i,k),k = 1,nodes)
10       CONTINUE
         CLOSE(10)

C initialize datagram routing assignments

         DO 20, i = 1,nodes
            DO 20, k = 1,nodes
            ifor(i,k) = 0
            rlen(i,k) = link(i,k)
20       CONTINUE

         CALL spath(rlen,ifor,dsrt)
```

```
         DO 25, i = 1,nodes
            DO 25, k = 1,nodes
            idfor(i,k) = ifor(i,k)
25       CONTINUE

C initialize VC related parameters and routing assignments

         DO 30, i = 1,ivcs
            DO 30, ln = 1,npt
            ivcflg((i-1)*npt+ln) = 0
                 rscnt((i-1)*npt+ln) = 0
                 rlcnt((i-1)*npt+ln) = 0
                 DO 30, k = 1,nodes
               ivfor((i-1)*npt+ln,k) = 0
               id = TF(8,REAL(i))
               ivfor((i-1)*npt+ln,k) = idfor(k,id)
30       CONTINUE

      ENDIF
      RETURN
      END
C directs different events to their corresponding subroutines

      SUBROUTINE EVENT(L,I)

      IF (I .EQ. 1) THEN
         CALL EV1(L)
      ELSEIF (I .EQ. 2) THEN
         CALL EV2(L)
      ELSEIF (I .EQ. 3) THEN
         CALL EV3(L)
      ELSEIF (I .EQ. 4) THEN
         CALL EV4(L)
      ENDIF

      RETURN
      END

C*********************************************************************
 C determines the outgoing link according to routing assignments

      SUBROUTINE EV1(L)

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,
      *             TFIN,J,NRUN
      COMMON/NETVAR/nodes,ivcs,link(10,10),idfor(10,10),ivfor(400,10),
      *               dellnk(10,10),rsp(10,10),rlp(10,10),edel(10,10),
      *               ivcflg(400),vcst(400),rscnt(400),rlcnt(400),npt

C determine the last node

      i = A(L,7)
```

```
C find the next node and hence the link number from tables first for
C datagram packets and then for VC packets depending on their lengths

      IF (A(L,5) .EQ. 1) THEN
         k = A(L,4)
         n = idfor(i,k)
         r = REAL(n)
         CALL SETA(L,7,r)
         J = TF(i,r)
      ELSE
         k = (A(L,1)-i)*npt + A(L,8)
         n = ivfor(k,i)
         r = REAL(n)
         CALL SETA(L,7,r)
         J = TF(i,r)
      ENDIF

      RETURN
      END

C*****************************************************************************
 C execute the rerouting procedure according to CODEX algorithm

      SUBROUTINE EV2(L)

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,
      *             TFIN,J,NRUN
      COMMON/NETVAR/nodes,ivcs,link(10,10),idfor(10,10),ivfor(400,10),
      *              dellnk(10,10),rsp(10,10),rlp(10,10),edel(10,10),
      *              ivcflg(400),vcst(400),rscnt(400),rlcnt(400),npt

      DIMENSION lflag(10,10),vlen(10,10),iuvfor(10,10),uvsrt(10,10)

C for each source-destination pair determine the VC no.

      ln = A(L,1) - 19
      IF (NRUN .GT. 1) THEN
      DO 20, iv = 1,nodes
         nsd = TF(ln+10,REAL(iv))
         DO 20, lc = 1,npt
            IF ((nsd .NE. 0) .AND.
      *          (RA(10) .LE. X(7))) THEN
            k = (nsd-1)*npt + lc
            IF ((ivcflg(k) .EQ. 1) .AND.
      *           ((TNOW - vcst(k)) .GE. 10000)) THEN
            p = REAL(nsd)
            is = TF(7,p)
            id = TF(8,p)

C find the VC loads in terms of short and long packets respectively
```

```
            rspvc = rscnt(k) / (TNOW - vcst(k))
            rlpvc = rlcnt(k) / (TNOW - vcst(k))

C assign link lengths to the links used by the VC. links indicated by
C setting lflag(link) to 1 from 0 as you go from source to dest.

            DO 35, i = 1,nodes
            DO 35, n = 1,nodes
               lflag(i,n) = 0
35          CONTINUE

            itmp1 = is
            itmp2 = ivfor(k,itmp1)
150         IF (edel(itmp1,itmp2) .LT. 0) THEN
            vlen(itmp1,itmp2) = 100000
            ELSE
            im = TF(itmp1,REAL(itmp2))
            rm = REAL(im)
            costwi = edel(itmp1,itmp2)
            flow = (rsp(itmp1,itmp2)-rspvc)*TF(10,1.0) +
     *                 (rlp(itmp1,itmp2)-rlpvc)*TF(10,2.0)
            costwo = flow / ((1/TF(9,rm))-flow)
            vlen(itmp1,itmp2) = costwi - costwo
            IF (vlen(itmp1,itmp2) .GT. 100000) THEN
               vlen(itmp1,itmp2) = 100000
            ENDIF
            ENDIF
            lflag(itmp1,itmp2) = 1
            IF (itmp2 .NE. id) THEN
            itmp1 = itmp2
            itmp2 = ivfor(k,itmp2)
            GOTO 150
            ENDIF
C assign link lengths to other links not used by the current VC
            DO 40, i = 1,nodes
            DO 40, n = 1, nodes
               IF ((link(i,n) .EQ. 1) .AND.
     *                (lflag(i,n) .EQ. 0)) THEN
                  im = TF(i,REAL(n))
                  rm = REAL(im)
                  IF (edel(i,n) .LT. 0) THEN
                  vlen(i,n) = 100000
                  ELSE
                  flow = (rsp(i,n)+rspvc)*TF(10,1.0) +
     *                (rlp(i,n)+rlpvc)*TF(10,2.0)
                  costwi = flow / ((1/TF(9,rm))-flow)
                  IF (costwi .LT. 0) THEN
                     vlen(i,n) = 100000
                  ELSE
                     costwo = edel(i,n)
                     vlen(i,n) = costwi-costwo
                     IF (vlen(i,n) .GT. 100000) THEN
```

```
                  vlen(i,n) = 100000
                  ENDIF
               ENDIF
               ENDIF
            ENDIF
40       CONTINUE
 C execute the shortest path algorithm

         CALL spath(vlen,iuvfor,uvsrt)

C find the current length of the VC

         itmp1 = is
         itmp2 = ivfor(k,is)
            curln = 0.0
350      curln = curln + vlen(itmp1,itmp2)
         IF (itmp2 .NE. id) THEN
            itmp1 = itmp2
            itmp2 = ivfor(k,itmp2)
            GOTO 350
         ENDIF

C if new length < current length, update VC routing assignments for the
C VC

         IF (uvsrt(is,id) .LT. curln) THEN
            itemp = is
300         IF (itemp .NE. id) THEN
            ivfor(k,itemp) = iuvfor(itemp,id)
            itemp = iuvfor(itemp,id)
            GOTO 300
            ENDIF
         ENDIF

      ENDIF
      ENDIF
20    CONTINUE
      ENDIF

      RETURN
C*********************************************************************
C do book keeping at the dest. when a packet reaches i.e. inc counter
      SUBROUTINE EV3(L)

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,
      *             TFIN,J,NRUN
      COMMON/NETVAR/nodes,ivcs,link(10,10),idfor(10,10),ivfor(400,10),
      *               dellnk(10,10),rsp(10,10),rlp(10,10),edel(10,10),
      *               ivcflg(400),vcst(400),rscnt(400),rlcnt(400),npt

C depending on the length of the packet increment appropriate counter
```

```
      k = (A(L,1)-1)*npt + A(L,8)
      IF (A(L,5) .EQ. 1) THEN
         rscnt(k) = rscnt(k) + 1
      ELSE
         rlcnt(k) = rlcnt(k) + 1
      ENDIF

      RETURN
      END

C*********************************************************************
C does  initializations when a VC is set up and also when a packet is C
created

      SUBROUTINE EV4(L)

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,
     *           TFIN,J,NRUN
      COMMON/NETVAR/nodes,ivcs,link(10,10),idfor(10,10),ivfor(400,10),
     *              dellnk(10,10),rsp(10,10),rlp(10,10),edel(10,10),
     *              ivcflg(400),vcst(400),rscnt(400),rlcnt(400),npt

      DIMENSION vlen(10,10),iuvfor(10,10),uvsrt(10,10)

C assign correct destination and VC no.

      nsd = A(L,1)
      p = DP(12,2)
      CALL SETA(L,8,p)
      k = (nsd-1)*npt + A(L,8)

C if a new VC, set the active flag, assign load estimates, start time

      IF (ivcflg(k) .EQ. 0) THEN
         ivcflg(k) = 1
         vcst(k) = TNOW

         IF (NRUN .GT. 1) THEN
            is = TF(7,REAL(nsd))
            id = TF(8,REAL(nsd))
            rspvc = 0
            rlpvc = (1/TF(17,REAL(nsd))) * 0.8 / npt

C find the routing assignments for the new VC

            DO 40, i = 1,nodes
            DO 40, n = 1, nodes
               IF (link(i,n) .EQ. 1) THEN
                  im = TF(i,REAL(n))
                  rm = REAL(im)
                  IF (edel(i,n) .LT. 0) THEN
                  vlen(i,n) = 100000
```

```
                ELSE
                flow = (rsp(i,n)+rspvc)*TF(10,1.0) +
     *              (rlp(i,n)+rlpvc)*TF(10,2.0)
                costwi = flow / ((1/TF(9,rm))-flow)
                IF (costwi .LT. 0) THEN
                   vlen(i,n) = 100000
                ELSE
                   costwo = edel(i,n)
                   vlen(i,n) = costwi-costwo
                   IF (vlen(i,n) .GT. 100000) THEN
                   vlen(i,n) = 100000
                   ENDIF
                ENDIF
                ENDIF
             ENDIF
40       CONTINUE
45       CONTINUE

         CALL spath(vlen,iuvfor,uvsrt)

            itemp = is
300         IF (itemp .NE. id) THEN
            ivfor(k,itemp) = iuvfor(itemp,id)
            itemp = iuvfor(itemp,id)
            GOTO 300
            ENDIF
         ENDIF

      ENDIF

      RETURN
      END

c*************************************************************************
C does the book keeping and parameter updating relevant to end of a run

      SUBROUTINE WRAPUP

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,
     *           TFIN,J,NRUN
      COMMON/NETVAR/nodes,ivcs,link(10,10),idfor(10,10),ivfor(400,10),
     *              dellnk(10,10),rsp(10,10),rlp(10,10),edel(10,10),
     *              ivcflg(400),vcst(400),rscnt(400),rlcnt(400),npt
      DIMENSION cost(10,10),iuvfor(10,10),uvsrt(10,10)

C store results of interest in corresponding files

      OPEN(UNIT=9,FILE='result.dat',STATUS='OLD')
80       READ(9,*,END=90)
      GOTO 80
90    WRITE(9,*) NRUN,'      ',TAVG(17)
      CLOSE(9)
```

```
C determine lnk flows and datagram routing assignments as per CODEX
C alg

      DO 10, i = 1,nodes
            DO 10, ln = 1,nodes
             IF (link(i,ln) .EQ. 1) THEN
             k = TF(i,REAL(ln))
             p = REAL(k)
             dellnk(i,ln) = TAVG(k)
             rsp(i,ln) = NC(k) / (TNOW-X(1))
             rlp(i,ln) = NC(k+16) / (TNOW-X(1))
             flow = rsp(i,ln)*TF(10,1.0) + rlp(i,ln)*TF(10,2.0)
             edel(i,ln)= flow / ((1/TF(9,p))-flow)
             IF (edel(i,ln) .LT. 0) THEN
                cost(i,ln) = 100000
             ELSE
                cost(i,ln) = edel(i,ln)
             ENDIF
             ENDIF
10    CONTINUE
17    CONTINUE

      CALL spath(cost,iuvfor,uvsrt)
 15   CONTINUE

C update the last updating time

      X(1) = TNOW
      RETURN
      END

C*********************************************************************
C computes shortset paths from every node to every other node given the
C link length assignments

      SUBROUTINE spath(length,for,sroute)

      COMMON/NETVAR/nodes,ivcs,link(10,10),idfor(10,10),ivfor(400,10),
      *                dellnk(10,10),rsp(10,10),rlp(10,10),edel(10,10),
      *                ivcflg(400),vcst(400),rscnt(400),rlcnt(400),npt

      REAL length(10,10),sroute(10,10)

C local vars
      INTEGER for(10,10),temp,head,first,last,top,list(10)

C initialize
      DO 10, i = 1,nodes
         DO 10, j = 1,nodes
            for(i,j) = 0
            sroute(i,j) = 3000000.0
10    CONTINUE
```

```
      DO 15, i = 1,nodes
         sroute(i,i) = 0.0
15    CONTINUE
C alg for each source
      DO 20, i = 1,nodes
         DO 30, j = 1,nodes
            list(j) = 0
30    CONTINUE

      last = i
      head = i

500   DO 40, j = 1,nodes
         IF ((link(head,j) .EQ. 1) .AND. (head .NE. j)) THEN
            IF (sroute(i,head)+length(head,j) .LT. sroute(i,j)) THEN
            sroute(i,j) = sroute(i,head) + length(head,j)
C set forwarding nodes
            for(i,j) = for(i,head)
            IF (head .GE. i) THEN
               for(head,j) = j
            ENDIF
C j, new node to tail
            IF ((j .NE. last) .AND. (list(j) .EQ. 0)) THEN
                  list(last) = j
                  last = j
            ELSE
               IF (list(j) .EQ. -1) THEN
                  IF (list(head) .GT. 0) THEN
                     list(j) = list(head)
                  ENDIF
                  list(head) = j
               ENDIF
            ENDIF
            ENDIF
         ENDIF
      DO 5, k = 1,nodes
5     CONTINUE
40    CONTINUE
         temp = head
         head = list(head)
         list(temp) = -1
         first = head
         IF ((head .NE. -1) .AND. (head .NE. 0)) GOTO 500
      DO 8, k = 1,nodes

8     CONTINUE
20    CONTINUE
      RETURN
      END
```

APPENDIX III

SIMULATION CODE FOR THE ARPANET ALGORITHM

APPENDIX III

This appendix contains the simulation codes for the standard ARPANET algorithm.This was implemented in SIMAN with FORTRAN Discrete Event Models. The Model file, the Experimental file and the Discrete Event file are presented below.

MODEL FILE

```
BEGIN;
            STATION,1-19;                              APPROP. SD PAIR
SPLIT       BRANCH,2:
               ALWAYS,SYSTEM:
               ALWAYS,GEN;                        ONE TO SYS,ONE TO GEN
GEN         DELAY:ED(M):NEXT(SPLIT);             EXP INTERARRIVAL TIME
;
SYSTEM      ASSIGN:A(3)=TF(7,A(1)):MARK(2);        ASSIGN THE SOURCE
            ASSIGN:A(4)=TF(8,A(1));                 ASSIGN THE DESTN
            ASSIGN:A(5)=DP(8,1);                  ASSIGN TYPE(LNG/SHT)
            ASSIGN:X(3)=A(3)+19;                   ASSIGN STN TO ENTER
           .ASSIGN:A(7)=A(3);
;
            ROUTE:0.0,X(3);                    PACKET ENTERS AT SOURCE
;
            STATION,20-25;                      EACH NODE IS A STATION
            BRANCH,1:
               IF, (A(4)+19).EQ.M, REACHED:
               ELSE,PROCEED;                   REACHED DEST OR GO ON ?
;
REACHED     TALLY:17,INT(2):DISPOSE;           ADD PACK DEL & DISPOSE
;
PROCEED     EVENT:1;                              DET FORWARDING LINK
            ROUTE:0.0,J+25;                     SEND TO CORRECT LINK
            STATION,26-41:MARK(6);              EACH LINK IS A STATION
            QUEUE,M-25;                           WAIT FOR IDLE LINK
            SEIZE:LINK(M-25);                           GRAB THE LINK
            ASSIGN:X(4)=M-25;                         ASSIGN LINK NO
            ASSIGN:X(6)=TF(9,X(4));                 ASSIGN LINK CAPACITY
            DELAY:X(6)*TF(10,A(5));                   PRODUCE LINK DELAY
            RELEASE:LINK(M-25);                         RELEASE THE LINK
            BRANCH,1:
               IF, A(5) .EQ. 1, SHT:
               ELSE, LNG;
SHT         COUNT:M-25:NEXT(CONT);                  COUNT SHORT PACKETS
LNG         COUNT:M-9:NEXT(CONT);                   COUNT LONG PACKETS
CONT        TALLY:M-25,INT(6);                        TALLY LINK DELAY
            ASSIGN:J=A(7)+19;
            ROUTE:0.0,J;                              SEND TO THE NODE
;
END;
```

EXPERIMENTAL FILE

```
BEGIN;

PROJECT,CODEX,B.P. MOHANTY,5/4/89;

DISCRETE,8000,7,16,41;

RESOURCES:1-16,LINK;

ARRIVALS:1,STATION(1),0.0,1,1:2,STATION(2),0.0,1,2:
         3,STATION(3),0.0,1,3:4,STATION(4),0.0,1,4:
         5,STATION(5),1000.0,1,5:6,STATION(6),1000.0,1,6:
         7,STATION(7),1000.0,1,7:8,STATION(8),1000.0,1,8:
         9,STATION(9),2000.0,1,9:10,STATION(10),2000.0,1,10:
         11,STATION(11),2000.0,1,11:12,STATION(12),2000.0,1,12:
         13,STATION(13),3000.0,1,13:14,STATION(14),3000.0,1,14:
         15,STATION(15),3000.0,1,15:16,STATION(16),3000.0,1,16:
         17,STATION(17),4000.0,1,17:18,STATION(18),4000.0,1,18:
         19,STATION(19),4000.0,1,19; ARRIVALS ON 19 STREAMS

DISTRIBUTIONS:1,EX(1,1):2,EX(2,2):3,EX(3,3):4,EX(3,3):
              5,UN(4,4):6,UN(4,4):7,UN(5,5):8,EX(6,6):
              9,EX(3,3):10,EX(3,3):11,EX(2,2):12,EX(3,3):
              13,EX(7,7):14,EX(7,7):15,EX(7,7):16,EX(7,7):
              17,UN(5,5):18,UN(4,4):19,EX(6,6);
;                                          ARRIVAL DISTRIBUTIONS

PARAMETERS:1,60.0:2,200.0:3,300.0:4,80.0,100.0:5,60.0,80.0:
           6,70.0:7,90.0:8,0.2,1,1.0,2; DISTRIBUTION PARAMETERS

SEEDS:1,2342,NO:2,56876,NO:3,35441,NO:4,7423,NO:5,95643,NO:
      6,41876,NO:7,58945,NO;  RANDOM NO. GENERATOR SEEDS

TABLES:1,1,1,0,1,2,0,0,3:
       2,1,1,0,0,4,5,0,6:
       3,1,1,7,0,0,8,0,0:
       4,1,1,0,9,10,0,11,0:
       5,1,1,0,0,0,12,0,13:
       6,1,1,14,15,0,0,16,0:
       7,1,1,1,1,1,1,2,2,2,3,3,3,4,4,4,5,5,5,6,6,6:
       8,1,1,3,4,5,6,1,5,6,1,2,5,1,3,6,1,2,3,1,2,4:
       9,1,1,1,1,5.83,0.78,5.83,5.83,1,1,0.78,1,0.78,
       5.83,1,0.78,1,1:
       10,1,1,2.29,18.29;

TALLIES:1,LNK 1 TO 2 DEL:2,LNK 1 TO 3 DEL:3,LNK 1 TO 6 DEL:
        4,LNK 2 TO 3 DEL:5,LNK 2 TO 4 DEL:6,LNK 2 TO 6 DEL:
        7,LNK 3 TO 1 DEL:8,LNK 3 TO 4 DEL:9,LNK 4 TO 2 DEL:
        10,LNK 4 TO 3 DEL:11,LNK 4 TO 5 DEL:12,LNK 5 TO 4 DEL:
        13,LNK 5 TO 6 DEL:14,LNK 6 TO 1 DEL:15,LNK 6 TO 2 DEL:
        16,LNK 6 TO 5 DEL:17,NETWORK DELAY;
```

```
COUNTERS:1,LNK 1 TO 2 STRAF:2,LNK 1 TO 3 STRAF:
         3,LNK 1 TO 6 STRAF:4,LNK 2 TO 3 STRAF:
         5,LNK 2 TO 4 STRAF:6,LNK 2 TO 6 STRAF:
         7,LNK 3 TO 1 STRAF:8,LNK 3 TO 4 STRAF:
         9,LNK 4 TO 2 STRAF:10,LNK 4 TO 3 STRAF:
         11,LNK 4 TO 5 STRAF:12,LNK 5 TO 4 STRAF:
         13,LNK 5 TO 6 STRAF:14,LNK 6 TO 1 STRAF:
         15,LNK 6 TO 2 STRAF:16,LNK 6 TO 5 STRAF:
         17,LNK 1 TO 2 LTRAF:18,LNK 1 TO 3 LTRAF:
         19,LNK 1 TO 6 LTRAF:20,LNK 2 TO 3 LTRAF:
         21,LNK 2 TO 4 LTRAF:22,LNK 2 TO 6 LTRAF:
         23,LNK 3 TO 1 LTRAF:24,LNK 3 TO 4 LTRAF:
         25,LNK 4 TO 2 LTRAF:26,LNK 4 TO 3 LTRAF:
         27,LNK 4 TO 5 LTRAF:28,LNK 5 TO 4 LTRAF:
         29,LNK 5 TO 6 LTRAF:30,LNK 6 TO 1 LTRAF:
         31,LNK 6 TO 2 LTRAF:32,LNK 6 TO 5 LTRAF;

REPLICATE,100,0,10000,NO;

END;
```

DISCRETE EVENTS FILE

```
      SUBROUTINE PRIME

      COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,
     *           TFIN,J,NRUN

      COMMON/NETVAR/nodes,link(10,10),idfor(10,10),dellnk(10,10),
     *              edel(10,10),thold(10,10)

      DIMENSION rlen(10,10),ifor(10,10),dsrt(10,10)

      IF (NRUN .EQ. 1) THEN

C Perform all initializations
C set run time = 0.0

        X(1) = 0.0

C Open result file and read network data from the topology file

        OPEN(UNIT=9,FILE='result.dat',STATUS='UNKNOWN')
        REWIND(9)
        WRITE(9,*) '          NRUN      AVG.DEL',X(7)
        CLOSE(9)

        OPEN(UNIT=10,FILE='topo.dat',STATUS='OLD')
        READ(10,*) nodes
        READ(10,*) ivcs
        DO 10, i = 1,nodes
           READ(10,*) (link(i,k),k = 1,nodes)
10      CONTINUE
        CLOSE(10)

C Initialize linl delays and thresholds

        DO 15, i = 1,nodes
           DO 15, k = 1,nodes
           edel(i,k) = 0.0
           thold(i,k) = 0.0
15      CONTINUE

C -nitialize nodal routing tables

        DO 20, i = 1,nodes
           DO 20, k = 1,nodes
           ifor(i,k) = 0
c          il = TF(i,REAL(k))
c          rl = REAL(il)
c          rlen(i,k) = TF(9,rl)
           rlen(i,k) = link(i,k)
20      CONTINUE
```

```
            CALL spath(rlen,ifor,dsrt)

               DO 25, i = 1,nodes
                  DO 25, k = 1,nodes
                  idfor(i,k) = ifor(i,k)
      25        CONTINUE

            ENDIF
            RETURN
            END

C*****************************************************************************

            SUBROUTINE EVENT(L,I)

            IF (I .EQ. 1) THEN
               CALL EV1(L)
            ENDIF

            RETURN
            END

C*****************************************************************************

      C Determine the outgoing link on which the packet is to be sent

            SUBROUTINE EV1(L)

            COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,
           *           TFIN,J,NRUN
            COMMON/NETVAR/nodes,link(10,10),idfor(10,10),dellnk(10,10),

           *              edel(10,10),thold(10,10)

            i = A(L,7)

               k = A(L,4)
               n = idfor(i,k)
               r = REAL(n)
               CALL SETA(L,7,r)
               J = TF(i,r)

            RETURN
            END

c*****************************************************************************

            SUBROUTINE WRAPUP

            COMMON/SIM/D(50),DL(50),S(50),SL(50),X(50),DTNOW,TNOW,
           *           TFIN,J,NRUN
            COMMON/NETVAR/nodes,link(10,10),idfor(10,10),dellnk(10,10),
```

```
     *              edel(10,10),thold(10,10)

          DIMENSION cost(10,10),iuvfor(10,10),uvsrt(10,10)

     C Write the mean delay result into the file

          OPEN(UNIT=9,FILE='result.dat',STATUS='OLD')
     80      READ(9,*,END=90)
          GOTO 80
     90   WRITE(9,*) NRUN,'      ',TAVG(17)
          CLOSE(9)

          DO 10, i = 1,nodes
               DO 10, ln = 1,nodes

     C Update the link lengths i.e. link delays

                  IF (link(i,ln) .EQ. 1) THEN
                  k = TF(i,REAL(ln))
                  dellnk(i,ln) = TAVG(k)
                  IF (dellnk(i,ln) .EQ. 0) THEN
                     dellnk(i,ln)=TF(9,REAL(k))*(0.2*2.29+0.8*18.29)
                  ENDIF

     C Threshold mechanism to decide whether to update or not

                  IF (ABS(dellnk(i,ln)-edel(i,ln)) .GE.
          *            thold(i,ln)) THEN
                     edel(i,ln) = dellnk(i,ln)
                     thold(i,ln) = 64.0
                  ELSE
                     thold(i,ln) = thold(i,ln) - 12.8
                  ENDIF

                  IF (edel(i,ln) .LT. 0) THEN
                     cost(i,ln) = 100000
                  ELSE
                     cost(i,ln) = edel(i,ln)
                  ENDIF
                  ENDIF
     10   CONTINUE

     C Comput shortest paths

          CALL spath(cost,iuvfor,uvsrt)

     C Update the routing tables
          DO 15, i = 1,nodes
             DO 15, ln = 1, nodes
                idfor(i,ln) = iuvfor(i,ln)
     15   CONTINUE
```

```
C Set the run time to its new value

      X(1) = TNOW

      RETURN
      END

C*********************************************************************

      SUBROUTINE spath(length,for,sroute)

      COMMON/NETVAR/nodes,link(10,10),idfor(10,10),dellnk(10,10),
     *              edel(10,10),thold(10,10)


      REAL length(10,10),sroute(10,10)

C local vars
      INTEGER for(10,10),temp,head,first,last,top,list(10)

C initialize
      DO 10, i = 1,nodes
         DO 10, j = 1,nodes
            for(i,j) = 0
            sroute(i,j) = 3000000.0
10    CONTINUE
      DO 15, i = 1,nodes
         sroute(i,i) = 0.0
15    CONTINUE

C alg for each source

      DO 20, i = 1,nodes
         DO 30, j = 1,nodes
            list(j) = 0
30    CONTINUE
      last = i
      head = i

500   DO 40, j = 1,nodes
         IF ((link(head,j) .EQ. 1) .AND. (head .NE. j)) THEN
            IF  (sroute(i,head)+length(head,j)  .LT.  sroute(i,j))
                THEN
            sroute(i,j) = sroute(i,head) + length(head,j)
C set forwarding nodes
            for(i,j) = for(i,head)
            IF (head .GE. i) THEN
               for(head,j) = j
            ENDIF
```

```
C j, new node to tail
            IF ((j .NE. last) .AND. (list(j) .EQ. 0)) THEN
                  list(last) = j
                  last = j
            ELSE
               IF (list(j) .EQ. -1) THEN
                  IF (list(head) .GT. 0) THEN
                     list(j) = list(head)
                  ENDIF
                  list(head) = j
               ENDIF
            ENDIF
            ENDIF
         ENDIF

      DO 5, k = 1,nodes
5     CONTINUE
40    CONTINUE
         temp = head
         head = list(head)
         list(temp) = -1
         first = head
         IF ((head .NE. -1) .AND. (head .NE. 0)) GOTO 500
      DO 8, k = 1,nodes
8     CONTINUE
20    CONTINUE
      RETURN
      END
```

APPENDIX IV

OPTIMAL ROUTING AND FLOW CONTROL IN NETWORKS WITH REAL-TIME TRAFFIC

OPTIMAL ROUTING AND FLOW CONTROL IN NETWORKS WITH REAL-TIME TRAFFIC [1]

Christos G. Cassandras and Michelle Hruby Kallmes
Dept. of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003
Tel. 413 - 545 1340

Don Towsley
Dept. of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

ABSTRACT

We address the problem of flow control and routing of *real-time traffic* in a network, where messages must arrive at their destination within given deadlines; otherwise they are considered lost. Performance in this case is measured in terms of the probability of losing a message. For the case of n parallel links, the problem is formulated as one of optimal flow allocation and solved under general conditions. It is shown that for a FCFS service discipline an admission policy rejecting messages before link assignment is optimal when the load exceeds a critical value. Thus, we take advantage of the fact that if some messages will exceed their deadlines anyway, it is beneficial not to admit them in the first place. An efficient algorithm for explicitly solving the problem is presented and specific examples are analyzed. We also discuss the applicability of on-line algorithms for this problem when modeling assumptions cannot be made.

1. INTRODUCTION

In this paper, we address issues of performance analysis and control for systems with *real-time traffic*. Real-time traffic is characterized by deadlines within which service must be completed, and is encountered in at least two settings: communication networks and multiprocessor computer systems [1],[2]. We will present the problem with a communication network application in mind, but all results can be applied to analogous problems in a computer system environment. Thus, when a message with a real-time constraint is transmitted, we require that its destination be reached by some given *deadline*, characteristic of the message itself or of the traffic class to which it belongs. If this condition is not satisfied, the message is useless and is considered lost. In such problems, the objective of interest is the minimization of the probability of loss, measuring the fraction of messages that exceed their deadlines.

Given this framework, we shall consider the problem of assigning messages to n links connecting a source node to a destination node so as to minimize the fraction of messages that do not reach the destination node by some deadline. There are two variations of this problem: (a) all messages are required to be transmitted, and (b) messages are allowed to be rejected instead of being assigned to a link. We shall consider case (b) only, and accomplish the flow control through an admission policy. A typical workload consists of voice and/or video packets with a time constraint by which they must be received to be useful. In current implementations, the scheduling policies at the link level are very simple and do not allow for deliberate packet rejection. It is, however, possible to include flow control at the source in the form of an admission policy that does reject a fraction of the packets whenever it is beneficial to do so.

A similar situation is encountered in static load balancing for a multiprocessor system, where if we consider messages to be jobs and links to be processors, the same results apply. In past analysis of computer systems, most work has focussed on the problem of balancing loads so as to minimize a metric such as average delay [3],[4]. However, a better objective seems to be minimizing the fraction of jobs whose response times fall above some threshold. Thus, if $E[x]$ is the average processing time, good performance may be determined by a response time that lies below some multiple of $E[x]$. In this application, it would be beneficial to impose a policy that prohibits admission to some fraction of jobs so as to increase the fraction of jobs that complete processing within their threshold response time.

To illustrate the potential advantage of an admission policy allowing rejection, we present the following simple example. Suppose the system consists of a single link. Upon arrival into the system, a message is either accepted with some probability p and placed in the FCFS queue, or rejected with probability $(1-p)$. The objective is to adjust p so as to minimize the *probability of loss*. A message can be *lost* either at the source by being rejected, or at the destination by violating its deadline. The crucial observation here is that a proper choice of p can result in *rejecting some messages that may not have made their deadlines anyway*. Note that as p is decreased, the link becomes less congested, resulting in a larger fraction of messages making their deadline. On the other hand, more messages are being rejected. Moreover, if we dealt with limited buffering space, then a third result of decreasing p would be a welcomed decrease in the blocking probability. Thus, there is a tradeoff between rejecting a message so that others may make their deadline and serving the message in hopes that it might make its deadline. As we shall show, it is generally beneficial to reject some fraction of

[1] This work is supported in part by the Office of Naval Research under contract N00014-87-K-0304 and by the ROME Air Development Center under contract F30602-88-D-0027.

CH2702-9/89/0000/0784$01.00 © 1989 IEEE

the incoming messages. It is only for the case of lightly loaded systems that the optimal admission policy is $p = 1$.

The paper is organized as follows. In section 2, we formulate the model and the optimization problem. We then characterize the optimal admission and link assignment policies under fairly general conditions. In section 3, we present an algorithm for explicitly solving this problem. We also discuss the problem of on-line optimization when knowledge of the system characteristics is limited. Section 4 includes some specific examples. Finally, we summarize our results in section 5, and indicate directions for future work.

2. MODEL DESCRIPTION AND PROBLEM FORMULATION

Consider a stream of messages arriving to n parallel links according to an arbitrary arrival process with rate λ, as shown in Fig. 1. The rate of transmission at link i is $\mu_i > 0$, and all links are independent. We will assume that there is the option of rejecting messages before they are assigned to any link; let λ_0 denote the corresponding flow. In addition, let λ_i be the flow to link i, for $i = 1,\ldots,n$. Finally, let D be a random variable that denotes the deadline associated with a random message.

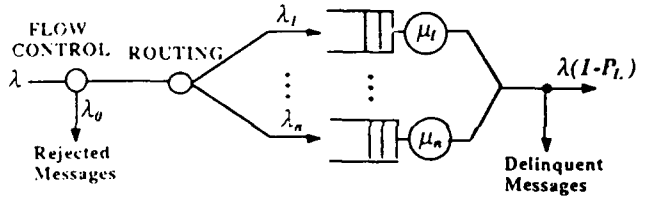


Figure 1: *Link Assignment Model with Admission Policy*

2.1. Problem Formulation

We wish to determine a flow allocation $(\lambda_0,\ldots,\lambda_n)$ so as to minimize the *probability of loss*, where a message is considered *lost* if it is either *rejected* upon arrival or *accepted* but is *delinquent* (i.e., misses its deadline). Let $P^r(\lambda_0)$ be the probability that a random message is rejected, and let $P^d(\lambda_1,\ldots,\lambda_n)$ be the probability that a message is delinquent given that it is accepted. Then the total probability of loss is

$$P_L = [P^r + (1-P^r)\cdot P^d].$$

We shall assume in what follows that a message is delinquent if its total delay T exceeds the deadline D, i.e. $P^d = P[T > D]$. However, the analysis also applies to other definitions of delinquency, such as the event that the queueing time of a message exceeds the deadline. For example, see section 4.2.

At this point, we translate the problem from one of minimizing the *probability of loss* to one of minimizing the *loss rate*, L, where

$$L = \lambda \cdot [P^r + (1-P^r)\cdot P^d].$$

Clearly, for a fixed load λ, the two problems are equivalent. Note that

$$\lambda\cdot P^r = \lambda_0\,, \qquad \lambda\cdot(1-P^r)\cdot P^d = \sum_{i=1}^{n}\lambda_i\cdot P_i^d(\lambda_i)\,,$$

where $P_i^d(\lambda_i)$ is the probability that a job is delinquent given that it is routed over link i. Thus, the performance metric can be written as

$$L(\lambda_0,\ldots,\lambda_n) = \lambda_0 + \sum_{i=1}^{n}\lambda_i\cdot P_i^d(\lambda_i).$$

Furthermore, let

$$f_i(\lambda_i) = \lambda_i\cdot P_i^d(\lambda_i), \qquad i = 1,\ldots,n, \tag{1}$$

$$f_0(\lambda_0) = \lambda_0. \tag{2}$$

Then a general statement of the optimal flow allocation problem is the following:

$$\min_{(\lambda_0,\ldots,\lambda_n)} \sum_{i=0}^{n} f_i(\lambda_i)$$

subject to

$$\sum_{i=0}^{n}\lambda_i = \lambda$$

$$\lambda_i \ge 0, \qquad i = 0,\ldots,n.$$

Note that the flows λ_i are not constrained to be less than the corresponding link capacity μ_i; this will naturally emerge as a property of the optimal flow allocation. Also note that by introducing the additional constraint $\lambda_0 = 0$ to the problem above, we can obtain a formulation where no rejections are allowed, as in variation (a) presented in the introduction. The analysis in this case would proceed along the same lines.

2.2. The Optimal Solution

We will make the following assumptions regarding the functions $P_i^d(\cdot)$ and $f_i(\cdot)$, $i=1,\ldots,n$:

A1 $P_i^d(x)$ is increasing for all $x \in [0,\mu_i)$.
A2 $P_i^d(\mu_i) = 1$.
A3 $f_i(x)$ is increasing and strictly convex for all $x \in [0,\mu_i)$.
A4 If $\lambda_i < \mu_i$, then $f_i(\lambda_i) < \lambda_i$.

Most of our subsequent results depend on A3 only. A1 simply requires that increasing the load to a link must increase the probability that messages on that link exceed their deadlines; this is natural, since increasing the load causes longer average delays. A2 requires that waiting times become infinite at capacity. It is satisfied for the FCFS service discipline and possibly by RSS, but it does not hold for a LCFS service discipline. Finally, from (1), A4 is equivalent to the requirement that if $\lambda_i < \mu_i$, then $P_i^d(\lambda_i) < 1$. This assumption is not critical, since we can always determine some $\mu_i' \le \mu_i$ such that $P_i^d(\mu_i') = 1$, and treat μ_i' as the effective capacity of the link.

Let

$$h_i(\lambda_i) = \partial f_i(\lambda_i)/\partial\lambda_i, \qquad i = 0,\ldots,n,$$

where the derivative is defined for $\lambda_i \in [0,\mu_i)$. Then, from A3, $h_i(x)$ is monotonically increasing for all $x \in [0,\mu_i)$, $i=0,\ldots,n$.

Moreover, $h_i(\lambda_i)$ has the following properties for i=0,...,n:

Lemma:

(a) $0 \le h_i(0) < 1$,

(b) $h_i(\mu_i) > 1$.

Proof: From the definition (1):

$$h_i(\lambda_i) = \frac{\partial f_i(\lambda_i)}{\partial \lambda_i} = P_i^d(\lambda_i) + \lambda_i \frac{\partial P_i^d(\lambda_i)}{\partial \lambda_i}.$$

Hence, $h_i(0) = P_i^d(0)$, where $0 \le P_i^d(0) \le 1$. Statement (a) then follows from **A4**. Similarly, $h_i(\mu_i) = 1 + \mu_i \cdot (\partial P_i^d / \partial \mu_i) > 1$, by **A1** and **A2**. **QED**

The following result identifies conditions satisfied by the optimal solution to the flow allocation problem (see also [3]).

Theorem 1: (a) The optimal solution $(\lambda_0^*,...,\lambda_n^*)$ to the above problem satisfies the following necessary and sufficient conditions:

$$h_i(\lambda_i^*) = \alpha \quad \text{if } \lambda_i^* > 0, \tag{3}$$

$$h_i(\lambda_i^*) \ge \alpha \quad \text{if } \lambda_i^* = 0, \tag{4}$$

such that:

$$\sum_{i \text{ s.t. } \lambda_i^* > 0} h_i^{-1}(\alpha) = \lambda, \qquad \alpha > 0. \tag{5}$$

(b) All optimal flows satisfy $\lambda_i^* < \mu_i$, $i = 1,...,n$.

Proof: Using a Lagrange multiplier $\alpha > 0$, we adjoin the flow conservation constraint to $L(\lambda_0,...,\lambda_n)$ to obtain

$$\tilde{L}(\lambda_0,...,\lambda_n,\alpha) = \sum_{i=0}^{n} f_i(\lambda_i) + \alpha \cdot \left(\lambda - \sum_{i=0}^{n} \lambda_i \right).$$

Thus, the necessary (Kuhn-Tucker) conditions require that at $\lambda_i = \lambda_i^*$

$$\frac{\partial \tilde{L}}{\partial \lambda_i} = h_i(\lambda_i) - \alpha = 0, \quad i = 0,...,n$$

provided $\lambda_i^* > 0$, and (3) is obtained. On the other hand, if the constraint $\lambda_i \ge 0$ is active (i.e. $\lambda_i^* = 0$), then at the feasible region boundary, defined by $\lambda_i = 0$, optimality requires that

$$\frac{\partial \tilde{L}}{\partial \lambda_i} \ge 0 \quad \text{at} \quad \lambda_i = 0,$$

which yields (4). Finally, for an optimal allocation $(\lambda_0^*,...,\lambda_n^*)$, the flow conservation constraint becomes

$$\sum_{i=0}^{n} \lambda_i^* = \sum_{i \text{ s.t. } \lambda_i^* > 0} \lambda_i^* = \lambda$$

and since all λ_i^* such that $\lambda_i^* > 0$ must satisfy (3), we immediately obtain (5). Note that $h_i^{-1}(\cdot)$ is well-defined in $(0, h_i(\mu_i))$, and, as we shov in part (b), $h_i^{-1}(\alpha) = \lambda_i^* < \mu_i$.

The sufficiency of these conditions follows directly from the convexity assumption A3. This guarantees that a unique α is determined through (5), and hence unique $\lambda_i^* > 0$ are determined from (3).

Part (b) of the Theorem is trivial if $\lambda_i^* = 0$. Thus, we consider only the case $\lambda_i^* > 0$. First, note from (2) that $h_0(\lambda_0) = 1$ for all $\lambda_0 \ge 0$. Since (3) and (4) imply that $h_0(\lambda_0^*) \ge \alpha$, it follows that $\alpha \le 1$. Thus, from (3), we get

$$h_i(\lambda_i^*) = \alpha \le 1.$$

From the Lemma above, $h_i(\mu_i) > 1$. Thus, $h_i(\lambda_i^*) < h_i(\mu_i)$. Since $h_i(\lambda_i)$ is monotonically increasing for all $\lambda_i \in [0,\mu_i)$, it follows that $\lambda_i^* < \mu_i$. **QED**

Remark. Since $h_0(\lambda_0) = 1$, it follows from (3) that if $\lambda_0^* > 0$, i.e. rejections do occur, then we have $\alpha = 1$.

An illustration of the Theorem is shown in Fig. 2. For any i=1,...,n, as long as $\lambda_i^* > 0$, the optimal flow λ_i^* is determined by the point where $h_i(\lambda_i)$ intersects the constant α line. Otherwise, $h_i(\lambda_i)$ lies above the constant α line and $\lambda_i^* = 0$. Furthermore, as pointed out above, $\alpha = 1$ whenever $\lambda_0^* > 0$. In the example shown in Fig. 2, α takes on a value less than 1 and $h_1(\lambda_1)$ does not intersect the line defined by α. Thus, $\lambda_0^* = \lambda_1^* = 0$, corresponding to a lightly loaded case with link 1 unutilized.

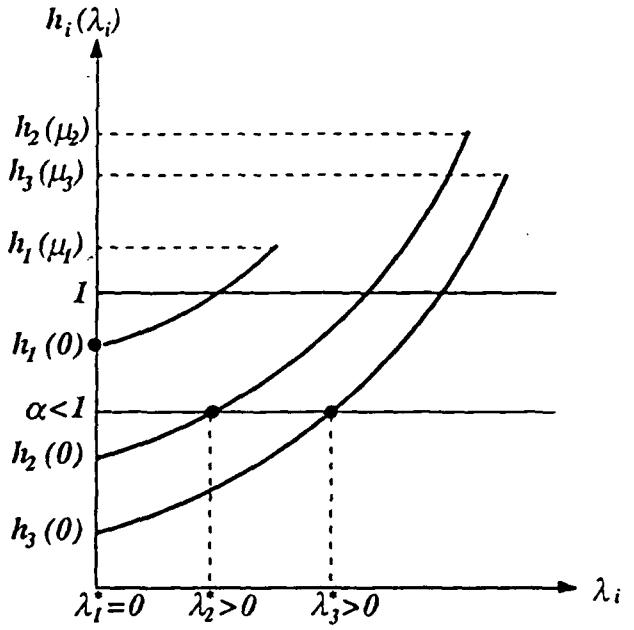


Figure 2: *Characterization of Optimal Flows*

Some interesting properties of the optimal flows are presented below as a Corollary of Theorem 1. First, we define $g_i(y)$ to be the bounded inverse of $h_i(x)$, defined for $0 \le y < h_i(\mu_i)$. More specifically,

$$g_i(y) = \begin{cases} x & \text{if } y = h_i(x),\ x > 0, \\ 0 & \text{if } y \le h_i(0). \end{cases}$$

Thus, (5) can be written as $\Sigma_i\, g_i(\alpha) = \lambda$. Observe that since $h_i(\cdot)$ is increasing, $g_i(\cdot)$ is non-decreasing.

Corollary 1.1: Suppose links are indexed so as to satisfy the following ordering:

$$h_0(0) \geq h_1(0) \geq \ldots \geq h_n(0).$$

Then the optimal flows, $\lambda_0^*,\ldots,\lambda_n^*$, have the following properties:

$$\lambda_i^* > 0 \text{ if and only if } \lambda > \sum_{j \geq i} h_j^{-1}[h_i(0)],\ i = 0,\ldots,n, \qquad (6)$$

$$\lambda_0^* > 0 \text{ if and only if } \lambda > \sum_{i=1}^{n} h_i^{-1}(1), \qquad (7)$$

$$\lambda_i^* = 0 \Rightarrow h_i(0) \geq \alpha,\ h_i(0) > \alpha \Rightarrow \lambda_i^* = 0,\ i = 0,1,\ldots,n. \qquad (8)$$

Proof: For any $i = 0,1,\ldots,n$, suppose $\lambda_i^* > 0$. Then (3) implies $\alpha = h_i(\lambda_i^*) \geq h_i(0)$ since $h_i(\cdot)$, $i=1,\ldots,n$ is monotonically increasing by A3 and $h_0(\cdot) = 1$. Thus, from (5)

$$\lambda = \lambda_i^* + \sum_{j \neq i} g_j(\alpha) > \sum_{j \geq i} h_j^{-1}[h_i(0)],$$

which establishes the "if" part in (6). We prove the converse by contradiction. Thus, assume that the inequality in (6) is satisfied and that $\lambda_i^* = 0$. From (4), this implies that $h_i(\lambda_i^*) \geq \alpha$, i.e. $h_i(0) \geq \alpha$. Because of the ordering we have established, it follows that

$$h_j(0) \geq \alpha \quad \text{for all } j < i.$$

Using this fact, $g_j(\alpha) = 0$ for all $j < i$, and (5) now gives:

$$\lambda = \sum_{j<i} g_j(\alpha) + \sum_{j \geq i} g_j(\alpha) \leq \sum_{j \geq i} h_j^{-1}[h_i(0)],$$

which contradicts the inequality in (6). This completes the proof of (6).

Since this proof holds for all $i = 0,1,\ldots,n$, (7) is simply a special case of (6) with $i = 0$, noting that $h_0(0) = 1$ from (2).

To establish (8), first assume that $\lambda_i^* = 0$. Then, by (4), $h_i(\lambda_i^*) \geq \alpha$, i.e. $h_i(0) \geq \alpha$. On the other hand, if $h_i(0) > \alpha$, then, by the monotonicity of $h_i(\cdot)$ we have $h_i(\lambda_i^*) > \alpha$, since $\lambda_i^* \geq 0$. Invoking (4), the result follows and the proof is complete. QED

An implication of this Corollary, specifically (7), is that there exists a critical load value, λ_{crit}, given by

$$\lambda_{crit} = \sum_{i=1}^{n} h_i^{-1}(1). \qquad (9)$$

The value λ_{crit} defines the *load threshold*, beyond which rejection of messages will always improve performance. This is illustrated in Fig. 3, where the effective throughput, $\lambda \cdot (1-P_L)$, is always higher with $\lambda_0 > 0$ than with $\lambda_0 = 0$ (no admission policy) as long as $\lambda > \lambda_{crit}$. That is, when the load exceeds the critical value, the optimal flow policy will reject some *positive* flow of messages.

Remark. For any $i = 0,\ldots,n$, the values of $h_i^{-1}(1)$ in (9) are independent of the flows $\lambda_0,\ldots,\lambda_n$, and the parameter λ; they depend only on the structure of the system (the distribution of interarrival times, service times, and deadlines), and on the parameters $\mu_1,\ldots,\mu_n$ and $E[D]$. Hence, provided that an analytical expression for $h_i^{-1}(\cdot)$ can be obtained, one can determine λ_{crit} without explicitly solving the optimization problem.

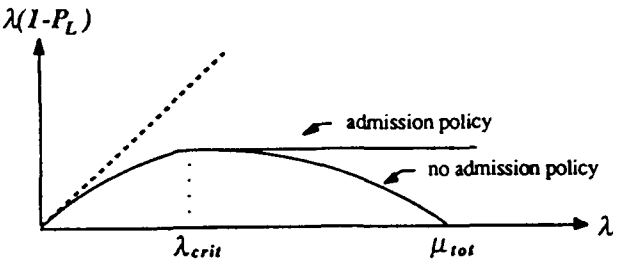


Figure 3: *Load Threshold in Optimal Admission Policy*

Some additional properties of the optimal flow allocation are stated below without proof. In expressing these properties, it is convenient to introduce the following definitions. Let $(\lambda_0,\ldots,\lambda_n)$ denote a feasible solution to our problem. With respect to this solution, there are two classes of links, *active* and *idle*. Link i is considered active if $\lambda_i > 0$; otherwise, it is considered idle. Let A denote the set of links that are active and I the set of links that are idle. We now present four properties of the optimal solution.

Property 1 The optimal flow λ_i^* is a nondecreasing function of λ. Furthermore, if $\lambda_0^* > 0$ for some λ, then as λ increases, λ_i^* remains constant for all $i = 1,\ldots,n$.

Property 2 The set of active links A is a nondecreasing function of λ. In particular, if the nodes are ordered according to the rule specified in Corollary 1.1, then as λ increases, links join A in the reverse order.

Property 3 Let $A' = A - \{j\}$. Then A' is a nonincreasing function of μ_j. Moreover, links transform from active to idle in the same order of $h_i(0)$ as μ_j increases. Furthermore, j always becomes active for sufficiently large μ_j, and remains active thereafter. Last, $A' \to \emptyset$ as $\mu_j \to \infty$.

Property 4 If $\lambda > \Sigma_i \mu_i$, then $\lambda_0^* > 0$.

3. OPTIMAL FLOW ALLOCATION ALGORITHMS

In this section, we address the issue of obtaining explicit solutions for the optimization problem presented above. Clearly, this depends on the availability of closed-form expressions for the link loss rate functions $f_i(\cdot)$, as well as the inverse functions $h_i^{-1}(\cdot)$. Such expressions are generally not easy to obtain, even under reasonable stochastic modeling assumptions [5].

In section 3.1, we present an efficient algorithm (of order $n\log n$) for obtaining an explicit optimal flow solution under the assumption that analytical expressions for $h_i(\cdot)$ are available. However, its inverse, $h_i^{-1}(\cdot)$, need not be available in closed form; numerical methods can be employed to evaluate it.

In section 3.2, we briefly address the issue of on-line optimization, when limited knowledge of the overall system is available. In this case, we invoke gradient-based stochastic optimization algorithms, taking advantage of recent developments in stochastic gradient estimation [6]-[8].

3.1. An Efficient Algorithm

The following algorithm can be used to obtain the solution to the problem of minimizing the message loss rate for the system of Fig. 1.

Algorithm.

1. Ordering of links:
   - order links so that $h_0(0) \geq h_1(0) > h_2(0) \geq \cdots \geq h_n(0)$.
2. Determination of the set of active links, A:
   - define $\lambda(m) = \Sigma_{i>m} g_i(h_m(0))$,
   - use binary search to find m such that $\lambda(m) \leq \lambda < \lambda(m+1)$,
   - $I = \{1,\ldots,m\}$, $A = \{m+1,\ldots,n\}$.
3. Determination of link flows:
   - solve equations (3)-(5) to obtain $(\lambda_0^*,\ldots,\lambda_n^*)$.

The algorithm is explained as follows. The first step orders the links so that the properties (6)-(8) will hold. Step 2 applies property (6) to determine which links have positive flow. For links $i \in I$, we can immediately determine $\lambda_i^* = 0$. In step 3, we notice that equation (3) holds for links $i \in A$, so those $(n-m)$ equations along with (5) uniquely determine λ_i^* for $i \in A$.

The algorithm is most easily understood by referring back to Fig. 2. In the example shown, the ordering of step 1 has already been applied. Step 2 determines the region of the constant α line, that is, m is found such that $h_m(0) \geq \alpha > h_{m+1}(0)$. Then step 3 calculates the intersection point of $h_i(\lambda_i)$ with the constant α line for all active links; λ_i at the intersection is optimal.

3.2. On-Line Optimization

Our analysis thus far relies on the availability of analytical expressions for $f_i(\lambda_i)$, $i=1,\ldots,n$. This means that specific link models can be developed; in section 4, for example, we assume each link to behave like an M/M/1 queueing system. Often, however, accurate models are not available and operating conditions change. In such cases, one is limited to observing actual system data (e.g. message arrival times at a node) and making admission and routing decisions based on that information alone. There are several algorithms suitable for such on-line control schemes, mostly dependent on estimating gradients (sensitivities) of the performance metrics with respect to the parameters of interest (see [9],[10]). Our purpose here is only to make brief mention of the fact that stochastic gradient estimation techniques can be developed and used in the context of our problem, and to illustrate the use of this information for the simple flow control problem corresponding to $n=1$.

If a single link is present, the critical load value in (9) becomes $\lambda_{crit} = h_1^{-1}(1)$. Recalling the definitions of $f_i(\cdot)$ and $h_i(\cdot)$, this implies that when $\lambda = \lambda_{crit}$, the accepted flow, λ_1^c, is determined from

$$\min_{\lambda_1} [\, (\lambda-\lambda_1) + \lambda_1 \cdot P_1^d(\lambda_1) \,],$$

yielding

$$\frac{\partial}{\partial \lambda_1}[\lambda_1 P_1^d(\lambda_1)] = 1,$$

which implies that

$$\lambda_1^c = [1 - P_1^d(\lambda_1)] \cdot \left(\frac{\partial P_1^d}{\partial \lambda_1}\right)^{-1}$$

with the right hand side evaluated at $\lambda = \lambda_{crit}$. If this system is directly observed, an estimate of $P_1^d(\lambda_1)$ can be obtained under any admission policy yielding a flow λ_1. Suppose that, at the same time, an estimate of the sensitivity of the delinquency probability with respect to the flow, $(\partial P_1^d/\partial \lambda_1)$, can also be obtained. Then, one would immediately be able to determine whether the load threshold point shown in Fig. 3 has been reached or not, and hence determine the optimal flow allocation. Furthermore, in an adaptive sense, flows can be adjusted on-line until the optimal allocation is found. In many cases, this gradient estimation is accomplished without knowledge of the actual load λ or the link capacity μ_1 [10]. The problem of determining stochastic gradient estimates along a single observation interval is addressed through Perturbation Analysis [7] and the Likelihood Ratio technique [8]. However, results currently available apply primarily to performance measures such as throughput and mean message delay. The case of $(\partial P_1^d/\partial \lambda_1)$ is considered in [9] and is the subject of ongoing research.

4. EXAMPLES

In what follows, we present three examples for the case of M/M/1 link models. That is, we assume that messages arrive into the system according to a Poisson arrival process with parameter λ and are assigned to links through a probabilistic rejection and routing mechanism. In addition, we model all links as exponential servers with parameters μ_i, $i=1,\ldots,n$ and a FCFS queueing discipline.

4.1 Constant Deadline on Total Delay

Let the deadline always take on the value $D = \tau$. In this case $P_i^d = P[T>\tau]$, and

$$f_i(\lambda_i) = \lambda_i\, e^{-(\mu_i-\lambda_i)\tau}$$

$$h_i(\lambda_i) = (1+\lambda_i\tau)\, e^{-(\mu_i-\lambda_i)\tau}$$

for all $i=1,\ldots,n$, and $f_0(\lambda_0) = \lambda_0$ as before. Even though we are unable to obtain an explicit expression for $h_i^{-1}(\cdot)$, and hence for $g_i(\cdot)$, it is easy to show that $h_i^{-1}(\alpha)$ is the root of the equation

$$(1+\lambda_i\tau)\, e^{-(\mu_i-\lambda_i)\tau} - \alpha = 0$$

that lies in $[0,\mu_i)$. This root exists and is unique provided

$$e^{-\mu_i\tau} \leq \alpha < 1+\mu_i\tau\,. \qquad (10)$$

In such a case, $g_i(\alpha) = h_i^{-1}(\alpha)$. Otherwise, if $\alpha < e^{-\mu_i\tau}$, then $g_i(\alpha) = 0$, and the case where $\alpha > 1 + \mu_i\tau$ does not occur because, as we have already shown, $\alpha \leq 1$.

It is easily shown that A1-A4 are satisfied. Furthermore, the optimal solution results in rejections if and only if $\lambda > \Sigma_i h_i^{-1}(1)$, where $h_i^{-1}(\alpha)$ is the root of the equation

$$(1+\lambda_i\tau)\, e^{-(\mu_i-\lambda_i)\tau} - 1 = 0$$

that lies in $[0,\mu_i)$, which we know to exist and be unique since (10) holds for $\alpha = 1$.

4.2 Constant Deadline On Queueing Time

We assume that deadlines are deterministic and equal to τ, and let $P_i^d = P[Q>\tau]$ where Q is the queueing time of a random message. In this case,

$$f_i(\lambda_i) = \frac{\lambda_i^2}{\mu_i}\, e^{-(\mu_i-\lambda_i)\tau}$$

$$h_i(\lambda_i) = \frac{\lambda_i}{\mu_i}\,(2+\lambda_i\tau)\, e^{-(\mu_i-\lambda_i)\tau}$$

for $i = 1,\ldots,n$. Assumptions A1-A4 are easily shown to hold for these functions. We are unable to obtain an explicit expression for $h_i^{-1}(\lambda_i)$, and hence for $g_i(\lambda_i)$. However, we can easily show that $h_i^{-1}(\alpha)$ is the root of the equation

$$\frac{\lambda_i}{\mu_i}\,(2+\lambda_i\tau)\, e^{-(\mu_i-\lambda_i)\tau} - \alpha = 0$$

that lies in $[0,\mu_i)$. This root exists and is unique provided $0 \le \alpha < 2 + \lambda_i\tau$, in which case we have $g_i(\alpha) = h_i^{-1}(\alpha)$. But since $0 < \alpha \le 1$, we can conclude that α is in the range, hence the root does exist uniquely for all choices of system parameters, and $\lambda_i^* > 0$, $i = 1,\ldots,n$.

4.3 Exponential Deadlines on Total Delay

We assume that D is an exponential random variable with mean $1/\gamma$, and, once again, $P_i^d = P[T>D]$. Then

$$f_i(\lambda_i) = \frac{\lambda_i\gamma}{\gamma+\mu_i-\lambda_i}$$

$$h_i(\lambda_i) = \frac{\gamma+\mu_i}{(\gamma+\mu_i-\lambda_i)^2}$$

for $i = 1,\ldots,n$. It is easy to show that P_i^d is such that A1-A4 are satisfied. Furthermore, we can write the following expression for $g_i(\alpha)$,

$$g_i(\alpha) = \begin{cases} \gamma+\mu_i - \sqrt{\dfrac{\gamma+\mu_i}{\alpha}}, & \alpha \ge \dfrac{1}{\gamma+\mu_i}, \\ 0, & \alpha < \dfrac{1}{\gamma+\mu_i}. \end{cases}$$

If the optimal solution has $A = \{m+1,\ldots,n\}$, and $\lambda_0^* = 0$, then

$$\lambda_i^* = \gamma + \mu_i - (\gamma+\mu_i)^{1/2}\left(\sum_{i=m+1}^{n} \frac{\gamma+\mu_i}{(\gamma+\mu_i-\lambda_i)^2}\right)^{-1/2}$$

for $i = m+1,\ldots,n$, and 0 otherwise. If $\lambda_0^* > 0$, then

$$\lambda_i^* = \gamma + \mu_i - (\gamma+\mu_i)^{1/2}.$$

The optimal solution results in rejections if and only if

$$\lambda > \sum_{i=1}^{n}\left[\gamma+\mu_i - \sqrt{\gamma+\mu_i}\right].$$

For the special case of a single link with capacity μ, the optimal flow control policy, $(\lambda_0^*, \lambda_1^*)$ is

$$\lambda_0^* = \begin{cases} \lambda - (\gamma+\mu-\sqrt{\gamma+\mu}), & \lambda \ge \gamma+\mu-\sqrt{\gamma+\mu}, \\ 0 & \text{otherwise}. \end{cases}$$

$$\lambda_1^* = \begin{cases} \gamma+\mu-\sqrt{\gamma+\mu}, & \lambda \ge \gamma+\mu-\sqrt{\gamma+\mu}, \\ \lambda & \text{otherwise}. \end{cases}$$

If we do not apply an admission policy (i.e. never reject), then the optimal loss rate is

$$L^* = \begin{cases} \dfrac{\lambda\gamma}{\gamma+\mu-\lambda}, & \lambda \le \mu, \\ \lambda & \text{otherwise}. \end{cases}$$

but allowing rejection gives

$$L^* = \begin{cases} \dfrac{\lambda\gamma}{\gamma+\mu-\lambda}, & \lambda \le \gamma+\mu-\sqrt{\gamma+\mu}, \\ \lambda - \left[\dfrac{(\gamma+\mu-\sqrt{\gamma+\mu})\cdot(\sqrt{\gamma+\mu}-\gamma)}{\sqrt{\gamma+\mu}}\right], & \text{otherwise}. \end{cases}$$

In Figures 4a and 4b, we have plotted the marginal loss rates, $h_0(\lambda_0) = 1$ and $h_1(\lambda_1)$, and the effective throughput, $\lambda\cdot(1-P_L)$, for the case where $\mu = 3$, $\gamma = 1$. Note that Fig. 4b is similar to Fig. 3, and clearly shows the existence of a critical load threshold for this model.

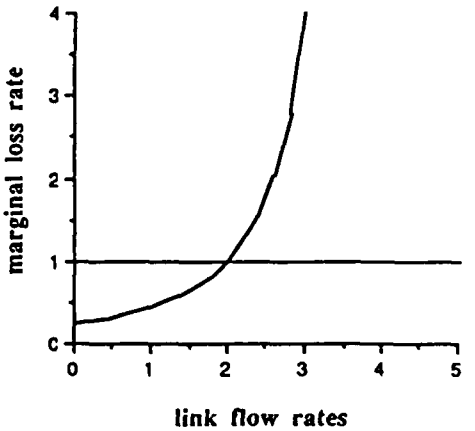


Figure 4a: *Marginal Loss Rates*

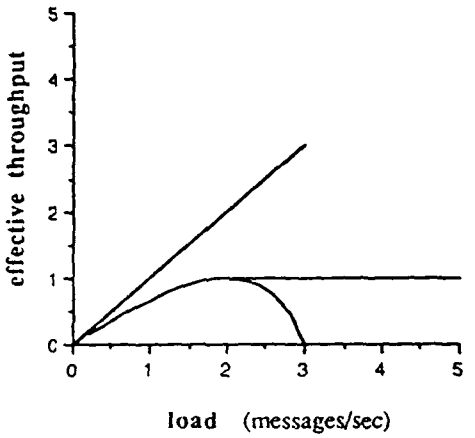


Figure 4b: *Effective Throughput*

5. CONCLUSIONS AND FUTURE WORK

We have shown that when the performance metric of interest is the probability of losing messages which exceed a given deadline, it is advantageous to apply flow control that deliberately rejects part of the incoming traffic. Thus, some messages which might be lost anyway are prevented from accessing the system, a fact which contributes to increasing the fraction of the accepted messages making their deadlines. The optimization problem we have studied applies to networks supporting real-time traffic, a situation which is expected to be commonly encountered. The solution to this optimization problem and its structural properties were analyzed in section 2. A specific criterion for determining load conditions under which it is indeed optimal to apply flow control was also provided (equation (9)). Approaches towards developing explicit algorithms to implement flow control and routing for the case of n parallel links were presented in section 3.

The analysis presented in this paper appears to be the point of departure for a variety of interesting extensions. First, as mentioned in the Introduction, we have considered the case where flow control is applied in the form of probabilistic message rejections at a source node. However, our analysis can easily be applied to the case where all messages must be assigned to a link. Second, we can introduce weights for the components $f_i(\lambda_i)$ of the performance metric so as to penalize rejection more than delinquency in some desirable fashion. This setup would model systems where messages are of some use when delinquent. Third, it is possible that our explicit results for the M/M/1 link models may be extended to more general situations, so as to obtain analytical solutions using the algorithm in section 3.1.

As discussed in section 3.2, of great interest is the wide applicability of stochastic gradient estimation and on-line optimization techniques that take advantage of observed system data with little or no information concerning parameters or distributions involved in an analytical model. Motivated by the results in [10],[11], we believe that such techniques will indeed be useful in this framework, in order to develop adaptive schemes for flow control and routing in networks with real-time traffic.

An obvious extension of great interest is the generalization of our results to more complex networks. Consider, for instance, a virtual circuit model, consisting of several links in tandem. Since a link is often shared by many virtual circuits, flow control at a source node presents the opportunity to improve performance for all real-time traffic sharing that link. Moreover, it is possible that rejection of messages can be implemented not only at the source node of a virtual circuit, but also at intermediate nodes. This appears to be a fruitful area for further research. In addition, the presence of finite buffers provides yet another opportunity to benefit from rejections by simultaneously reducing blocking probabilities.

Another challenging problem concerns the extension of our flow allocation problem for real-time traffic to the case of dynamic policies, i.e. policies taking into account some system state information (e.g. instantaneous queue lengths in making link assignments). Some of the techniques used in [9] seem promising for this situation.

REFERENCES

[1] J. F. Kurose and R. Chipalkatti, "Load Sharing in Soft Real-Time Distributed Computer Systems," *IEEE Trans on Computers*, vol. C-36, 8, pp. 993-1000, 1987.

[2] P. Nain and K. W. Ross, "Optimal Priority Assignment with Hard Constraint," *IEEE Trans. on Automat. Contr.*, vol. AC-31, 10, pp. 883-888, 1986.

[3] A. Tantawi and D. Towsley, "Optimal Static Load Balancing in Distributed Computer Systems," *J. Ass. Comput. Mach.*, vol. 32, pp. 445-465, 1985.

[4] D. Eager, E. Lazowska, and J. Zahorjan, "Dynamic Sharing in Homogeneous Distributed Systems," *IEEE Trans. on Software Engin.*, vol. SE-12, pp. 662-675, 1986.

[5] B. Gavish and P. Schweitzer, "The Markovian Queue with Bounded Waiting Time," *Management Sci.*, vol. 23, pp. 1349-1357, 1977.

[6] C. G. Cassandras and S. G. Strickland, "On-Line Sensitivity Analysis of Markov Chains," *IEEE Trans. on Automat. Contr.*, vol. AC-34, 1, pp. 76-86, 1989.

[7] Y. C. Ho, "Performance Evaluation and Perturbation Analysis of Discrete Event Dynamic Systems," *IEEE Trans. on Automat. Contr.*, vol. AC-32, 7, pp. 563-572, 1987.

[8] M. I. Reiman and A. Weiss, "Sensitivity Analysis via Likelihood Ratios," *Proc. 1986 Winter Simulation Conf.*, pp. 285-289, 1986.

[9] C. G. Cassandras and J.-I. Lee, "Applications of Perturbation Analysis to Optimal Resource Sharing in Discrete Event Systems," *Proc. 1988 American Control Conf.*, pp. 450-455, 1988.

[10] C. G. Cassandras, M.V. Abidi, and D. Towsley, "Distributed Routing with On-Line Marginal Delay Estimation," to appear in *IEEE Trans. on Commun.*, 1989.

[11] C. G. Cassandras, S. G. Strickland, and J.-I. Lee, "Discrete Event Systems with Real-Time Constraints: Modeling and Sensitivity Analysis," *Proc. 27th Conf. Decision and Control*, pp. 220-225, 1988.

APPENDIX V

ROUTING IN NETWORKS WITH REAL-TIME TRAFFIC: A COMPARATIVE STUDY BETWEEN TWO EFFICIENT METHODS FOR ASSIGNING ROUTING VARIABLES

ROUTING IN NETWORKS WITH REAL-TIME TRAFFIC: A COMPARATIVE STUDY BETWEEN TWO EFFICIENT METHODS FOR ASSIGNING ROUTING VARIABLES [1]

Michelle Hruby Kallmes and **Christos G. Cassandras**
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003
Tel. (413) 545-1340
Telefax (413) 545-0724

ABSTRACT

We address the problem of routing *real-time traffic* in a network, where jobs must arrive at their destination within given deadlines, otherwise they are considered delinquent. The overall objective is to route jobs such that the probability of delinquency is minimized, but such an optimization is often impossible to perform. For the case of N parallel servers, we present and compare two sub-optimal methods for assigning routing variables. The first method is quite popular; routing variables are chosen so as to minimize the mean system time. The second method and its comparison with the first constitute the major contribution of the paper. We offer evidence that a sub-optimal approach of maximizing a function of the mean and variance of the system time produce results that are better than those obtained through a method of minimizing the mean system time.

April 1989

[1] This work is supported in part by the Office of Naval Research under contract N00014-87-K-0304 and by the Rome Air Development Center under contract F30602-88-D-0027.

1. INTRODUCTION

In this paper, we address issues of performance analysis and control for systems with *real-time traffic*. Real-time traffic is characterized by *deadlines* within which service must be completed. A job's deadline is defined to be the maximum amount of time the job can spend in the system; if a job's system time exceeds the deadline, the job is considered *delinquent*. Real-time traffic is encountered in at least two settings: in communication networks where messages are required to reach their destination within a specified time interval, and in multiprocessor computer systems where tasks must be executed under real-time constraints [1]-[3]. We will limit ourselves to systems that can be modeled by a network of N parallel servers; our ideas can be applied to communication networks consisting of N parallel links, computer systems comprising N shared processors, or in general, any system that fits the model.

Traditionally, much work has been done on systems with non-real-time traffic where emphasis is placed on serving jobs quickly and efficiently. A problem that has received considerable attention is that of routing jobs to N servers where the objective is to minimize the mean delay of a job. Several routing algorithms that minimize mean job delay have been developed [4]-[6]. Mean job delay, which is both meaningful and easy to calculate, is a conventional choice for an objective function. However, for systems with real-time constraints, a more natural objective function is the fraction of jobs that miss their deadline. Therefore, we consider the problem of routing jobs to N servers so as to minimize the fraction of jobs that do not complete service within some specified deadline.

We will label jobs that do not complete service within their deadline as *delinquent*. That is, a job is considered delinquent if its *system time* exceeds the deadline, where the system time is the total time that a job is in the system, from the time it enters the network of N servers until the time it completes service at one of the servers and leaves the network. Then the problem is to route jobs so as to minimize the *probability of delinquency*. Unfortunately, such an optimization is often impossible to perform due to a lack of information about the objective function (the probability of

delinquency) and how it is affected by the routing variables. There are two approaches to gathering this information: (a) derive an analytical expression for the objective function in terms of the control variables, and (b) estimate the sensitivity of the performance measures with respect to the control variables with on-line measurements. A time-domain closed-form expression for the probability of delinquency as a function of the routing variables is attainable only in the case where the N parallel queues are M/M/1 and the system parameters are known. There is always the possibility of estimating the objective function (the tail of the system time distribution) in terms of the system parameters but such an approach has the disadvantages that it requires knowledge of the system parameters and yields an inexact expression. An on-line estimation for the affect of the routing variables on the probability of delinquency requires approximating the sensitivity of the tail size with respect to the routing parameters, which is difficult and requires a large number of samples [7]. Clearly, for non-M/M/1 queues, one cannot use an exact expression for the objective function in the optimization problem. Either one can be satisfied optimizing the approximate expression for the objective function, or one can consider sub-optimal policies. Both approaches introduce error.

It has been suggested that employing a sub-optimal method for determining routing variables would simplify the optimization process without fully compromising the overall objective of minimizing the probability of delinquency [8]. A routing allocation may be determined using the limited information that is available, and one hopes that the resulting policy gives near optimal performance. Consider the case were the mean and variance of the system time can be either calculated or estimated. Certainly for M/G/1 queues, the Laplace transform of the system time is known, from which one can derive the mean and variance. Furthermore, there have been several advances in on-line derivative estimation of statistics such as the mean and variance for the case of general queues [9],[10]. These estimation techniques, which could be incorporated in an on-line routing algorithm, do not require knowledge of system parameters and are simpler and quicker than estimating the sensitivity of the tail of the distribution.

We present two alternative routing methods that result in sub-optimal routing variables.

The first method, which consists of resorting to a policy that minimizes the mean system time, has been suggested by others [8]. The second method is our contribution and relies on more information about the system time distribution, namely the variance; the method consists of maximizing a metric that is a natural function of the mean and variance. We require the sub-optimal methods to satisfy the following properties: (a) the associated performance measure must be easy to compute or estimate, and (b) the resulting routing allocation must *nearly* minimize the true objective function, the probability of delinquency. We offer evidence that the second method often results in better routing variables than does the first. The objectives of this paper are to develop the second sub-optimal method, to study when and why the two methods differ, and to suggest which method to use in a given situation.

The paper is organized as follows. In section 2, the model and the objective function are formalized. We then present the method for determining the optimal routing variables, develop two sub-optimal methods for determining routing variables, and make a conjecture concerning the sub-optimal methods. In section 3, we test the conjecture for the special case of Poisson arrivals. Theoretical support for the sub-optimal methods is given in section 4. Finally, in section 5, we summarize our results and indicate directions for further work.

2. MODEL DESCRIPTION AND PROBLEM FORMULATION

Consider a stream of jobs arriving to N parallel servers according to an arbitrary arrival process with rate $\lambda > 0$, as shown in Fig. 1. Servers are independent with finite service rates $\mu_i > 0$ for $i = 1,\ldots,N$, where the total system capacity, the sum of the individual service rates, is at least λ. Other than the restrictions stated above, service-time distributions are arbitrary. Let ϕ_i be the fraction of incoming jobs sent to server i, for $i = 1,\ldots N$. In addition, let τ denote the finite deadline associated with each job, which we assume to be fixed. Finally, assume infinite queues.

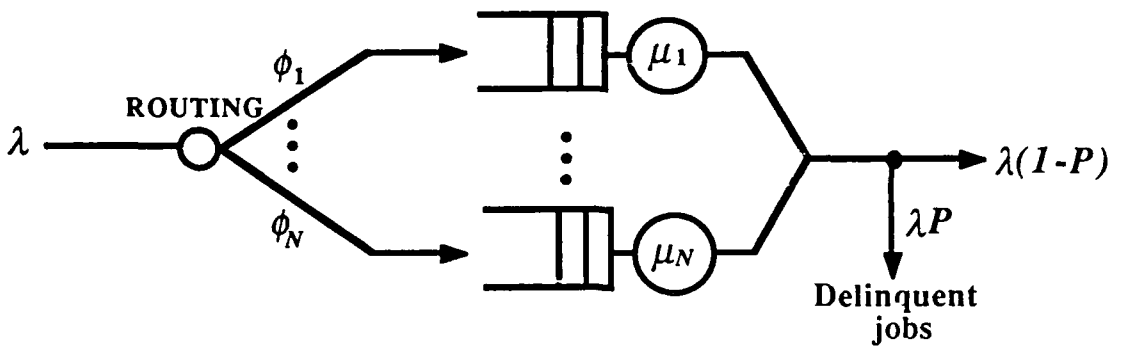


Figure 1: *Routing in N-Server Model*

2.1. The Formal Objective

Ideally, our *objective* is to determine a routing vector $\Phi = (\phi_1,\ldots,\phi_N)$ so as to minimize the *probability of delinquency*, where a job is considered delinquent if it misses its deadline. Let P_τ be the probability a job is delinquent with respect to a deadline of τ, and let T be the system time of a random job. Then we define

$$P_\tau = \Pr\{T > \tau\} = 1 - F_T(\tau)$$

where $F_T(\cdot)$ is the cumulative distribution of T. We assume that the fixed deadline τ is greater than the mean E[T] that results from our choice of routing variables. Note that for a problem to be interesting, τ must be large enough to be a realistic deadline, and the assumption would hold. Clearly P_τ depends on the routing vector Φ since the distribution of T is affected by the amount of traffic routed to each server. We will refer to $P_\tau(\Phi)$ as the *objective function.* A formal statement of the optimization problem is

$$\min_{\Phi} P_\tau(\Phi) \tag{1}$$

subject to

$$\sum_{i=1}^{N} \phi_i = 1, \tag{2}$$

$$0 \le \phi_i \le \frac{\mu_i}{\lambda}, \qquad i = 1,\ldots,N. \tag{3}$$

Equation (2) is a conservation constraint, and (3) arises because the routing variables must be non-negative and sufficiently small that the capacity of a server is not exceeded. Note that although job

flow may not exceed the capacity of a server, it may saturate the server. Let $\Phi^*(\tau)$ be the routing vector that is optimal with respect to the overall performance objective $P_\tau(\Phi)$ for deadline τ. Unfortunately, for most systems of interest, the distribution of T, $F_T(\cdot)$, is unknown and difficult to estimate, hence the objective function cannot be evaluated, and a closed-form expression for $\Phi^*(\tau)$ cannot be found.

2.2. Two Sub-Optimal Methods

Since the optimal routing problem is often difficult to solve, we present two alternative routing schemes that result in sub-optimal routing variables. Suppose that the first and second moments of T can be found, either by calculation or by estimation. In addition, assume that T has a finite mean and variance. Then define two *performance measures* that depend exclusively on the first two moments of T and not on the distribution itself: (1) the mean system time, which we will denote by

$$M = \mathrm{E}[T], \tag{4}$$

and (2) the normalized distance of the mean system time from the deadline,

$$D_\tau = \frac{\tau - \mathrm{E}[T]}{\sigma_T}, \tag{5}$$

where σ_T^2 is the variance of T. Once again, M and D_τ depend on Φ, as does σ_T. Note that the performance measure M is independent of the value of the deadline, τ. In choosing (5), we sought a measure that is a function of the mean and variance; (5) is a natural choice because of its normalizing qualities: it is the distance of the mean system time from the deadline normalized by the standard deviation [11]. Using the performance measures defined above, we present two sub-optimal methods for routing jobs to the N servers:

$$\min_{\Phi} M(\Phi) \qquad \text{subject to (2) and (3),} \qquad \text{(Method 1)}$$

$$\max_{\Phi} D_\tau(\Phi) \qquad \text{subject to (2) and (3).} \qquad \text{(Method 2)}$$

Method 1 is the existing method of minimizing the mean; method 2 is a new method of

maximizing a *distance* function of the mean and variance. We question which sub-optimal method for choosing Φ best satisfies the overall objective of minimizing P_τ. That is, if we define Φ^M and $\Phi^D(\tau)$ to be the routing allocations that are optimal with respect to performance measures M and D_τ for deadline τ, then the issue we investigate is whether $P_\tau(\Phi^M) < P_\tau(\Phi^D(\tau))$ or vice versa. To do so, define the fractional error in method 1 as

$$\Delta^M(\tau) = \frac{P_\tau(\Phi^M) - P_\tau(\Phi^*(\tau))}{P_\tau(\Phi^*(\tau))},$$

and, similarly for method 2,

$$\Delta^D(\tau) = \frac{P_\tau(\Phi^D(\tau)) - P_\tau(\Phi^*(\tau))}{P_\tau(\Phi^*(\tau))}.$$

Clearly both $\Delta^M(\tau)$ and $\Delta^D(\tau)$ will be positive for all $\tau > 0$ since $P_\tau(\Phi^*(\tau))$ is optimal. For a given τ, we desire to use the method producing the smallest error. We have evidence that there is a critical τ above which method 2 has a smaller error, and below which method 1 has less error. In subsequent sections, we present examples and a theoretical argument as support.

3. EXAMPLES: N PARALLEL M/M/1 SERVERS

If we consider systems with Poisson arrivals and with known service-time distributions and system parameters, we are able to find a closed-form expression for $P_\tau(\Phi)$, and using numerical methods, we can find the optimal routing allocation $\Phi^*(\tau) = (\phi_1^*, \ldots, \phi_N^*)$. We use such systems as a starting point for comparing how well the two methods, minimizing the mean M and maximizing the distance D_τ, approximate the optimal solution.

Given a Poisson arrival stream and probabilistic routing of jobs that enter the system, arrivals to each server i form a Poisson process of rate $\phi_i \lambda$. Let T_i denote the random variable describing the system time of a job routed over link i, and let $F_{T_i}(\cdot)$ denote the cumulative distribution function for T_i. Then, by the Law of Total Probability, and by the definitions of $F_{T_i}(\cdot)$ and ϕ_i,

$$F_T(\tau) = \Pr\{T \le \tau\} = \sum_{i=1}^{N} \Pr\{\text{routed to server } i\} \cdot \Pr\{T_i \le \tau \mid \text{routed to server } i\}$$

$$= \sum_{i=1}^{N} \phi_i F_{T_i}(\tau). \tag{6}$$

It follows that

$$P_\tau(\Phi) = \Pr\{T > \tau\} = 1 - F_T(\tau) = \sum_{i=1}^{N} \phi_i [1 - F_{T_i}(\tau)]. \tag{7}$$

The last equality follows from (6) and the fact that the routing variables must sum to one. Since $F_{T_i}(\cdot)$ can be calculated from knowledge of the arrival rate and the service-time distribution then, for a system with Poisson arrivals, one can find $P_\tau(\Phi)$.

The subsequent analysis holds for $0 \le \phi_i \le \mu_i/\lambda$, $i = 1,\ldots,N$. If we further assume exponential servers and a FCFS service discipline, then T is a hyperexponential random variable and (7) becomes

$$P_\tau(\Phi) = \sum_{i=1}^{N} \phi_i e^{-(\mu_i - \phi_i\lambda)\tau}, \tag{8}$$

and the probability density function of T, $f_T(\cdot)$, is

$$f_T(t) = \sum_{i=1}^{N} \phi_i (\mu_i - \phi_i\lambda) e^{-(\mu_i - \phi_i\lambda)t}.$$

Then we can calculate the first and second moments of T:

$$\mathrm{E}[T] = \int_0^\infty t f_T(t)\, dt = \sum_{i=1}^{N} \frac{\phi_i}{\mu_i - \phi_i\lambda}, \tag{9}$$

$$\mathrm{E}[T^2] = \int_0^\infty t^2 f_T(t)\, dt = \sum_{i=1}^{N} \frac{2\phi_i}{(\mu_i - \phi_i\lambda)^2}. \tag{10}$$

From (9) and (10) we can find analytic expressions for the performance measures in methods 1 and 2.

$$M(\Phi) = \mathrm{E}[T] = \sum_{i=1}^{N} \frac{\phi_i}{\mu_i - \phi_i \lambda}, \tag{11}$$

$$D_\tau(\Phi) = \frac{\tau - \mathrm{E}[T]}{\sigma_T} = \frac{\tau - \sum_{i=1}^{N} \frac{\phi_i}{\mu_i - \phi_i \lambda}}{\left[\sum_{i=1}^{N} \frac{2\phi_i}{(\mu_i - \phi_i \lambda)^2} - \left(\sum_{i=1}^{N} \frac{\phi_i}{\mu_i - \phi_i \lambda} \right)^2 \right]^{\frac{1}{2}}}. \tag{12}$$

3.1. Example 1: Two M/M/1 Servers.

Consider a system consisting of two servers in parallel, each exponential, with rates $\mu_1 = 1.0$ and $\mu_2 = 0.5$. Let the overall arrival process be Poisson with rate $\lambda = 1.0$. By (2), $\phi_2 = 1 - \phi_1$. Then it is easily shown that for $0.5 \le \phi_1 \le 1.0$,

$$P_\tau(\phi_1) = \phi_1 \, e^{-(1-\phi_1)\tau} + (1-\phi_1) \, e^{-(\phi_1 - 0.5)\tau},$$

$$M(\phi_1) = -\frac{4\phi_1^2 - 5\phi_1 + 2}{2\phi_1^2 - 3\phi_1 + 1},$$

$$D_\tau(\phi_1) = -\frac{(2\tau+4)\phi_1^2 + (-3\tau-5)\phi_1 + (\tau+2)}{\sqrt{-16\phi_1^4 + 40\phi_1^3 - 25\phi_1^2 - 2\phi_1 + 4}}.$$

By setting $\phi_2 = 1 - \phi_1$, we remove the constraint (2) from the optimization problems to get the following single-variable unconstrained boundary-valued optimization problems:

$$\min_{\phi_1} P_\tau(\phi_1) \quad \text{subject to} \quad 0.5 \le \phi_1 \le 1.0,$$

$$\min_{\phi_1} M(\phi_1) \quad \text{subject to} \quad 0.5 \le \phi_1 \le 1.0,$$

$$\max_{\phi_1} D_\tau(\phi_1) \quad \text{subject to} \quad 0.5 \le \phi_1 \le 1.0.$$

First consider the problem of minimizing $P_\tau(\phi_1)$. Clearly $P_\tau(\phi_1)$ is convex. In addition, it can be shown that the optimal routing variable $\phi_1^*(\tau)$ lies on neither boundary. Therefore the necessary and sufficient condition for optimality given the unconstrained statement of the problem

is

$$\frac{\partial P_\tau(\phi_1)}{\partial \phi_1} = 0. \tag{13}$$

That is, $\phi_1^*(\tau)$ is the root of (13) lying in (0.5,1.0). One cannot solve for $\phi_1^*(\tau)$ analytically, but, given τ, one can find $\phi_1^*(\tau)$ numerically, and we do so for several values of τ.

Next consider the problem of minimizing $M(\phi_1)$. Once again, $M(\phi_1)$ is convex and the optimal routing variable ϕ_1^M for the unconstrained problem lies in the open interval (0.5,1.0). Therefore ϕ_1^M is the root of

$$\frac{\partial M(\phi_1)}{\partial \phi_1} = 0$$

that lies in (0.5,1.0). Here we can analytically solve for ϕ_1^M, and we find that $\phi_1^M = 1/\sqrt{2}$.

Finally, consider the problem of maximizing $D_\tau(\phi_1)$. One can show that $D_\tau(\phi_1)$ is unimodal in the interval [0.5,1.0] and that the optimum $\phi_1^D(\tau)$ does not lie on the boundary. Therefore, $\phi_1^D(\tau)$ is the root of the equation

$$\frac{\partial D_\tau(\phi_1)}{\partial \phi_1} = 0$$

that lies in (0.5,1.0). The problem reduces to finding the root of the quartic equation

$$(8\tau)\phi_1^4 + (-42\tau-64)\phi_1^3 + (66\tau+132)\phi_1^2 + (-44\tau-87)\phi_1 + (11\tau+18)$$

that lies in the interval. We have done so for various values of τ.

Recalling that $P_\tau(\Phi)$ and $D_\tau(\Phi)$ depend on τ, we conclude that Φ^* and Φ^D also depend on τ; Φ^M, however, does not. Fig. 2 illustrates the behavior of $P_\tau(\phi_1)$, $M(\phi_1)$, and $D_\tau(\phi_1)$ for the case of $\tau = 15$ sec. With the knowledge of the sub-optimal routing variables ϕ_1^M and $\phi_1^D(\tau)$ and the optimal routing variable $\phi_1^*(\tau)$, we can calculate the error in the objective function resulting from each method. In both methods, the absolute errors, $[P_\tau(\Phi^M) - P_\tau(\Phi^*(\tau))]$ and $[P_\tau(\Phi^D(\tau)) - P_\tau(\Phi^*(\tau))]$ approach zero as $\tau \to \infty$. Of more interest are the scaled or fractional errors $\Delta^M(\tau)$ and $\Delta^D(\tau)$ (defined in section 2.2), which we have graphed in Fig. 3 as a function of τ. Clearly there is a critical deadline, τ_c, above which $\Delta^D(\tau) < \Delta^M(\tau)$ and below which $\Delta^M(\tau) < \Delta^D(\tau)$. From the graph, we observe that $\tau_c \approx 10$. This example supports our conjecture of section 2.2, that method

2 results in better performance when the deadline is above some critical value.

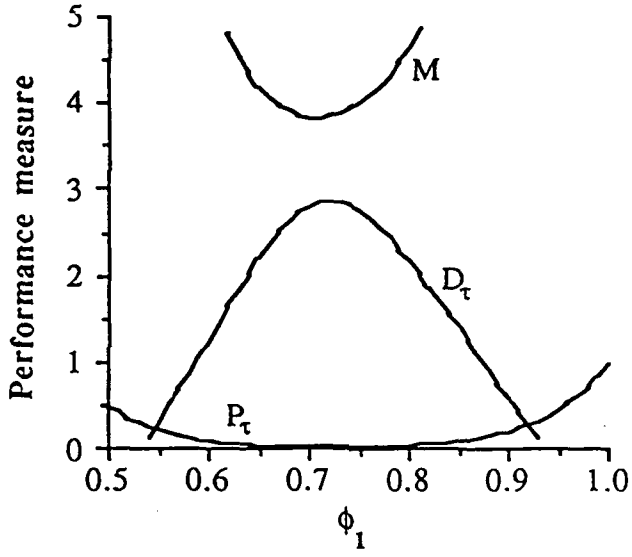


Figure 2: *Behavior of Performance Measures*

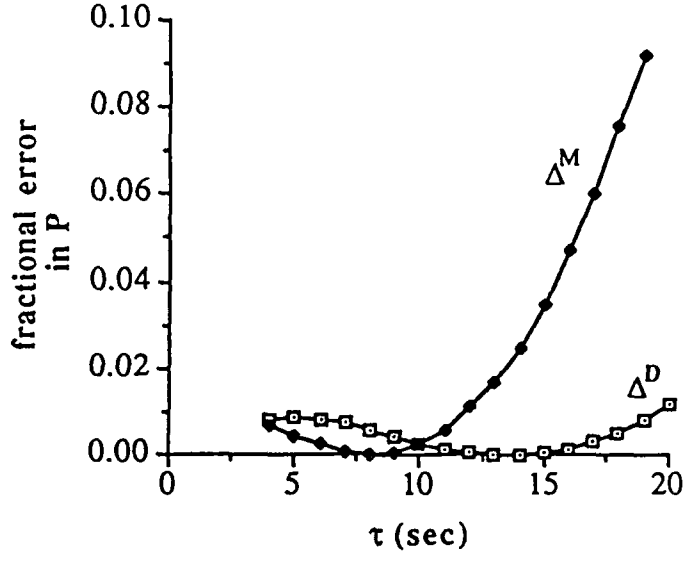


Figure 3: *Error Resulting from Sub-Optimal Methods (Example 1)*

3.2. Example 2: Three M/M/1 Servers.

Let the system consist of three parallel servers with capacity $\mu_1 = 1.00$, $\mu_2 = 0.75$, and $\mu_3 = 0.50$, and let the service times be exponential. Assume that the overall arrival process is Poisson with

rate $\lambda = 1.00$. Then, from (8), the objective function is

$$P_\tau(\phi_1,\phi_2,\phi_3) = \phi_1 e^{-(1-\phi_1)\tau} + \phi_2 e^{-(0.75-\phi_2)\tau} + \phi_3 e^{-(0.5-\phi_3)\tau}; \qquad \begin{array}{l} 0 \le \phi_1 \le 1, \\ 0 \le \phi_2 \le 0.75, \\ 0 \le \phi_3 \le 0.5. \end{array}$$

Once again, the problem of minimizing $P_\tau(\Phi)$ cannot be solved analytically, but $P_\tau(\Phi)$ is convex and it can be shown that $\Phi^*(\tau)$ does not lie on a boundary. Here we do not reduce the problem to its unconstrained counterpart, but instead leave it as a multivariable constrained optimization problem specified in (1)-(3) and apply Marquardt's method to minimize $P_\tau(\Phi)$. Marquardt's method is an optimization algorithm that combines the universal convergence properties of Cauchy's algorithm with the speed of Newton's algorithm [12]. Using Marquardt's method, we solve for $\Phi^*(\tau)$ for several values of τ.

In the problem of minimizing the mean, replace ϕ_3 with $1-(\phi_1+\phi_2)$ to get the unconstrained multivariable boundary-value problem

$$\min_{(\phi_1,\phi_2)} M(\phi_1,\phi_2) \quad \text{subject to} \quad \begin{array}{l} \max\,(0,\, 0.5-\phi_2) \le \phi_1 \le 1-\phi_2, \\ 0 \le \phi_2 \le 0.75 \end{array}$$

with

$$M(\phi_1,\phi_2) = \frac{a(\phi_1,\phi_2)}{b(\phi_1,\phi_2)} \qquad \begin{array}{l} 0 \le \phi_1 \le 1, \\ 0 \le \phi_2 \le 0.75, \end{array}$$

where $a(\phi_1,\phi_2)$ and $b(\phi_1,\phi_2)$ are fourth-order polynomials in ϕ_1 and ϕ_2. Since $M(\phi_1,\phi_2)$ is convex and since it can be shown that (ϕ_1^M,ϕ_2^M) does not lie on a boundary then (ϕ_1^M,ϕ_2^M) is the root of the simultaneous equations

$$\frac{\partial M(\phi_1,\phi_2)}{\partial \phi_1} = 0, \qquad \frac{\partial M(\phi_1,\phi_2)}{\partial \phi_2} = 0$$

that lies in the region $\max(0,0.5-\phi_2) \le \phi_1 \le 1-\phi_2$, $0 \le \phi_2 \le 0.75$. The analytically obtained solution for $\Phi^M = (\phi_1^M,\phi_2^M,\phi_3^M)$ is (0.5142, 0.3293, 0.1565).

For the problem of maximizing $D_\tau(\Phi)$, we set ϕ_3 to $1-(\phi_1+\phi_2)$ and apply Cauchy's best-step steepest-descent algorithm to the unconstrained optimization problem with two independent variables, ϕ_1 and ϕ_2. So the problem is

$$\max_{(\phi_1,\phi_2)} \left\{ D_\tau(\phi_1,\phi_2) = \frac{c_\tau(\phi_1,\phi_2)}{\sqrt{d(\phi_1,\phi_2)}} \right\} \quad \text{subject to} \quad \begin{array}{l} \max(0, 0.5-\phi_2) \le \phi_1 \le 1-\phi_2, \\ 0 \le \phi_2 \le 0.75, \end{array}$$

where $c_\tau(\phi_1,\phi_2)$ is a third-order polynomial in ϕ_1 and ϕ_2 involving τ, and $d(\phi_1,\phi_2)$ is a polynomial in ϕ_1 and ϕ_2 of the sixth order. Using Cauchy's algorithm, we solve for $(\phi_1^D(\tau),\phi_2^D(\tau))$ for several values of τ. Then $\phi_3^D(\tau) = 1-(\phi_1^D(\tau)+\phi_2^D(\tau))$.

In Fig. 4, the fractional errors, $\Delta^M(\tau)$ and $\Delta^D(\tau)$, are plotted as a function of τ. Note that these curves have the same general nature as do the curves in Fig. 3. Once again, there exists a critical deadline, $\tau_c \approx 6$, that divides the graph into regions according to which method results in the least error.

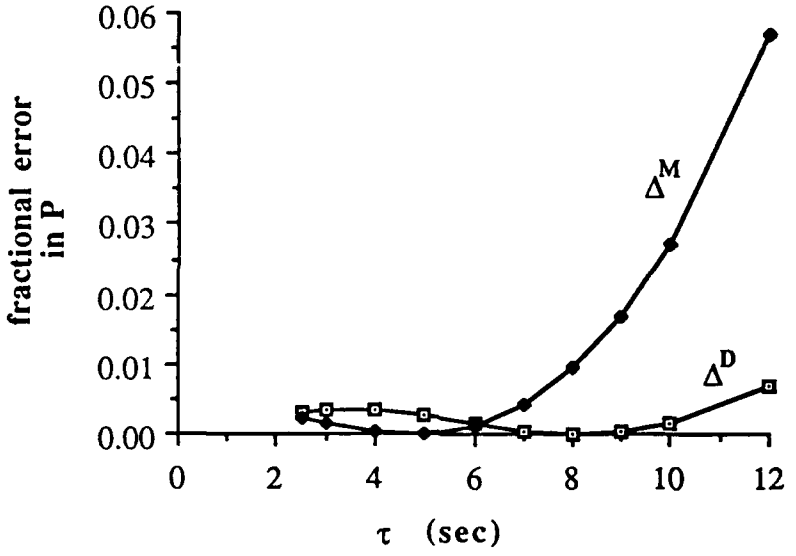


Figure 4: *Error Resulting from Sub-Optimal Methods (Example 2)*

4. THEORETICAL JUSTIFICATION FOR THE CHOICE OF THE SUB-OPTIMAL METHODS

The crossover results begin to make sense if one considers certain well-known results from probability theory, Markov's Inequality and the One-Sided Inequality. Whereas the ultimate objective is to minimize the probability of delinquency, the two sub-optimal methods presented in section 2 effectively minimize certain upper bounds on the probability of delinquency, and the

bounds themselves display a crossover behavior.

Markov's Inequality [13]: *Let T be a random variable with expected value* E[T] *and such that* $\Pr\{T<0\} = 0$. *Then, for each* $t > 0$,

$$\Pr\{T \geq t\} \leq \frac{\mathrm{E}[T]}{t}.$$

If we consider $t = \tau$ where τ is the deadline, then

$$\frac{\mathrm{E}[T]}{\tau} \tag{14}$$

bounds $\Pr\{T \geq \tau\}$, the area of the tail above τ, which in our case is the probability of delinquency, P_τ, for a deadline of τ. By choosing the routing variables that minimize E[T], as is specified by method 1, we are minimizing the limiting value of the Markov inequality that bounds the probability of delinquency.

One-Sided Inequality [13]: *Let T be a random variable with mean* E[T] *and variance* σ_T^2. *Then, for* $t >$ E[T],

$$\Pr\{T > t\} \leq \frac{\sigma_T^2}{\sigma_T^2 + (t - \mathrm{E}[T])^2}.$$

Since we have assumed that $\tau >$ E[T], we can consider the case where $t = \tau$. Then

$$\frac{\sigma_T^2}{\sigma_T^2 + (\tau - \mathrm{E}[T])^2} \tag{15}$$

bounds the probability of delinquency, P_τ, for a deadline of τ, as does (14). By maximizing D_τ as proposed in method 2, we are minimizing (15) because

$$\frac{\sigma_T^2}{\sigma_T^2 + (\tau - \mathrm{E}[T])^2} = \frac{1}{D_\tau^2 + 1}$$

from the definition of D_τ in (5), so we are choosing the routing variables that minimize the one-sided bound.

Since the distribution of the system time is affected by the choice of routing variables, so too are the mean, variance, tail, and bounds on the tail. By "minimizing a certain bound," we are

choosing the routing variables that give the particular distribution with the smallest bound on the tail.

When we question which is the tighter bound, we find that the bounds, like our results in section 3, have a crossover point. For small t, the Markov inequality provides a tighter bound on the size of the tail. Otherwise, the one-sided inequality is a stronger statement. Solving for the values of t where

$$\frac{\mathrm{E}[T]}{t} = \frac{\sigma_T^2}{\sigma_T^2 + (t - \mathrm{E}[T])^2},$$

we find that the curves cross at

$$t_1 = \mathrm{E}[T] \quad \text{and} \quad t_2 = \mathrm{E}[T] + \frac{\sigma_T^2}{(\mathrm{E}[T])^2}.$$

The second crossing point is the one of interest. It can be shown that for $t_1 < t < t_2$, the Markov bound is tighter, and for $t > t_2$, the one-sided bound is tighter. Fig. 5 illustrates the relationship between the two bounds for the case where $\mathrm{E}[T] = 3.8$ and $\sigma_T = 3.9$.

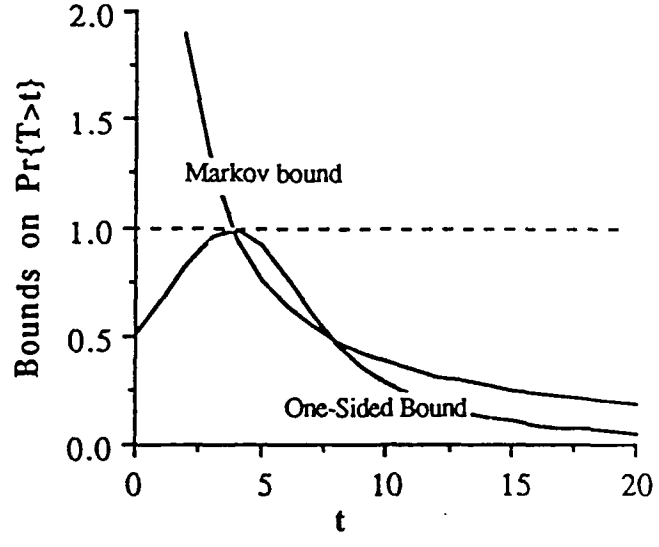


Figure 5: *Markov and One-Sided Bounds on Pr[T>t] where E[T] = 3.8 and σ_T = 3.9*

Note that τ is a parameter of the system, and $\mathrm{E}[T]$ and σ_T^2 are a result of the routing variables chosen.

In the problem we address, T is the system time, and we let $t = \tau$. Then E[T] and σ_T depend on the choice of routing variables, which in turn depends both on the method used for choosing the routing variables and on the deadline. For the sake of illustrating this result, we will assume the simple system of Example 1 where we can compute the probability of delinquency, hence we can find the optimal routing variables and the corresponding mean and variance. This allows us to draw a graph of the bounds, Fig. 6. In order to show how the inequalities bound the probability of delinquency, we have graphed the optimal probability of delinquency, $P_\tau(\Phi^*(\tau))$, along with the bounds on P_τ which were computed using the mean E[T] and variance σ_T^2 that correspond to the optimal routing, $\Phi^*(\tau)$.

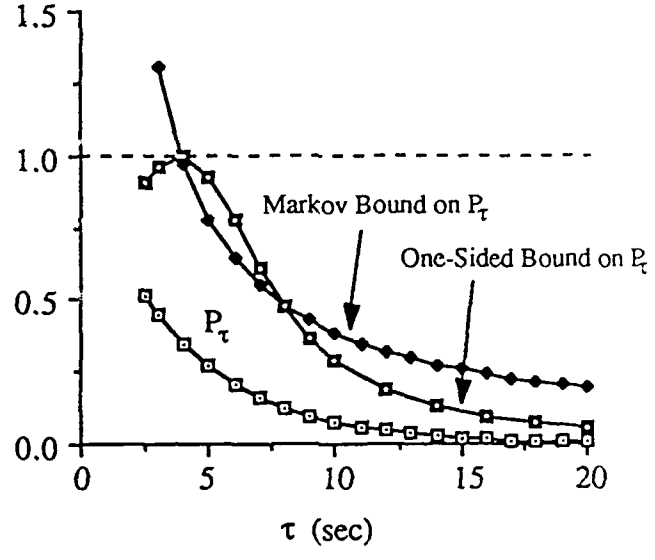


Figure 6: *Bounds on P_τ for the System of the First Example*

Of most interest is the fact that the bounds, like the curves in section 3, have a crossover point, which we know to occur at

$$\tau_2 = \mathrm{E}[T] + \frac{\sigma_T^2}{(\mathrm{E}[T])^2}.$$

As evidenced by the subsequent figure, the critical deadline, τ_c, lies near τ_2. Hence, by calculating τ_2, we know approximately where the method of maximizing the distance starts to outperform the method of minimizing the mean. Define the fractional error of the Markov bound and the one-

sided bound:

$$\Delta^{MB}(\tau) = \frac{\frac{\mathrm{E}[T]}{\tau} - P_\tau(\Phi^*(\tau))}{P_\tau(\Phi^*(\tau))} \quad \text{and} \quad \Delta^{OB}(\tau) = \frac{\frac{\sigma_T^2}{\sigma_T^2 + (\tau - \mathrm{E}[T])^2} - P_\tau(\Phi^*(\tau))}{P_\tau(\Phi^*(\tau))}.$$

In Fig. 7, we have superimposed the plots of these error curves with those of Δ^M and Δ^D. Although the crossover points do not exactly correspond, there is certainly a one-to-one correspondence between the two pairs of curves. Note that the two pairs of curves have different orders of magnitude, hence in superimposing them, we lose the quantitative quality of the independent axis.

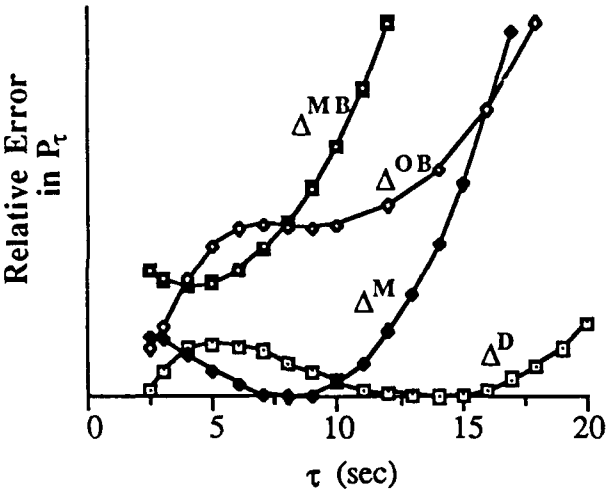


Figure 7: *Comparison of Crossover Points for Two Sets of Error Curves*

5. CONCLUSIONS AND FUTURE WORK

In systems where the ultimate goal is to route jobs to N servers so as to minimize the fraction of jobs that do not complete service within some specified deadline, it is not always possible to find the optimal routing allocation. One alternative approach to assigning routing variables, sub-optimal method 1, is to route jobs so as to minimize the mean system time. We present a second alternative approach, sub-optimal method 2, where the routing variables are chosen such that the normalized distance of the mean from the deadline is maximized. Through examples, we have shown support

for our conjecture that if the deadline is above a critical value, τ_c, then method 2 outperforms method 1.

To further test the conjecture, we plan to analyze systems with Poisson arrivals and non-exponential servers. Analysis of such systems can be accomplished using a combination of analytical and numerical methods. In general, relaxing the Poisson assumption leads to models that are hard to analyze, and one must resort to simulation. Alternatively, however, one may use recently developed on-line sensitivity estimation techniques to build gradient-based optimization algorithms where closed-form expressions of the performance measures are not required. This approach was successfully applied in [14] and could be employed to test our hypothesis for non-Poisson systems.

Another avenue for further research involves the development of new performance measures. For example, if there exists a bound on the tail of a distribution where the bound is tighter than both the Markov bound and the one-sided bound, then the bound could be transformed into a performance measure that may outperform those already in existence.

An obvious extension to our work is the generalization of our results to more complex systems with non-constant deadlines. Further, the deadlines could apply to quantities other than the system time. For example, deadlines could be imposed on queueing delay or on service time.

REFERENCES

[1] J. F. Kurose and R. Chipalkatti, "Load Sharing in Soft Real-Time Distributed Computer Systems," *IEEE Trans. on Computers*, vol. C-36, 8, pp. 993-1000, 1987.

[2] K. J. Lee and D. Towsley, "On the Analysis of a Decentralized Load Balancing Policy in Heterogeneous Distributed Systems," subm. to *IEEE Trans. on Comput.*, 1987.

[3] P. Nain, and K. W. Ross, "Optimal Priority Assignment with Hard Constraint," *IEEE Trans. on Automat. Contr.*, vol. AC-31, 10, pp. 883-888, 1986.

[4] R. G. Gallager, "A Minimum Delay Routing Algorithm Using Distributed Computation," *IEEE Trans. on Comm.*, vol. COM-25, pp. 73-85, Jan. 1977.

[5] D. P. Bertsekas, E. M. Gafni, and R. G. Gallager, "Second Derivative Algorithms for Minimum Delay Distributed Routing in Networks," *IEEE Trans. on Comm.*, vol. COM-32,

pp. 911-919, Aug. 1984.

[6] M. Schwartz and T. E. Stern, "Routing Techniques Used in Computer Communication Networks," *IEEE Trans. on Comm.*, vol. COM-28, pp. 539-552, Apr. 1980.

[7] C. G. Cassandras and J-I. Lee, "Application of Perturbation Analysis to Optimal Resource Sharing in Discrete Event Systems," *Proc. 1988 American Control Conf.*, pp. 450-455, 1988.

[8] G. Van Leemput, "Design Issues for Higher-Level Protocols in High Speed Communication Networks." Master's thesis, University of Massachusetts, Jan. 1988.

[9] Y. C. Ho, "Performance Evaluation and Perturbation Analysis of Discrete Event Dynamic Systems," *IEEE Trans. on Automat. Contr.*, vol. AC-32, pp. 563-572, 1987.

[10] *Proc. of IEEE: special issue on Dynamics of Discrete Event Systems*, vol. 77, January 1989.

[11] W. Feller, *An Introduction to Probability Theory and its Applications*, vol. I. New York: Wiley, 1964.

[12] G. V. Reklaitis, A. Ravindran, and K. M. Ragsdell, *Engineering Optimization: Methods and Applications*. New York: Wiley, 1983.

[13] A. O. Allen, *Probability, Statistics, and Queueing Theory*. New York: Academic Press, 1978.

[14] C. G. Cassandras, M. V. Abidi, and D. Towsley, "Distributed Routing with On-Line Marginal Delay Estimation," *Proc. 1988 INFOCOM Conf.*, 1988.

APPENDIX VI

OPTIMALITY OF THE LAST-IN-FIRST-OUT (LIFO) SERVICE DISCIPLINE IN QUEUEING SYSTEMS WITH REAL-TIME CONSTRAINTS

OPTIMALITY OF THE LAST-IN-FIRST-OUT (LIFO) SERVICE DISCIPLINE IN QUEUEING SYSTEMS WITH REAL-TIME CONSTRAINTS [1]

Michelle Hruby Kallmes
Dept. of Electrical and Computer Engineering

Don Towsley
Dept. of Computer and Information Science

Christos G. Cassandras
Dept. of Electrical and Computer Engineering

University of Massachusetts
Amherst, MA 01003

ABSTRACT

We consider a G/G/1 queueing system where the objective is to maximize the probability that a customer's system time does not exceed a given *deadline*. The deadline is defined to be the maximum amount of time the customer can spend in the system. We show that for deadlines that are i.i.d. random variables with concave cumulative distribution functions, LIFO gives the highest probability of success, and FIFO the lowest, over the class of all work-conserving non-preemptive service disciplines that are independent of service time and deadline. Extensions to policies that allow preemption in G/M/1 systems, deadlines imposed on queueing (rather than system) time, and multi-server queues are provided. Results are illustrated by the simple example of an M/M/1 queue with exponential deadlines. In addition, the case of a G/G/1 queue with constant deadlines is considered.

February 1989

28th CDC

---

[1] This work is supported in part by the Office of Naval Research under contracts N00014-87-K-0304 and N00014-87-K-0796 and by the Rome Air Development Center under contract F30602-88-D-0027.

1. INTRODUCTION

We consider a G/G/1 queueing system where every arriving customer is characterized by a *deadline D*. A customer's deadline is defined to be the maximum amount of time the customer can spend in the system. Deadlines are assumed to be i.i.d. random variables with a cumulative distribution function $G_D(\cdot)$. If a customer's system (sojourn) time exceeds the assigned deadline, then the customer is considered to be *lost*. (Note: a customer that exceeds his deadline is not removed from the system.) This model is particularly useful in communication systems where messages are required to reach their destination within a specified time interval, and in computer systems where tasks must be executed under real-time constraints.

Let Π_{np} denote the class of work-conserving, non-preemptive service disciplines that are independent of the service times and deadlines. Thus, when the server adopts a policy $\pi \in \Pi_{np}$, it selects at every service completion instant a customer among those presently in the queue without any information regarding service times or deadlines of the customers. Letting R_π denote the ergodic system time of a customer under $\pi \in \Pi_{np}$, the problem is to determine a policy π^* optimal in Π_{np} such that the probability $\Pr[R_\pi \le D]$ is maximized. Our main result in this paper is that the Last-In-First-Out (LIFO) policy is optimal, as long as $G_D(t)$ is concave in t. Moreover, the First-In-First-Out (FIFO) policy provides the worst $\Pr[R_\pi \le D]$ in this case.

The result is particularly interesting in view of the fact that for the common case of constant deadlines ($D = \tau$), no such universal claim about the optimality of a certain policy can be made without specific knowledge of the parameters of the system and deadline. For example, although LIFO outperforms FIFO in the case of *tight* deadlines (small τ), FIFO provides a higher $\Pr[R_\pi \le D]$ than LIFO for *loose* deadlines (large τ).

With our key result as a starting point, it is possible to provide extensions to the class of policies allowing preemption, and to the problem of maximizing $\Pr[Q_\pi \le D]$, where Q_π is the queue waiting time. In addition, we extend the results to single-server queues with bulk arrivals, and to multiple-server queues. Furthermore, the result raises a number of questions pertaining to the properties of the LIFO policy in models where deadlines play a critical role in determining

system performance. For instance, can the LIFO policy still be optimal if $G_D(t)$ does not satisfy the condition above? It is also possible that LIFO outperforms FIFO even when the latter policy is used in conjunction with state-dependent admission control to the queueing system.

This paper is organized as follows. Section 2 defines a convex ordering and states some existing results. The main result is derived in section 3; extensions to this result are provided in section 4. In section 5, we present examples of queues with (a) exponential deadlines, and (b) constant deadlines.

2. DEFINITIONS AND PRELIMINARY RESULTS

Our main result is based on the idea of establishing a certain type of partial ordering between two random variables X and Y defined over all non-negative real numbers. In particular, we shall use the following definition of a *convex* ordering [1]:

Definition: *X is smaller than Y in the convex ordering, denoted by $X \leq_c Y$, if and only if:*

$$E[g(X)] \leq E[g(Y)]$$

for any convex function g: $[0,\infty) \rightarrow \Re$.

The following Theorem, due to Shanthikumar and Sumita [1], establishes a convex ordering for the ergodic system time in a G/G/1 queueing system. Recall that Π_{np} is the class of work-conserving non-preemptive service disciplines that are independent of the service times and deadlines. Clearly, the FIFO and LIFO policies belong to this class.

Theorem 1a: *For any $\pi \in \Pi_{np}$, let R_π be the ergodic system time in a G/G/1 queueing system under π. Then:*

$$R_{FIFO} \leq_c R_\pi \leq_c R_{LIFO}.$$

Shanthikumar and Sumita [1] derive a similar result for the G/M/1 queue and the more general class Π of work-conserving service disciplines that are independent of the service times and deadlines. In this case, preemption is allowed, and we use LIFO-P to represent the LIFO preemptive/resume policy, which belongs to Π.

Theorem 1b: *For any $\pi \in \Pi$, let R_π be the ergodic system time in a G/M/1 queueing system under π. Then:*

$$R_{\text{FIFO}} \leq_c R_\pi \leq_c R_{\text{LIFO-P}}.$$

The proofs of these Theorems in [1] are based on interchange arguments applied to sample paths of the queueing system under consideration. It can be shown, for instance, that if some policy $\pi \in \Pi_{np}$ other than FIFO is used in a G/G/1 system, then interchanging the first customer not served under FIFO with the customer satisfying the FIFO policy results in a lower value of the ergodic system time in the convex ordering sense.

3. NON-PREEMPTIVE G/G/1 QUEUES

The following theorem establishes the optimality of the LIFO service discipline; furthermore, it shows FIFO to be the worst.

Theorem 2: *Let deadlines be i.i.d. random variables with a cumulative distribution function $G_D(\cdot)$. For a G/G/1 queue and any service discipline $\pi \in \Pi_{np}$, if $G_D(t)$ is a concave function of t, then*

$$\Pr[R_{\text{LIFO}} \leq D] \geq \Pr[R_\pi \leq D] \geq \Pr[R_{\text{FIFO}} \leq D]. \tag{1}$$

Proof: Define the following function:

$$h(r) = \Pr[D \geq r] = 1 - G_D(r).$$

But $h(r)$ is convex since $G_D(r)$ is concave; therefore $h(R_\pi)$ is convex and it follows by Theorem 1a that

$$\mathrm{E}[h(R_{\text{LIFO}})] \geq \mathrm{E}[h(R_\pi)] \geq \mathrm{E}[h(R_{\text{FIFO}})].$$

Let $F_\pi(\cdot)$ be the cumulative distribution function of R_π under policy π. Observing that

$$\mathrm{E}[h(R_\pi)] = \int_0^\infty \Pr[D \geq R_\pi \mid R_\pi = r]\, dF_\pi(r) = \Pr[D \geq R_\pi],$$

the result (1) follows. **QED**

Remark: Constraining the cumulative distribution function to be concave is equivalent to requiring the probability density function to be non-increasing for $t > 0$. For example, a deadline

that is exponentially distributed would satisfy this condition, as would one that is distributed uniformly on the interval $[0,b]$, $b > 0$.

4. EXTENSIONS

The following results are easily derived as extensions of Theorem 2. We omit the proofs.

G/M/1 Queues with Preemption Allowed. Consider the larger set of policies, Π, where preemption is allowed. If we restrict ourselves to exponential servers, then with application of Theorem 1b, the following ordering result can be obtained for concave $G_D(\cdot)$:

$$\Pr[R_{\text{LIFO-P}} \le D] \ge \Pr[R_\pi \le D] \ge \Pr[R_{\text{FIFO}} \le D]. \tag{2}$$

Queues with Deadlines on Queueing Time. We can extend the results of of the previous sections to systems where the deadline applies to queueing time rather than system time. That is, we can replace R_π by Q_π where Q_π is the stationary queueing time, and the results will still hold. The proof relies on certain properties of convex ordering that can be found in [2, p.272].

Queues with Bulk Arrivals. Here we show that the results presented in the previous sections can be extended to bulk arrival queues. For single-server queues where the bulk size of the nth arrival is random, Theorems 1a and 1b still hold [1]. Therefore, it can be shown that the inequalities in equations (1) and (2) hold for queues with bulk arrivals assuming that the conditions on the deadline distributions are met.

Multiple-Server Queues. We can show that the results of Theorem 2 can be extended to G/G/c queues. The proof entails first extending the convex ordering result of Shanthikumar and Sumita [1] to apply to multiple-server queues.

5. EXAMPLES

We will look at the two special cases of an M/M/1 queue with exponential deadlines, and a G/G/1 queue with constant deadlines. In the case of exponential deadlines, the required property is satisfied for Theorem 2 to hold, and we give an example to show the magnitude of the difference

between the FIFO, LIFO, and LIFO-P policies. In addition, we will show that in the case of a constant deadline, whose cumulative distribution function does not satisfy the concavity property, no policy is always optimum over all choices of the deadline. In fact, we can show that there exists a critical deadline, τ_{crit}, above which one policy outperforms the other, and below which the converse is true.

5.1. M/M/1 Queue with Exponential Deadlines

Consider an M/M/1 queueing system with all deadlines drawn from an exponential distribution. Let the arrival rate be λ; the service rate μ; and the mean deadline $\frac{1}{\theta}$. Using standard queueing theoretic techniques, we can show that the analytical expressions for the probabilities of interest are:

$$\Pr[R_{\text{LIFO-P}} \le D] = \frac{1}{2\lambda}\left(\theta + \lambda + \mu - \sqrt{(\theta+\lambda+\mu) - 4\lambda\mu}\,\right), \tag{3}$$

$$\Pr[R_{\text{LIFO}} \le D] = \begin{cases} \dfrac{1}{2(\theta+\mu)}\left(\theta - \lambda + 3\mu - \sqrt{(\theta+\lambda+\mu) - 4\lambda\mu}\,\right), & \lambda < \mu, \\ \dfrac{\mu}{2\lambda(\theta+\mu)}\left(\theta + \lambda + \mu - \sqrt{(\theta+\lambda+\mu) - 4\lambda\mu}\,\right), & \lambda \ge \mu, \text{ and} \end{cases} \tag{4}$$

$$\Pr[R_{\text{FIFO}} \le D] = \begin{cases} \dfrac{\mu - \lambda}{\theta + \mu - \lambda}, & \lambda < \mu, \\ 0, & \lambda \ge \mu. \end{cases} \tag{5}$$

Through algebraic manipulation of equations (3)-(5), one can show that, indeed, the following ordering is satisfied for any choice of λ, μ, and θ:

$$\Pr[R_{\text{LIFO-P}} \le D] \ge \Pr[R_{\text{LIFO}} \le D] \ge \Pr[R_{\text{FIFO}} \le D].$$

To illustrate the difference in performance under the different policies, we plot $\Pr[R_\pi \le D]$ for $\pi =$ LIFO-P, LIFO, and FIFO as a function of λ for the case where $\mu = 1$ and $\theta = 1/2$ (see Fig. 1).

5.2. Constant Deadlines

Consider a G/G/1 queue where deadlines are constant and equal to τ. We can show that for two policies $\pi_1, \pi_2 \in \Pi_{np}$ such that $R_{\pi_1} \le_c R_{\pi_2}$, there exists a critical deadline, τ_{crit}, such that

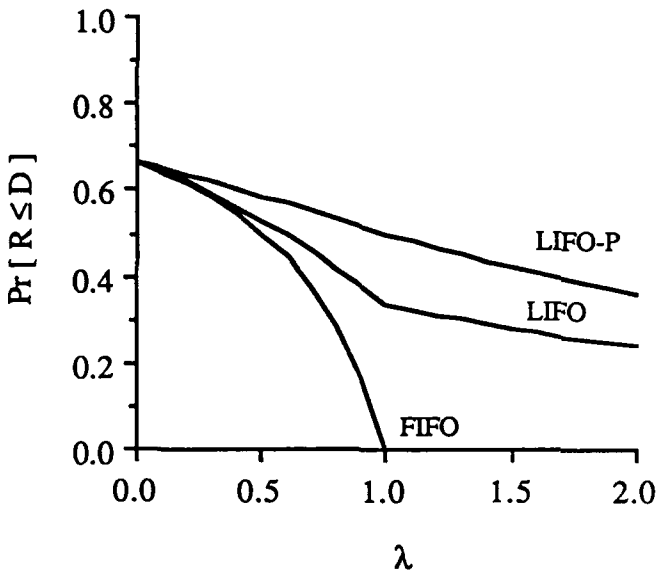


Figure 1: *Comparison of LIFO-P, LIFO, and FIFO Service Disciplines for an M/M/1 Queue with Exponential Deadlines*

$$\Pr[R_{\pi_1} \le \tau] \begin{cases} < \Pr[R_{\pi_2} \le \tau], & \tau > \tau_{\text{crit}}, \\ = \Pr[R_{\pi_2} \le \tau], & \tau = \tau_{\text{crit}}, \\ > \Pr[R_{\pi_2} \le \tau], & \tau < \tau_{\text{crit}}. \end{cases} \tag{6}$$

The proof comes from the fact that $\Pr[R_\pi \le \tau]$ is just the cumulative distribution function (cdf) of R_π and a result in [3, p.107] stating that if two random variables with the same mean have a convex ordering, then their cdf's cross exactly once and in a direction consistent with (6).

Remark: The critical deadline, τ_{crit}, depends on the parameters of the interarrival and service-time distributions. Therefore, to compare two policies, one needs exact values for (a) the deadline, and (b) the system parameters.

6. CONCLUSIONS

The LIFO service discipline in G/G/1 systems is often regarded as undesirable, since it results in a system time distribution with the same mean but higher variance compared to FIFO. In the context of the problem we have considered, however, this feature can be exploited. Our key result is that for G/G/1 queues where customers have deadlines drawn from a concave cumulative distribution

function, a LIFO service discipline is optimal over the set of work-conserving non-preemptive policies that are independent of service demands and deadlines. We also show FIFO to be the worst policy. In addition, we extend the basic result to policies that allow preemption and to more general systems.

Our results suggest several interesting questions pertaining to the desirable properties of the LIFO service discipline. For example, we can ask how a LIFO policy compares to a FIFO policy with state-dependent admission control. There is, in fact, evidence that LIFO can do better even in this case. Another idea is to make the same comparisons in a system where customers are removed from the queue as soon as they are detected to have exceeded their deadline. Further, we can question whether LIFO is optimal for any deadline distributions that do not satify the concavity constraints.

REFERENCES

[1] J. G. Shanthikumar and U. Sumita, "Convex Ordering of Sojourn Times in Single-Server Queues: Extremal Properties of FIFO and LIFO Service Disciplines," *J. Appl. Prob.*, vol. 24, pp. 737-748, 1987.

[2] S. M. Ross, *Stochastic Processes*, Wiley (New York), 1983.

[3] R. Barlow and F. Proschan, *Statistical Theory of Reliability and Life Testing*, To Begin With (Silver Spring, MD), 1981.