UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(12)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: 3940
2. GOVT ACCESSION NO.: AD-A086 338
3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): (6) ARPANET ROUTING ALGORITHM IMPROVEMENTS
5. TYPE OF REPORT & PERIOD COVERED: Semiannual Technical Report 4-1-78--10-1-78
6. PERFORMING ORG. REPORT NUMBER: 3940
7. AUTHOR(s): (10) J. M. McQuillan, E. C. Rosen, I. Richer, D. P. Bertsekas
8. CONTRACT OR GRANT NUMBER(s): (15) MDA903-78-C-0129
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Bolt Beranek and Newman Inc., 50 Moulton St., Cambridge, MA 02138
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 060 100 ARPA Order-3491, ARPA Order No. 3491
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1404 Wilson Blvd., Arlington, VA 22209
12. REPORT DATE: (11) Oct[illegible]978
13. NUMBER OF PAGES: 271
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Defense Supply Service--Washington, Room 1D 245, The Pentagon, Washington, DC 20310 (12) 278
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:
16. DISTRIBUTION STATEMENT (of this Report): UNCLASSIFIED/UNLIMITED
(14) BBN-3940
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
(9) Semiannual technical rept. no. 2, 1 Apr-30 Sep 78
18. SUPPLEMENTARY NOTES

DTIC ELECTE JUL 9 1980 S D C

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
computer networks, routing algorithms, ARPANET, protocols, congestion control, updating, line up/down procedures, network delay, network measurement, logical addressing, group addressing, physical addressing, multi-destination addressing, broadcasting (continue

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
This report describes progress during the second six months of a contract to make several improvements to ARPANET routing. A new line up/down protocol has been designed and implemented, and its performance has been extensively measured under actual network conditions. The protocol is described and the measurements are presented. Software routines to measure and report average packet delays on a per-hop basis have been designed and imple- (continu

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

060100

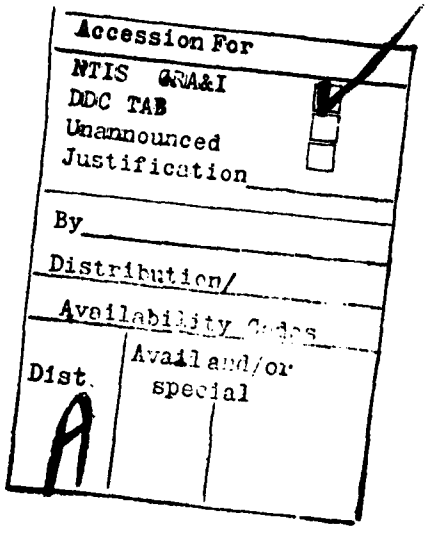
(19.....continued)

flooding, distributed data base, stability of routing algorithms

---

(20.....continued)

mented. The routines are described, and measurements which have been performed using the routines are presented. A rigorous mathematical analysis of the stability of the SPF algorithm is presented. A protocol to ensure the rapid and reliable transmission of routing update messages has been designed. It is described, and the reasons for choosing it are discussed. A preliminary design of enhanced message capabilities for the ARPANET (logical addressing, broadcast addressing, and group addressing) is presented. Finally, the interactions between routing and congestion control are discussed.



BBN Report No. 3940

ARPANET Routing Algorithm Improvements

Second Semiannual Technical Report

October 1978

SPONSORED BY
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY AND
DEFENSE COMMUNICATIONS AGENCY (DOD)
MONITORED BY DSSW UNDER CONTRACT NO. MDA903-78-C-0129

ARPA Order No. 3491

Submitted to:

Director
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

Attention: Program Management

and to:

Defense Communications Engineering Center
1860 Wiehle Avenue
Reston, VA 22090

Attention: Dr. R. E. Lyons

TABLE OF CONTENTS

INTRODUCTION

This report covers work performed during the period from April 18, 1978 through October 15, 1978 on the contract to study, design, and implement improvements in the ARPANET routing algorithm. Our progress to date is summarized in six main sections and an appendix.

In September, a new line up/down protocol was installed in the network, after a lengthy period of analysis and measurement. The analysis is described in the First Semi-Annual Technical Report. Section 1 of the present report describes the new protocol in detail. The techniques we used to measure the performance of the protocol are also discussed, and the results of our measurements are presented.

As we indicated in the First Semi-Annual Technical Report, the routines which measure packet delay in the network are an important part of the new routing algorithm. These routines have been designed, implemented, and installed in the network. They are described in Section 2. The routines are highly parameterized, and we have performed an extensive series of measurements in order to determine the optimimum values for the parameters. Section 2 also contains a report on these measurements.

Section 3 summarizes some mathematical analysis of the stability of the SPF algorithm (and similar algorithms). The analysis indicates some possible instabilities and sub-optimalities in the algorithm. It also suggests techniques for eliminating these problems. The full details of the analysis are presented in the Appendix.

The new routing algorithm (SPF) requires each IMP to maintain a data base which specifies the topology and the delay on each network line. It is important for all IMPs to have the identical data base, even though the data base is being continuously updated by routing update messages. This requires a carefully designed protocol for transmitting the updates quickly and reliably. Section 4 describes the protocol we have designed, and compares it to other possible protocols which we have rejected as unsatisfactory.

We have continued our preliminary design of enhanced message addressng capabilities for the ARPANET, as reported in Section 5. Logical addressing, broadcast addressing, and group (or multi-destination) addressing are discussed with particular attention to implementation considerations.

Finally, we have begun to investigate congestion control techniques for the ARPANET and networks in general. This topic is one of the major open questions in networking; Section 6

presents some preliminary thoughts on the nature of this problem and how it interacts with routing.

1. LINE UP/DOWN PROTOCOL

The previous Semiannual Report (BBN Report No. 3803, April 1978) proposed some new procedures for bringing lines down and up. Specifically, we recommended using "k-out-of-n" counters, denoted (k,n). During each fixed interval of time, a line-protocol packet should be received. The (k,n) counter triggers, causing the line state to change, if within any block of n or fewer intervals, k events occur. For a line going down, an event is missing a packet, and thus the line is brought down if k packets are missed in n or fewer intervals; we showed that for proper performance, $k > 2$ should be used. For a line coming up, the appropriate event is receiving a packet; in this case we recommended a consecutive counter (i.e., $k = n$), with $n > 60$.

During the past six months we have tested and implemented these procedures on the ARPANET. This section describes in detail the work that was performed. We begin by presenting the technique we used to implement the proposed counters. The second section below discusses the measurement and testing method we employed, a method that is more powerful than conventional simulations and that will also be employed when we test the new routing algorithm itself. The final section summarizes the measurement results and the current status.

1.1 Protocol Description

The technique we have developed for implementing the recommended line up/down strategies is not symmetric for the two sides of a line, but it is straightforward and effective, and requires only modest processor resources. The basic idea is that the IMP on one side of each line (e.g., the higher numbered IMP) is considered the Master, and the other IMP is the Slave. From information in the protocol packets, an IMP determines his neighbors' numbers. The Slave acts as an "echo" for Hello's sent from the Master, and the Master is responsible for declaring the line up or down.

In a symmetric protocol, the IMPs at each end of a line perform the same tasks, and they transmit and expect to receive packets at the same rate. However, the clocks at the two IMPs generally run at slightly different rates and in addition, they are not exactly synchronized. Therefore, during the interval between successive clock ticks at one IMP, the clock at the other IMP usually ticks once, but may tick zero or two times. A significant amount of protocol logic is needed to handle all these situations. However, with the asymmetric protocol we are using, the Master is the only time keeper, so there is no problem with skewed clocks or with different clock periods on each side of a line. The elimination of these potential problems is a principal virtue of our protocol.

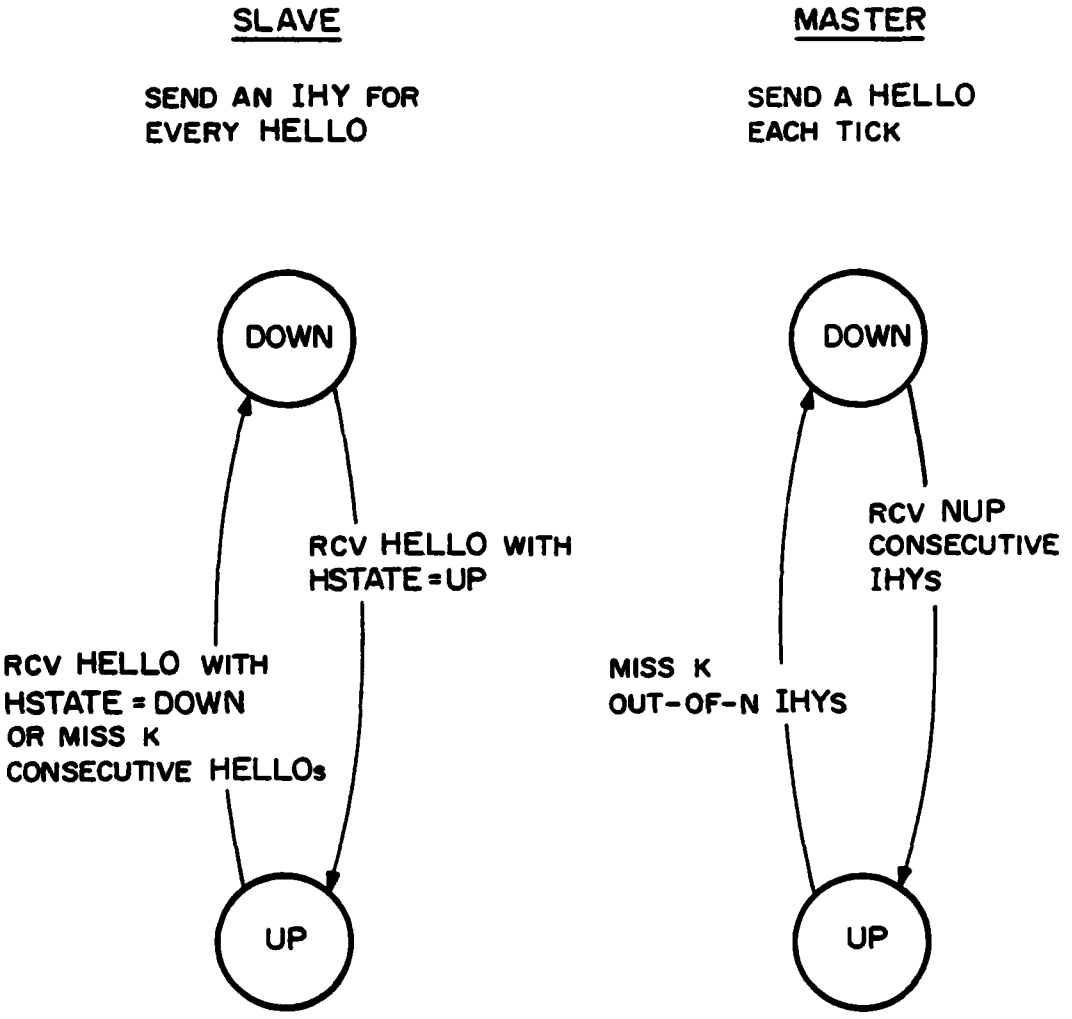
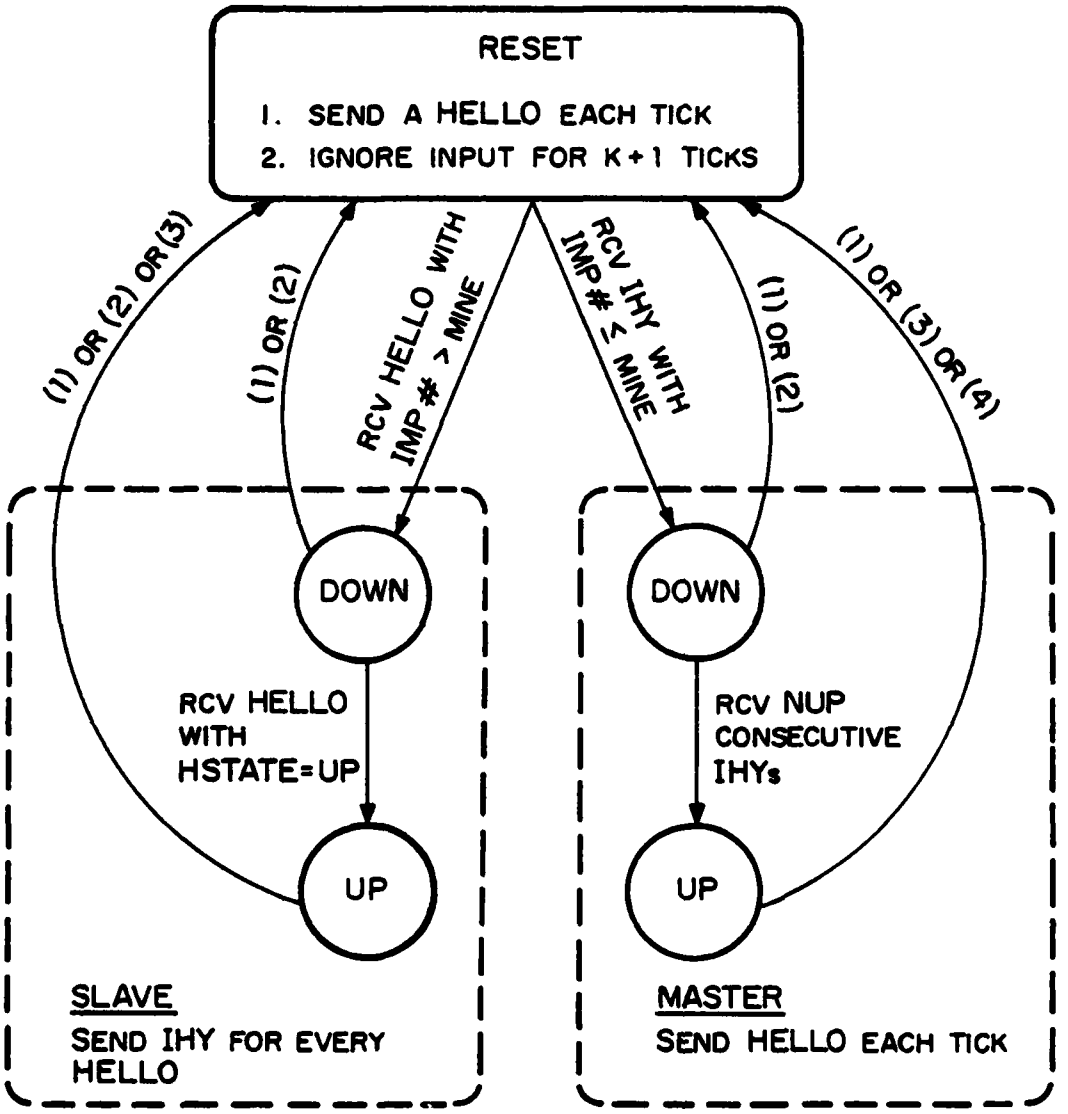


Figure 1-1 Simplified Line State Diagram

Figure 1-1 is a simplified state diagram of the protocol. As indicated in the figure, a Master sends only Hellos, a Slave only IHYs. A single bit can thus be used to distinguish between Master (or Hello) and Slave (or IHY). HSTATE denotes a bit in the Hello indicating the state of the Master. In this simplified scenario, the Slave declares the line up so long as it receives Hellos with the HSTATE bit indicating that that the Master declares the line up. The Slave declares the line down if the Master declares it down or if k consecutive Hellos are missed. The latter provision enables Slaves to detect and bring down lines that are actually inoperative. The Master changes the line status according to the (k,n) and consecutive counters. (The parameter NUP denotes the number of IHYs in the consecutive counter.)

In a more realistic scenario we must, of course, allow for other events that might necessitate a Slave bringing a line down (e.g., the sudden appearance of a new neighbor at the other end of a line, an event that occasionally occurs because of telephone company operations). In addition, an IMP must be able to dynamically determine whether it is the Master or the Slave on each of its lines. Figure 1-2 shows a complete state diagram that handles these and other situations. In the RESET state, each IMP acts as a Master, sending a Hello each tick. By transmitting Hellos, a Slave forces its Master to enter the RESET



(1) MISCELLANEOUS CAUSE - E.G., WRONG IMP # IN PACKET
(2) MISS K CONSECUTIVE HELLOS (SLAVE) OR IHYS (MASTER)
(3) RECEIVE HELLO WITH HSTATE = DOWN
(4) MISS K OUT OF N IHYS

Figure 1-2 Complete Line State Diagram

state. Also, after entering the RESET state, an IMP does not respond to any input for k+1 ticks. These rules ensure that the IMPs on each end of a line will agree on the state of the line. In effect, the RESET state replaces the READY state discussed in the previous Semiannual Report. The non-responsive interval is set at k + 1 ticks to allow for possible clock discrepancies at the two IMPs. Each Hello/IHY contains the number of the transmitting IMP, thereby enabling an IMP to determine its status on each line. In addition, if a Hello's IMP number is equal to the receiver's number, then the packet is treated as an IHY; this allows looped lines to be handled in a consistent manner.

The scheme described above depends upon the Master receiving a response to its Hello before the next tick occurs. We must therefore examine the delays involved with different links. The following tabulation assumes worst-case values and a 200-bit Hello/IHY:

Delays in Msec

	230 Kbps Land	50 Kbps Land	50 Kbps Sat.	9.6 Kbps Land	9.6 Kbps Sat.
Latency	6	25	25	130	130
Transmission	1	4	4	21	21
Propagation	20	20	260	20	260
Processing	5	5	5	5	5
2 x Total	64	108	588	352	832

Latency is the time spent waiting for the previous packet to be transmitted. The total of each column is doubled because the same delays could occur in each direction. The table shows that with terrestrial and with 50 Kbps satellite links, there is adequate margin to use slow ticks (640 ms.) to govern the Master's clock. For 9.6 Kbps satellite links, however, we must use a slower clock rate; a reasonable period for this clock is every second slow tick (1280 ms.). Finally, we note that it might be advantageous to be able to detect problems on 230 Kbps lines more quickly than on 50 Kbps lines because of the potentially greater traffic volume that might be routed toward these higher-speed lines; as discussed in Sec. 1.3 below, the clock period for line protocol packets on these lines is currently 128 ms.

1.2 Testing Technique

The line up/down procedures are basic to the proper operation of the ARPANET, and therefore any change in these procedures must be installed carefully in order to eliminate possible degradation or disruption of service. Thus, prior to their installation on the network, the procedures must be thoroughly tested, and this testing must include the basic design, the coding, the interactions with other aspects of IMP and network operation, and the selection of parameter values. Because of a variety of limitations, stand-alone tests and

conventional simulations are useful for only a small fraction of the testing and debugging. For example, there are too many related variables, some of their values unknown, for a meaningful simulation; and even if an adequate simulation could be generated, it would be very difficult to characterize "real-world" situations that occur in the ARPANET.

Our solution to this dilemma is to use the network itself for "simulations," obtaining performance estimates of new techniques while the network is functioning. In other words, our test procedures run as background tasks to normal network operation, so that we can obtain reasonably realistic and reliable data. In testing the line up/down procedures during the past few months, we have found this technique to be quite powerful, and during the subsequent months we plan to apply it as we install the new routing algorithm itself.

The specific steps that we used were as follows: The first pertinent IMP release contained the new line up/down procedures operating in a so-called "phantom" mode. That is, in addition to sending the old form of Hellos and IHYs, each IMP also sent appropriate packets as dictated by the new protocols. The old procedures actually controlled line status, but special messages ("traps") reported to the NCC when the new procedures would have changed the line status, if these procedures had been in control. These traps also contained relevant data such as the time since

the previous up or down-trap. In addition, for each line, data was gathered on the overall packet error rate and on the error rates of line-protocol packets alone. The parameters controlling the new line procedures could easily be modified, so that as we gathered data, we adjusted parameter values to achieve proper performance.

We are aware, of course, that our testing procedure is still a simulation in the sense that the observed performance of the algorithms may not be quite the same as when they actually controlled the status of all lines. For example, since the new protocol brings lines down more quickly, network traffic patterns may be different from those that presently exist; and as noted below, heavy traffic at an IMP can sometimes affect line-protocol operation at that IMP. In other words, the new protocol could affect routing, which in turn could affect line status. Nevertheless, this on-line simulation technique does produce the most realistic data that can be obtained in a practical manner. Moreover, by gathering data over long periods of time, we can observe the effects of day-to-day variations and of occasional unusual events.

The second pertinent IMP release contained a feature that allowed either the new or the old line procedure to control individual lines at each IMP, with the selection determined by the setting of a logical switch. Thus, having determined

reasonable parameters from the first release, we could now use these parameters in the new procedures for actually controlling the lines, and we could observe the effect, if any, on network operation. Recall that the old line procedures are (1) too slow in bringing poor lines down, and (2) too fast in bringing poor lines up. We were somewhat concerned that the new procedures (which eliminate these shortcomings) might alter the topology more frequently than the old, and that as a consequence, routing might be adversely affected. We therefore switched lines in a very conservative way in several stages:

(1) Gradually several lines were switched to the new procedures, but with "conservative" parameters that would bring poor lines down slowly (e.g., a (4,4) counter).

(2) The parameters on these lines were then modified to the design values.

(3) All lines were switched to the new procedures, with conservative parameters.

(4) The parameters on all lines were modified to the design values.

We ran each of these stages for several days, carefully monitoring network operation. The final stage of course corresponds to the new procedures controlling all lines.

The third IMP release (5 September 1978) contains only the new line procedures. This release also allowed different types of lines to have different parameters and different protocol clock rates. Again we proceeded cautiously, operating with

conservative parameter values for several days before finally substituting the design values. With this latest release, each IMP sends a conventional trap to the NCC whenever the status of a line changes. We plan to monitor line behavior during the next few months in order to determine whether any further adjustment of the protocol parameters is necessary.

1.3 Measurement Results

Extensive measurements were made during a six-week period of the following quantities for each of the lines in the networks:

- the average error rate of all packets
- the average rate of missed IHYs
- the number of phantom line-downs and ups
- the number of clock periods separating the first and the k-th tick in a line-down
- the durations of line up and line down periods

The data on packet errors was approximate (an under-estimate) because only checksum errors, and not missed packets, were included. Packets can be missed entirely if their first two bytes are damaged or if the IMP is not responsive enough. In the first case, since the first two bytes signal the

start of a packet to the hardware of the receiving interface, if these are damaged, the IMP is unaware that a packet is on the line. (Similarly, if the last two bytes are damaged, the following packet may be lost, and only one checksum error will be recorded.) In the second case, if two successive packets are closely spaced, the IMP might not have sufficient time between the completion of the first packet and the start of the second to allocate a buffer for the second. If this happens, the second packet is lost.

Some of these measurements were continued beyond the six-week period, and several will be collected during the subsequent months. During the measurement period, the size of the line-protocol packet was 280 bits (on the line). This size was used because we had planned to allow 128 bits for update acknowledgments in each packet. As discussed in Section 4, we have since adopted a different ACK scheme, so that protocol packets are now 152 bits. This change does not affect any of our conclusions.

As noted earlier, the measurements were intended to aid us in debugging the implementation and to guide us in selecting parameters. In addition to accomplishing these goals, the measurements also led us to an understanding of an important aspect of network operation. Our observations were as follows:

(1) Shortly after we initiated the measurements, we observed that the high-speed (230 Kbps) lines went phantom down significantly more often than 50 Kbps lines. (When the new protocol operated in the phantom node, the line status, as determined by this protocol, was called "phantom up" or "phantom down.") Also, the high frequency with which they went phantom down was inconsistent both with the measured average missed IHY rate and with the average error rate of all packets. Moreover, the missed IHY rate on these lines was substantially greater than the measured packet error rate. (As explained above, the packet error rate is a lower estimate, so we expect it be a little bit smaller than the missed IHY rate.)

(2) After three weeks of data had been collected, we noted which lines (excluding the high-speed lines) had gone phantom down most often. The results are shown in the ARPANET map of Fig. 1-3, where very heavy lines are those that went down very often, and the mid-weight lines "moderately" often. Examination of this map reveals that (a) the lines with the worst behavior are those that tend to carry the most traffic; and (b) each weighted line is contiguous with one or more other weighted lines - that is, the problem is probably associated with the IMPs rather than with the lines.

The explanation of both observations is the same and was mentioned earlier: If two successive packets arrive on a

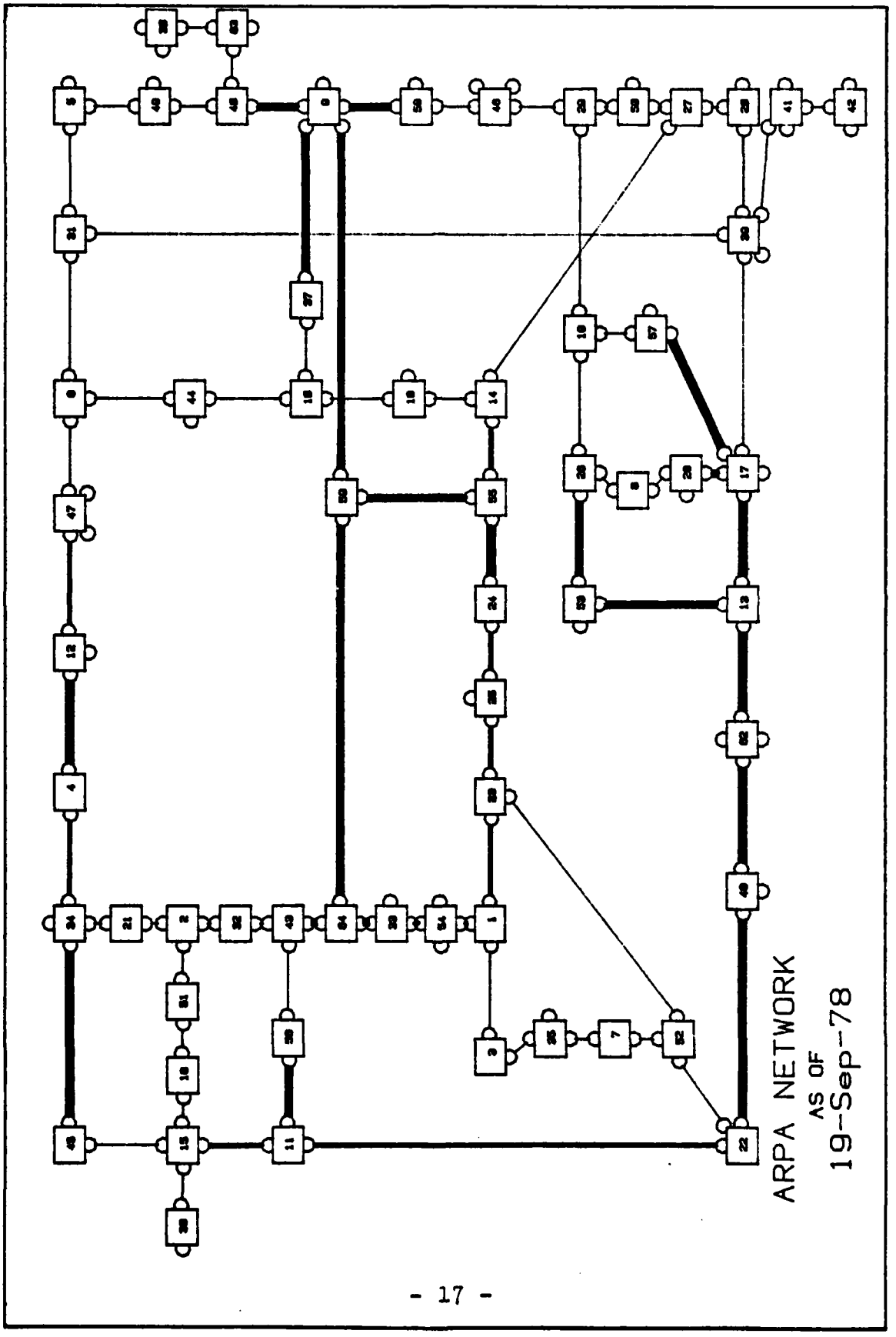


Figure 1-3 Map Showing Which Lines Went Down Most Often

particular line in short succession, then the receiving IMP may not respond quickly enough to a completion interrupt of the first packet, and the second packet may be dropped. The minimum required response time is 5-6 character-times, or roughly 0.2 ms. on a high-speed line, 1 ms. on a 50 Kbps line. Clearly, missed packets will be more likely on high-speed lines, but our measurements also showed that under heavy traffic, an IMP might have difficulty in responding to a packet on a 50 Kbps line.

Of course we had been aware of the fact that packets could be lost because of missed completion interrupts, but prior to these measurements, we had no data on the rate of occurrence. Some investigation revealed that a significant fraction of the missed line protocol packets could be traced to the fact that in the initial release, a phantom Hello was transmitted immediately after the periodic routing message. (That is, both packets were sent each slow tick.) When the order of these two packets was reversed, the number of phantom downs dropped considerably, and the average missed IHY rate also dropped, becoming comparable with the overall average packet error rate. (This last point indicates that the principal cause of missed IHYs is now line errors rather than actual missed packets; this average error rate is quite low, approximately 2×10^{-4}.) Efforts are being made to reduce the likelihood of missed packets by shortening those segments of IMP code which run with interrupts locked out. Such

segments include both long sequences of instructions as well as short sequences that are iterated many times. Despite these efforts, we must remain cognizant of the fact that as an IMP's traffic increases, the probability that packets on its incoming lines will be missed also increases, and that this miss probability is not independent on each of the IMP's lines. These considerations in fact entered into the design of the updating policy we have designed.

On the basis of the above considerations and the measurements we have conducted we have currently set the parameters on the 50 Kbps line to be (4,20) for bringing a line down. Figure 1-4 shows an approximation for the expected number of intervals E(N) for a (4,20) counter to bring a line down as a function of the packet error probability p. The curve is based upon the equations

$$E(N) \geq k/p$$

$$E(N) \approx \left[\binom{n-1}{k-1} p^k \right]^{-1} \qquad np << 1$$

(We believe, but have not proven, that the second equation above is also a lower bound.) Note that the knee of the curve is near p=0.1, so that lines are allowed to remain up unless there is a significant likelihood of a packet error. Also shown in Fig. 1-4

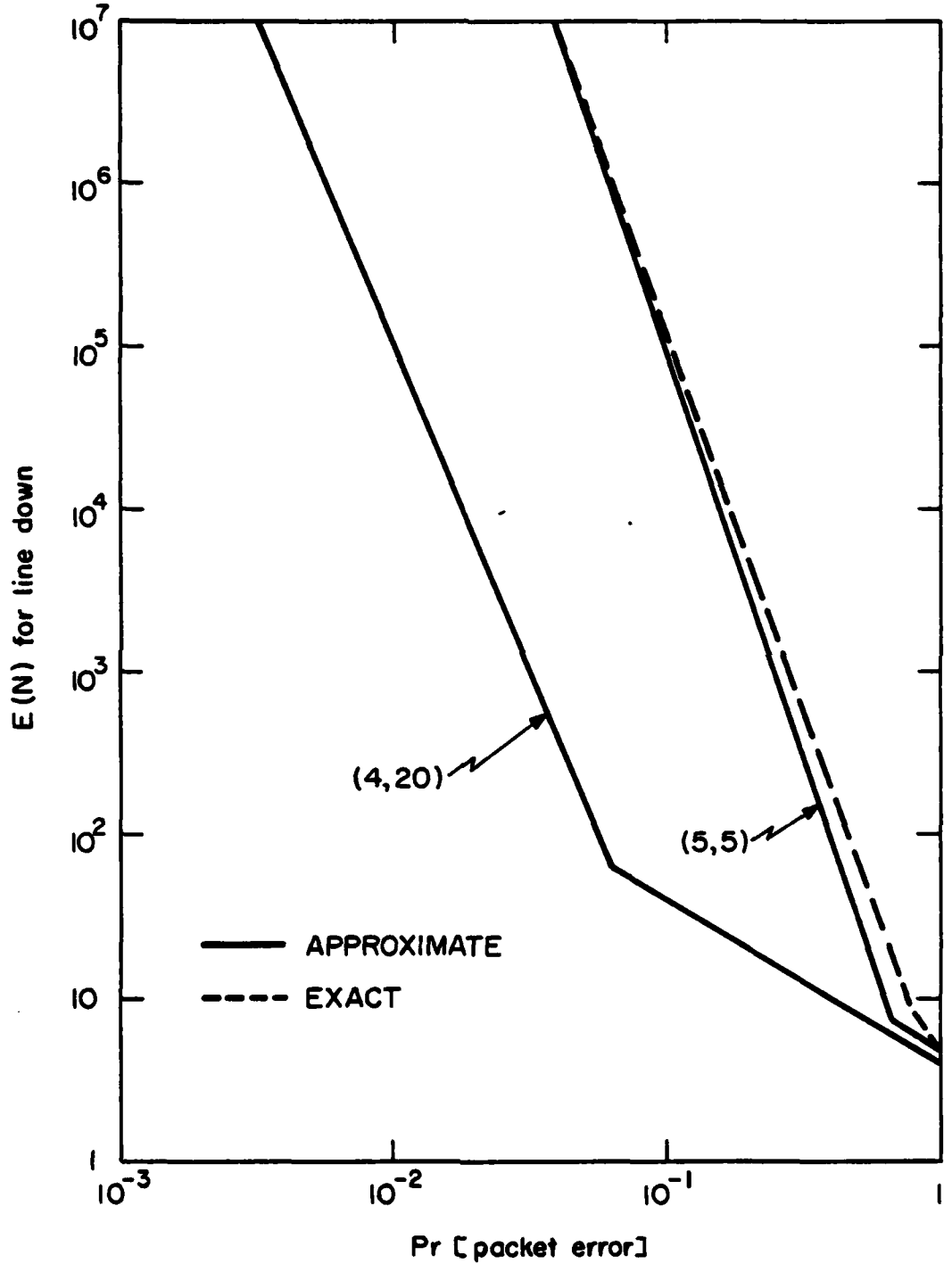


Figure 1-4 Performance of Present Line Down Counters

is an approximate and an exact curve for a (5,5) counter, which corresponds to the old line down protocol; we used these parameters when the new protocol was first installed in the network. The following table compares the performance of these two counters over successive periods of network operation:

	(5,5)	(4,20)
number of days in measurement	8	8
average number of times down per line per day	0.33	0.54
average excluding lines from IMP53	0.26	0.46

The averages above were taken only over the 50 Kbps lines. The second row of averages excludes the two lines from IMP53 (EGLIN) because these lines are exceptionally troublesome, contributing almost 25% to the total number of line downs; that is, these averages are more representative of overall network performance. The measured number of line downs due to the new protocol also includes some downs which were introduced when lines were looped and some which were caused by the slave bringing a line down for a "miscellaneous" reason. Thus, the averages above are slightly greater than those caused by the line protocol alone. The data show that (1) the new (4,20) counter does increase the rate at which lines are brought down, and (2) the average rate of line

downs is not excessive. (Moreover, on a typical day, one or two different lines would contribute a significant fraction of line downs, and thus the "average" line is somewhat better than the above figures indicate.)

For a 50 Kbps line we have achieved satisfactory performance with the line up counter operating with NUP=90 consecutive IHY's. Our measurements indicated that a smaller value of NUP provides equally good performance, but because the updating procedure necessitates approximately a one-minute period for bringing a line up, we are initially using a value of NUP which guarantees this period (90 ticks takes about 58 sec).

The parameters for the high-speed lines are different from those on the 50 Kbps lines because these lines send protocol packets at a higher rate, and because the probability of a missed packet is greater. The clock rate on these lines will initially be set at five times that of 50 Kbps lines, i.e., one tick every 1/5 slow tick. Since the clock is running at a faster rate, the expected number of clock intervals between downs must be proportionately greater, and therefore we are currently using a (5,5) counter. We plan to test other parameters, e.g. (8, 16) in the coming weeks.

2. MEASURING AND REPORTING DELAY

2.1 Delay Measurement Routines

Routines to measure the delay experienced by packets in the ARPANET have been designed and implemented. These routines implement the block-average smoothing discussed in our First Semi-Annual Technical Report. That is, every n seconds, the average delay of all packets that have been transmitted on a given line during the last n seconds is computed. This average delay is then compared to the reported delay for that line. The reported delay is the value of delay that existed when an update was last generated - it corresponds to the value of delay that would be used in the SPF calculation. If the reported delay differs by a significant amount from the average delay that was just computed, an update is generated.

The measured delay values are quantized twice, once when the delay for each packet is measured, and again more coarsely when the average delay is computed.

The delay is considered to have changed "by a significant amount" whenever the change exceeds a certain threshold. The threshold is not a constant, but a decreasing function of time. The reason for this is the following. Whenever there is a large change in delay, it is desirable to report the new delay as soon as possible so that routing can react quickly. When the delay

changes by only a small amount, it is not important to report it quickly, since it is not likely to result in an important routing change. However, whenever a change in delay is long-lasting, it is important that it be reported eventually, even if it is small; otherwise additive effects can introduce large inaccuracies into routing. What is needed is a scheme which reports large changes quickly, small changes slowly, and moderate changes in a moderate amount of time. A threshold value which is initially high but which decreases to zero over a period of time has this effect. In the scheme that has been implemented, the threshold is originally set to an initial value. Whenever a change in delay is less than the threshold, the threshold is decreased by a decay value, except that it is never decreased below zero. Whenever a change in delay equals or exceeds the threshold, an update is generated, and the threshold is reset to its initial value. Since the threshold will eventually decay to zero, an update will always be sent after a certain amount of time, even if there is no change in delay. The rate at which the decay to zero occurs establishes the minimum frequency of routing updates.

It has been decided for efficiency reasons that an update from a given IMP shall contain the latest data on all the IMP's lines, rather than the data for only a single line.

One of our principal goals has been to compare the effects of different sets of parameters on the delay measuring and

reporting process in the real network environment. Such a comparison can only be meaningful if the input to the delay measuring process can be held constant as the parameters are changed. Otherwise, differences may be attributed to differences in the input as well as the parameters. However, the input to the delay measuring process is something which cannot be held constant, viz., the actual delays of packets in the network. Since there is no way for us to contol the input, there are only two means of obtaining meaningful comparative results:

a. Collect all the input in raw form, and simulate the effects of different parameters.

b. Maintain parallel data structures in the IMPs, so that different sets of parameters can be applied in parallel to the same input.

While the former procedure might seem to have the advantage of providing greater flexibility, it has the disadvantage of being only a simulation. One can never be as confident of results produced by simulation as of results produced by actual network measurement. Furthermore, there does not seem to be any realistic way of collecting on our TENEX PDP-10 the voluminous amount of data required (i.e., the individual delay of each packet which traverses a particular line in the network). TENEX does not seem to be able to accept data from the network at a

high enough rate to enable us to gather satisfactory amounts of data. Therefore, we have opted for the procedure of maintaining parallel data structures. These enable us to test the effects of different averaging intervals, different quantization units, and different threshold parameters on the very same data. One of the two parallel structures will be removed when the delay measurement routines become operational.

Similarly, we have been sending updates to a single collection point, rather than to all IMPs. When the routines become operational, the collection mechanism shall be removed, and replaced with the updating mechanism described elsewhere in this report.

The following paragraphs present a precise design specification of the delay measurement routines.

2.1.1 Parallel Data Structures and Parameters

In order to be able to run comparative experiments, we maintain parallel data structures and parallel sets of parameters. We refer to the parallel structures as the A-structure and the B-structure. In general, each invocation of a procedure will deal either with the A-structure or the B-structure. There are also parallel sets of parameters.

2.1.2 Stamping a Packet with its Arrival Time

The arrival time is stamped in each packet. For packets not originating locally, the arrival time is defined as the time the packet enters the IMP. For packets originating locally, the arrival time is defined as the time the packet is queued for routing.

2.1.3 Computing and Storing a Packet's Delay (Sampling)

In order to store the per-packet delays on each line for later averaging, three words per line are needed: a count of the packets transmitted on the line, the sum of the delays of all the transmitted packets, and a count of the number of times the sum overflowed. Actually, in order to maintain the A-structure and B-structure in parallel, six words are needed for each line -- three for the A-structure and three for the B-structure.

Whenever a packet is transmitted, the packet is stamped with the time at which transmission begins. Whenever a packet is retransmitted, this "sent time" is overwritten with the new sent time. When an acknowledgment is received for the packet (or when the packet is discarded due to 32 retransmissions with no acknowledgment), the delay is calculated as follows: subtract the arrival time from the sent time and add in the propagation delay (a constant for each line, depending only on line speed) and the transmission delay (a tabled function of line speed and

packet length). The delay will now be in units of 100 microseconds. Then the delay must be shifted right. The number of bits to right-shift the delay is specified by a single parameter which applies both to the A-structure and the B-structure.

After computing and shifting the delay, the following steps are carried out, first with the A-structure, then with the B-structure:

a) The count of transmissions is incremented.
b) The delay is added to the sum of delays for that line.
c) If this causes an overflow, the overflow counter is incremented.

Note that although parallel data structures are maintained while sampling, they are treated exactly the same.

2.1.4 Taking the Average (Smoothing)

The smoothing routine will be called periodically. The frequency with which it is called depends on the setting of two parameters: the A-smoothing-frequency and the B-smoothing-frequency. Suppose that the former is set to 10 seconds and the latter to 100 seconds. Then the smoother is invoked every 10 seconds, and operates only on the A-structure during each such invocation. In addition, the smoother is

invoked every 100 seconds to operate on the B-structure. The smoother operates on all lines each time it is called.

The average is computed by dividing the delay sum by the transmission count. However, before an average can be computed, the delay sum must be adjusted for overflows. If the overflow counter contains an n-bit number, the sum is right-shifted by n bits, and then the overflow counter is ORed into the n leftmost bits.

It is not necessary for the average to have as much precision as the individual packet delays. As described in section 2.1.3, the individual packet delays are already shifted by some amount before being added to the sum. There is another parameter to specify how much additional right-shifting should be done in computing the average. If there were no danger of overflows, this additional right-shifting could be done on the sum before the average is computed. However, after adjusting the sum of delays for overflow, the sum may be already right-shifted by some amount (i.e., by the amount equal to the number of bits in the overflow count). Let n be the number of bits which the average should be right-shifted (over and above the amount which the delays were shifted before being summed). Then there are three cases to consider:

a) The count of overflows is exactly n bits long. Then the additional right-shifting was done in adjusting for overflows, and no further shifting is required. The average is computed immediately.

b) The count of overflows is less than n bits long. Then additional right-shifting is required before the average can be computed.

c) The count of overflows is more than n bits long. Too much shifting has been done. Compensatory left-shifting is done after the average is computed.

To compute the average, the sum of delays is divided by the transmission count. Since the IMP has no divide instruction, a version of the following division algorithm has been implemented (note that all multiplications and divisions which appear in the specification of the algorithm can be done by shifting):

```
LEFT = 0;
POSITION = 15;
QUOTIENT = 0;
RIGHT = NUMERATOR;

#ALL DIVISIONS BELOW ARE INTEGER DIVISIONS

WHILE ((LEFT > 0) | (RIGHT > 0))
  {LEFT = 2*LEFT + RIGHT/2**POSITION; #TAKE MOST SIGNIFICANT
   IF (RIGHT/2**POSITION ==1)         #BIT FROM RIGHT AND MAKE
      RIGHT = RIGHT - 2**POSITION;    #IT LEAST SIGNIFICANT
   IF (LEFT >= DENOMINATOR)           #BIT ON LEFT
      {LEFT = LEFT - DENOMINATOR;
       QUOTIENT = QUOTIENT + 2**POSITION;
      }
   POSITION = POSITION - 1;
  }
```

This algorithm is just the one we usually use for long division, except that it is restricted to binary integers. It scans the numerator from the left. When it encounters an initial segment of leftmost digits (bits) which, taken as a number in its own right, is larger than the denominator, a 1 is placed in the appropriate bit position of the quotient. (Note that 1 is the only non-zero binary digit.) Then the denominator is subtracted from the most significant part of the numerator, and the process is repeated until all digits of the numerator have been scanned.

It is obvious that if the numerator is a 16-bit word, the WHILE loop cannot be repeated more than 16 times. As implemented in the IMPs, the algorithm can never run for more than half a millisecond for each line, even in the very worst case.

In addition to taking averages, the smoother zeroes out all packet transmission counters, sum of delays, and overflow counters that it uses, thereby initializing them for the next sampling interval.

After the average delay <u>for each line</u> is computed, the sampler calls the comparer, providing it with these delays as input. The smoother also tells the comparer whether the average has been obtained from the A-structures or the B-structures.

Note: 1) If a line is down, the smoother reports "infinite" (i.e., $2^{16}-1$) delay. If the line is up, the delay is not allowed to exceed $2^{16}-2$.

2) If the transmission counter for any line is zero, the smoother reports a delay for that line equal to the propagation delay on the line.

2.1.5 Comparing

The comparer is called directly by the smoother, which tells it whether to work with the A-structures or the B-structures. The other input to the comparer is a list containing the current average delay for each line.

The comparer works by comparing the current delay on each line with its "base delay", where base delay is the delay that was last reported to the network, i.e., the delay that all other IMPs think the line has. The comparer must therefore maintain a table of the base delay for each line; rather, it must maintain two such tables -- one for use with the A-structure and one for use with the B-structure. On any given call, however, it uses only one of the tables.

The comparer decides whether an update message is called for by determining whether the absolute value of the difference between the current delay for a line and the base delay for that line exceeds a threshold. There is a separate threshold for the A-structure and the B-structure. Within a structure, however, the same threshold applies to all lines. The value of the threshold for a given structure depends on two parameters, the

threshold-initial-value and the threshold-decay-value. The threshold is initially set to the initial-value. Whenever the difference between the current delay and the base delay for any line exceeds the threshold, the threshold is reset to its initial value. Whenever the difference for every line is less than the threshold, the threshold is decreased by the decay value.

If the difference between the base delay and the current delay for any line exceeds the threshold, the base delay is set to the current delay, and an update message is sent.

2.1.6 Line up/line down

Whenever a line goes down, all counters and sums associated with that line are zeroed out. The base delay is set to the maximum value, and an update message immediately generated.

Whenever a line comes up, the base delay for that line is set to the propagation delay for the line, and an update message immediately generated.

2.1.7 Utilization

The present routing algorithm currently measures the utilization on each line by keeping a count of the number of milliseconds each modem is busy during each 640 millisecond period.

Before the counter is zeroed, it is added to a counter which is zeroed only when the comparing routine is called. The values of these counters are reported in routing update messages. However, the counters are zeroed each time the comparing routine is called, whether or not a message is sent. This feature will not be part of the new routing algorithm when it becomes operational.

2.1.8 List of Parameters which can be Modified in Experiments

1) The propagation delay for each line, in 100 usec unit (used by the sampling routines).

2) The table of transmission delays, in 100 usec units, indexed by line speed and packet size (used by the sampling routine).

3. The number of bits to right-shift each packet's delay before adding it to the sum of delays (used by the sampling routines).

4) The number of additional bits to right-shift the average delay (used by the smoothing routines).

5) The frequency with which to call the averager. This will actually be two "parallel" parameters:

A - frequency: the frequency with which the averager is applied to the A-structure

B - frequency: the frequency with which the averager is applied to the B-structure

6) The comparison threshold initial value (used by the comparer). Again, this will be a pair of A and B parameters.

7) The comparison threshold decay value (used by the comparer). Again, a pair of A and B parameters are required.

2.2 Delay Measurement Experiments

In the past few months, we have been gathering data from the network using the delay measurement routines. We have used these measurements to test various settings of the parameters, as well as to gather information on the relationship between delay and utilization in an actual network. Our observations are reported below.

2.2.1 Delay vs. Utilization

For experimental purposes, we included a means of measuring line utilization in the delay measurement modules. Utilization

was measured by actually computing and summing the transmission times of all packets sent on a line during a given interval. The transmission times were computed in 100 microsecond units, and were reported via update packets in 1.6 microsecond units. This enabled us to compute the fraction of time the line was busy during the interval. The same update packets that reported the utilzation also reported the average delay over the interval. Figures 2-1 through 2-5 show delay plotted against utilization for 5 data samples. The interval of measurement is 9.6 seconds. Individual packet delays were quantized to 800 microseconds, and the average delay was quantized to 6.4 ms.

Figures 2-1 and 2-2 show delay vs. utilization under actual network conditions (no artificial test traffic). Both the delay and the utilization are low, as would be expected. The amount of scatter is somewhat surprising, as is the fact that delay shows little tendency to increase as utilization increases. Figures 2-3, 2-4, and 2-5 show data samples in which an attempt was made to saturate the line with artificially generated traffic for part of the sampling time. In Figure 2-3, we see that the delay does have a tendency to increase as utilization exceeds 0.8. However, the data diverges from theoretical predictions in two respects. First, there are a large number of intervals which have a relatively low delay, even at high utilization. Second, the delay does not seem to be increasing to infinity as utilizaton

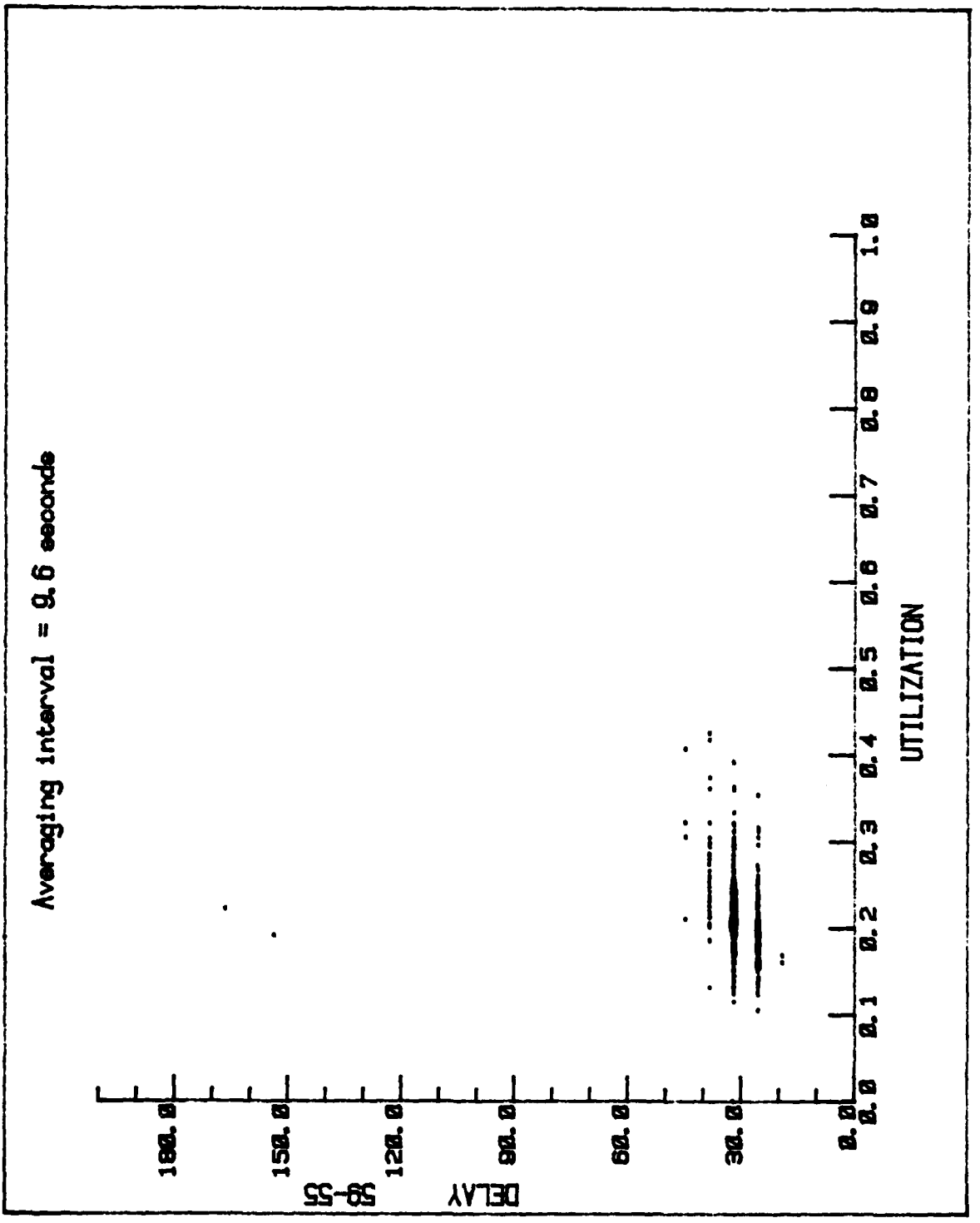

Averaging interval = 9.6 seconds
DELAY
UTILIZATION
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
30.0
60.0
90.0
120.0
150.0
180.0

Figure 2-1

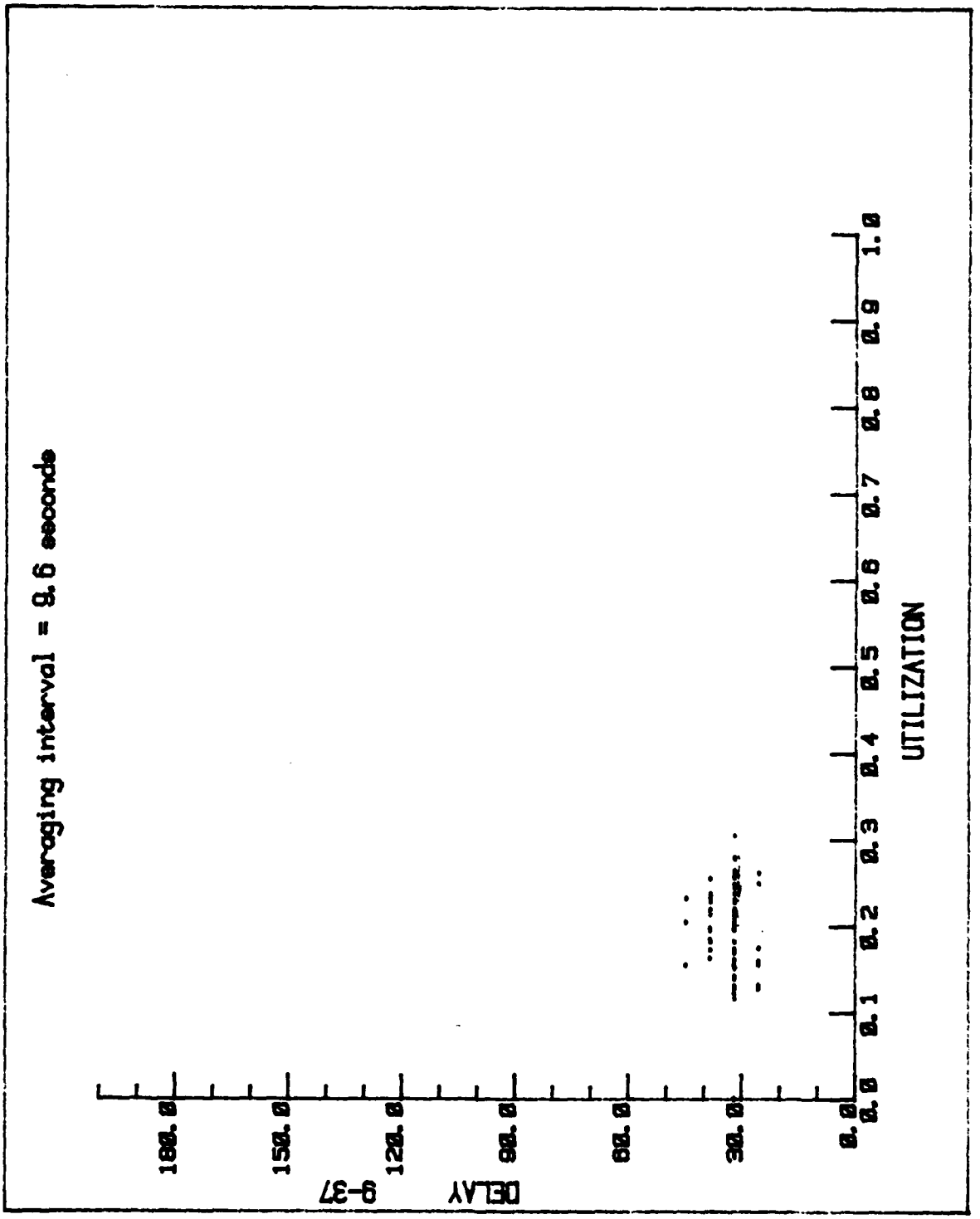

Averaging interval = 9.6 seconds
DELAY
180.0
150.0
120.0
90.0
60.0
30.0
0.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
UTILIZATION

Figure 2-2

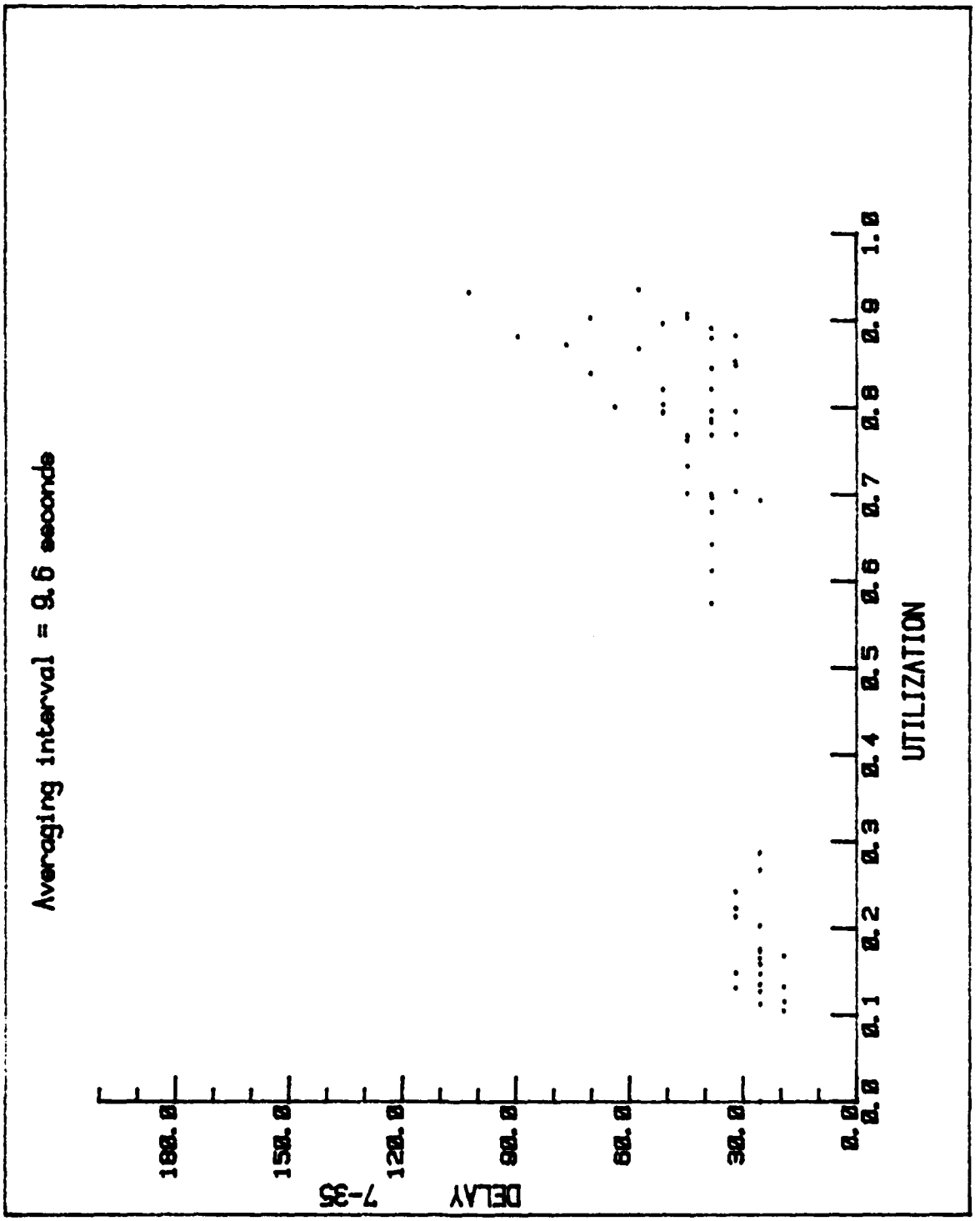

Averaging interval = 9.6 seconds
DELAY
180.0
150.0
120.0
90.0
60.0
30.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
UTILIZATION
7-35

Figure 2-3

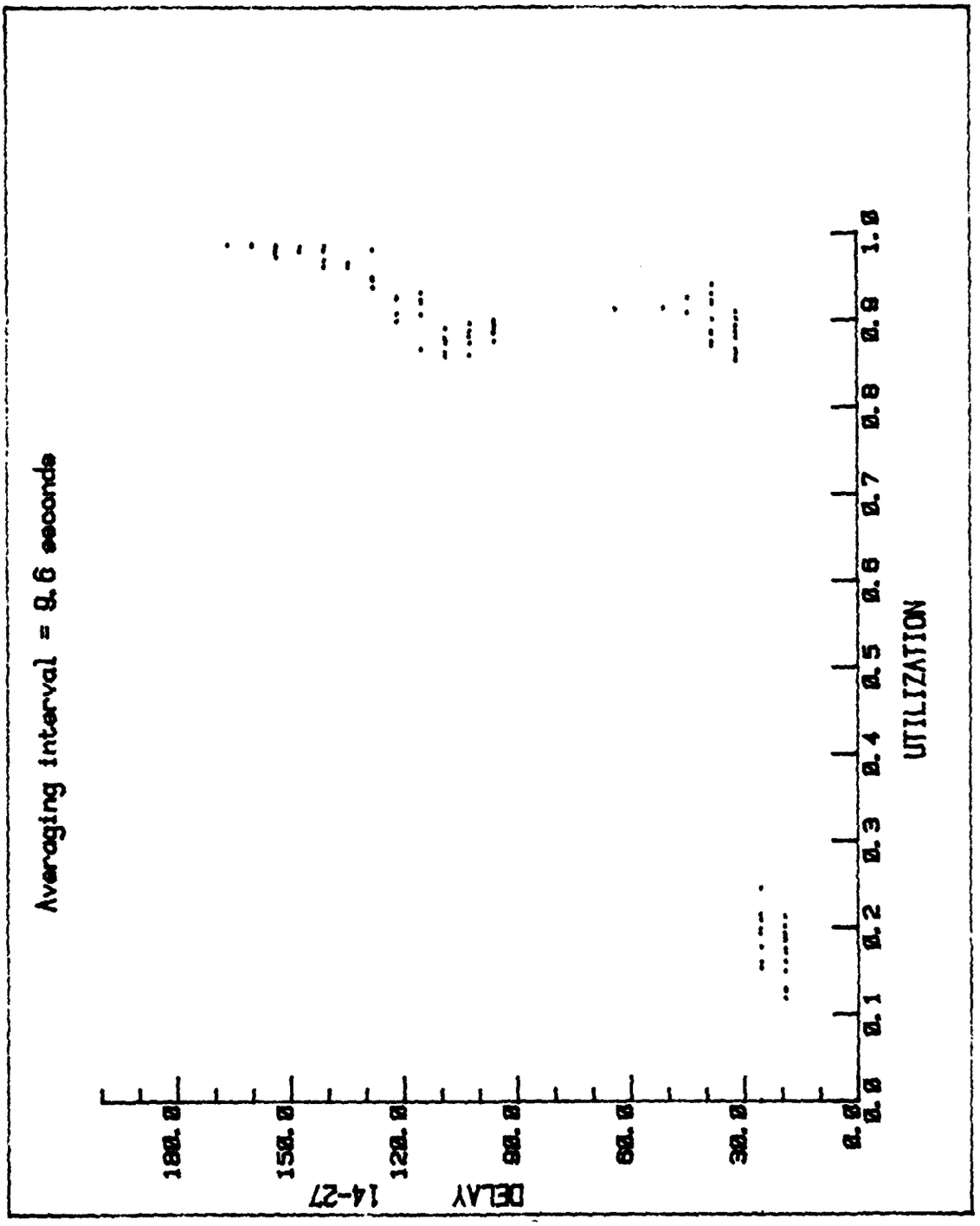

Averaging interval = 9.6 seconds
DELAY
180.0
150.0
120.0
90.0
60.0
30.0
0.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
UTILIZATION
14-27

Figure 2-4

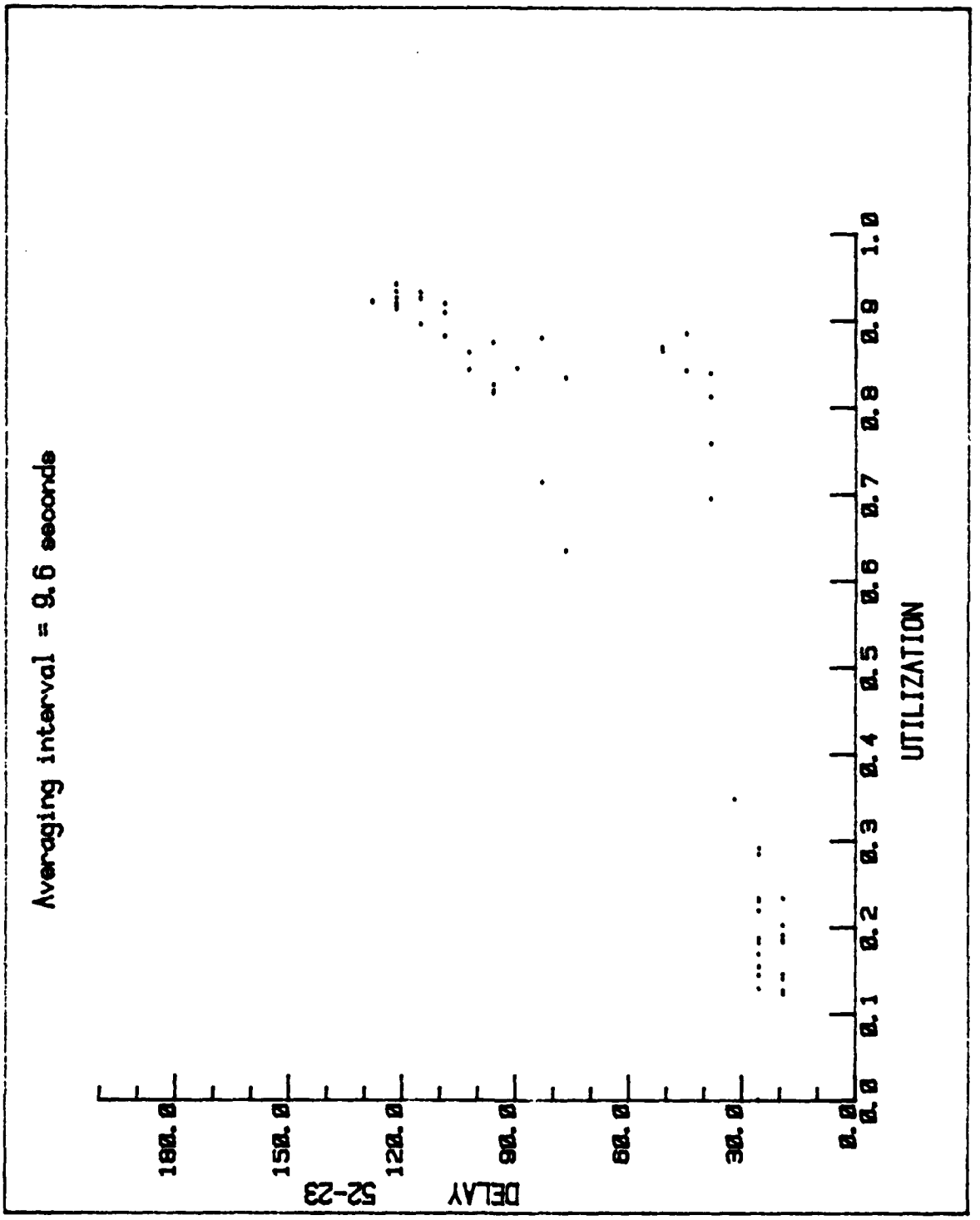

Averaging interval = 9.6 seconds
DELAY
52-23
180.0
150.0
120.0
90.0
60.0
30.0
0.0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
UTILIZATION

Figure 2-5

approaches 1.0. Figures 2-4 and 2-5 show data which correspond more closely to theory, in that there is a much steeper rise in delay as utilization exceeds 0.8. In both figures, however, there is a cluster of points showing relatively low delay at high utilization.

2.2.2 Quantization Units

As described above, delay data is quantized twice. There is quantization in measuring the delay of each individual packet, and further quantization in computing the average delay. All measurements in the IMP are performed with the 100 microsecond clock, and the only practical means of quantization is right-shifting. Therefore, the unit of quantization is always a power of two times 100 microseconds.

The unit of quantization used when measuring the delay of each individual packet is not too crucial to the performance of the routing algorithm. For the sake of accuracy, we want it to be small reltive to the actual delay experienced by a packet. Yet we do not want it to be so small that we risk overflowing the field which is used to sum the delays over the averaging interval. (The risk of overflow is reduced, however, by the fact that the summation is done in double precision arithmetic.) We have chosen to quantize the individual packet delays to 800 microseconds.

The unit of quantization which is applied to the average delay is more important to the performance of the routing algorithm, since it is the quantized average delay which is reported to the network. If the quantization is too coarse, the reported average delay is insufficiently accurate. Since the reported delays are summed up by the SPF algorithm, errors can accumulate, resulting in poor routing. These considerations argue for making the unit small. On the other hand, if the quantization is too fine, then we need a large number of bits to store the value of delay. We want to be able to store the delay value in a small number of bits, in order to conserve memory, but we also want the reported values of delay to be able to take on a wide range of values. These considerations argue for making the unit larger.

A further argument for making the unit small has to do with the effect of statistical noise on the delay measurement process. Our measurements show that many changes in average delay are changes of only one unit, no matter what that unit is. Furthermore, the size of the unit does not seem to affect the number of these changes. Many of the one unit changes are undoubtedly due to noise in the measurement process, and are not real changes. As the unit of quantization gets large, noise has a correspondingly larger effect.

We have settled tentatively on a unit of quantization of 6.4 ms. If we store the value of delay in 8 bits, as currently planned, this gives delay a dynamic range of 6.4 ms to 1.6 seconds, which seems to be a large enough range to enable the routing algorithm to distinguish between congestion and a heavy but uncongested load. The next largest possible unit would be 12.8 ms. Since it is possible for a packet to experience less delay than this, this figure is probably too large to ensure sufficient accuracy. The next smaller possibility after 6.4 ms is 3.2 ms. Using 3.2 ms. rather than 6.4 would halve the dynamic range of the delay values while resulting in only a small increase in accuracy. Our collections of data from the network show no important distinction between these two units - that is, 6.4 seems "small enough" as well as "large enough". We may, of course, wish to revise this opinion after further testing.

2.2.3 Averaging Interval

In our First Semi-Annual Technical Report, we discussed the extreme variability of individual packet delays, and the consequent need for averaging. In order to obtain a meaningful measurement of average delay, it is necessary to have a relatively long averaging period. If the period is too short, stochastic noise in the data is likely to make the measurement inaccurate. On the other hand, if the averaging interval is too long, it will take a corresondingly long time to detect changes

in the averge delay, and the responsiveness of the routing algorithm is slowed.

We have collected data from the network to compare the effect of different averaging intervals. Figure 2-6 compares an interval of 96 seconds (Figure 2-6a) with an interval of 9.6 seconds (Figure 2-6b). The data represented in these figures is due only to user traffic - no artificially generated test traffic was used. The reported delays never disagree by more than a single unit of quantization. Undoubtedly the data based on the 96 second interval is more accurate, with the discrepencies being due to noise. The noise does not seem very significant, however, and even with the shorter interval, reported changes in delay occur only once every 163 seconds, on the average.

Figure 2-7 compares the same two averaging intervals, but the data was collected from a line which we attempted to saturate with test traffic. We used the IMPs' internal message generators to send maximum length multi-packet messages as frequently as possible. Message generators were on from approximately minutes 20-44, and from approximately minutes 51-55. Comparing Figure 2-7a with Figure 2-7b shows several differences between the 96 second interval and the 9.6 second interval. The larger interval gives much smoother results than the shorter. The larger interval causes a reported change in delay once every 145 seconds, on the average; the shorter, once every 57 seconds.

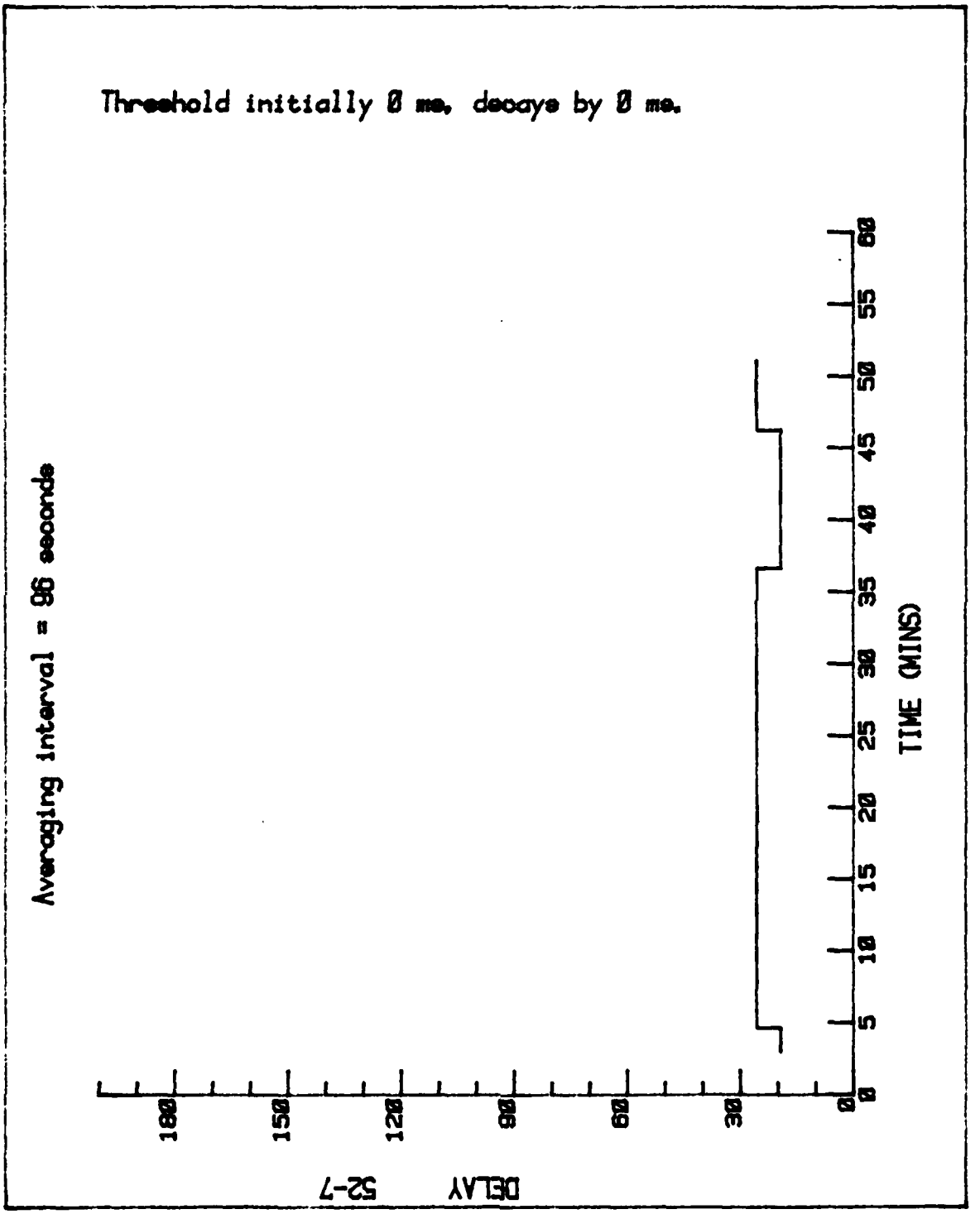

Threshold initially 0 ms, decays by 0 ms.
Averaging interval = 96 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY
S2-7
180
150
120
90
60
30
0

Figure 2-6a

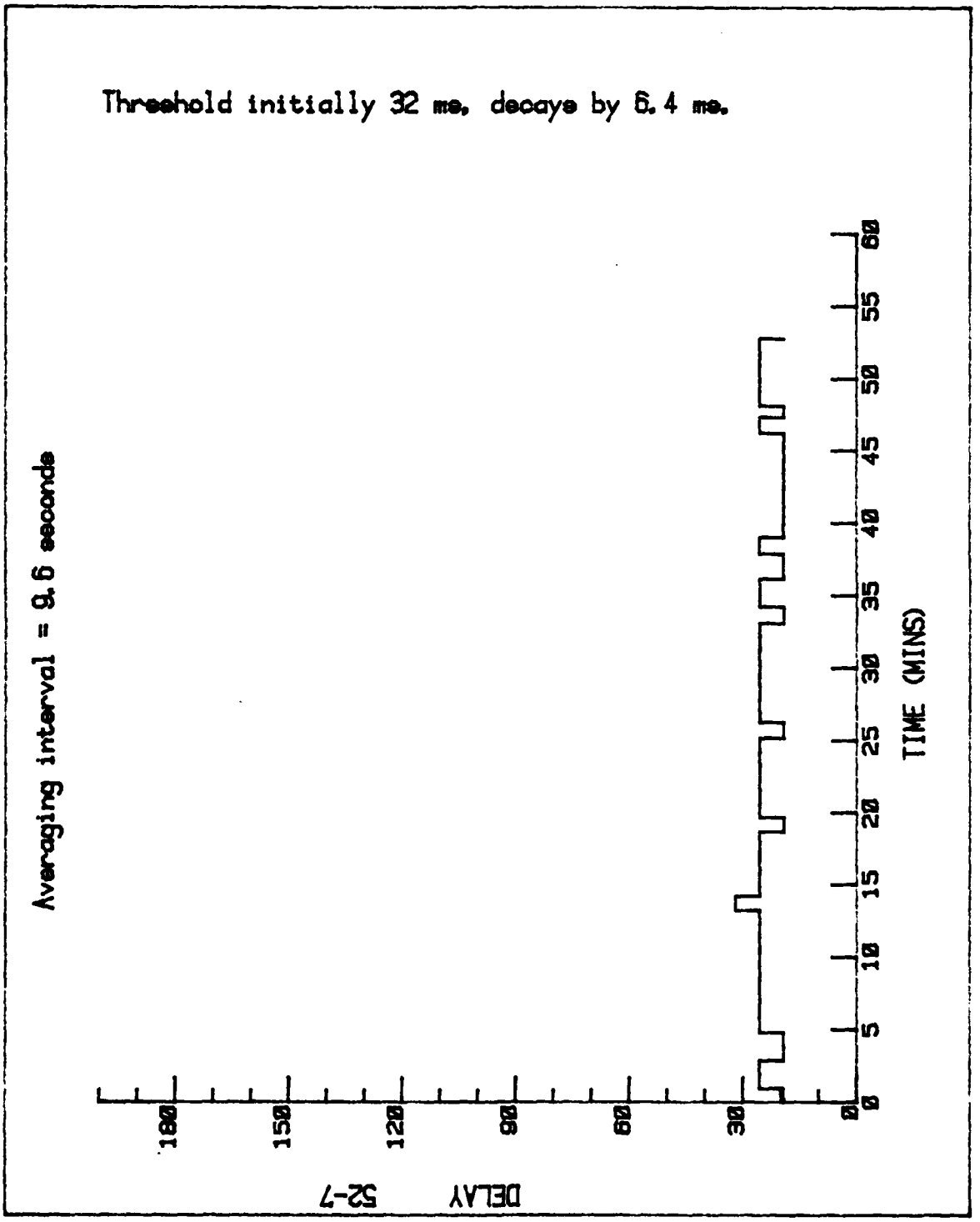

Threshold initially 32 ms. decays by 6.4 ms.
Averaging interval = 9.6 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY
S2-7
180
150
120
90
60
30
0

Figure 2-6b

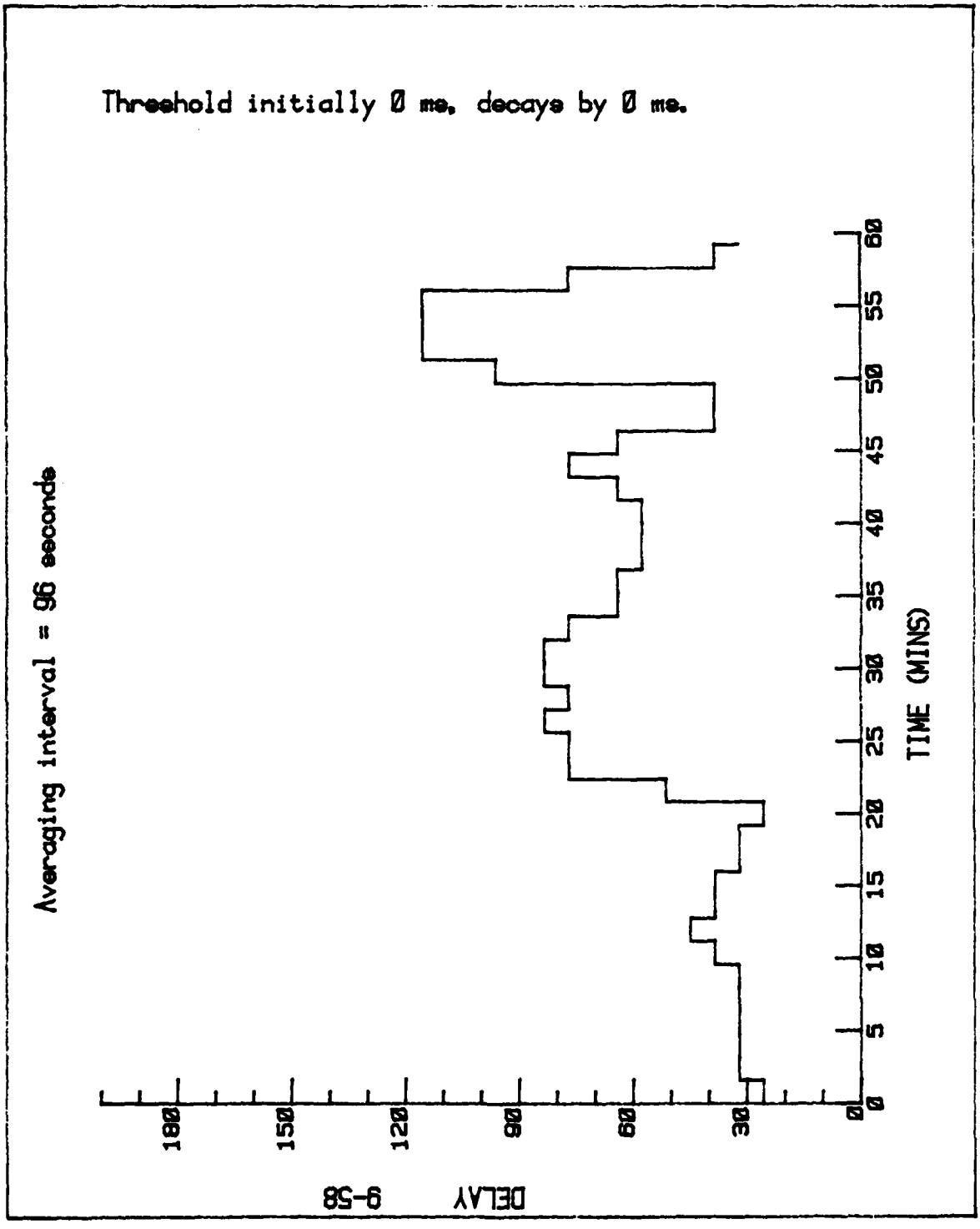

Threshold initially 0 ms, decays by 0 ms.
Averaging interval = 96 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY
9-58
180
150
120
90
60
30
0

Figure 2-7a

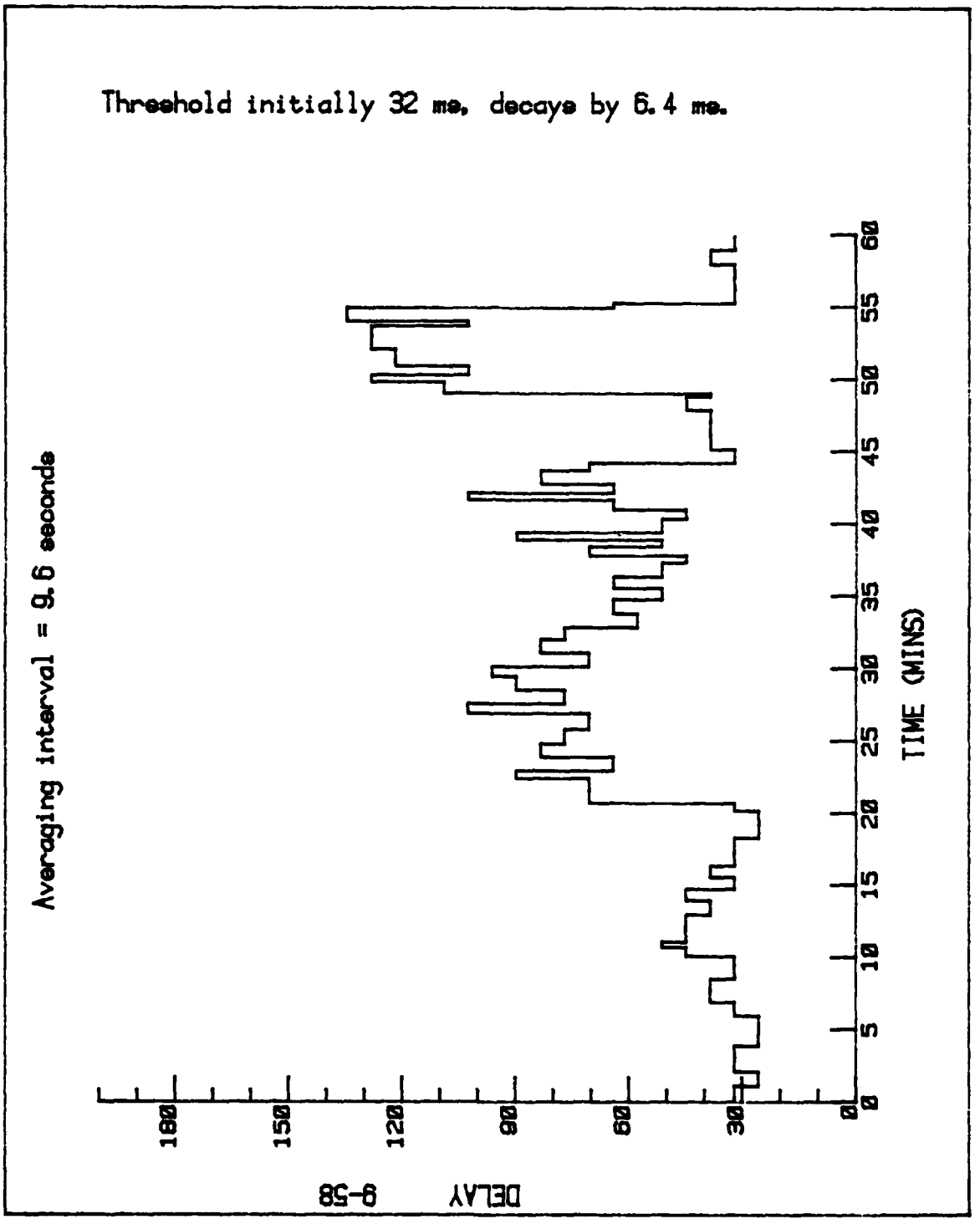

Threshold initially 32 ms, decays by 6.4 ms.
Averaging interval = 9.6 seconds
DELAY 9-58
180
150
120
90
60
30
0
0
5
10
15
20
25
30
35
40
45
50
55
60
TIME (MINS)

Figure 2-7b

However, although Figure 2-7a is smoother than 2-7b, the data presented on 2-7b seems adequately smooth, and does not cause an excessive number of updates to be generated. The two graphs track each other quite well. Furthermore, the shorter interval has a crucial advantage over the larger, in that it reacts much more quickly to large changes in delay. (This is quite clear if the graphs are superimposed.) This increased responsiveness is very important to the routing algorithm. Although the shorter averaging interval results in more "noise" in the measurement, it seems that this is simply the price one must pay for increased responsiveness. It is our judgement therefore that the 9.6 second interval is preferable to the 96 second interval.

We have also investigated whether an even shorter interval might be preferable. In Figure 2-8, we compared an interval of 2 seconds (Figure 2-8a) with an interval of 9.6 seconds (Figure 2-8b), with no test traffic. Figures 2-9a and 2-9b compare the same two averaging intervals, with test traffic running from minute 8 to minute 53. The two-second interval is unsatisfactory; there is too much variability, and too many meaningless changes are reported. The two-second interval yields a smoothed delay which bears too close a resemblance to the raw data which was reported on in the First Semi-Annual Report.

In Figures 2-10 and 2-11, a 5-second averaging interval is compared to a 10-second interval. The measurements shown in

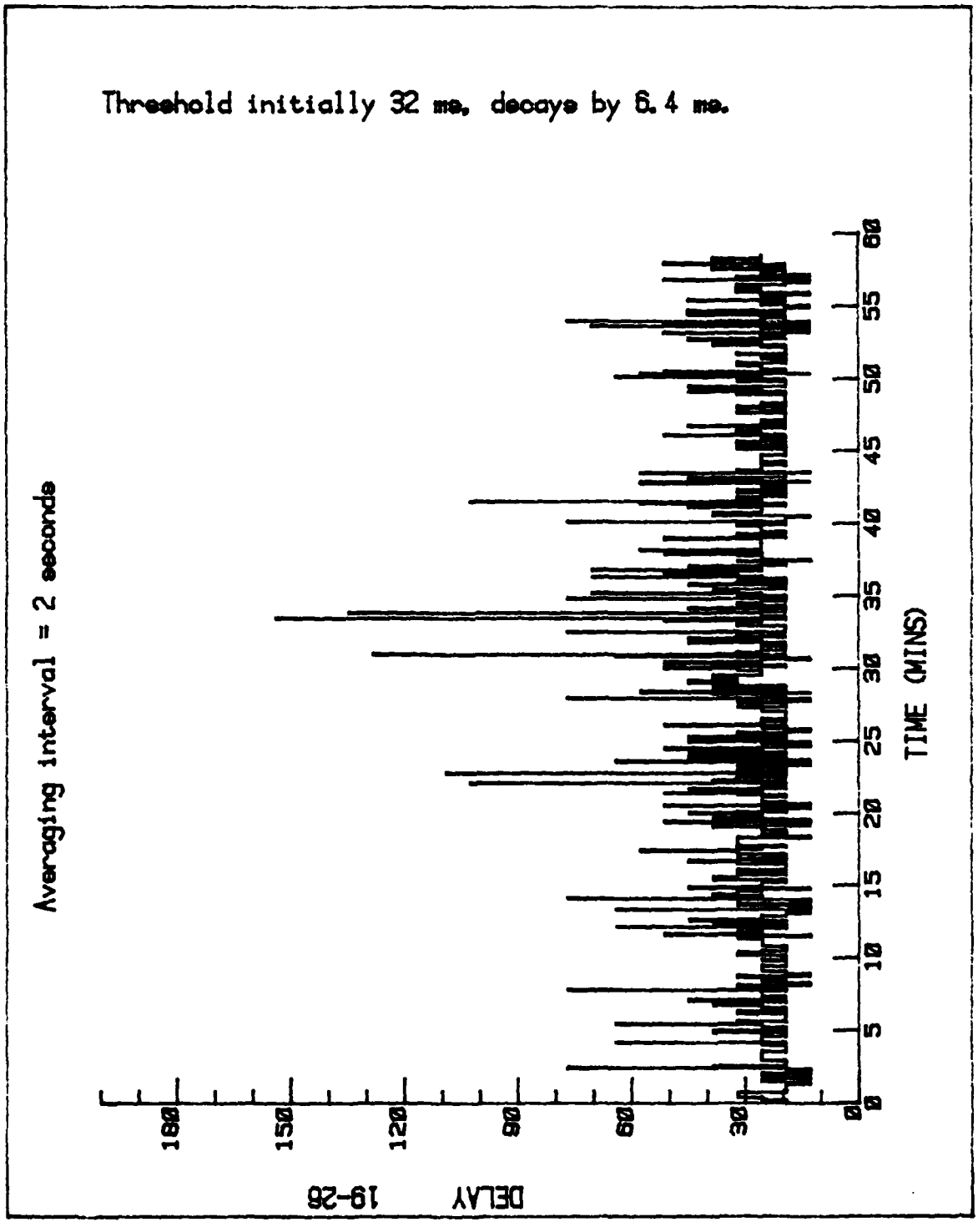

Threshold initially 32 ms, decays by 6.4 ms.
Averaging interval = 2 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY
19-28
180
150
120
90
60
30
0

Figure 2-8a

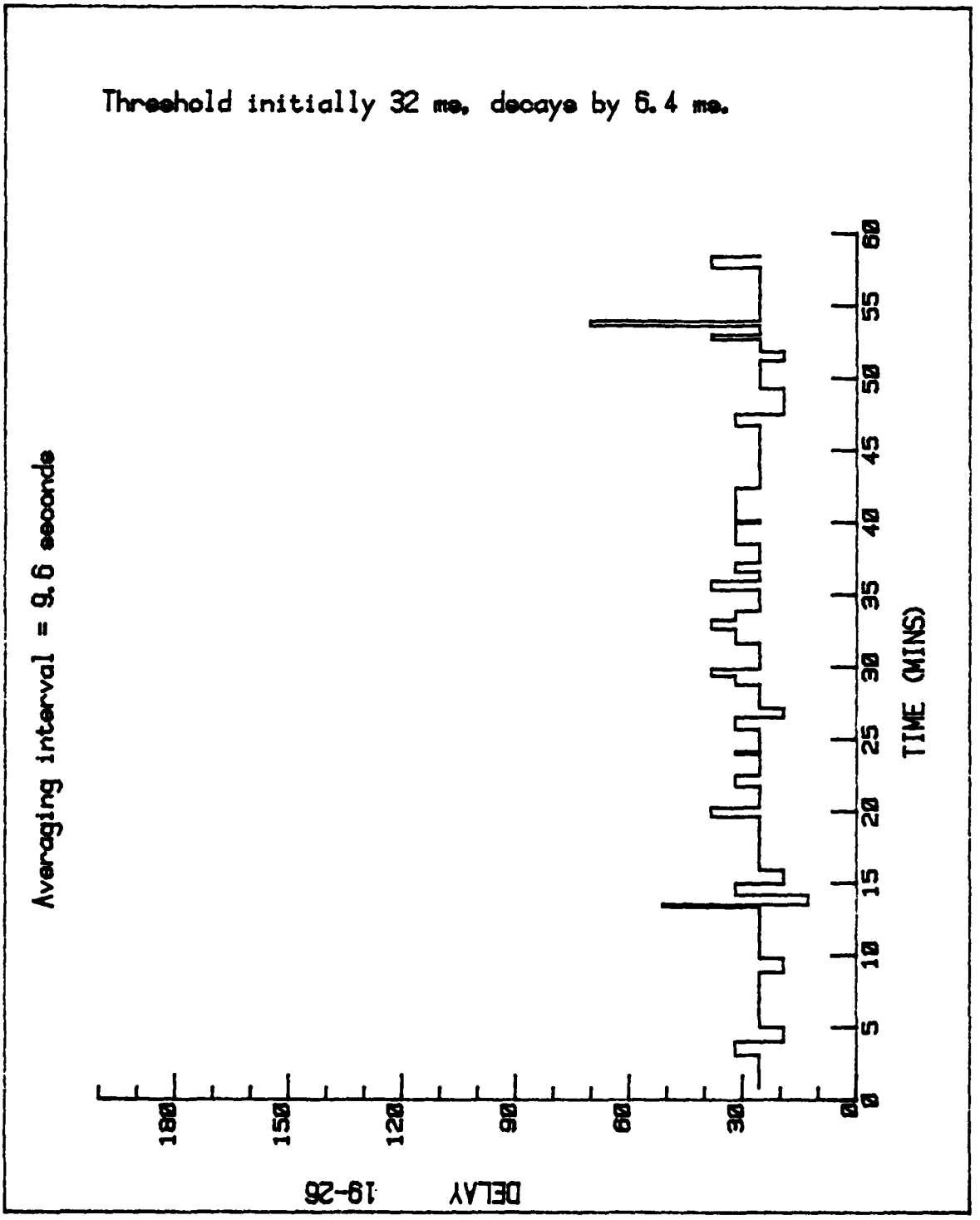

Threshold initially 32 ms, decays by 6.4 ms.
Averaging interval = 9.6 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY
0
30
60
90
120
150
180

Figure 2-8b

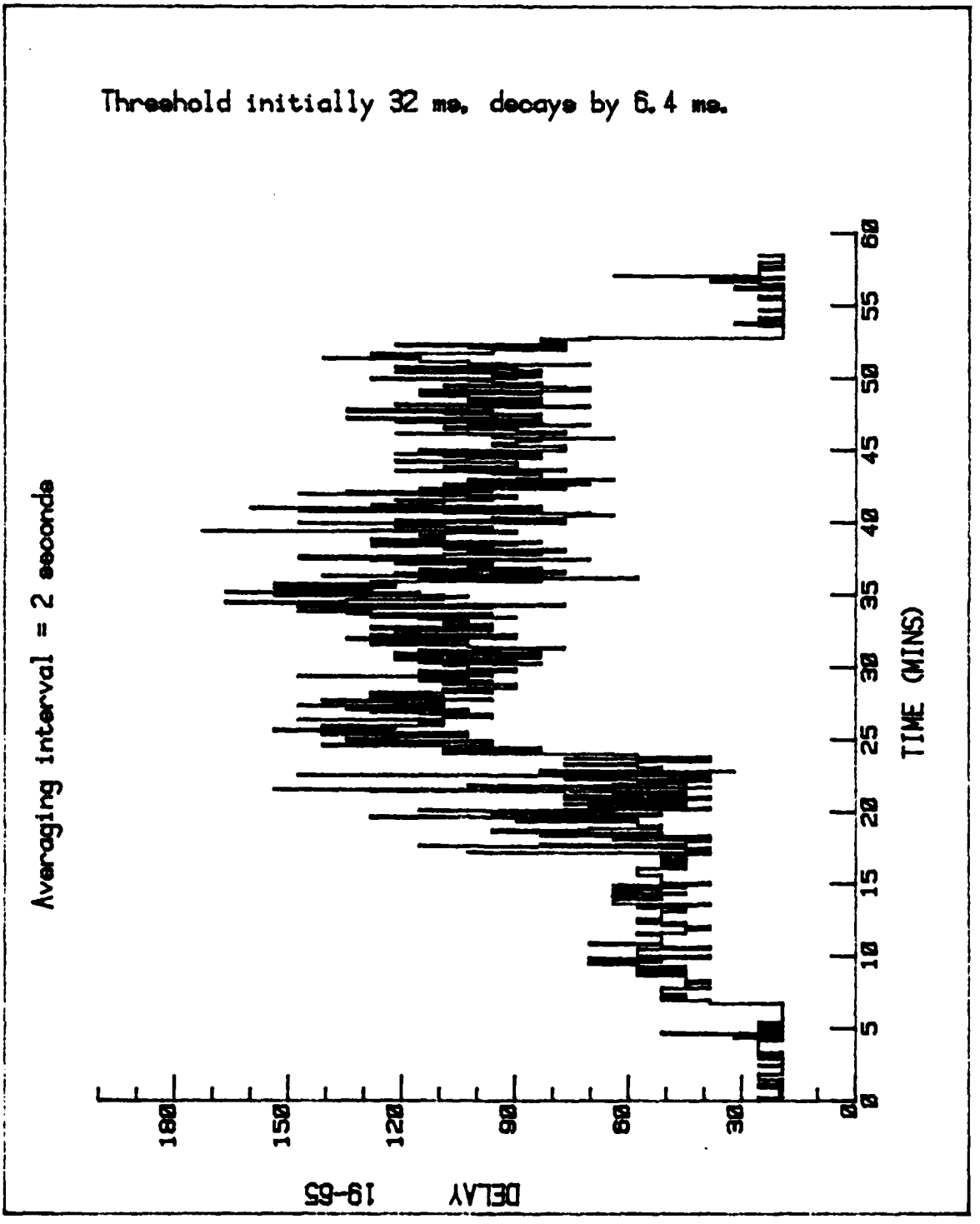

Threshold initially 32 ms. decays by 6.4 ms.
Averaging interval = 2 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY
19-65
180
150
120
90
60
30
0

Figure 2-9a

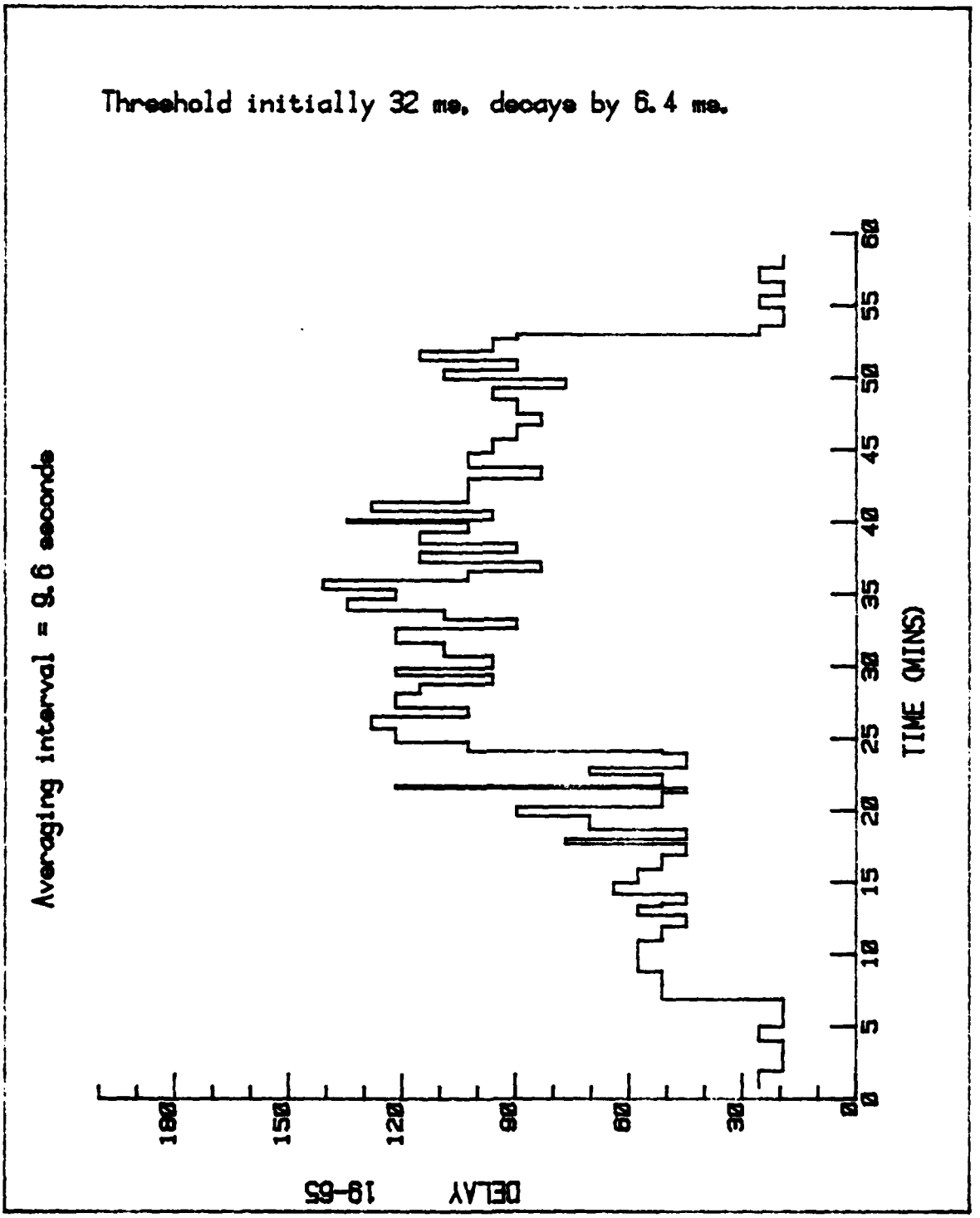

Threshold initially 32 ms, decays by 6.4 ms.
Averaging interval = 9.6 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY
19-65
180
150
120
90
60
30
0

Figure 2-9b

Figure 2-10 were taken under ordinary conditions, while those in Figure 2-11 were taken under conditions of artificially induced heavy load. In both cases, the larger interval yields a smoother result, while the shorter interval does not appear to have any countervailing advantage.

Since an averaging interval of approximately 10 seconds appears to be long enough to yield a meaningful measurement, while also short enough to detect changes quickly, we have decided to use an interval of 10 seconds in our initial implementation.

2.2.4 Threshold Parameters

There are two threshold parameters - the initial value of the threshold, and the value by which the threshold is decreased each time a measurement is made which does not differ significantly from the reported delay value. The values of the threshold parameters have two important effects. Since the two parameters together control the rate at which the threshold decays to zero, they determine the minimum routing frequency. Also, they determine the responsiveness of the routing algorithm to changes in delay.

Our measurements have shown that it is very rare for the delay to change, on the average, more often than the minimum updating frequency. For instance, if the initial threshold is 5

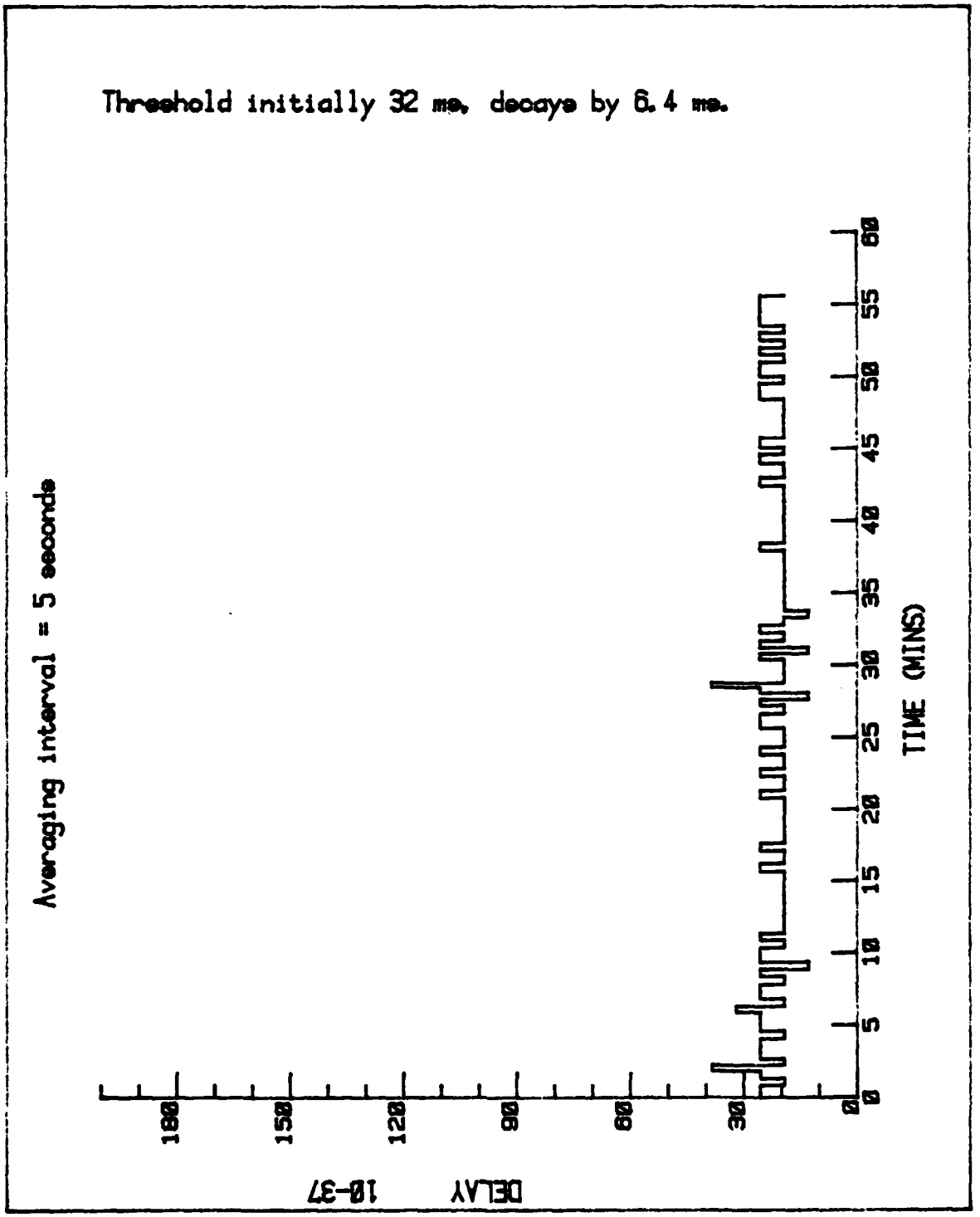

Threshold initially 32 ms. decays by 6.4 ms.
Averaging interval = 5 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY 10-37
180
150
120
90
60
30
0

Figure 2-10a

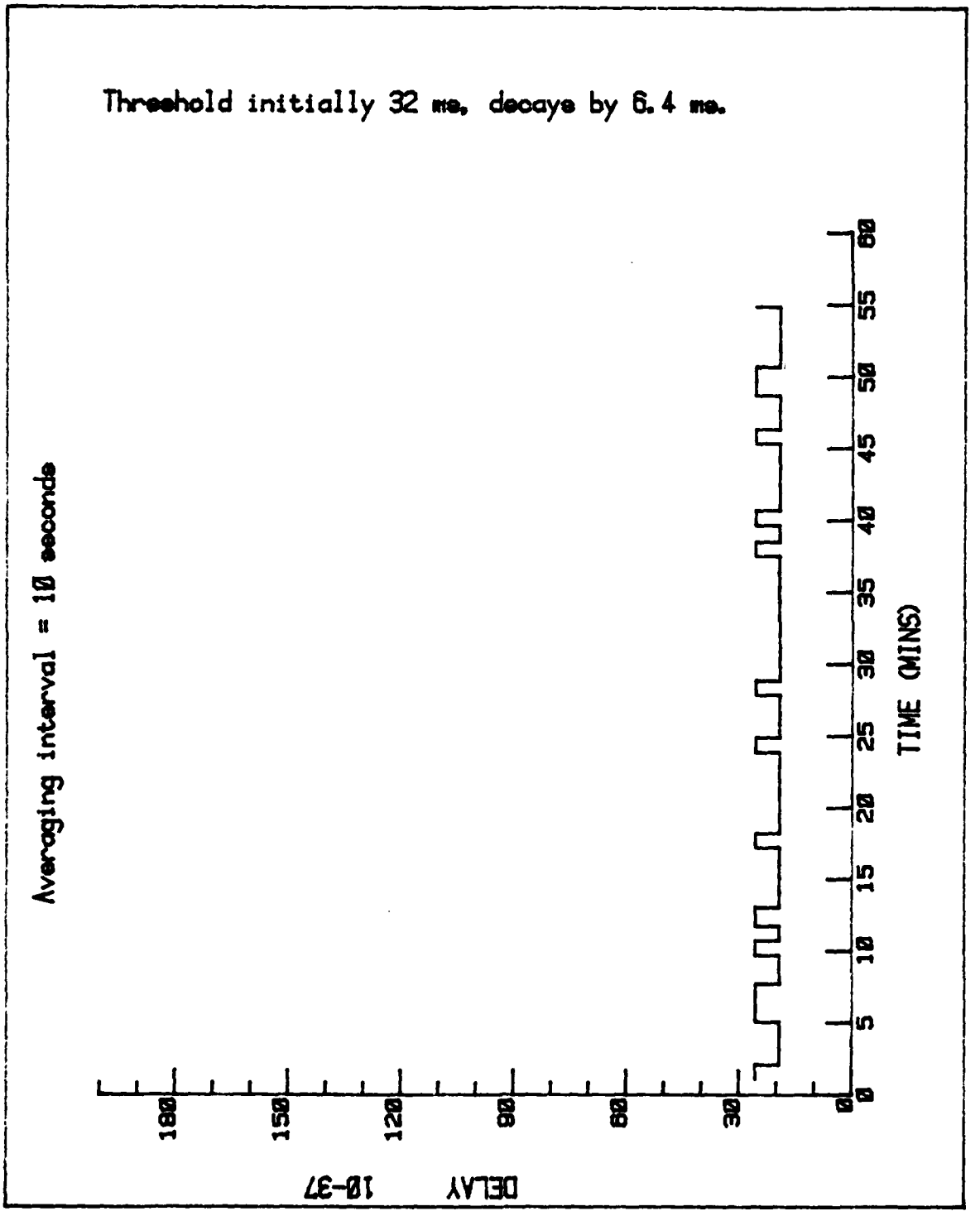

Threshold initially 32 ms, decays by 6.4 ms.
Averaging interval = 10 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY
10-37
180
150
120
90
60
30
0

Figure 2-10b

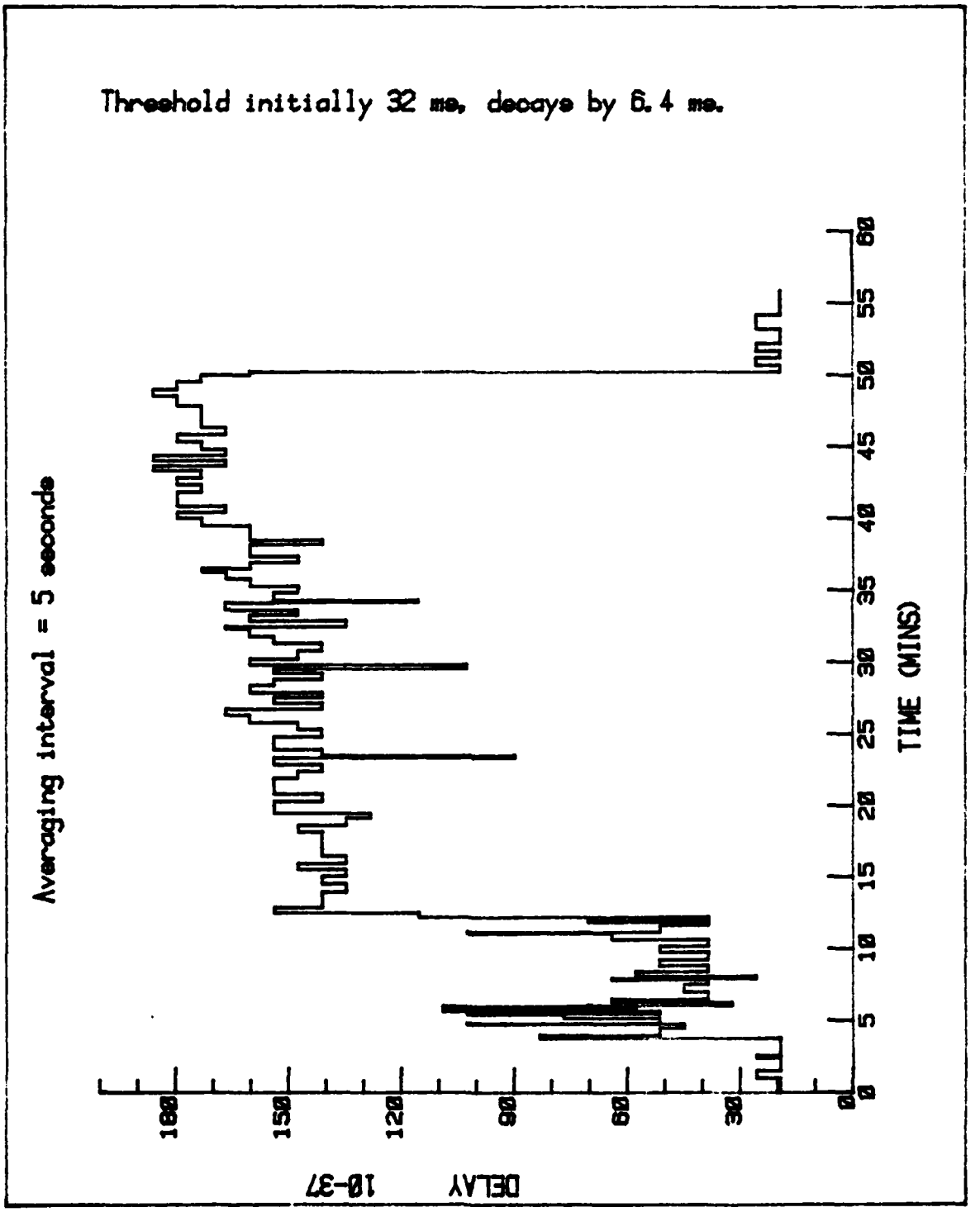

Threshold initially 32 ms, decays by 6.4 ms.
Averaging interval = 5 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY 10-3?
0
30
60
90
120
150
180

Figure 2-11a

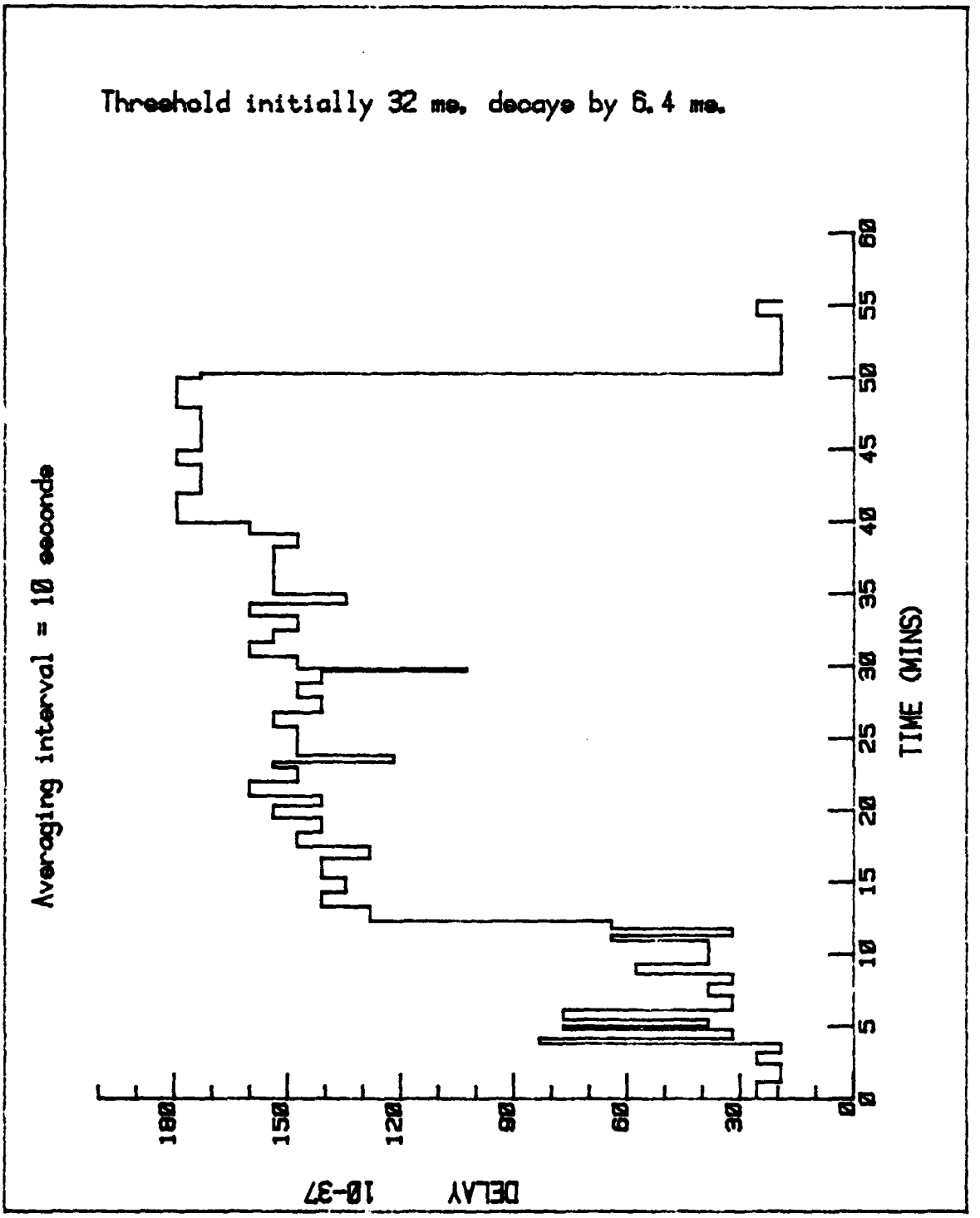

Threshold initially 32 ms. decays by 6.4 ms.
Averaging interval = 10 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY 10-37
0
30
60
90
120
150
180

Figure 2-11b

times greater than the threshold decay value, and the averaging interval is 10 seconds, an update (which may not, however, report a change) will be generated once every 60 seconds. Our measurements show that, no matter what the traffic load and no matter what the actual values of the threshold parameters are, the number of reported delay changes rarely exceeds one per minute, on the average, as long as the parameters are related as specified, and the initial value is reasonably large. This suggests that once the averaging interval is chosen, the ratio between the initial threshold and the threshold decay should be chosen so as to ensure the desired minimum updating frequency. The only additional choice to make is the actual value of the initial threshold. This should be large enough so that only major changes are detected, but small enough so that they are not missed.

Figures 2-12a, 2-13a, and 2-14a show data for which the initial threshold is 32 ms., and the decay is 6.4 ms. Figures 2-12b, 2-13b, and 2-14b show the same delay data respectively, but with the initial threshold set to 64 ms, and the decay to 12.8 ms. In the data shown in Figure 2-12, the choice of threshold value makes no appreciable difference at all. In Figure 2-12a, there is a reported change in delay once every 68 seconds; in Figure 2-12b, once every 69 seconds. In the data shown in Figure 2-13, the difference in threshold values is more

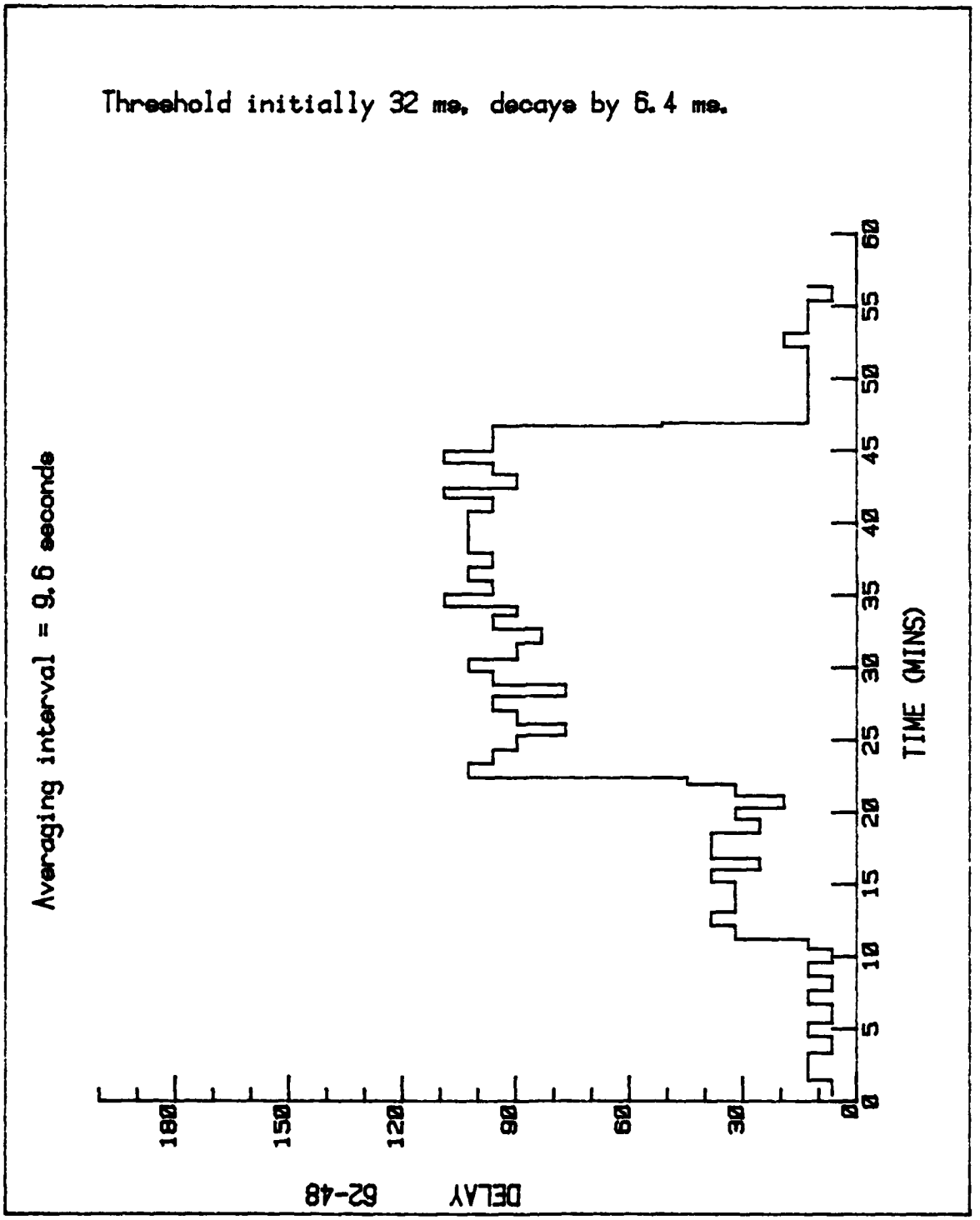

Threshold initially 32 ms, decays by 6.4 ms.
Averaging interval = 9.6 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY
62-48
0
30
60
90
120
150
180

Figure 2-12a

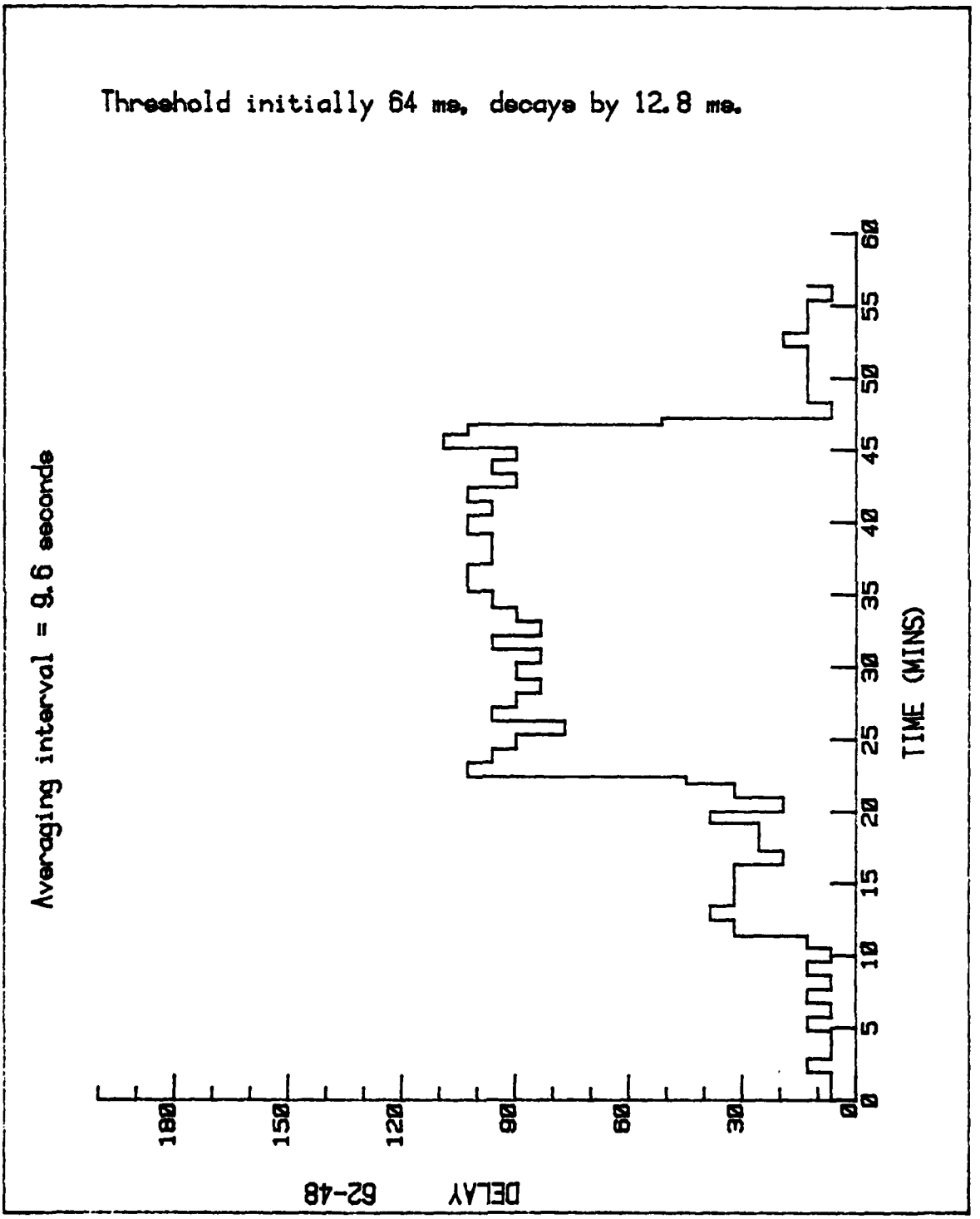

Threshold initially 64 ms, decays by 12.8 ms.
Averaging interval = 9.6 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY
62-48
180
150
120
90
60
30
0

Figure 2-12b

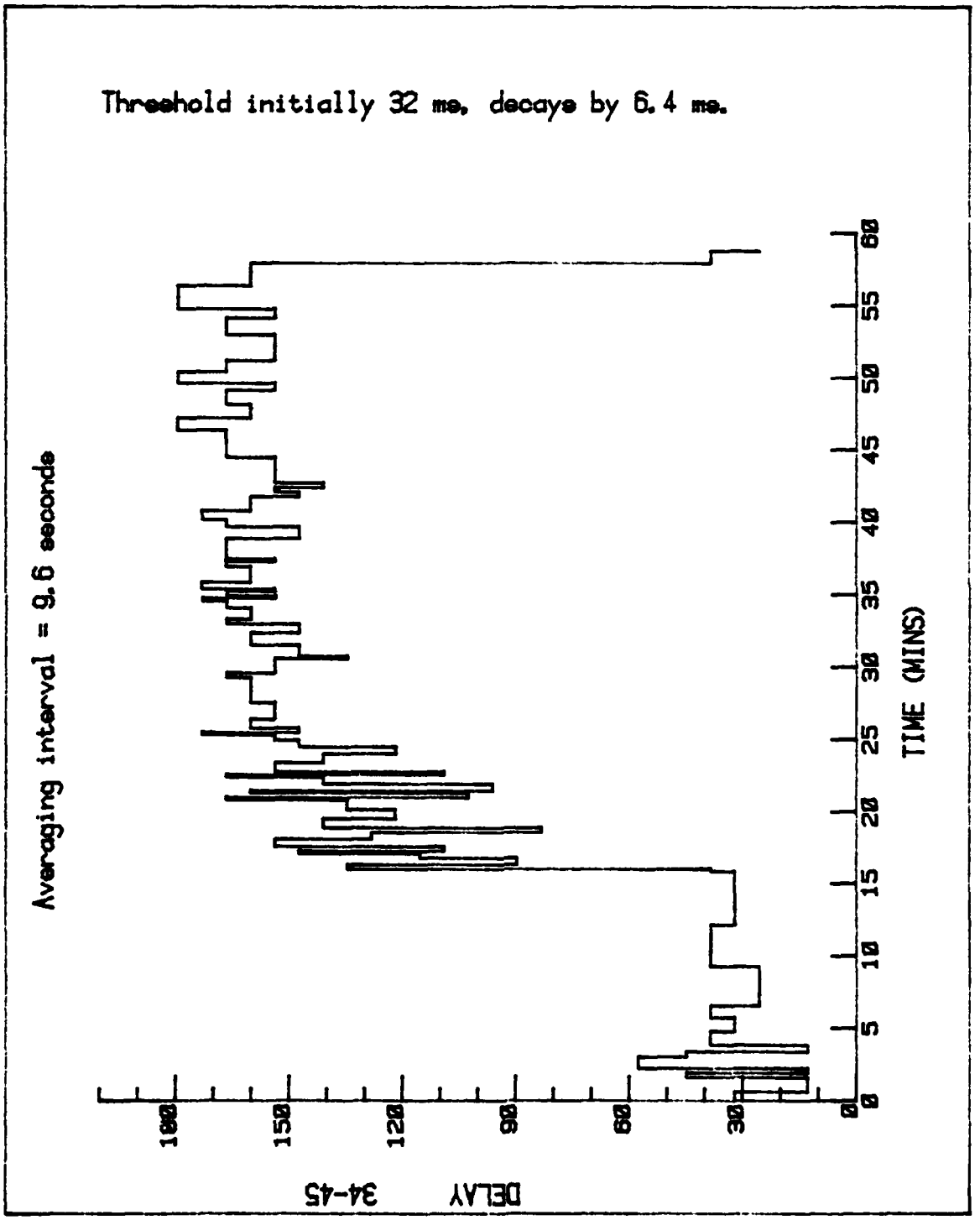

Threshold initially 32 ms, decays by 6.4 ms.
Averaging interval = 9.6 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY 34-45
180
150
120
90
60
30

Figure 2-13a

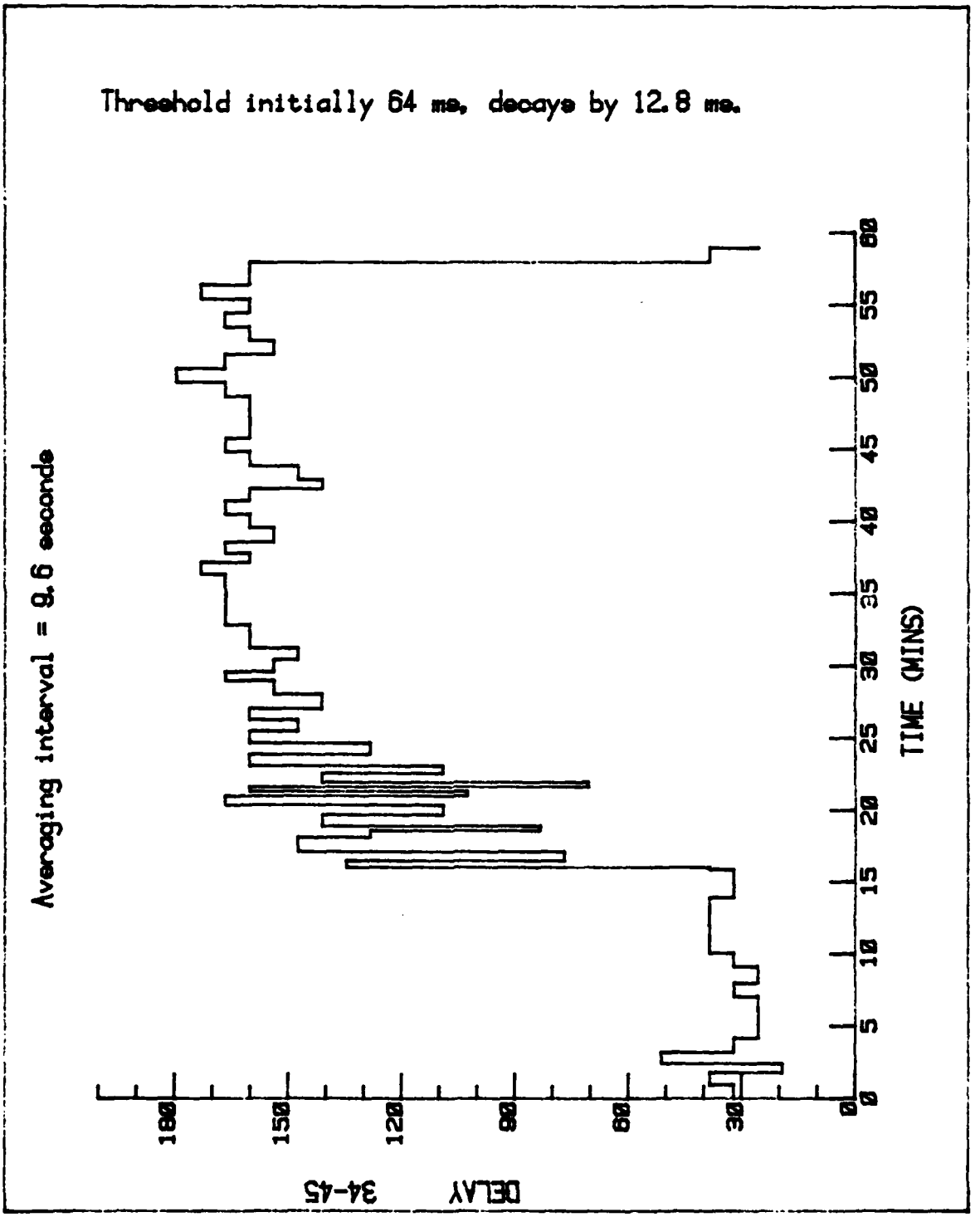

Threshold initially 64 ms, decays by 12.8 ms.
Averaging interval = 9.6 seconds
TIME (MINS)
0
5
10
15
20
25
30
35
40
45
50
55
60
DELAY 34-45
0
30
60
90
120
150
180

Figure 2-13b

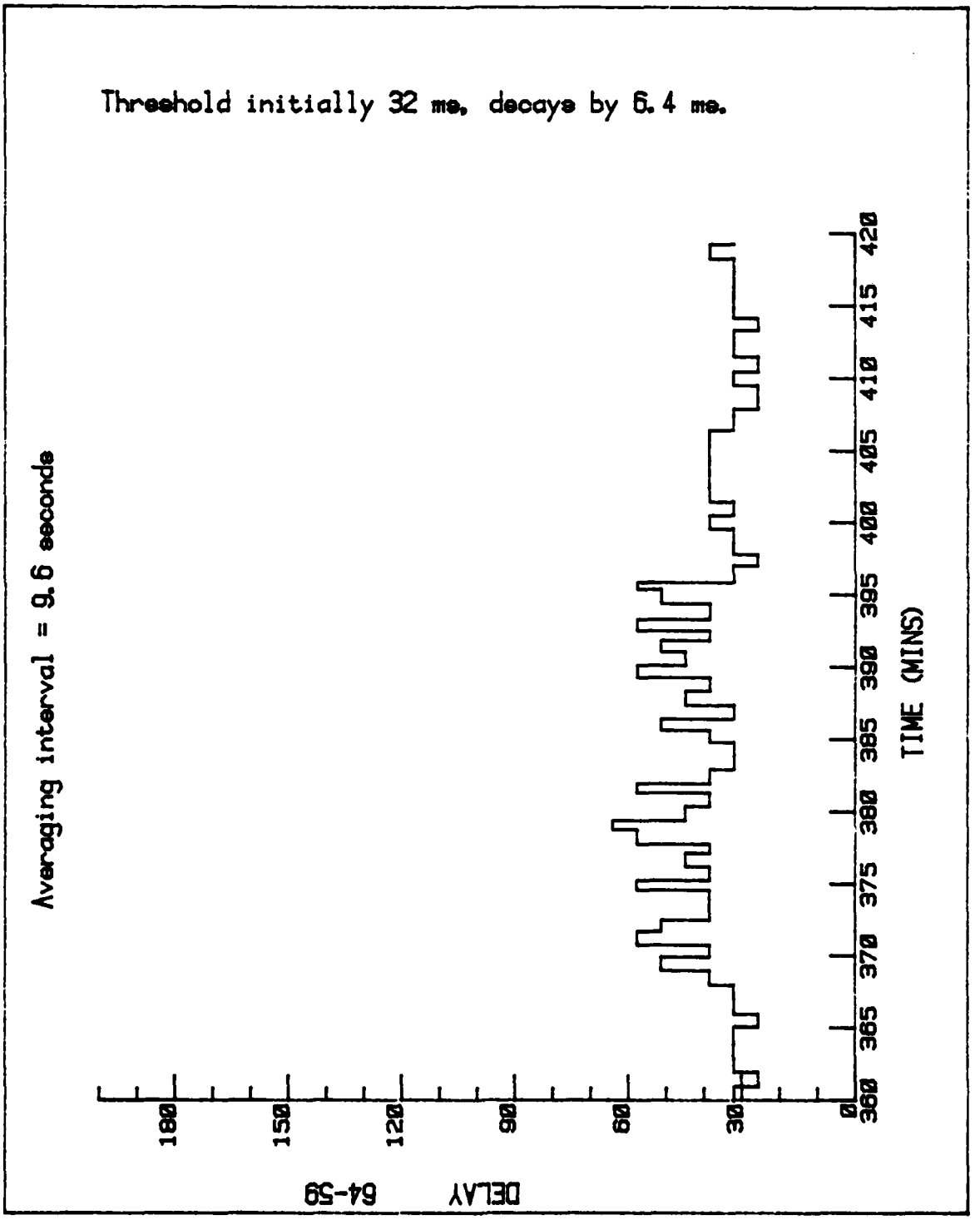

Threshold initially 32 ms, decays by 6.4 ms.
Averaging interval = 9.6 seconds
DELAY
64-59
180
150
120
90
60
30
0
360
365
370
375
380
385
390
395
400
405
410
415
420
TIME (MINS)

Figure 2-14a

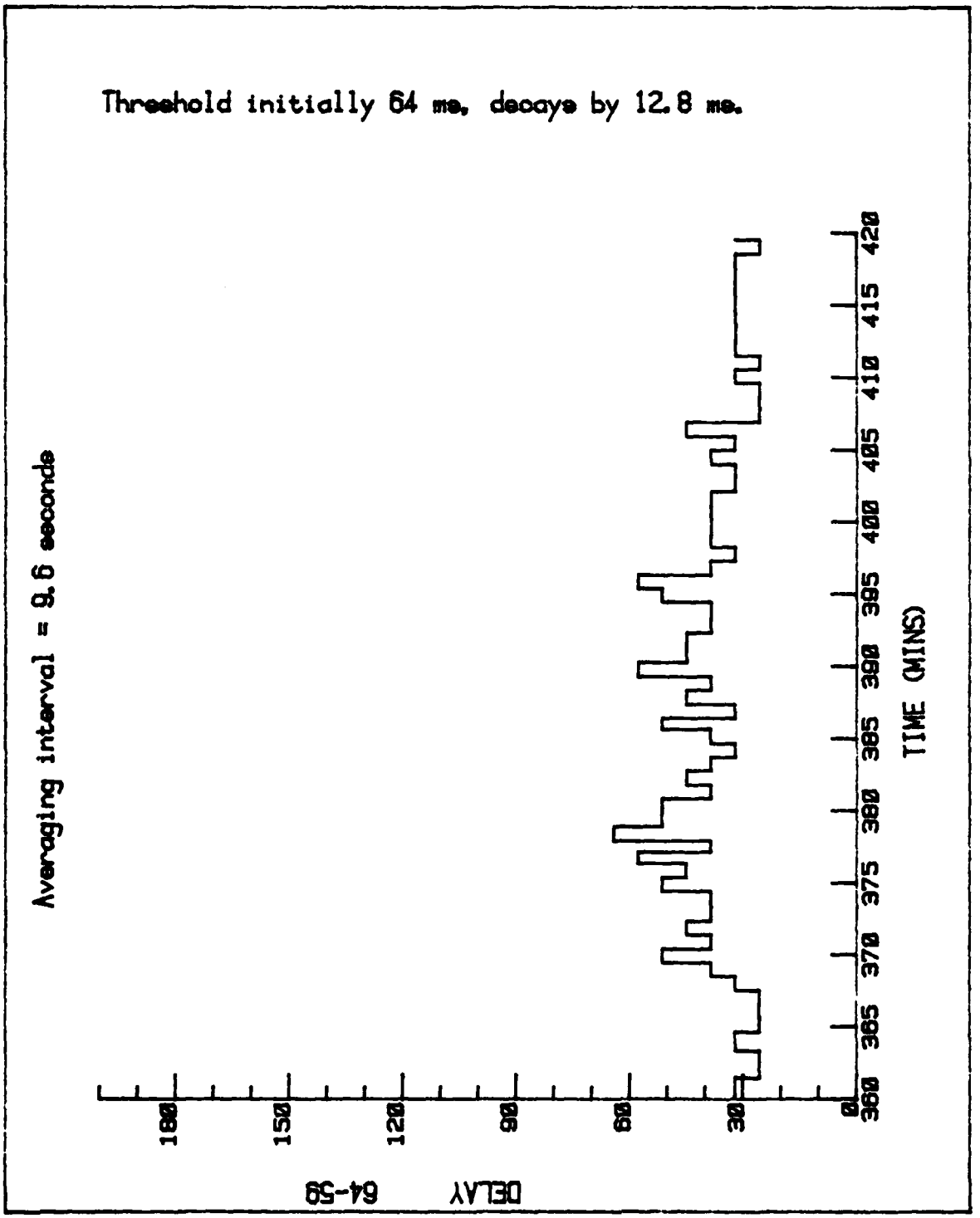

Threshold initially 64 ms, decays by 12.8 ms.
Averaging interval = 9.6 seconds
DELAY 64-59
180
150
120
90
60
30
0
360 365 370 375 380 385 390 395 400 405 410 415 420
TIME (MINS)

Figure 2-14b

significant. The larger initial threshold value resulted in one reported delay change every 58 seconds; the smaller, once every 44 seconds. This data is unusual in that there are so many changes over an hour-long period. Using the larger initial threshold, however, does gain considerable smoothness, without losing any responsiveness to large changes in delay. In Figure 2-14, the larger initial threshold also results in a smoother graph than the smaller (one change every 82 seconds, as opposed to one every 73 seconds). However, if Figures 2-14a and 2-14b are superimposed, it is seen that the smaller threshold does yield a somewhat quicker reaction to significant changes.

We intend in our initial implementation to use an initial threshold value of 64 ms. and a threshold decay value of 12.8 ms. This yields the minimum updating frequency we want (once per minute). It gives a smoother reported-delay curve than would a smaller value of the initial threshold, without losing much responsiveness. While larger values of the initial threshold would yield an even smooother curve, they lose too much responsiveness as they get larger.

2.3 Conclusions

Although the parameter settings are very important to the performance of the delay measurement routines, and hence to the

SPF routing algorithm itself, there is no easy way to determine the optimal values of the parameters. The measurements reported on in this chapter have served us as very important guidelines, but they cannot be regarded as the last word. As the new routing algorithm is installed, it will be instrumented so that we can monitor its performance and continue to examine the effects of changing the parameters. The algorithm will be implemented so that we can easily modify the parameters, if necessary, on the basis of our future experience.

It is also important to realize that the measurements we have taken so far were taken while the old routing algorithm was still in control. As we emphasized in our First Semi-Annual Technical Report, various characteristics of the network may change drastically when the new routing algorithm takes control, and this may invalidate previous measurements. Therefore, it will be necessary to continue to perform measurements and to tune the parameters after the network is cut over to the new routing algorithm.

3. DYNAMIC BEHAVIOR OF THE SPF ALGORITHM (SUMMARY)

During a five-week period, Prof. Dimitri Bertsekas of MIT worked with us on the dynamic behavior of the SPF algorithm. Although his work is strictly analytical in nature, it indicates some possible instabilities and sub-optimalities of the SPF and other shortest path algorithms. His analysis also indicates several techniques that can reduce both the likelihood and the severity of these degradations. Now that we have been alerted to some potential problems, we are planning a variety of tests on the SPF algorithm in real networks, in order to determine whether these problems can arise in practice, and if so, how they can be alleviated. This section summarizes the major results obtained by Prof. Bertsekas. Appendix A contains a detailed description and derivation of this material.

The basic stability problem can be most easily explained by means of a simple example. Consider the ring topology of Figure 3-1, and assume that each node, except for node 4, sends one unit of traffic (per second) to destination node 8, and node 4 sends u units. Further assume (1) that the "length" of a link, a dynamically varying quantity, is equal to the traffic flowing on the link, plus a constant bias b; and (2) that periodically and simultaneously each node measures the length of each of its outgoing links, and reports that information to all other nodes. In other words, this example could represent a simple network

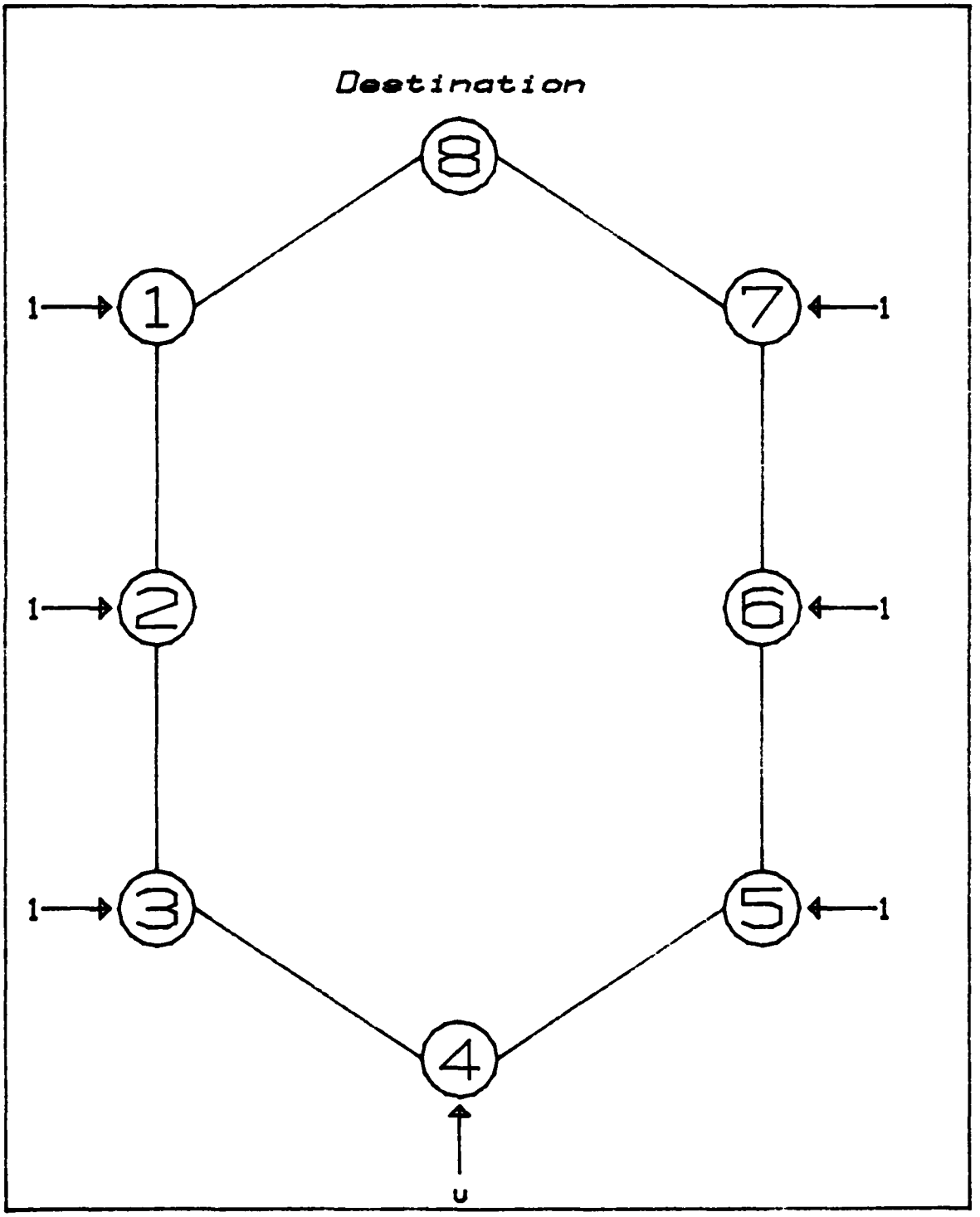


Figure 3-1 Example illustrating dynamic behavior of routing algorithm

running the SPF algorithm, with simultaneous updates that are propagated quickly and that are based upon the traffic flowing immediately prior to the updates. Finally, let R(i), $1 \leq i \leq 8$, denote the routing pattern in which the traffic of all nodes $k \geq i$ is routed counterclockwise, with all other traffic routed clockwise; for example R(3) means that traffic from nodes 1 and 2 flows clockwise to node 8, and traffic from nodes 3-7 flows counterclockwise.

It is easy to see that for the assumed length function, the optimal routing is R(4) or R(5). However, the stability of this optimal routing, and whether or not the initial routing will actually lead to this optimum, both depend on the choice of bias. Some specific results, easily derived for this example, are:

- if initial routing is R(4) or R(5) and if u = 0, then the routing is stable (and optimal).

- if the initial routing is R(k), $k \neq 4,5$, and if u = 0 and b = 0, then the routing will oscillate between R(k) and R(9-k).

- if $u \neq 0$ and b = 0, then regardless of the initial routing, the routing will eventually oscillate between the two worst-case (most sub-optimum) values, R(1) and R(8); note that this conclusion is valid even if $u \ll 1$.

- if $u \neq 0$, then very small bias values result in these same oscillations; intermediate values of bias lead to stable equilibrium, provided that the initial routing is sufficiently near the equilibrium point; and large values of bias guarantee convergence to the optimum.

In the Appendix, these specific results are generalized to other traffic flows and topologies. Because the stability analysis of discrete networks is difficult, a continuous model is developed. The major conclusions, although derived from a continuous model, should nevertheless be applicable to discrete networks.

The fundamental result of the analysis, suggested by the simple example above, is that there is an interplay between the traffic pattern, the bias value, and the stability of the routing. Clearly, a large value of bias leads to an equilibrium routing which is stable or, more precisely, static and which is close to the minimum-hop routing, whereas a small value of bias could result in an unstable equilibrium (or no equilibrium at all). As the bias increases, however, the routing adapts less readily to congestion, and therefore there is a trade-off involved in the choice of bias. In general, it can be shown that for stability, the bias should increase as the total traffic level increases and as the marginal delay increases. The marginal delay is defined as the additional "length" that would be introduced by sending additional traffic on a link; (i.e. it

is the derivative of the length function with respect to traffic flow).

The assumptions utilized in the above example simplified the analysis. We are in the fortunate (and unusual) position, however, that relaxing some assumptions leads to a more practicable implementation, results in a more realistic model, and actually yields improved performance! Specifically, if the reported link lengths are based upon measurements that have been averaged over several past routings, then the bias required to achieve a stable equilibrium is significantly reduced. A similar reduction occurs if updates are generated asynchronously by the nodes. The smaller the bias, the more sensitive routing can be made to congestion. Also, even if the algorithm is unstable, its dynamic behavior is dramatically improved by averaging and asynchronous operation. That is, if oscillations exist (as in the example above), they will be less violent.

The above remarks notwithstanding, the analysis indicates that the choice of the bias value may not be an easy task. For example, in a simple loop network it can be shown that the total delay can be minimized by choosing the length function to be the marginal delay, provided that the algorithm actually converges to the equilibrium routing. On the other hand, for stability (and good convergence), the bias must be increased if the traffic level increases. But it can be shown that if the bias is fixed,

then there exists some level of traffic beyond which that level of bias is inadequate. Even if traffic levels are limited, as they are in a real network, a very large bias may be needed to prevent instability under all circumstances. This obvious "solution" of using a constant high bias might result in non-adaptive min-hop routing. Alternatively, the use of different bias on different links may also lead to difficulty. In order to better understand these tradeoffs and achieve good routing performance, we plan to experiment with a variety of schemes.

4. RELIABLE TRANSMISSION OF ROUTING UPDATE MESSAGES

In choosing the SPF algorithm, we presupposed that all IMPs will be able to run the algorithm from the same data base. That is, we presupposed that there will be no period of time (except for an insignificantly short "transition period") during which the IMPs will disagree about the delay on any given line. If this presupposition does not hold, the IMPs may make conflicting routing decisions, possibly resulting in long-lasting loops. Therefore, reliable transmission of the updates for the SPF algorithm is of the utmost importance. It is much more important than with the previous ARPANET algorithm. With SPF, even a single lost update can cause a long-term loop to form, thereby causing a catastrophic degradation in network performance.

In our First Semiannual Technical Report (BBN Report No. 3803) we argued that flooding updates through the network was a more reliable means of transmission than using some form of multi-address routing. While it is true that flooding is an inherently very reliable method of transmission, it is not sufficient by itself to guarantee delivery. It must be augmented with a retransmission/acknowledgment scheme, to ensure that line errors cannot prevent updates from reaching some IMP in the network. Furthermore, there are certain situations involving lines going up and down in rapid succession in which flooding is not reliable. Consider the following scenario. Let A, B, and C

be three IMPs such that A and B are neighbors, and B and C are neighbors. Let u be an update from some other IMP in the network. Suppose that at time t_0, A receives u, but the line from A to B is down (although the line from B to C is up). Clearly A cannot transmit u to B. Then at time t_1, the line from A to B comes up. At t_2, the line from B to C goes down. At t_3, C receives u. Now C cannot transmit u to B. Unless the flooding scheme is augmented in some way, B may never receive u, even though there was never any moment at which B was totally disconnected from the network.

Augmenting the flooding scheme with a protocol to ensure reliable transmission is not, however, sufficient to ensure that all IMPs have the same SPF data base. It may occasionally happen that two or more updates from the same IMP are being transmitted around the network at the same time (because a second update had been generated before the first had been received by all IMPs). Once this happens, it is possible that some IMP will receive the updates out of order. If different IMPs can receive the updates in different orders, it is essential that they always keep the update which was generated most recently (rather than the one received most recently), while discarding the others. This can be effected by having the updates carry serial numbers. However, the addition of serial numbers gives rise to a further problem. Suppose IMP A sends out update a with serial number s(a), and

this update is received by IMP B. Then A and B become disconnected due to network partition. Sometime later the partition ends, and B receives from A update a' with serial number s(a'). The problem is that during the partition, A's serial numbers may have wrapped around one or more times. Thus if B simply compares s(a) with s(a') to determine whether a or a' is the more recent update, it may come to the wrong decision. Some scheme must be developed to ensure that the serial numbers get re-synchronized after a partition.

A similar problem arises if an IMP goes down and needs to be restarted. When it comes back up it may not remember the serial number it used last. If it starts sending updates again with a serial number that is too low, the other IMPs may not recognize the new update as being the most recent. So some way of resynchronizing the serial numbers is also needed for this case.

Thus we have two different protocol problems to solve. The first is the problem of ensuring that the most recent update from a given IMP is transmitted reliably, and that when updates are received out of order only the one which was generated most recently is accepted. A protocol to ensure this will involve serial numbers, thereby giving rise to the second protocol problem, that of re-synchronizing serial numbers after a partition. Various proposed solutions to these two problems will be discussed separately in what follows.

4.1 Reliable transmission

We have considered three different ways of augmenting the basic flooding scheme to ensure reliable transmission. These are discussed below. An important feature of all the schemes is that updates need not be kept buffered pending retransmission. Since all the information in an update is kept by the IMP in its routing tables, updates which need to be retransmitted can be reconstructed from the tables.

I) Logical channels

This protocol is modeled on the IMP-IMP protocol. It divides each line into NN logical channels in each direction, where NN is the number of IMPs. When an update from IMP n is transmitted, it travels on channel n. The channel is blocked between the time the update is transmitted and the time an acknowledgment (ACK) for it is received. While the channel is blocked, no other update may be transmitted on it. If an update receives a negative acknowledgment (NAK), it is retransmitted. The ACK/NAKs are carried in the line up/down protocol packet (which are sent once every 640 ms.), one bit per logical channel.

When any given update is being transmitted in a particular direction on a particular line, one IMP is the transmitter and one is the receiver. Each IMP must keep NN transmit bits and NN receive bits (one for each logical channel) for each of its

lines. (The transmit bit for a channel is also copied into each packet transmitted on that channel.) The protocol is specified as follows:

1) Initialization

When a line comes up, all the transmit and receive bits for that line are set to zero.

2) Transmitter

a) When an update is ready for transmission, a check is made to see whether its logical channel is blocked.

i) If it is blocked, a flag is set indicating that an update was received while the channel was blocked (call this the "update-waiting" or "UW" flag for that channel). The update may be processed internally, but it should not be transmitted on that line.

ii) If the channel is not blocked, the transmit bit for that channel must be copied into the packet, and the packet transmitted. The channel must be marked "blocked".

b) When a line up/down protocol packet is received, the acknowledgment bit for each blocked channel (recall

that each protocol packet carries one acknowledgment bit for each channel) must be compared with the transmit bit for that channel.

i) If the bits do not match, the channel has been ACKed. The transmit bit must be flipped, and the channel unblocked. If the UW flag is set, the flag should be cleared, and the latest update from the appropriate IMP should be re-constructed from the routing tables and then transmitted.

ii) If the bits match, then the channel has been NAKed. The latest update from the appropriate IMP should be re-constructed from the routing tables and then transmitted. The UW flag should be cleared. (Note that this always results in the transmission of the most recent update. Updates which have been superseded are never re-transmitted.)

3) Receiver

a) When an update is received, the transmit bit in the update packet is compared to the receive bit for its logical channel.

i) If the bits match, the packet is a duplicate (i.e., a spurious retransmission). The packet should simply be discarded.

ii) If the bits do not match, the receive bit for that channel should be flipped. A determination should be made as to whether the update is the most recently received from the source IMP (by comparing serial numbers). If not, the packet is discarded. Otherwise the packet is processed internally, and transmitted on all lines except the one it was received on, subject to the transmitter protocol described above. This is the part of the protocol that ensures the flooding of recent updates.

b) When a line up/down protocol packet is about to be sent, the receive bits for all logical channels are copied into the packet as the acknowledgment bits.

This protocol ensures reliable transmission on an IMP-IMP basis, similar to the ordinary IMP-IMP protocol. It differs from it in only three important respects. It does not guarantee delivery of every update packet, but only of those update packets which are not superseded by more recent updates before they are NAKed. (However, this is all that is needed in the present

context.) It does not require packets to be buffered while awaiting acknowledgment, since all the information in a NAKed update can be obtained from the routing tables. Finally, the acknowledgment comes periodically rather than immediately.

This protocol shares with the IMP-IMP protocol the advantage of being a very reliable low-overhead scheme. The line bandwidth required is only NN bits every 640 ms. (Since the line up/down protocol packets are sent every 640 ms. anyway, there is no additional overhead incurred.) This is much less than would be required by explicit ACKs. The protocol ensures that if an update is missed due to line error, it will be re-transmitted (or a more recent update from the same source IMP will be transmitted) within 640 ms., thereby making lost updates impossible.

However, the protocol has several disadvantages:

1) It does not solve the problem discussed above where an IMP fails to get an update because one of its lines comes up just as another of its lines goes down. To solve this, it is necessary to augment the protocol with a special mechanism, such as "re-transmitting" all updates over a line which has just come up.

2) The protocol may not respond quickly enough to missed updates. Although a retransmission is assured within 640 ms.,

this is a long period relative to network transmit times. If an update is missed due to a line error, IMPs may disagree about the state of some lines for 640 ms. longer than they otherwise would, which may mean an additional 640 ms. of unacceptable routing. (On the other hand, it must be noted that the flooding scheme will, in general, deliver more than one copy of each update to each IMP. Therefore, missing one copy of an update due to a line error would not mean that the update would be totally lost, even if there were no retransmission.)

3) The protocol is very expensive in terms of its IMP memory requirement. It requires four flags for each logical channel on each line of an IMP - the transmit bit, the receive bit, the "channel-blocked" flag, and the UW flag. In an 80-IMP network, an IMP with four lines would need 80 words just for these flags. Furthermore, the protocol requires a great deal of bit manipulation, which requires a large amount of code in the IMP. Since memory in the IMPs is a scarce resource, this is a major disadvantage.

4) Since a logical channel is blocked until an ACK arrives, the flow of updates is artificially slowed down in some cases.

II) Exchanging Serial Numbers

A much simpler protocol could be based on the periodic exchange of serial numbers between neighboring IMPs. Suppose

that, at a fixed interval, each IMP prepared a packet containing the serial numbers of the most recent updates that it has received from all the IMPs in the network. This packet would then be transmitted to each of the IMP's neighbors. When an IMP receives one of these packets from a neighbor, it compares its neighbors' most recent serial numbers with its own. In this way an IMP can determine whether or not it has received an update which its neighbor has not yet received (or, more accurately, had not yet received a little while ago). If it has, it re-constructs the update from its routing tables and transmits it to the neighbor, thereby ensuring that its neighbor is at least as up-to-date as it is.

This protocol is just as effective as protocol I in ensuring that line errors do not cause updates to get lost. It has the further advantage of automatically handling the case where one of an IMP's lines goes down just as another of its lines has come up. Since the protocol is based on a periodic exchange of serial numbers, rather than one-time-only acknowledgments of transmissions, missing updates are automatically detected.

However, this protocol has quite a serious disadvantage. The periodically transmitted packet which contains the serial numbers is rather long, much longer than the periodic packet required by protocol I. While this may not result in using an excessive amount of bandwidth, when measured in absolute terms,

it uses the bandwidth in an unfortunate way. The periodic transmission of long packets can have an effect on network performance which is much worse than would seem to be indicated by considering just the absolute bandwidth used. (This is discussed in our first Semiannual Technical Report.) Any scheme using periodically transmitted long packets must be avoided.

The size of the periodic packet can be made shorter if it is not required to send all the serial numbers at once. For instance, if the serial numbers are six bits each, and one-sixth of the serial numbers are sent at a time (say, in the line up/down protocol packet), then the utilization of line bandwidth can be made to be exactly the same as in protocol I. If protocol II is modified in this way, however, it loses much of its attractiveness, since it becomes much less responsive to missed updates. If it takes six intervals to cycle through all the serial numbers, and each interval is 640 ms. long, then a missed update might not be retransmitted for 3.8 seconds. This is much too long a period to wait. If the missed update causes routing loops to form and to persist for this long a period of time, areas of congestion can form, causing packets to be discarded from the subnet. The situation is made even worse if one of the protocol packets containing the serial numbers is not received because of a line error. Another 3.8 seconds will go by before the protocol cycles around again to the same group of serial

numbers. This version of the protocol does not seem to be adequately responsive.

III) Explicit Acknowledgement by Echoing

The third protocol we considered uses a sort of explicit acknowledgment which fits very neatly into the basic flooding scheme. Recall the way in which flooding works. When an update is received, a check is made to see whether that update was created more recently than any other update which has been received from the same source IMP. If not, the update is simply discarded. If so, the update is transmitted over all lines except the line on which it was input. There is no need to transmit it over the input line, since it is known that the neighbor on that line has already seen the update. In protocol III, however, the updates are transmitted over all lines, including the input line. The "echo" over the input line serves as an acknowledgment. If the echo is not received in a given amount of time, the update can be retransmitted. The retransmitted update must carry a special bit (the "Retry" bit) indicating that it is a retransmission. The purpose of this bit is to force an echo, even if the update has been seen before and would ordinarily just be discarded. This is needed to cover the case where an "acknowledgment" (i.e. the echo) is missed, causing a spurious retransmission. Unless a second echo is forced, the retransmissions may go on without end.

To implement this protocol, an IMP with k lines must maintain k retransmission timers for each IMP (i.e., k x NN timers). The specification of the protocol is as follows. When an IMP receives an update from some IMP J, it first checks to see whether the Retry bit is on. If so, it is turned off, and the update packet is "echoed" back over the input line.

After this is done, or if the Retry bit was not on, the serial number of the update must be checked. If a more recent update from IMP J has already been seen, the update should simply be discarded.

If this update is the most recent from IMP J, but has already been seen, then it should also be discarded. It may, however, be an echo-acknowledgment. Therefore the retransmission timer for IMP J which corresponds to the input line should be set to zero.

If this update is the most recent update from IMP J, but has not been seen before, then it must be flooded. It should be echoed back over the input line, and the timer corresponding to the input line set to 0. It should be transmitted over all other lines, and the re-transmission timers for IMP J corresponding to those lines should be set to their maximum value.

Periodically (say every 25 ms. or so) all non-zero retransmission timers must be decremented. Whenever a timer gets

decremented from one to zero, an update must be re-transmitted (with its Retry bit turned on).

This protocol is more responsive than either of the two protocols previously discussed. Acknowledgements are sent immediately, rather than periodically, and the retransmission time-out interval can easily be tuned for best effect. This protocol uses more line bandwidth than protocol I, but that does not seem to be a problem. If the retransmission timers are implemented as two-bit counters, this protocol requires only half as much memory as protocol I. It does not have the channel-blocking problem of protocol I, and is generally much simpler to understand and to implement. For these reasons it is preferred to protocol I.

It is also preferable to protocol II. Although it has the disadvantage of requiring some additional mechanism to handle the case where one of an IMP's lines goes down just as another of its lines comes up, protocol II's disadvantages (discussed in the previous section) are enough to outweigh this one relatively minor advantage. This special case can be handled by transmitting (as "re-transmissions") <u>all</u> updates over a line which has just come up.

4.2 Re-synchronizing after Network Partition

Suppose that certain IMPs become disconnected from each other due to a network partition, and that the partition ends an unspecified period of time later. This means that for that period of time, certain IMPs have been unable to receive routing updates from certain other IMPs. When the partition ends, the IMPs in one segment of the network will remember the serial numbers of the last updates they received from IMPs in the other segment. However, if the partition lasted a long enough time, the serial numbers may have wrapped around one or more times. If there has been wrap-around, it is meaningless to compare the serial numbers of new updates with the serial numbers of old updates. Some method must be developed to force all IMPs to discard the pre-partition updates in favor of the post-partition ones.

The following scheme suggests itself immediately. The SPF computation enables each IMP to know whether another IMP is reachable or not. When an IMP becomes unreachable, all updates from it shall be ignored. When it becomes reachable again, the next update received from it shall be accepted. This automatically resynchronizes the serial numbers.

Although the scheme seems initially attractive, it has serious difficulties, as would any scheme which requires some

updates to be ignored. Recall that if there is any long-term discrepancy in the data bases maintained by the IMPs, the SPF calculation may result in routing loops which may make the network useless for long periods of time. The proposed scheme enables such discrepancies to exist after a partition ends. Suppose (for concreteness) that the network is partitioned East-West. When the partition ends, the Eastern IMPs will initially appear unreachable to the Western IMPs, and vice versa. Then updates will begin to flow across the East-West boundary. Eventually, all IMPs will have processed updates from all other IMPs, and they will all see each other as reachable again. The problem arises because Western IMPs cannot begin to process updates from Eastern IMPs as soon as the partition ends. Rather, they must wait until the Eastern IMPs become reachable. But as updates flow from East to West, different Western IMPs will process the updates in different orders, and at different rates. This means that if E is an Eastern IMP, and W1 and W2 are Western IMPs, there may be some time interval during which W1 can accept, process, and forward updates from E, while W2 must ignore them, neither processing nor forwarding them.

The result is that it may be a very long time before updates from E are able to reach <u>all</u> the Western nodes (even though they are able to reach <u>some</u> Western nodes in a very short time). During this time, the IMPs' data bases are inconsistent. Exactly

how long it takes depends on which reliable transmission protocol is used. If protocol I or III are used, then once an update is ignored, an update from the same source may not be seen again until one is generated at the source, possibly a minute later. As a result, the IMPs may have inconsistent data bases for several minutes. If protocol II is used, it may be possible to reduce this period to several tens of seconds, but that still is far from satisfactory.

It must be understood that the problem is not merely that it will take a long time to re-integrate the two segments after a partition. The point is that when a partition ends, incorrect routing patterns may form which affect communication even between IMPs in the same segment. For example, two Eastern IMPs which are communicating with each other may begin routing their traffic to each other via a series of Western IMPs. But if the Western IMPs hold inconsistent information about the Eastern IMPs, the traffic may never get through. As a result, some IMPs which were able to communicate during the partition would not be able to communicate after it ends.

This also shows that when a partition ends, some action must be taken to ensure that the most recent updates from each segment are immediately sent into the other. It is important not to wait until new updates are generated. Otherwise, there may be some long period of time during which the IMPs have inconsistent data

bases, and the network will be useless for this period. However, as discussed in the previous section, whenever a line comes up it is necessary to transmit all recent updates on that line, in order to ensure reliable transmission. This same procedure ensures quick transmission of all updates across a partition boundary after the partition ends.

The source of the problem with the initial scheme is that some IMPs are forced to ignore certain updates while others are not. What is needed is a scheme which allows all IMPs to process all updates they receive as soon as a partition ends. The following scheme has been suggested:

> Let zero serve as a canonical lowest serial number. No update packet shall ever carry a serial number of zero. However, when an IMP A is determined by an IMP B to be unreachable, B shall set its tabled copy of A's most recent serial number to zero. Then when B next receives an update from A, the new update will automatically be accepted as a recent update, and can be processed normally.

The intent of this scheme is that when a partition ends and updates begin to flow again between the segments, they can be accepted and processed as soon as they are received. Thus the scheme does not allow the formation of long-term discrepancies in the data bases. However, it has a different sort of problem which is just as serious.

Suppose again that the network is partitioned East-West. Let M be an Eastern IMP which is on the border of the partition. Let A, B, and C be three other Eastern IMPs which are connected in a triangle, and let W be a Western IMP. Let m be an update from M which reports the partition. That is, the other Eastern IMPs notice the partition as a result of processing m. (Presumably m reports that the line between M and its Western neighbor M' has gone down.) Let w be an update from W which reached the Eastern segment of the network just before partition, and let s(w) be its serial number. Now it is certainly possible that m gets to A before w does, and that w actually follows m around the triangle. As update m travels around the triangle, IMPs A, B, and C will determine that W is unreachable, and will set their tabled copies of W's most recent serial number to zero. But as soon as they do this, they will receive update w. Since zero is the canonical lowest serial number, s(w) > 0, so w always looks like a recent update. I′ is accepted and forwarded. However, the next time A, B, or C does a routing computation, they will again realize that W is unreachable, and reset their copies of its most recent serial number to zero. Once they do this, they no longer remember that they have seen w before. When w comes around the loop again, it again looks like a recent update (s(w) > 0), and is again accepted and forwarded. There is nothing to stop this process, which may continue indefinitely. In fact, w may still be traveling around when the partition ends.

Once the partition is over, W will eventually send out another update, w'. This may result in w and w' being in the network at the same time. If the partition lasted long enough for the serial numbers to wrap around, then it is meaningless to compare s(w') with s(w). As a result, the IMPs may incorrectly believe w to be more recent than w', and routing will be based on very old and out-of-date information. Depending on the exact values of s(w) and s(w'), this problem may persist for a very long time, causing extremely bad performance throughout the whole network (for instance, if w' reports that one of W's lines has gone down, lots of traffic may be routed to a non-existent line).

We see from this that it is not enough to allow all updates to be processed as soon after a partition as they are received. In addition, we must be able to ensure that if the partition has lasted long enough for serial number wrap-around to have occurred, then no pre-partition updates are still around. One way of ensuring that updates do not stay around the system too long is simply to time them out. When the last received update from a given IMP becomes "too old", the next update from that IMP is automatically accepted as more recent, no matter what serial number it has. This would eliminate the problem of an old update circulating around forever. That problem was caused by artificially setting certain serial numbers to zero, a feature which would no longer be required. And in the example above, by

the time the partition ended, w would be "too old", so w' would be automatically accepted as more recent when it is received.

The most accurate way to determine the age of an update would be to maintain a globally synchronized clock. Each update packet would carry the time of its creation at its source, as well as a serial number. Then each IMP would know exactly how long ago an update was generated, subject to the resolution of the clock and possible inaccuracies of synchronization. (Note that the clock-time could not be used instead of a serial number, since it is hard to ensure that the clock time will increase monotonically as each update is sent out.) Use of a globally synchronized clock has several problems, however. One problem is simply the difficulty of maintaining synchronization. But the most serious problem is that of re-synchronizing after a partition. When a partition forms, there is no way of ensuring that the clocks in the two segments stay synchronized. If, when the partition ends, updates flow between the two segments before re-synchronization is established, the results are unpredictable. So not only must there be a method of re-establishing sync, there must also be some way for the IMPs to determine whether or not sync has been re-established, so they know whether or not it is safe to pass on updates. While such methods can no doubt be developed, they add a significant amount of complexity to the scheme. It is worthwhile then to investigate means of

determining the age of an update which do not require globally synchronized clocks.

Suppose IMP A transmits update a which is received at IMP B. At any given time, the age of update a (as determined by B) can be divided into two components - transit time and holding time. Transit time is the time it took the update to travel from A to B. Holding time is the time since it arrived at B. If we may assume that, within a certain amount of time after an update is initially created, its holding time at any given IMP will be very much larger than its transit time to that IMP, then we may neglect the transit time, and equate the update's age to its holding time. But the holding time can be computed from a purely local clock. No global synchronization is necessary at all.

In the ARPANET, network transit times are on the order of 100 milliseconds. We would not want to consider an update "too old" unless it was at least a minute or so old (justification for this is given below). Within a minute after any update is created, its holding time at any IMP would almost always dominate its transit time to that IMP by so much that the transit time could be neglected. There is only one exceptional case. That is when an update has to be re-constructed and re-transmitted because a line has come up. If A has held an update for 59 seconds before transmitting it to B, we do not want B to have to wait yet another minute before regarding the update as too old. The time the update was held at A must be figured in, somehow.

This problem is resolved by having the update packets carry around some indication of their age. Suppose we allocate a k-bit field in each packet, and each IMP has a clock which ticks once every t seconds. When an update is first generated, the "age field" is loaded with 2^k-1. When an update is received, its age field is stored, and decremented once each tick of the clock. When a packet is re-transmitted, the current stored value of the age field is placed in the packet. An update is considered "too old" when its age field has been decremented to zero. This scheme ensures that the age of an update as seen by a given IMP is determined by the time it has been held in the given IMP, plus the time it was held in any IMPs from which it was re-transmitted. It must be realized though that this method of computing the age of an update is subject to an error which is a function of the coarseness of the resolution of the clock-tick. These errors can accumulate if an update is re-transmitted many times. For a given time-out period, the coarseness of the clock resolution can be traded off against the number of bits in the age-field. Optimum values must be determined by experimentation.

The use of a time-out scheme like the one just described places several constraints on various parameters used by the routing scheme. The constraints are:

a) It should be impossible for an update's serial number to wrap around (i.e., to get half-way through the sequence number

space) before the time-out period expires. After all, the whole point is that old updates should get timed out before serial numbers can wrap around.

b) The time-out period should be somewhat longer than the maximum period between updates from a single IMP. This prevents good, recent updates from reachable IMPs from timing out. Failure to meet this constraint may result in failure to transmit a valid update when a partition ends. This in turn may result in long-term data base inconsistencies.

c) It should be impossible for an IMP to crash and be restarted within the time-out interval. This ensures that all of the IMP's old updates will time out before any new updates are sent.

4.3 Final Specification of Protocol

The following is a complete and precise specification of the updating protocol to be used in the ARPANET. Particular values of the parameters are chosen to meet the constraints described above. No claim is made that these choices of parameter values are optimal, but only that they seem safe enough for use in the initial implementation.

1. Determining whether an update is acceptable

An update u from IMP I should be acceptable at IMP J if and only if u was sent more recently from I than was the update from I which J already has tabled. Acceptability will be determined by the use of a three-bit age field and a 6-bit serial number field. These 9 bits must be allocated both in the update packet and in the IMP's tables (i.e., the tables must have 3 bits per IMP for the age and 6 bits per IMP for the serial number.)

a) The age field

i) When an update packet is created, the age field in the packet is set to a max value.

ii) When an update is received and accepted (i.e., judged acceptable), the age field of the packet is copied into the tabled age field for that IMP.

iii) Each IMP will have a slow-ticking clock, which ticks once every t seconds. This clock is local to each IMP and is not synchronized with other IMPs. At each tick of the clock, the tabled age field for each IMP is decremented, unless it is already zero, in which case it shall not be further decremented.

iv) When an update is re-created from the tables in order to retransmit it, the tabled age field is copied into the packet age field. Updates with age 0 shall never be re-transmitted.

b) The serial number

Each IMP shall maintain a serial number for the updates it originates. When the IMP is about to generate an update, it increments the serial number by 1 (modulo 64) and copies it into the packet. In order to determine which of two serial numbers is more recent, the following algorithm will be used.

Let n and m be two serial numbers. If n = m, then neither is "more recent" than the other. Otherwise, suppose n > m. Then n is "more recent" than m if an only if n - m $\leq$ 32, and m is "more recent" than n if and only if n - m > 32.

c) Determining acceptability

Suppose IMP I has already accepted update u from IMP J, and now receives update u' from J. The question is whether u' is acceptable. There are several cases:

i) u has age 0, but u' has non-zero age. Then u' is acceptable.

ii) u' has age 0. Then u' is not acceptable. This is an error condition which should never occur.

iii) u and u' both have non-zero age. Then u' is acceptable if and only if its serial number is more recent than that of u.

d) Parameters

i) Clock ticks once every 8 seconds

ii) Maximum age = 8

2. Transmission and acknowledgment

Suppose IMP i has k lines. Then i will have to maintain k 2-bit timers (one for each line) for each IMP j.

When a node receives an update from IMP j it first checks to see whether the RETRY bit is on. If so, it is turned off, and the update packet is "echoed" back over the input line.

After this is done, or if the RETRY bit was not on, the IMP must determine whether the update is acceptable. There are three cases to consider:

a) The update is not acceptable because its serial number is identical to the tabled serial number of the update last accepted from IMP J (where both the age-field in the packet and the tabled age-field are non-zero). The timer for IMP J corresponding to the input line is set to zero. The packet is discarded.

b) The update is not acceptable for some other reason (i.e. serial number not recent enough, or age of zero). The packet is discarded.

c) The update is acceptable. Its age, serial number, and delay information are copied into the tables. The timer for IMP J corresponding to the input line is set to zero. The timers for IMP J corresponding to the other lines are set to 3. The packet is "flooded" over all the lines. Note that in the special case where an update is first being transmitted from its source IMP, there is no input line, and the RETRY bit is initially set off. There is one more special case - if the packet has already been sent over the input line because the RETRY bit was on, it need not be sent again.

The timers for land-lines must be decremented every 25.6 ms., and for satellite lines, once every 128 ms. When a timer ticks down to zero, the update must be re-created and transmitted over the line whose timer reached zero. The RETRY bit must be turned on before the transmission proceeds, and the timer must be set back to 3.

When a line comes up, all updates which do not have age zero should be re-created and "re-transmitted" over that line, with the RETRY bit on.

When an IMP crashes and is restarted, it must wait 90 seconds before coming back up. This ensures that any old updates from that IMP have aged sufficiently so that the new updates will be recognized as more recent.

Each IMP must send an update at least often enough to ensure that its most recent update does not appear to be too old (i.e., at least once per minute).

5. ENHANCED MESSAGE ADDRESSING CAPABILITIES

How should one user of a network address messages to other users? The answer to this question is fundamental in defining the appearance of the network to its users. For example, does one user have to know exactly where the other is located, or just the region of the network, or is the address independent of location? Can he identify himself to the network or does the network know who he is automatically? If self-identification is possible, can he have several addresses corresponding to several roles or functions? Can he have multiple connections to the network, and can he move from one location to another without changing his address(es)? Can he send a single message to a group or list of other users (e.g., a mailing list) automatically? Can he set up "conference calls" with other users, and join conferences in progress? Can he send a message to all other users?

These questions are important for several reasons: some addressing modes allow functions which would not be available otherwise (e.g., the ability to send a message to a distribution list without knowing the identity or location of the members of the list), and which are essential for certain types of users and applications. Furthermore, these addressing capabilities offer opportunities for efficient implementations that would not exist otherwise (e.g., a message addressed to a group can be

transmitted with fewer packets than the equivalent separately addressed messages).

We distinguish between the addressing mode, how the user identifies the intended recipient(s) of a message, and the addressing implementation, how the network processes the message. The former is an interface between the user and the network; the latter is a protocol within the network. It is also useful to distinguish between addressing, how the network selects the destination(s) of the message, and routing, how the network selects the path(s) over which the message travels.

Our investigations have led us to the conclusion that the following three addressing methods would be valuable additions to the ARPANET which now supports only physical addressing (as shown in Figure 5-1a):

1. Logical addressing, in which a permanently assigned logical address denotes one or more physical addresses (see Figure 5-1b). The sender does not need to know the physical location of the destination subscriber, and subscribers can relocate without change of address. Since one logical address can refer to several physical addresses, subscribers can connect to the network by multiple lines ("multiple homing"), increasing reliability and traffic capacity (Figure 5-1c).

PHYSICAL ADDRESSING

A Sends a Message To D, Addressed To 3.2
(Node 3 ,Access Line 2)

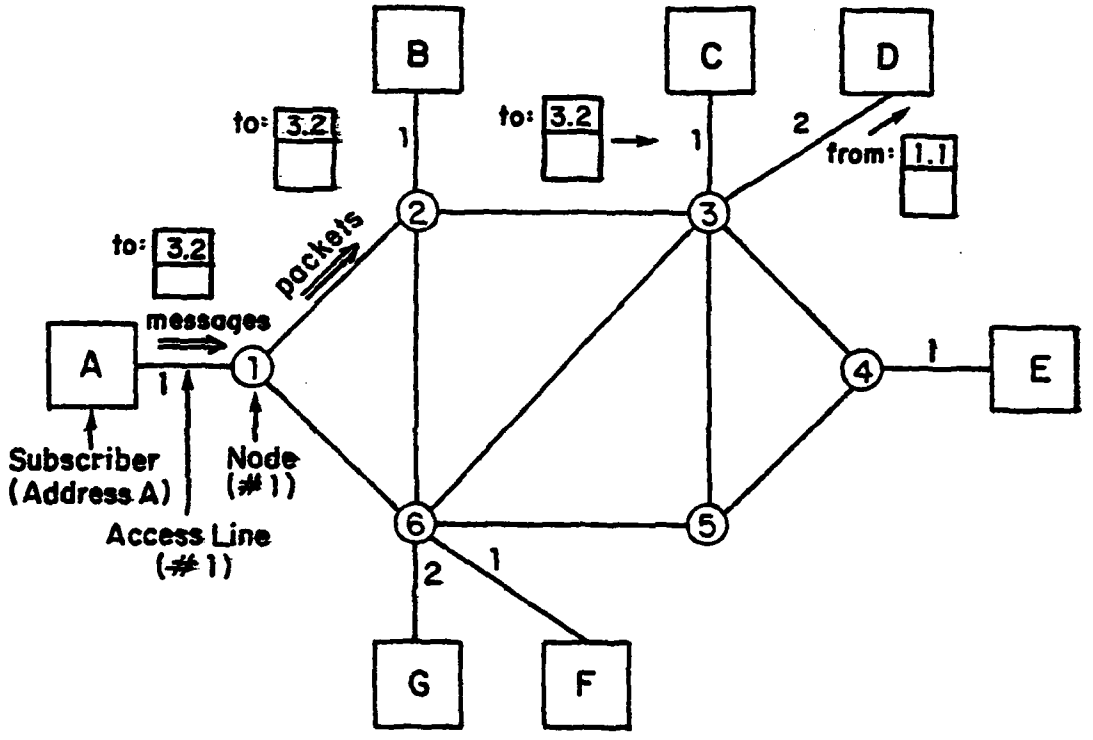


Figure 5-1a Message Addressing Modes -- Terminology

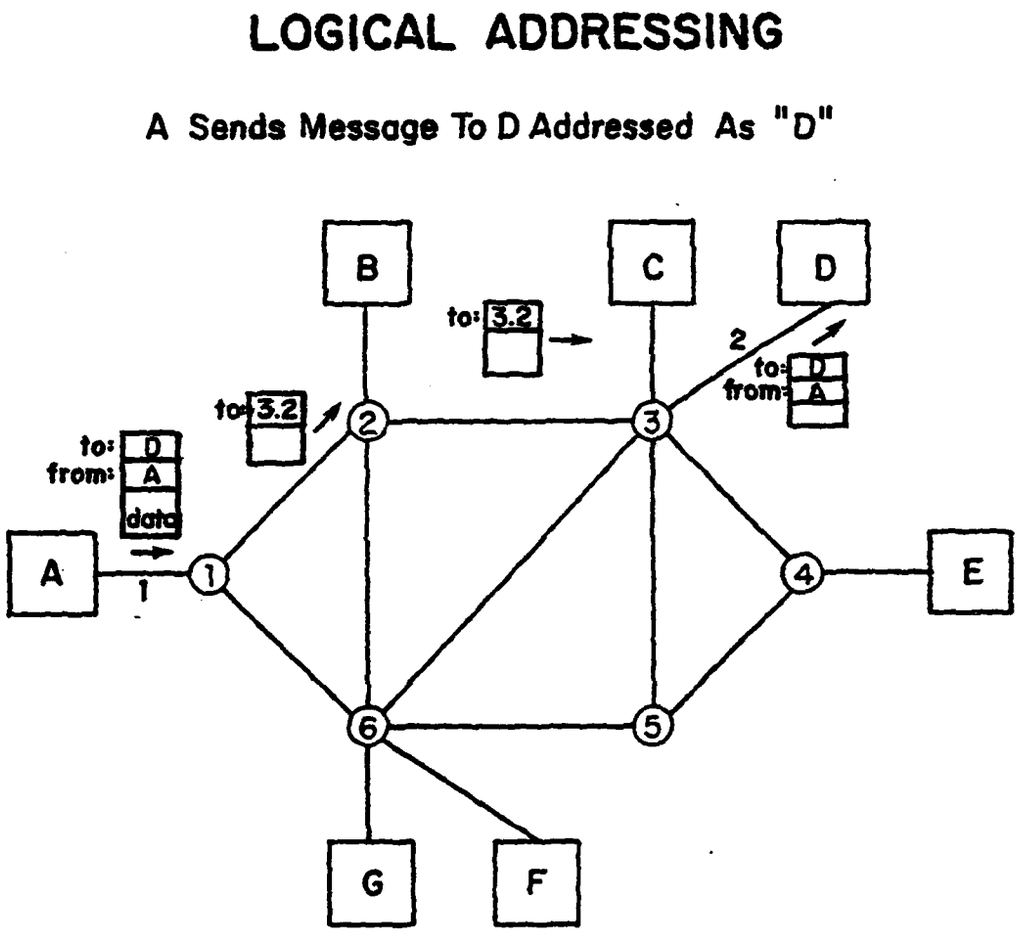


Figure 5-1b Message Addressing Modes -- Terminology

MULTIPLE HOMING

Subscribers D,E and F Are Multiply-Homed

Possible Traffic Flow From A to D Indicated

D Has Multiple Physical Addresses, One Logical Address

G, G1, G2, G3, Have One Physical Address, Multiple Logical Addresses

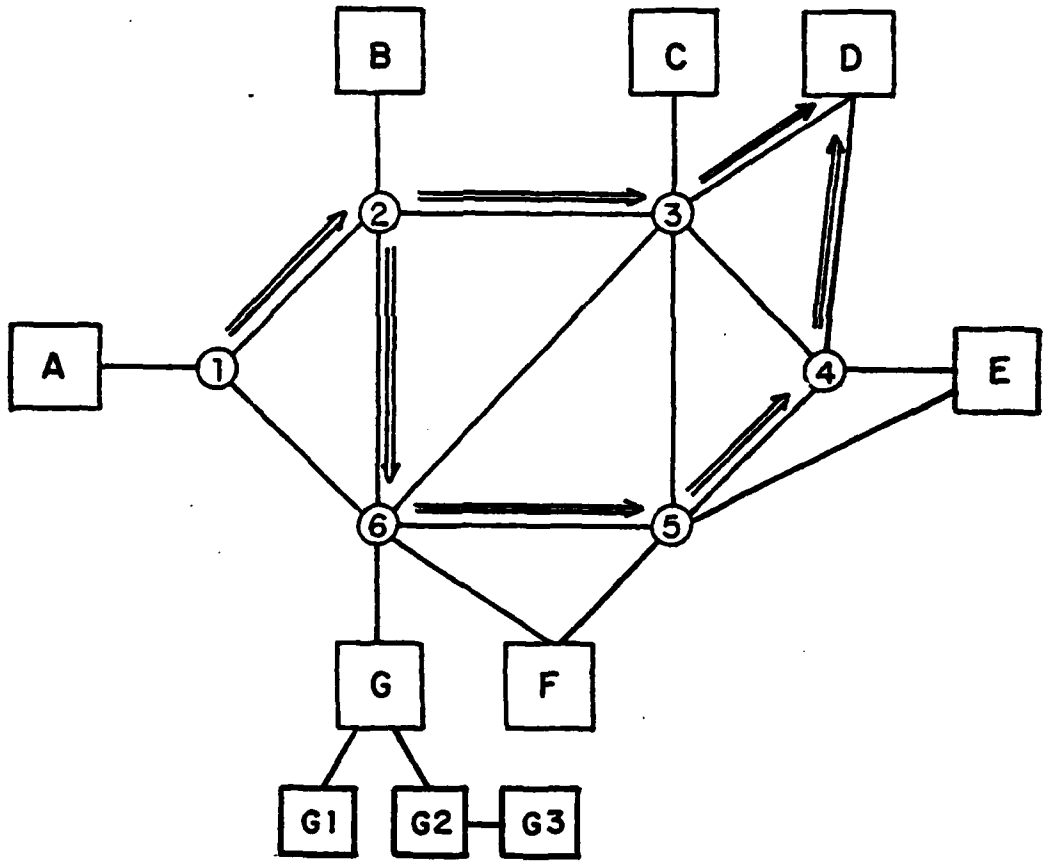


Figure 5-1c Message Addressing Modes -- Terminology

2. Broadcast addressing, in which a message is addressed to all other nodes or subscribers (see Figure 5-1d). If combined with an efficient implementation, this can reduce network traffic significantly compared with separately addressed messages.

3. Group addressing (Figure 5-1e) and multi-destination addressing (Figure 5-1f), in which a message carries the name of a list of addresses, or the list itself. When implemented with an appropriate delivery strategy, this also improves performance. It also facilitates electronic mail, conferencing, and similar applications.

This section describes the implementation considerations affecting the design of the three methods described above. The treatment we present is general and serves as a foundation for specific application to the ARPANET, which will take place in the subsequent months. Where appropriate, of course, pertinent comments are made regarding implementation in the ARPANET.

While there are many issues to be considered, we place the emphasis on efficiency and reliability, since these are the points that lead us to our design choices. If simplicity were the main criterion, an all-purpose addressing mechanism with a single implementation technique would be most desirable. However, we conclude that a different implementation is required

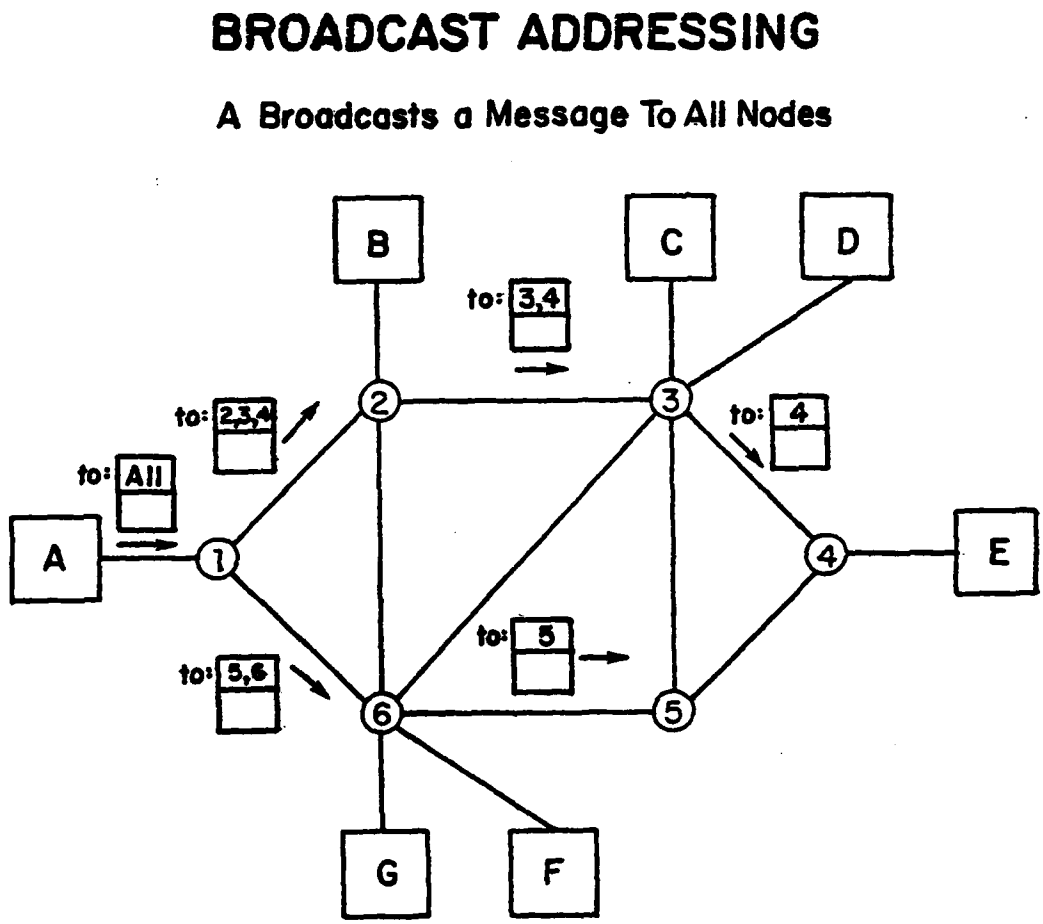


Figure 5-1d Message Addressing Modes -- Terminology

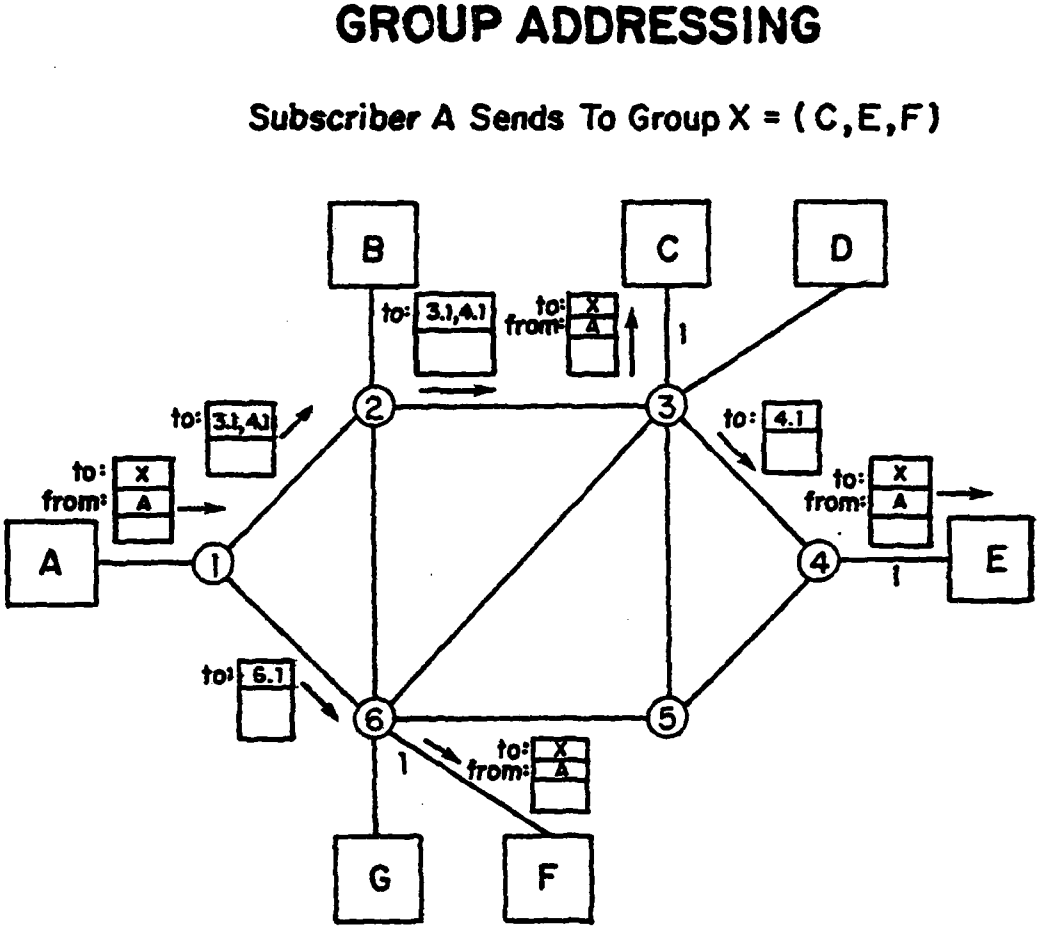


Figure 5-1e Message Addressing Modes -- Terminology

MULTI-DESTINATION ADDRESSING

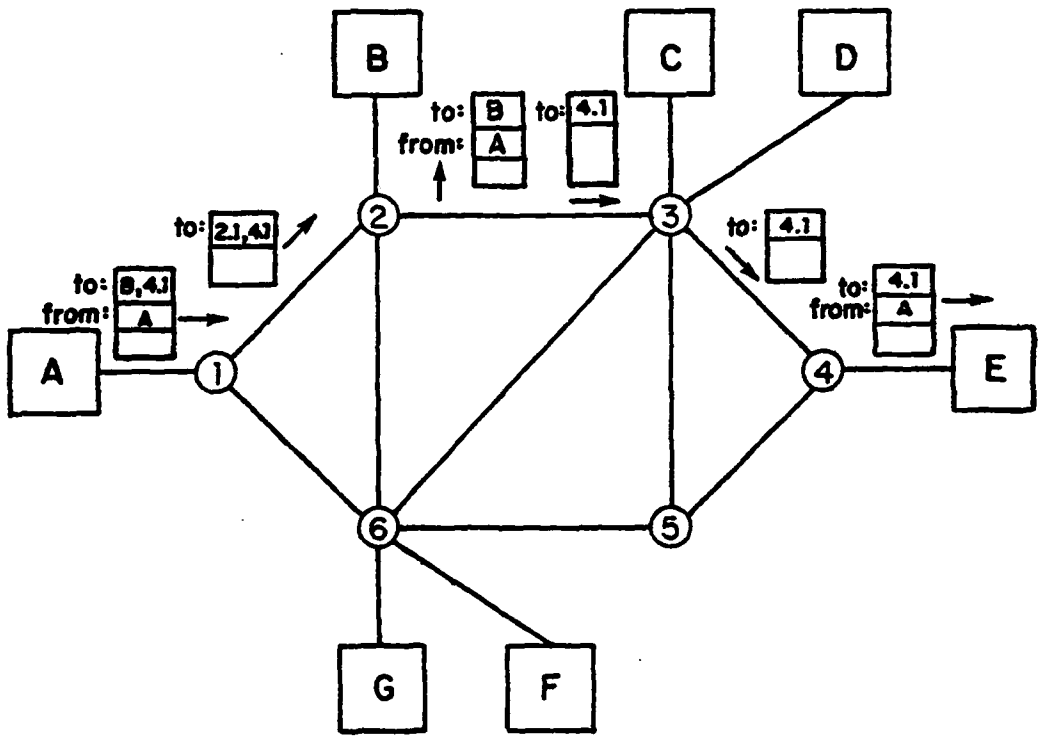


Figure 5-1f Message Addressing Modes -- Terminology

in each case to provide the best efficiency levels (to minimize the traffic flowing in the network) and to ensure adequate network reliability (to minimize loss of data due to errors or network failures).

Virtual circuits (in which messages are handled as part of connections analogous to telephone conversations) can support very efficient logical addressing mechanisms because logical addressing information needs to be sent only once per conversation. On the other hand, datagrams (in which each message is handled independently, like letters in the mail) are less efficient for logical addressing and yet can support broadcast and group addressing more readily because it is unnecessary to set up a complex set of pair-wise conversations. In fact, it may be so unwieldy to install a general multi-destination addressing method for virtual circuit service, due to the extensive control required for each circuit, that few virtual circuit networks will offer this service. Since the ARPANET supports both virtual circuits and datagrams, we will consider the full range of addressing methods outlined above.

5.1 Logical Addressing and Multiple Homing

A general logical addressing structure can translate many physical addresses into a single logical address and one physical address into many logical addresses. In a virtual circuit network the logical address is translated by the source node once per connection, permitting all messages in a given virtual circuit to flow to a particular physical address. In a datagram network, on the other hand, the addresses of messages are translated one by one and messages can flow to any physical address. The source node may perform the translation, or it may leave the logical address untranslated in the message. In the latter case, each intermediate node performs the translation (without changing the logical address in the packet header) before routing the message on the next line; this may result in slightly better route selection. On the other hand, it does not allow subscribers to refer to logical addresses as a part of a group address (as explained in Section 5.3). In this report, we will assume that the source node performs the translation to permit the delivery mechanism for group addressing proposed in Section 5.3. Logical addressing also permits multiple homing of subscribers to network ports and the use of one network port for the connection of several distinct subscribers. It is necessary for the source subscriber to identify itself by means of its logical address in the message header, as well as stipulating the

destination logical address, if a completely general mapping is desirable.

Physical addressing represents one end of the spectrum of message addressing. One difficulty with this approach is that changes in physical addresses must be announced to all subscribers, with the inevitable operational problems such changes entail. Logical addressing for every subscriber is at the other end of the spectrum. In this case, the communications network takes the responsibility for keeping track of the location of each subscriber and translating the logical addresses used by subscribers into physical addresses for internal data routing. It is also possible to design hybrid approaches which intermix physical and logical addressing.

5.1.1 Implementation Considerations

Logical addressing of subscribers requires some form of "mapping table" for translation between logical and physical addresses. These tables must be stored at one or more locations in the network and updated when changes occur. The cost of maintaining these tables depends on the size of the network and the implementation of logical addressing selected. A number of possible implementations are considered below and their costs compared. We select among several possible locations for the mapping tables: partial tables at each node, complete tables at

each node, a distributed data base of tables at the nodes, and a centralized table at one or several locations.

Physical and Logical Addressing. This is a hybrid approach which may be useful when a network designed for physical addressing only is modified to permit logical addressing. There may be a transition period during which subscribers may use either method, and there may be a requirement to keep both methods if certain subscribers do not implement logical addressing. A subscriber uses either a physical or a logical address for each message which it transmits to its node, and identifies the type of address transmitted with an indicator in the message header. Logical to physical address mapping is performed at the source node for certain subscribers. The mapping table consists of N entries giving node number and port number, where N is the number of logically addressable subscribers in the entire network. When N is relatively small the cost of the table in terms of storage required is insignificant.

One shortcoming of this hybrid approach is that it does not solve the problem of physically addressed subscribers moving from one port to another. Since many subscribers have to change ports or nodes from time to time, there may be considerable operational difficulty in keeping all subscribers informed about physical addresses (a "telephone book" may have to be published

regularly). The next two paragraphs suggest techniques for permitting all subscribers to use the logical addressing capability.

Logical Addressing - Complete Mapping. The complete mapping approach to providing a logical addressing capability for subscribers extends the ideas above to include all subscribers. The mapping table structure is the same, though its size is considerably larger since there is one entry for each of the subscribers. In the ARPANET, however, the IMPs do not have sufficient memory for such a table.

Logical Addressing - Partitioned Mapping. A different structure for the address table can be developed by taking advantage of the fact that, for routing purposes, a source node needs only the node information in the physical address, and the destination node needs only the port information. Routing is naturally partitioned into two stages; the mapping process can be partitioned in a similar fashion. The mapping table at node X can be divided into two tables, the first with K entries containing node numbers, the second with M entries containing port numbers, where K is the total number of logical addresses except those addresses of subscribers connected to node X, and M is the total number of logical addresses of subscribers connected to node X. Multiply-homed subscribers would be associated with one node at a time. The actual implementation of these two

logical tables could be a single table with an entry for every logical address containing a data field and a boolean variable to distinguish node entries from port entries.

Logical Addressing - Information Service. This approach is based on the existence of one or more information service centers on the network. The center(s) would maintain the address mapping information for the network subscribers and provide it to the nodes upon demand. Under IBM's SNA, the System Services Control Point (SSCP) provides such a function. This approach is probably most useful for large networks in which there are a few large central nodes and many smaller nodes with reduced capabilities. Each smaller node might maintain a set of address transformations used recently, together with those for its active connections and perhaps some others, to avoid access to the service centers for every message.

Relation to Multiple Homing. In many cases it is desirable to connect subscribers to more than one network node to improve reliability (and also to provide additional bandwidth over the multiple paths if they can be used simultaneously). Several connections to the same node can also be used. Any of the logical addressing techniques can be used to support multiple homing, provided that multiple entries are present in the address translation table. There are three approaches to routing messages over multiple access lines. The simplest approach is to

use only one at a time. For datagram networks it is possible to route each message to the "best" access line, e.g., the one which minimizes delay. For virtual circuit networks an alternative approach is to route entire virtual calls to one access line or the other, independently selecting the access line for each virtual circuit.

5.1.2 Efficiency Considerations

For a virtual circuit network, logical addressing can be implemented by exchanging the appropriate mapping information between the source and destination nodes as part of the connection setup procedure. The result of this exchange is that the source and destination nodes each remember the physical address and logical address of the subscriber at the other end. They can be used without reference to the address mapping table for the duration of the logical connection; this is an efficiency advantage not shared by datagram networks. Specifically, in a virtual circuit net the packets flowing in the network can be addressed with the physical address of the destination subscriber only, and the message header for the destination subscriber can be constructed at the destination node. This message header must contain the logical address of both source and destination subscribers. In a datagram net, the packet header must contain <u>both</u> the logical address information for the subscribers and the physical address information for network routing. Since the main

address translation table is referenced only at call setup time in the virtual circuit case, it may be practical to store the logical to physical mapping table on secondary storage if available. Thus it appears that virtual circuits, once established, are more efficient for logical addressing than datagrams.

With respect to maintaining the address mapping table, the alternatives are central vs. distributed and automatic (adaptive) vs. manual updating. A distributed adaptive approach, similar in concept to the ARPANET routing algorithm, is attractive. In this method, each node is responsible for the subscribers connected to it. When the set of subscribers connected to it changes, it attempts to pass this information (automatically) to the other switches in the network.

5.1.3 Reliability Considerations

The key difference between the network processing for multiply-homed subscribers and for singly-homed subscribers is that procedures must be defined for switching logical connections from one line to another in case of a failure. Four cases must be distinguished. The dual-homed subscriber may be the source or destination. In each case, the failure that necessitates switchover may occur in the node or on the access line. These four cases are explained below, where we define how failure is

detected and what action is taken to re-establish the existing logical connections via the alternate line.

Case 1 - Source node fails

The source subscriber can learn that switchover is required by observing the flow of actual or artificial traffic exchanged over its access line. It may have imperfect information concerning the disposition of the messages it has sent into the network. Subscriber level protocol should have sufficient error-control capability so that missing or duplicate messages can be detected.

Case 2 - Source access line fails

This case could be treated exactly like Case 1; however, since state information on all interrupted logical connections is still available at the source node, a better plan of action is possible. A special control mechanism can be added to the network to permit the source node to forward all of its state information on logical channels associated with the source subscriber to another node to which the subscriber is connected. The subscriber could then continue merely by retransmitting any message that was unacknowledged when the access circuit failed. Though subscriber level protocol must be prepared to deal with Case 1, it may be more efficient to deal with Case 2 at the network level, making it invisible to the subscriber.

Case 3 - Destination node fails or is inaccessible

The source node will learn that the destination node is unavailable via the network routing information. The source node can decide to use the alternate address for the destination subscriber. If copies of these messages are kept at the source node, either the source node or the source subscriber can initiate retransmission. The subscriber protocol must then be prepared to handle duplicates.

Case 4 - Destination access line fails

The source subscriber can learn of the problem via a "Destination Down" message in response to one of its data messages. The source node can decide to use the backup destination access line; connection resynchronization is similar to Case 3 except that additional state information about each logical channel is available at the destination node. This information may be sent to the source node (or possibly to the new destination node, but that would be more complex) in order to return to the source subscriber all acknowledgements queued at the destination node. Message duplication is still a possibility and either the source node itself or the source subscriber might actually effect the retransmission.

5.1.4 Datagram Considerations

Routing each message independently to one access line or the other satisfies the reliability objective and is attractive in terms of the goal of providing flexible allocation of access line bandwidth. However, there are costs associated with the more complex routing and message processing required.

Routing. Each message is independently routed to one of the access lines connecting the destination subscriber to the network. The source subscriber or the source node can direct each message to its destination; however, in neither case does the source entity have information about the present or future traffic over the access lines to the destination subscriber. To eliminate these problems the network routing algorithm may be designed to incorporate routing information concerning multiply-homed subscribers so that each node knows its best route to each multiply-homed subscriber. In other words, if a subscriber has more than one access line, and if any message can flow over any access line, then the access line selection can be treated as a routing problem rather than an addressing problem. The routing process must deal with choosing routes to nodes with more than one line, so it can be augmented to deal with multiply-homed subscribers as well. Note that the topology tables required by the SPF algorithm might be greatly enlarged if this technique is used.

Message Processing. By message processing we mean error control, flow control and sequencing. Dynamically assigning datagrams to access lines requires dual-homed subscribers to do message processing themselves. Any attempt to use multiple logical channels for a single logical data stream requires the destination subscriber to be involved in reordering and related functions. An error control mechanism is required to handle both missing and duplicate messages. Reordering at the destination subscriber is required if either the destination subscriber or the source subscriber is multiply-homed; sequence numbers can be attached to messages by the source subscriber to allow the messages to be identified and reordered by the destination subscriber.

Switchover Management. Detection of a failure and switchover are as described in Section 5.1.3. Since an alternate logical connection from the source to the destination already exists, the source subscriber needs to retransmit only those messages whose disposition is unknown at the time of the failure.

5.1.5 Virtual Circuit Considerations

Associating all of the messages of a virtual circuit or "conversation" with a single access circuit is a compromise which not only provides high reliability and makes relatively efficient use of existing access circuit bandwidth, but also is well suited to the logical addressing schemes described in Section 5.1.1.

Routing. If a multiply-homed subscriber is connected to different nodes, the source node establishes a logical channel for the entire conversation to a destination node selected from among alternatives in its logical-to-physical address mapping table, based on current routing data. If a subscriber is multiply-homed to a single node, only a single entry exists in the mapping table at the source, and access circuit selection occurs at the destination node. The destination node records multiple port numbers for each of its multiply-homed subscribers in its address mapping table and selects one when the "call request" message associated with the conversation is received. The structure of the address mapping table must permit multiple nodes and ports for a given address in order to implement this scheme.

Message Processing and Switchover. As in Section 5.1.3, no message processing functions are required of the subscribers. The method of detecting that logical channels need to be switched from one access line to another and the procedure for effecting the resynchronization are also identical to the methods and procedures described above.

5.2 Broadcast Addressing

Broadcast addressing means the capability for one node to send a message to all other nodes by marking it with the address

"ALL" rather than by sending separate messages to each node. This topic has not yet received much attention. Dalal's 1977 dissertation, "Broadcast Protocols in Packet-Switched Computer Networks" [5-1], discusses the design and analysis of broadcast routing algorithms for use in packet-switched computer networks. Five alternatives are considered in terms of qualitative implementation and quantitative performance. The SPF algorithm, as well as many other internal network algorithms and subscriber applications, requires an identical data base at all nodes. For instance, the logical addressing algorithms discussed above assumed an identical address translation table at each node. Although broadcast addressing can be used for the propagation of routing information throughout the network to all nodes, for a number of reasons we have discarded this approach, as explained in Section 4. In general, though, broadcast techniques can be used to maintain a common, distributed data base, so we will sometimes refer to the broadcast messages as "updates" in the discussion below.

5.2.1 Implementation Considerations

There are two general types of approaches to the problem of transmitting a message to all possible addresses: to route it once to each node (termed "broadcasting" here), or to send a copy over each network line (termed "flooding"). Flooding may be simpler to implement, but the nodes receive multiple copies of the message.

Broadcasting is a system in which the source explicitly addresses the message to all nodes, by labelling one or more copies with the appropriate addresses for each of its output lines, and routing them to each node along the best path (see Figure 5-2a). Such a scheme can require as little as N-1 packet hops for the broadcast, which is optimal (N is the number of nodes in the network). Each broadcast address is represented as an N-bit vector, each bit indicating whether the message should be sent to the corresponding node. The N bits are needed to indicate which nodes have received the message so far and which nodes have not; bits are turned off as the message flows through the network. The source of the message sets the address of the message transmitted on each of its lines to have bits corresponding to those nodes for which that line is the best route. Other nodes receiving such messages turn off their bit and then perform the following operation on the resulting address: for each adjacent node, take the logical AND of the received address and the bit vector of nodes for which the adjacent node is the best route. If non zero, send a message with the resulting ANDed address to that adjacent node. Broadcasting is the basis for sending a message to multiple destinations, as described in Section 5.3. However, in the special case of addressing all nodes, the next method may be preferable.

BROADCASTING

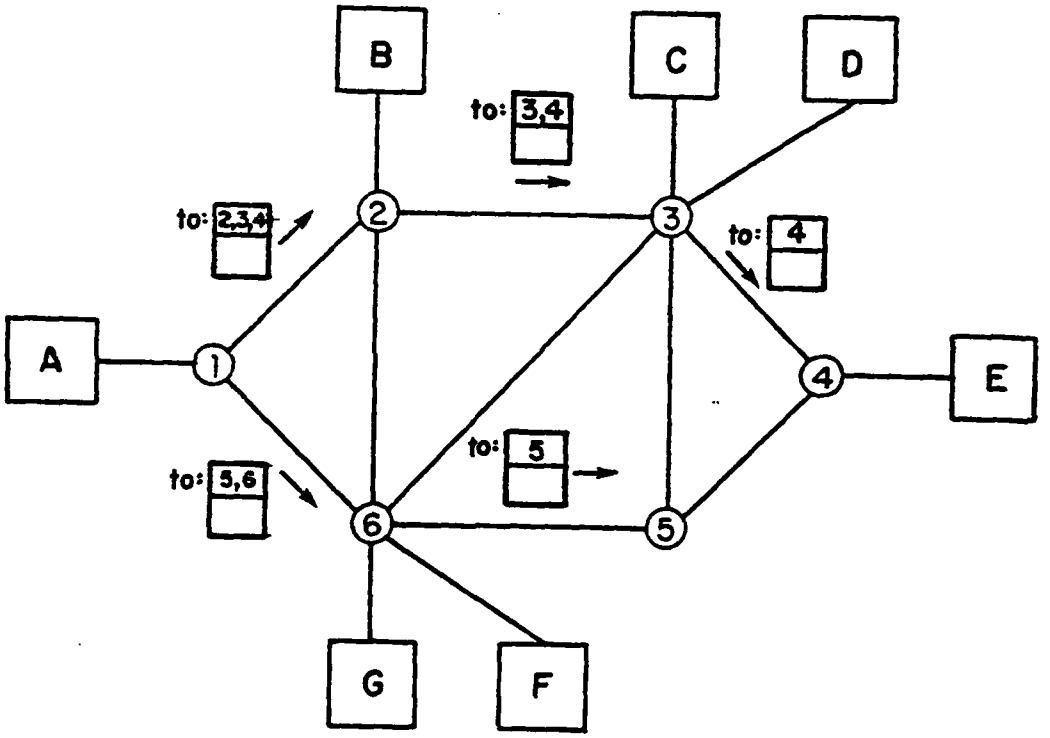


Figure 5-2a Broadcasting and Flooding

Flooding, as discussed in Section 4, is a method in which each node sends each "new" update on all its lines except the line on which the update was received (see Figure 5-2b). A new update is one the node has not seen before. This requires L-N+1 packet hops (where L is the number of lines in the net, counting each direction separately), since an update will flow on all lines except "backwards" on the N-1 lines of the broadcast tree from the source. If we define L=cN, where c=average node connectivity, then this number of updates is cN- (N-1) = (c-1)N +1.

FLOODING

Sequence Of Message Flow Indicated by Numbers
(1 Before 2 , Before 2', Before 3 , Before 3', Before 4)

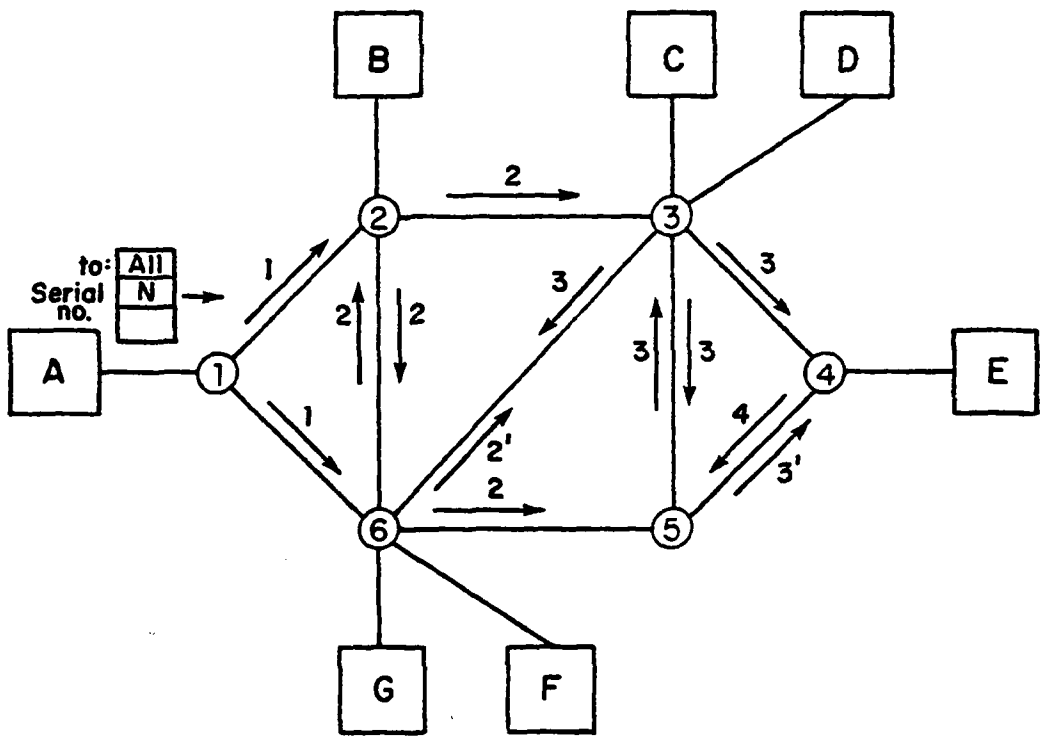


Figure 5-2b Broadcasting and Flooding

5.3 Group Addressing and Multi-Destination Addressing

For reasons of convenience and efficiency, it is desirable to provide a facility for addressing messages with the name of a group of addresses (logical and physical addresses, singly-homed and multiply-homed subscribers). Such a group may correspond to an on-going conference call or distributed working group of some kind, or it may be a simple distribution list for certain messages. In addition to such pre-established group addresses, it may also be useful to provide a general capability for addressing messages to a list of subscribers. This multi-destination addressing can cut down on network traffic and subscriber overhead by substituting a single transmission for several separately-addressed messages (see Figure 5-3).

5.3.1 Implementation Considerations

In a virtual circuit net, group addressing and multi-destination addressing are unwieldy -- both inefficient and difficult to control. The two basic alternatives are to set up (a)*(a) virtual circuits when a addresses are present in the group, or to modify the packet header to permit multiple message numbers, acknowledgments and allocations to flow over the same multi-destination virtual circuit. Both methods appear to be so complex that it is difficult to justify their implementation. For a datagram with many addresses the problem is simply to route the datagram efficiently to the destinations.

GROUP ADDRESSING AND MULTI-DESTINATION ADDRESSING

Subscriber A Sends a Message to C,E and F

4 Packet-Hops Required (Instead of 6 Packet-Hops For Separately-Addressed Messages)

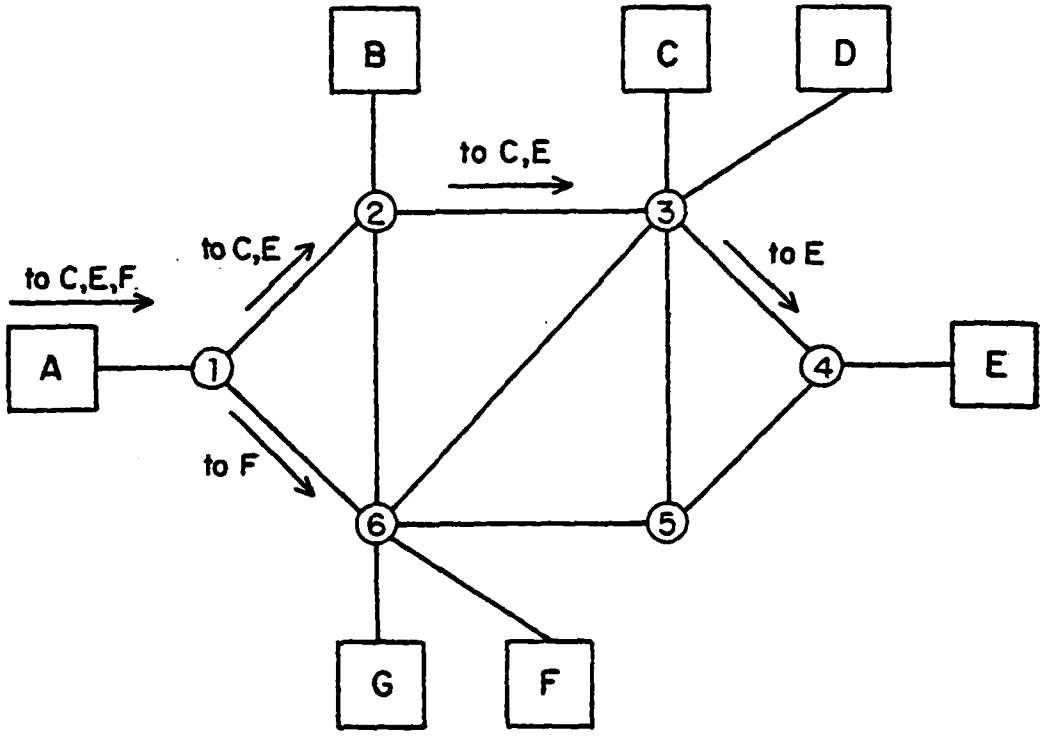


Figure 5-3 Group Addressing and Multi-Destination Addressing

The issues of formatting packets and messages with logical addresses and multi-destination addresses deserve some consideration. Group addressing is simpler to implement in the network since it requires a relatively small change to the subscriber software--the group address replaces the usual physical or logical address. On the other hand, multi-destination addressing is more flexible and useful to the subscribers but requires a fairly major change in the subscriber-to-network format, since a new variable-length address format is needed. Careful attention must also be given to the interaction between logical addressing and group addressing, since group addressing in general should permit reference to logical as well as physical addresses. As an example of this interaction, if a group address refers to several logical addresses as well as physical addresses, then the translation of logical to physical addresses must take place at the source node.

5.3.2 Efficiency Considerations

In the First Semiannual Technical Report, we showed that the use of multi-addressed packets can result in a significant packet-hop savings over the scheme of sending a separate packet to each destination, even if multi-addressed packets are routed according to the same algorithm as singly-addressed packets. Although multiple addressing yields a significant improvement over sending a separate packet to each destination, it may not

result in the minimum number of packet-hops. To get the minimum number of packet-hops, it is necessary to route the multi-addressed packets along a minimum spanning tree containing the source and destinations. While this would, by definition, result in the least number of packet-hops, it would not necessarily contain the shortest path from the source to each destination. This can be seen by considering the 8-node network shown in Fig. 5-4. The solid lines represent links which are on the shortest-path tree of node 1. The dotted lines represent links which are not on the shortest-path tree. The arrows show how a multi-addressed packet from node 1 to nodes 7 and 8 reaches its destination under (I) ordinary routing, and (II) minimum spanning tree routing. We see that the shortest path from 1 to 7 diverges at node 2 from the shortest path from 1 to 8. Therefore, if ordinary routing is used, the packet must be duplicated at node 2, for a total of 7 packet-hops. On the other hand, if the packet is routed along the minimum spanning tree which contains nodes 1, 7, and 8, it need never be duplicated, and 5 packet-hops is all that are required. However, when routed along the minimal spanning tree, the packet must travel the link from 7 to 8, which is not on node 1's shortest path tree. That is, the packet does not travel the shortest distance from node 1 to node 7.

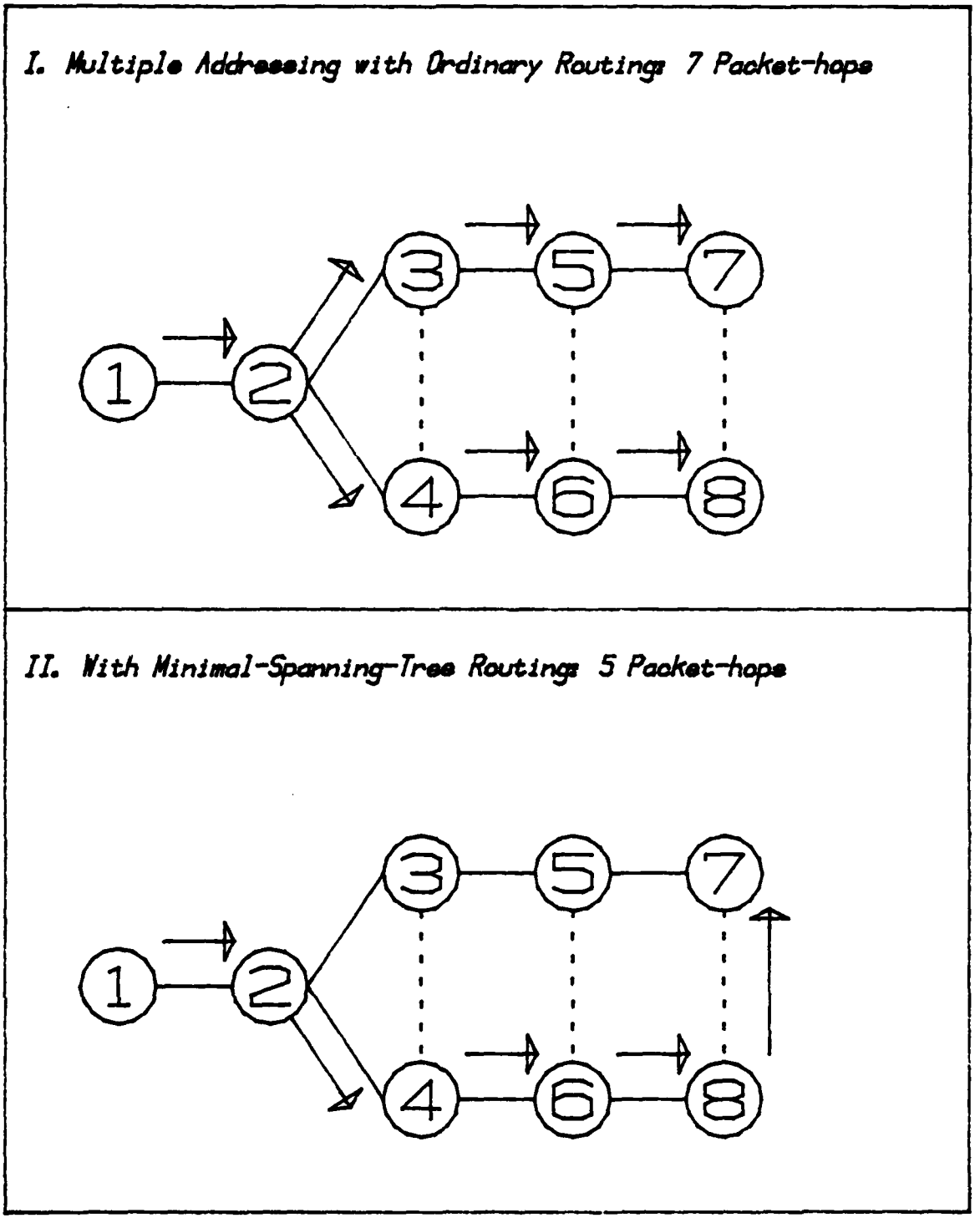


Figure 5-4 Multi-Address Routing --
Shortest Path vs. Minimum Spanning Tree

In order to determine how great a savings can be obtained by routing along a minimum spanning tree relative to routing via the shortest path tree, a RATFOR program to compute the spanning tree has been implemented on TENEX. The task of the program is the following: given a network, and a subset of nodes (call them the addressable nodes), create the tree which spans the addressable nodes in the least number of hops. Note that this tree is not necessarily included in a minimum spanning tree for the network as a whole.

The algorithm used to create the spanning tree is the following:

1. With all arc-lengths equal to 1, create the SPF tree for each addressable node.

2. Reduce these SPF trees by collapsing all chains of non-addressable nodes into single arcs. The length of one of these new arcs is equal to the sum of the lengths of all the arcs that make it up.

3. Form the base network by taking the union of all these reduced SPF trees (i.e., the base network is the network containing all and only the addressable nodes, as well as all and only the arcs which appear in the reduced SPF trees). Note that the base network contains the shortest path between each pair of addressable nodes.

4. Remove the shortest arc from the base network.

5. If placing that arc in the minimal spanning tree would result in a circuit, go to 4.

6. Place the arc in the minimal spanning tree.

7. If the spanning tree is complete, STOP.

8. If the arc is really a chain of non-addressable nodes, then
   a. Make the nodes in the chain addressable.
   b. Replace the arc in the spanning tree with the chain of newly addressable nodes.
   c. Create the SPF tree for each of the newly addressable nodes, as in (1) above.
   d. Reduce each of these SPF trees as in (2) above.
   e. Create a new base network by forming the union of the current base network with the new reduced SPF trees, as in (3) above.

9. Go to 4.

This algorithm is expensive in terms of computation. In order to tell whether putting a particular arc in the spanning tree would result in a circuit, an NxN boolean reachability matrix is maintained. Whenever an arc is added to the spanning tree, the endpoints of the arc are marked reachable from each

other, and everything which is already reachable from one of the endpoints is marked reachable from the other. As currently implemented, therefore, the task of maintaining the reachability matrix is quadratic in N, and hence can dominate the computation for large networks. (It does, in fact, dominate the computation for the ARPANET.)

In order to be able to select the shortest arc from the base network, it is necessary to sort the arcs each time a new base network is created. The time required for this sort increases more than linearly with the number of arcs in the base network. The number of times a new base network has to be created is inversely related to the connectivity of the whole network. The task of reducing an SPF tree is inversely related to the connectivity of the network and directly related to the number of nodes in the network.

Following are some results for different networks. The network NcC is an n-node network of uniform connectivity c; RING27 is a 27-node ring network. Each figure is the average for 500 runs in which sources and destinations were chosen at random. The results show that the expected <u>percentage</u> savings offered by using the minimum spanning tree are small. Neverless, the absolute savings might be great, and we therefore discuss below some implications of this technique.

Network	# of Destinations	# of Packet-Hops	
		Ordinary routing with multi-addressed packets	Minimum Spanning Tree
N8C3	1	1.81	1.81
	2	3.08	2.81
	3	4.24	3.71
	4	5.08	4.40
	5	5.82	5.17
	6	6.47	6.00
N8C4	1	1.44	1.44
	2	2.66	2.42
	3	3.79	3.19
	4	4.68	4.00
	5	5.52	5.00
	6	6.31	6.00
N8C5	1	1.31	1.31
	2	2.42	2.14
	3	3.43	3.02
	4	4.40	4.00
	5	5.30	5.00
	6	6.13	6.00
N22C3	1	3.18	3.18
	2	5.70	5.07
	3	7.76	6.68
	4	9.35	7.73
	5	10.56	8.72
	6	11.92	9.70
RING27	1	6.88	6.88
	2	11.72	10.91
	3	14.71	13.30
	4	17.21	15.26
	5	19.09	16.90
	6	20.35	17.72
ARPANET topology of 6 months ago	1	5.58	5.58
	2	9.66	8.82
	3	13.07	11.46
	4	15.98	13.66
	5	18.58	15.80
	6	20.58	17.27
Recent ARPANET	1	5.65	5.65
	2	9.92	9.07

Topology	3	13.20	11.53
	4	16.28	13.88
	5	18.89	15.89
	6	20.96	17.54

A major difference between the minimum spanning tree scheme and a multi-addressing scheme which uses the network's ordinary routing has to do with the packet forwarding procedures. If ordinary routing is used, a multiply addressed packet would always travel along the branches of a shortest path tree. Therefore, each node could decide on its own where to forward such a packet, by using its standard routing tables. However, the minimum spanning tree for a given group of nodes may not be coincident with the shortest path tree of any node. Therefore, if multi-destination routing is to be done via a minimum spanning tree, the entire tree will have to be pre-specified by the source node, and a coded representation of the tree will have to be carried in the packet. Intermediate nodes would route the packet according to the pre-specified tree, rather than according to the standard routing tables. Although a single path between two points can be represented in a relatively concise manner, representing an entire tree is more difficult, and can be expected to result in a significant amount of overhead in each packet. This large amount of overhead must be taken into account in judging the feasibility and desirability of implementing a minimum spanning tree routing scheme.

Apart from the inefficiency arising from this additional overhead, there is a further inefficiency involved in performing the computation itself. Since the minimum spanning tree

calculation takes much longer than an ordinary SPF calculation, it is not practical to perform the calculation for every multi-destination packet which enters the net. However, if it is known that a conference among n particular nodes is about to begin, and that it will last for a length of time, then the minimum spanning tree can be computed once and used for the entire duration of the conference. It would only have to be re-computed when a line in the tree fails. (Of course, some protocol would have to be developed to handle the packets which are already in transmission when the failure occurs.) This implies that minimum-spanning-tree routing is less suitable for communication which takes place in pure datagram mode than for more virtual circuit oriented communication. On the other hand, using multi-destination addressing with virtual circuit traffic gives rise to the serious control and protocol problems alluded to in Sec. 5.3.1. These problems would have to be resolved before a minimal-spanning-tree routing scheme could be implemented.

5.3.3 Reliability Considerations

Since we have assumed that group addressing and multi-destination addressing are not practical for virtual circuit service, and should be implemented only for datagrams, the subscribers using these services must take responsibility for providing reliable transmission for the end users. A host-level

protocol is necessary between the sending subscriber and each receiver to ensure an error-free, sequenced flow of messages between each pair of subscribers.

5.4 Conclusions

The enhanced message addressing modes discussed above have several important advantages over physical addressing. Logical addressing provides for considerable operational flexibility and reliability. The use of multi-destination and group addressing has been shown to lead to significant reduction in network traffic, even for the case of relatively few destinations per message. One of the important conclusions from this work is that while virtual circuit networks have some efficiency advantages over datagram networks for logical addressing, datagram networks facilitate the use of broadcast addressing and multi-destination addressing. These important points of comparison have not yet been fully considered by the network design community.

REFERENCES

5-1 Y.K. Dalal, "Broadcast Protocols in Packet-Switched Computer Networks," Ph.D. Dissertation, Stanford University, Digital Systems Laboratory Technical Report No. 128, April 1977.

6. INTERACTIONS BETWEEN ROUTING AND CONGESTION CONTROL

This section presents only our preliminary thinking on the difficult subject of interactions between routing and congestion control. During the coming months, we hope to pursue further the questions discussed below.

6.1 Introduction

This section explores some of the inter-relationships between routing and congestion control in the ARPANET, and in computer networks in general. Routing can be defined as the process of picking the best paths for traffic flow in the network. Congestion control can be defined as avoiding conditions which lead to traffic being refused at one IMP and backing up throughout the network.

One important starting point in any discussion about routing and congestion control is the set of assumptions one makes about the computer network. We assume that the communications links in the network may run at different speeds and the IMPs may be of different sizes, both in terms of memory capacity and processing power. We also assume that this heterogeneity implies that some IMPs may be (temporarily or permanently) underconfigured, that is, they may not possess enough memory or CPU capability to process all of the traffic that may flow in on their circuits during a peak traffic period.

This assumption is realistic for several reasons. First of all, it may be uneconomical to configure the switching IMPs to be able to process a continuous stream of very short packets flowing in on all of the circuits simultaneously. Second, the IMP may suffer a partial failure of its processing or memory power (especially if it is a multiprocessor) and run in a degraded configuration for some period of time. Third, the network may change from time to time as higher speed circuits are introduced or circuits are removed and added. Finally, network growth, in traffic or in size, may result in unanticipated overloads. These changes may not coincide exactly with retrofits to the computer equipment at the IMPs. There may be times when an IMP will have a very high speed line on one side and a slower speed line on the other, or a computer configuration intended for a somewhat slower set of circuits.

Several distinctions can be made between routing and congestion control. Routing is inherently a macroscopic process; that is, it should deal with average trends in traffic flow, since it is not appropriate to change the routing strategy used in a network to deal with momentary (second-by-second) traffic fluctuations. For this reason, the routing procedure is best performed by some form of global algorithm which has information about conditions throughout the entire network. Congestion control, on the other hand, is inherently a microscopic process,

since it should deal with the variance in traffic flows, often in a small section of the network, in addition to the average or expected traffic flow. Congestion is caused when the computer processing power, memory storage, or line bandwidth available at a particular IMP is not adequate to deal with the traffic flow entering that IMP. Thus, congestion control is inherently local in nature, since it must deal with moment-to-moment fluctuations which may be visible only at the local level. This is not to say, of course, that a more global algorithm should not be constructed in addition to this local control (for instance, to detect a global traffic overload).

Routing and congestion control constitute a natural system of checks and balances within the network, such as must be found in any successful computer system. On the one hand, routing is designed to provide the best possible level of service by the subnetwork to the subscribers (e.g., minimum delay). On the other hand, congestion control is designed to protect the subnetwork from traffic overloads and congestion leading to poor performance, which might be brought on by the behavior of the subscribers (as well as by other factors). Routing operates continuously to identify which path(s) traffic should use; congestion control operates during overload to identify which subset of the traffic should use those paths. We assume that there is also a flow control mechanism to regulate source-destination flows to the rate of the slower end.

6.2 Routing Algorithms

One basic premise of network operations is that it is necessary to make the most efficient possible use of the available bandwidth of the circuits. Routing algorithms have therefore been designed to avoid congestion, i.e., cases in which the input traffic mix and the output trunk rate at an IMP are not matched. There are two cases to consider: a momentary mismatch and a prolonged inequality. The factors involved in the relationship between input and output rates are, on the one hand, the arrival and departure rate of packets, and, on the other, the bit rate of the circuits, their error rates and failure characteristics. The situation with a single input line and a single output line is simple: either the output line is underutilized or it is oversubscribed. In the second case, the congestion control mechanism must prevent the whole network from filling up with packets for the oversubscribed line. An IMP with multiple output lines must have additional mechanisms to ensure that each output line has the highest possible effective throughput (i.e., that the lines are not all reduced to the effective rate of the slowest). Figure 6-1 shows the situation in a very simple case.

The input traffic shown is in the ratio of 2 packets for IMP 3 for every 1 for IMP 2. If we assume that all the trunks have the same capacity and a zero error rate, this is just an example

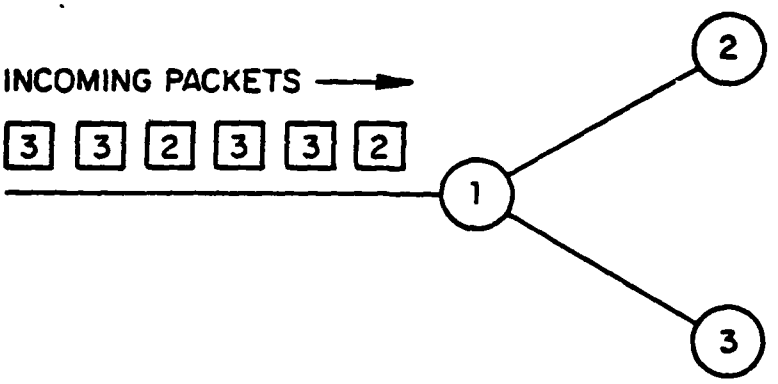


Figure 6-1 A Switch with Multiple Output Trunks

of underutilized output lines -- utilized at 2/3 and 1/3 the rate of the input line. However, if we assume that the other two lines both have 1/2 the capacity of the input line (or effective capacity, due to the presence of other traffic, line errors, etc.), then the line from IMP 1 to IMP 3 will be oversubscribed, while the line from IMP 1 to IMP 2 will run partially empty. This inefficiency is due to the fact that the input traffic mix does not correspond to the output line capacities. Clearly, the desired solution in this case is to arrange for the traffic to be equally balanced between destinations 2 and 3. This may not always be possible, due to input fluctuations. Traffic flowing between IMPs should be selected in such a way that there are no

artificial bottlenecks or underutilized lines. The traffic must be "metered" so that it is matched closely to the capacity of the network trunks.

Whenever the traffic arriving at an IMP is such that the IMP is not able to assign it to output lines without some queueing, then a certain amount of buffering is necessary. In a short time the IMP's buffers will fill up and any further input traffic must be rejected. Thereafter this IMP will accept the first packet which arrives after a buffer has been emptied. If we assume that several traffic streams are competing for the buffers, and that all traffic is backed up and therefore is competing equally for line bandwidth, then the packet that is accepted is chosen at random from the incoming traffic streams. The effect is that traffic flows at the rate of the slowest output line (each individual flow is proportional to the fraction of input traffic it represents times the speed of the slowest output line). One cannot circumvent this problem by dedicating buffers to each output line, because the problem exists in the general case for topologies in which the blocked traffic flows over an additional line before the block occurs, as shown in Figure 6-2. Even an IMP with only one output line must make the decision to accept the other traffic but hold off the blocked traffic; the problem is to quench according to destination IMP, not according to output line.

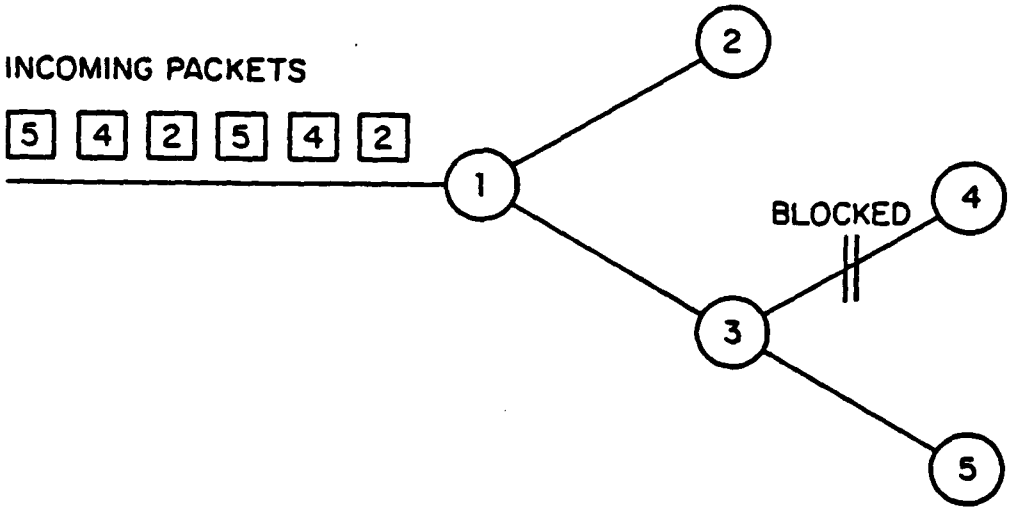


Figure 6-2 Blocked and Unblocked Traffic on the Same Trunk

Note that if the line between IMPs 3 and 4 in Figure 6-2 goes down, there is a buffering problem analogous to that caused by packet congestion. At the moment that the line fails, there will generally be some number of packets queued at IMP 3 waiting to be sent. In the time immediately following the failure, more packets will arrive and be routed on to the line. The IMP cannot declare the trunk to be "dead" (unusable for packet traffic) until some finite amount of time has passed, say 3 seconds, to differentiate a true failure from a momentary outage. During those 3 seconds the IMP may fill up with traffic for the failed trunk, to the point that further packets cannot be accepted. At this stage all other output lines at that IMP are completely

blocked. That is, for some period of time all lines are reduced to the effective rate of the slowest, which in this particular case is zero.

In order to be able to operate the ARPANET at high traffic levels, it is necessary to install controls which will guarantee the following:

1. On a long-term basis (minutes): overall network traffic is within overall network line capacities -- excess traffic is kept outside the network;
2. On a medium-term basis (many seconds): traffic on each line is metered so that no network line is oversubscribed -- excess traffic is kept outside the IMP;
3. On a short-term basis (few seconds): problems of suboptimal line use are left to the buffering mechanism -- excess traffic is kept inside the IMP.

The reason for the time scales above is the size of the storage at each IMP, the magnitude of the traffic flows in the network, and network delays. One method for providing both the medium and long term solutions is a routing algorithm which explicitly allocates line bandwidth to traffic flows to given destination IMPs. Such an algorithm can be constructed to route packets so as to maximize the effective bandwidth of the network,

and, as a by-product, to allocate the use of this bandwidth among competing streams. Unfortunately, the design and implementation of such an algorithm is not possible within the scope of the present contract. For this reason, we must consider explicit congestion control techniques.

6.3 Congestion Control Algorithms

The following principles can be put forward as basic design goals for any congestion control algorithm:

1. The congestion avoidance mechanisms should be quick to take effect.
2. The control of congestion should be smooth; that is, it should be an incremental rather than a simple "on/off" procedure.
3. The congestion control procedures should require a low level of overhead in the network, preferably a constant level rather than more overhead as traffic load grows.
4. Congestion control should take effect in a manner consistent with traffic precedence classes; that is, lower priority traffic should be refused before higher priority traffic. Also, within a particular precedence class, all traffic should be treated fairly.

These are goals which any congestion control procedure shares in common with the routing algorithm for the network.

6.3.1 Storage Allocation Methods

We next discuss the limitations of the techniques outlined above, and the need for augmenting them with local, packet-by-packet control techniques. The remaining problems can be stated as follows:

1. There is never "enough" buffering to handle extreme cases of line errors or failures, or traffic fluctuations;
2. There is no such thing as "perfect" routing, due to inaccuracies in measurement, finite time delays, and unpredictable events such as traffic surges and line errors.

One way to view the congestion control problem is that a successful congestion control strategy matches the incoming traffic to the traffic handling capacity of the network. Specifically, the following three correspondences must be made:

1. Input traffic rate must be less than output line rate.
2. Input traffic rate must be less than IMP processing rate.
3. Input traffic variance must be less than nodal storage capacity to hold fluctuating traffic.

Like many network flow control systems, congestion control has been thought of as a storage-based mechanism. (We examine this assumption critically in section 6.3.1.) Figure 6-3 shows four possibilities for the location of controlling storage for network packets, which dictates how the flow control system operates in each case:

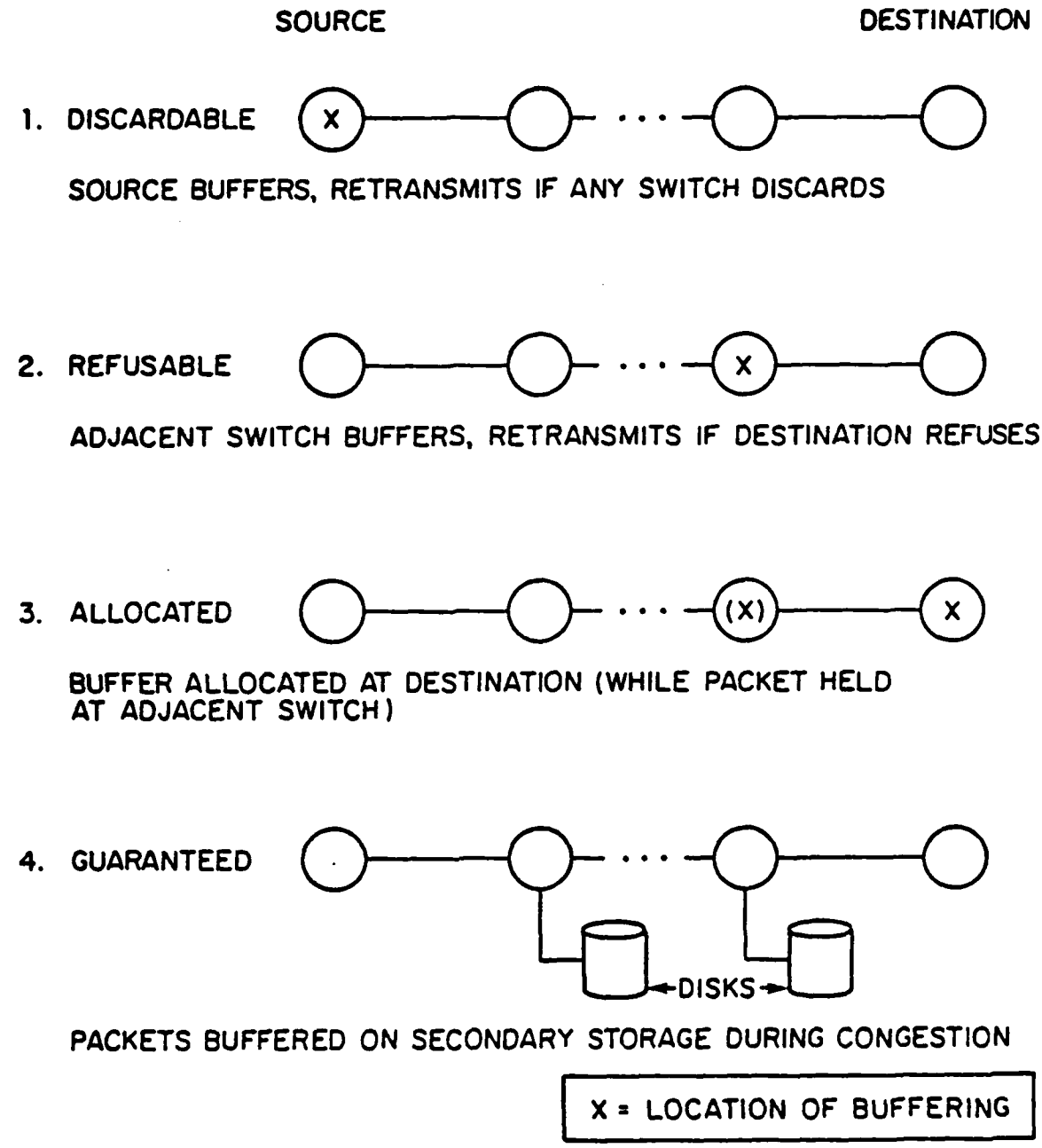


Figure 6-3 Four Possible Switch-to-Switch Flow Control Systems

1. Discardable. In this approach, the source IMP holds a copy of every packet sent into the network and this copy is used for IMP to IMP flow control. Whenever any intermediate IMP encounters a congested state (e.g., finds that it has no buffering for an incoming packet), it is able to discard the packet, since the source can retransmit a copy.

2. Refusable. This approach is similar to the first, but the copy is held at the adjacent IMP. Instead of discarding the packet, the intermediate IMP refuses it from its neighbor. The neighbor later retransmits the packet. This is the scheme in use at present, with a limit of 32 refusal/retransmission cycles; the packet is then discarded (it is not retransmitted from the source IMP, but is dropped entirely).

3. Allocated. This scheme is based on the concept of acquiring a buffer at the next IMP before the packet is sent along. Since the packet must be held at the first IMP until the next buffer is allocated, this technique tends to use more storage than the refusable method. This can be generalized to the concept of pre-allocating buffer space along the entire path.

4. Guaranteed. A different kind of solution to the problem of network congestion is to provide an amount of buffering which is essentially infinite, probably secondary storage of some kind, in order to hold packets at moments of congestion. The routing

in the network would still be responsible for holding traffic levels to the point that the disk storage would be necessary only a small fraction of the time.

What are the advantages and disadvantages of these approaches?

1. Discardable. One advantage of this technique is that it provides a way of dissipating congestion completely, removing all blocked packets from the area of congestion. This eliminates the possibility of the congestion backing up and causing a more global problem, or even a deadlock. Two specific problems are possible in this regard: direct and indirect store-and-forward lockups. A direct store-and-forward lockup is possible when two IMPs send packets to each other at a high rate. The lockup occurs when each IMP fills up all its buffers with packets for the other, and has no more storage in which to accept incoming packets. We installed a simple solution to this many years ago in the ARPANET: some buffers are permanently assigned to input and output on each trunk.

Indirect store-and-forward lockup is much less likely to occur, and much more difficult to prevent or correct. Indirect store-and-forward lockup occurs when each IMP is filled with packets destined for some other IMP in the network, but no IMPs have packets destined for adjacent IMPs. A mechanism for dealing

with this problem was designed for the ARPANET in 1971 but was never put into practice because of its complexity and expense, and the low probability of indirect store-and-forward lockup. The reason to make note of this problem is that the "discard" approach to congestion eliminates all such storage-based deadlocks since congestion does not back up.

If the discardable approach is used for congestion control, then a threshold must be defined beyond which incoming packets are discarded. The threshold is probably best designed as a combination of several factors:

1. A maximum number of packets per output line.
2. A maximum number of packets in total per IMP.
3. A maximum number of packets over an entire network path.
4. A maximum number of packets in total in the entire network.

No single one of these thresholds is sufficient, since congestion can be formed if any of the thresholds are violated, and since only one of the thresholds may be violated during a period of congestion. When viewed in this perspective the distinction between local control (e.g. discarding) and end-to-end control (e.g. complete allocation) seems somewhat artificial. Congestion control needs to be exercised at every point in the network at which congestion can take place. Some resources are local to a

particular IMP (e.g., the number of buffers that can be allocated to a particular output line), and therefore control is necessary at that point. Other resources (e.g., storage for virtual circuits) may be dedicated to a particular source or destination path, and congestion control is equally necessary at that point.

There are some important disadvantages of this scheme, primarily related to performance. If the source IMP must retransmit the packet, then it uses twice as much network trunk bandwidth as usual. Compared to the refusable method, discarding uses up trunk bandwidth on all circuits to the point of the congestion, instead of only on the congested line. As a consequence, it requires more storage as well, due to the longer delay before receiving the end-to-end ACK. Further, there is the question of retransmission timeout; the source IMP must determine when to retransmit its copy, and there is an obvious tradeoff here. Retransmitting too soon may be unnecessary, because the packet may still be in transit; retransmitting too late causes inordinately long delivery delays. Even if the source knows that the packet has encountered blocking and has been discarded (via a "discarded" reply message), it may not be desirable to retransmit right away, since the congestion may persist, and the packet should not reenter the congested area immediately. This problem can be alleviated somewhat if the IMP which discards the packet sends a small notification packet to the source, which then

chooses how long to wait before retransmission. The source can even adopt a set of rules that governs the wait time as a function of discard rate. This method has the disadvantage of requiring more overhead as congestion grows. We note again that discarding is intended to be a stop-gap solution only; routing should handle the problem of network congestion on a long-term basis, and discarding should be a rare event.

2. Refusable. The main advantage of the refusable approach over discarding packets is that the control of traffic flow is local to the problem, so that the response can be more rapid and more precise. The refusal of packets can be implicit (no ACK returned) or explicit (NAK returned), and the sending IMP can react quickly to the problem and retry or send other traffic and then retry. The problems mentioned above for the discardable method, especially the inefficiency of retransmission and the difficulty in setting the timeout value, hold true for refusable control also. However, the problems may be simpler since retransmission is local, and there is less variability in timing. On the other hand, traffic is allowed to back up through the network, which may lead directly to more congestion.

3. Allocated. This approach does not appear to have any advantage over the first two, since it is still necessary to hold a copy of the packet at one IMP until a buffer is reserved at the next IMP. While this kind of technique is useful for source to

destination protocol, it does not seem appropriate for datagram-style IMP-to-IMP protocols. Virtual circuit networks like Tymnet have used it with apparent success.

4. Guaranteed. The last approach, having a semi-infinite amount of storage in which to buffer blocked packets, would be attractive if the IMPs had disks for some other functions. Since the secondary storage would be essentially free in that case, guaranteed buffering would provide a way of eliminating all storage deadlocks just as the discardable approach does, but without any of its attendant performance drawbacks. It also leaves control at the IMP which is local to the congestion and best able to react to it, as the refusable method does. However, this is not a realistic option for the ARPANET.

6.3.2 Storage Allocation--The Assumption Questioned

We next examine the conventional assumption we have been using, that congestion control can be modeled as a storage allocation problem. Several ideas have been put forward in support of this model:

1. If all IMPs had infinite storage capacity there would be no congestion control problem.

2. All congestion sooner or later causes the storage at an IMP to fill with packets.

3. There are only two mechanisms which can be used to accomplish traffic control: reserving buffers in advance (storage at the receiver) and discarding packets which cannot be accepted (storage at the sender).

In addressing the first point, a very important technical assumption is whether the IMP is capable of accepting all the incoming packets and storing them in memory. This point is related to, but not the same as, the assumption that the IMPs have enough processing power to complete the processing for each of the incoming packets. The distinction here is between the capability of the IMP to store each packet in memory as it arrives (termed "responsiveness") and the capability of the IMP to forward each packet on some output line (termed "throughput"). The reason this distinction is important for congestion control is that an IMP with infinite responsiveness and infinite storage capability can avoid congestion control problems for an arbitrarily long period of time (traffic will not back up), even if it does not have the processing capability to be able to deal with traffic surges. On the other hand, an IMP with very limited responsiveness and memory capability will be susceptible to traffic congestion in a computer network with traffic rates that fluctuate from moment to moment even if it has enough processing capability to be able to deal with the average traffic flow. The only realistic assumption to make is that the IMP has limited

responsiveness (it may "miss interrupts", and thereby lose incoming traffic) and limited throughput (it may fall farther and farther behind in its store-and-forward function). Both factors can lead to congestion.

To summarize, an IMP needs both infinite storage capability and infinite responsiveness in order to be able to copy each of the incoming data packets into memory and prepare for the next input. Furthermore, it is not sufficient simply to store away all the incoming packets in memory, since the traffic flowing into an IMP may consist of data packets at several different priority levels as well as various kinds of control information, including routing and congestion control data. Therefore, the IMP must have enough capability to be able to deal with all of the control information flowing into the IMP as well as storing the data flowing into the IMP. If we ignore these points and assume that IMPs are endowed with infinite storage, then we can consider the computer network as an ideal message switching system (as opposed to a packet switching system) in which traffic flows from the source through intermediate IMPs, where it is buffered, until it reaches the destination. At each step of the way the entire data message can be stored. Congestion control is not a problem since each link transmission is independent. The entire data transmission is reduced to the rate of the slowest intermediate link (or to the rate of the source or the destination if either is the slowest element).

When we relax the assumptions about an ideal message switching system in order to model the network as a real packet switching system, does the conclusion that congestion control is really a storage allocation problem still hold true? The second point raised at the beginning of this section is that any of the several causes of congestion, namely, limited CPU processing power, memory capacity or line bandwidth, shows up as a queue of packets on the output line. Congestion control manifests itself as excess demand for packet storage at the IMP adjacent to the point of congestion. When a given IMP becomes congested, the adjacent IMPs find that their queues in the direction of the congested IMP begin to grow without limit. For this reason, it seems that congestion control can be viewed as a storage allocation problem. However, there are a number of reasons to believe that storage allocation is a necessary but not sufficient component of any successful congestion control procedure.

The following arguments can be used to show that more than a simple storage allocation procedure is needed for congestion control:

1. Since the IMPs have limited responsiveness (may miss incoming packets), allocating storage at the receiver does not guarantee avoiding congestion at the sender.

2. When a long queue forms in an IMP it is not possible by examining the entries of that queue to discover which sources or destinations of traffic are contributing to the congestion. All that can be determined is the set of sources and destinations which are being affected by congestion.

3. In the case that the total level of traffic being offered to the network is more than the network can carry, congestion control becomes a general resource allocation problem rather than a storage allocation problem. For instance, the network may be limited by channel bandwidth or IMP processing bandwidth; it is the limiting resource(s) which need to be allocated ("scheduled").

In summary, congestion control has often been implemented by means of a storage allocation procedure, since this is often a simple technique to implement. However, congestion control is actually a more general problem requiring a more general solution.

6.4 Effects of Routing Design Choices on Congestion Control

One of the main choices in designing a routing algorithm is that of the objective function. What is the routing algorithm designed to achieve? The optimality criterion used in the algorithm (for instance, minimizing average delay) can be seen as a congestion avoidance mechanism. Several objective functions are possible in computer networks:

1. Maximize throughput
2. Minimize delay
3. Minimize blocking probability
4. Maximize the ratio throughput/delay
5. Maximize the function x * throughput - delay (a weighted average of throughput and delay).

Any objective function which prefers low traffic levels on a given link to higher levels tends to avoid congestion on that link (but not necessarily on the network as a whole). However, simple measures, such as minimizing average delay, may not take into account high levels of congestion which might persist for short periods of time when traffic overloads are permitted. Functions which rise more sharply than linearly with delay might be an effective technique for congestion avoidance, though they would not be sufficient for congestion control.

A second major design choice in developing routing algorithms is how to insure that routing choices lead to a stable routing flow under steady-state traffic conditions. In general, the less stable a routing algorithm is, the more difficulty will be encountered in designing and implementing a successful congestion control procedure. Although this is obvious, it may be quite difficult to develop a routing algorithm which is stable enough not to introduce artificial levels of congestion into the network at times when the routing procedure leads to oscillating choices in traffic routes.

When using the discardable or allocated approach to congestion control, one of the most important decisions to be made is which traffic flows should be accepted and which should be rejected. An important component of this decision is the part of the algorithm which detects which traffic flows are causing congestion and which traffic flows are being affected by congestion. In other words, it is essential for the congestion control algorithm to identify those traffic flows which if slowed down or stopped altogether would result in an overall increase in network traffic flow or a decrease in network delay or otherwise improved network performance. This is a very difficult problem. As an example of how to determine what traffic is causing congestion, consider the relatively simple expedient of examining the packet buffers which are queued in an IMP. This might be

desirable, for instance, when it it necessary to discard a packet, or when it is necessary to send off an allocation message for a new packet. It is very difficult to decide on the basis of such local information which traffic flows are actually causing congestion. If one adopts the strategy of examining packets and identifying the most popular destination of traffic and labelling that as a "congested destination," the congestion control technique will proceed to discard packets for that destination, to allocate fewer packets for that destination, or to otherwise slow down traffic intended for that destination. However, that may be counterproductive. It may be that the traffic for that destination is merely blocked behind some other traffic which is not present in such numbers at that moment at the IMP. Likewise, any more elaborate system of evaluating buffered packets (at a small set of IMPs) on the basis of source-destination pairs or on the basis of source host or source IMP is also too myopic to be effective for congestion control. Clearly a memory-less system such as examining the buffers present in an IMP at any given time cannot take into account the previous flow of traffic. Also a purely local system cannot take into account the more global causes and effects of network congestion. Therefore, it would seem that a more complete congestion control algorithm would involve a record-keeping system at each IMP in the network which keeps track of traffic flowing between various sources and destinations, and a communications procedure which distributes

this information from each IMP to all other IMPs in the network.

6.5 Effects of Design Choices for Congestion Control on Routing

Any congestion control technique must be able to reject traffic entering a given IMP if sufficient resources do not exist for processing that traffic. Likewise, the control technique must be able to determine when the refusal of traffic is no longer necessary. One of the main points at which congestion control may interact with routing is the extent to which this traffic rejection information is made available to the routing algorithm. On the one hand, a tight coupling is possible between congestion control and routing: when a traffic overload is detected by the congestion control technique, the routing algorithm is informed and a new route selection is made. On the other hand, a loose coupling is possible: the routing algorithm establishes certain basic routes and the congestion control technique acts simply to control the total flow accepted on those routes in the network. An alternate strategy within the framework of a loose coupling might permit the congestion control technique to alter the fractional flows on different routes if multiple path routing is designed.

In general, it seems preferable to establish a loose coupling between routing and congestion control so that each technique can be developed independently and tuned independently in practice. Then, if each mechanism is operating correctly, one can obtain good network performance without risks to the

stability of the traffic flows and to complicated interactions.

As one example of the effects on network performance of the congestion control mechanism, we present a few simple numerical results concerning packet retransmissions. Consider the case of packet refusal between adjacent IMPs. If p is the probability that a transmission will be refused, then the expected number of transmissions is $1/(1-p) = 1 + p/(1-p)$. That is, the total number of transmissions equals one initial transmission and $p/(1-p)$ retransmissions. If we further assume that the system has some standard delay for the final successful transmission, and a waiting time (larger than twice the delay to the neighbor IMP) before retransmission, we can compute the total expected delay between neighboring IMPs. The table below gives some representative values for the increase in this delay due to retransmissions:

	Increase in Delay	
p (Prob of Refusal)	Wait = 2*Delay	Wait = 5*Delay
5%	10%	26%
10%	22%	56%
20%	50%	125%
30%	86%	215%

where Wait is the timeout period before retransmission. Clearly, it is important to keep the rate of packet retransmission down, probably to less than 10%. (This is also true for the discard

approach.) In addition to the delay experienced by a packet, refusal also increases the buffering required.

Retransmissions also impact the effective IMP-to-IMP throughput. If an IMP is able to continue sending other packets while timing out unacknowledged ones, the effective throughput of a channel with capacity C will be reduced to (1-p)*C due to packet refusal. If the IMP is unable to keep the trunk busy during packet timeout periods (due, for example, to a small number of IMP-to-IMP logical channels), the effective throughput of the trunk will be reduced still further.

The overall conclusion is that the measurement procedures by which the routing algorithm estimates network capacities and traffic delays should take into account the effect of the congestion control measures employed, especially retransmissions.

6.6 New Directions for Research

One possibility which has not been explored to date is that the routing algorithm should take cognizance, not only of average traffic flows in the network, but also of variances in those traffic flows. Likewise, the congestion control technique should be sensitive not only to average traffic flows which can be sustained at a particular IMP (a function of computer processing power and line bandwidth) but also to variations in that traffic flow which can be tolerated for a given period of time (a function of peak computer processing power and memory available for buffering traffic). If the congestion control mechanism makes available to routing an estimate of the traffic-carrying potential of the network, in terms of average flow and variance of that flow, then more closely engineered traffic flow can be obtained.

DYNAMIC BEHAVIOR OF SHORTEST PATH ROUTING ALGORITHMS FOR THE ARPANET

by

Dimitri P. Bertsekas

A.1 INTRODUCTION

This report considers the dynamic behavior of the following algorithm for routing packets in a store and forward communication network.

(A) Every x seconds a nonnegative length $D_{i\ell}$ of every link (i, ℓ) becomes available to each node. Based on these lengths each node computes a shortest path to each destination and routes the corresponding packets over that path during the next x seconds.

The standing assumptions for algorithm (A) are that *the lengths $D_{i\ell}$ depend exclusively on a fixed number of preceding shortest paths* via some as yet unspecified rule, and that the shortest path algorithm has an unambiguous rule for breaking ties between equidistant paths.

In algorithm (A) it is also implicitly assumed that the lengths $D_{i\ell}$ become simultaneously available to all nodes, that the shortest path computation is done instantaneously, and that the new routing takes effect immediately. In the framework of the ARPANET, these assumptions can be nearly satisfied in practice as will be explained in the next section.

The length $D_{i\ell}$ need not be the actual average delay per packet in link (i, ℓ). For example a bias (possibly different for each link) and dependent on global conditions such as for example sum of delays over all links could be added to the actual measured average delay per packet of a link to yield $D_{i\ell}$. Thus one may view in general $D_{i\ell}$ as a number which bears some (as yet unspecified) relation to traffic conditions that correspond to one or more preceding routings.

Routing algorithms such as (A) are currently under consideration for use in the ARPANET (see [A.1]). In the next section we discuss how the assumptions underlying algorithm (A) relate to actual operating conditions in an effort to determine the extent to which the subsequent analysis can provide the basis for design of a practical ARPANET routing algorithm.

The analysis is geared primarily towards studying the effect of the choice of lengths $D_{i\ell}$ on the dynamic behavior of the algorithm. Specific questions that we wish to answer, at least qualitatively, are:

1. For a particular choice of lengths will the algorithm tend to stabilize to a single routing path or will it cycle between two or more paths?

2. If it stabilizes to a single path, how close to "optimal" will this path be? If it cycles how close to "optimal" are the paths of the cycle?

Answers to these questions can suggest various procedures for choosing the lengths $D_{i\ell}$ in a practical setting.

The report is organized as follows:

In Sec. A.3 we provide a theoretical framework for studying the algorithm. We give a number of examples which show that if the lengths $D_{i\ell}$ are not chosen appropriately the algorithm may naturally tend to oscillate between bad routing paths and become itself a major contributor to network congestion. These examples show that *proper choice of $D_{i\ell}$ is probably the single most important design aspect of the routing algorithm.* They also show that *there is a basic design tradeoff between algorithmic stability and adaptation to traffic conditions.*

For the purpose of a more refined analysis we introduce in Sec. A.4 a model of a ring network with an infinite number of nodes. This allows us to employ techniques of stability analysis of discrete systems with continuous state space (see e.g. [*A.2*, *A.3*]), and enables us to quantify the relationship between choice of lengths and algorithmic behavior.

The ring topology is fundamental for analysis of more general network topologies. This is shown in Sec. A.5 where the results of Sec. A.4 are extended to arbitrary topologies and multiple destinations.

In Sec. A.6 we transfer some of the results of Sec. A.4 to finite node ring networks and we confirm the relevance of the continuous model analysis. We also present computational results that generally support the conclusions of the theoretical analysis.

The conclusions section summarizes the results of the analysis, and suggests guidelines for experimentation with practical routing schemes for the ARPANET.

A.2 DISCUSSION OF ASSUMPTIONS

In the SPF algorithm [*A.1*] currently under consideration for use in the ARPANET it is envisioned that nodes broadcast at regular intervals (say x sec long) the values of measured delay per packet on their outgoing links, averaged over the preceding x second period. This is done only if the delay value of some link has changed by more than some threshold amount over that period. Otherwise the update is foregone. Once an updated delay of a particular link reaches a node in the network, this node computes a length for that link (for example by adding a bias factor to reported delay), and modifies its shortest path tree to take into account the updated link length.

This mode of operation differs from that of the idealized algorithm (A) of the previous section in a number of ways. Algorithm (A) assumes that the link lengths $D_{i\ell}$ become simultaneously available to all nodes, and that the routing change is effected instantaneously. On the other hand in the practical setting some time elapses before the broadcast delays reach other nodes, the new shortest paths of these nodes are calculated and packet transmission according to the new routing is initiated. The propagation time of a delay broadcast, the calculation time for a node to update the shortest path tree, and the time necessary for a node to transmit the packets assigned to store and forward buffers according to the previous routing are all very small relative to the time interval between two updates of the same node, and they will be neglected in what follows.

In the case where nodes broadcast their delays synchronously, then, once the transients discussed in the preceding paragraph are neglected, it can be seen that the broadcast delays depend

only on the preceding routing and not on earlier routings. In practice nodes will not possess synchronized clocks so that synchronous broadcasting is impossible. On the other hand, analysis of the synchronous case is of interest both because it is simpler and because it provides a yardstick against which performance of asynchronous schemes can be measured. In the case where nodes broadcast asynchronously, then by associating each node broadcast with a routing update we see that in a single x-second period there are N routing updates where N is the number of nodes. Thus, each broadcast average delay depends on the preceding N routings and as a result, each shortest path calculation is based on delays that depend on a total of (2N-1) past routings. This follows from the fact that if B_{kN+1}, $B_{kN+2}, \cdots, B_{kN+N}$ are the broadcast delay vectors in the k-th x-second period, then the delay vector B_{kN+N} depends on the N routings corresponding to $B_{kN}, B_{kN+1}, \cdots, B_{kN+N-1}$, and, proceeding backwards, the delay vector B_{kN+1} depends on the N routings corresponding to $B_{(k-1)N+1}, \cdots, B_{kN}$. Thus we see that the routing corresponding to B_{kN+N} will be calculated on the basis of delays that depend on the routings corresponding to $B_{(k-1)N+1}, \cdots, B_{kN+N-1}$ for a total of (2N-1) routings.

Both cases of synchronous and asynchronous operation of the SPL algorithm are covered by the subsequent analysis of algorithm (A) provided the following additional simplifying assumptions are made.

1. The statistics of the outside (host) traffic and the network topology remain constant over a time interval including several updating periods.

2. Measured link average delays equal actual link average delays.

3. All link delays are broadcast (perhaps asynchronously) during each period, (i.e., the threshold change for a link delay to be broadcast is zero).

4. The shortest path algorithm has a fixed rule for determining all link lengths $D_{i\ell}$ as functions of reported link delays, and for breaking ties among equidistant paths.

Thus the subsequent analysis of algorithm (A) should be judged in the light of the preceding assumptions (1 through 4). These assumptions, however, appear to be realistic enough to allow confidence in the analytical conclusions.

For the most part of our analysis, we will also need to assume that link average delay (in sec/packet) depends exclusively on the average link flow (in bits/sec), in monotonically increasing fashion. This assumption is commonly made in communication network analyses, and we do not believe that it affects materially the basic nature of the results obtained.

A.3 A DISCRETE MODEL – EXAMPLES

Consider first algorithm (A) applied to a given network for the case where the lengths $D_{i\ell}$ depend exclusively on the preceding routing. Then each routing uniquely determines the lengths $D_{i\ell}$ and these in turn uniquely determine the next routing. There is a finite number of possible routings which we denote by $R_1, R_2, \ldots, R_M$, where M is some integer. To any routing, say R_{i_0}, there corresponds a unique sequence of subsequent routings $R_{i_1}, R_{i_2}, \ldots$ as shown below

$$R_{i_0} \rightarrow R_{i_1} \rightarrow R_{i_2} \rightarrow \cdots .$$

Because the set of routings is finite eventually some routing (say $R_{i_k} = R_{ik+n}$) will be repeated and once this happens the sequence will become periodic as shown in Fig. A.1.

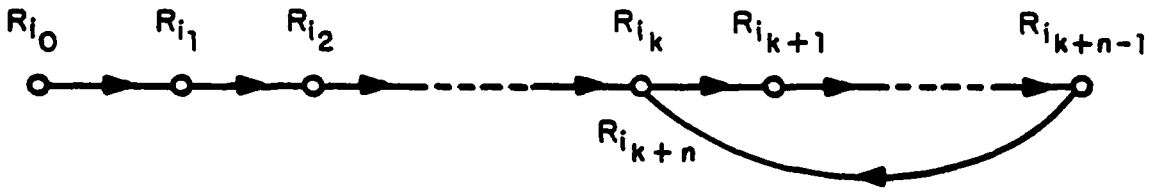


Figure A.1

Thus starting at R_{i_0} the algorithm will eventually end up cycling through $R_{i_k}, \cdots, R_{i_{k+n-1}}$. Of course, it is possible that R_{i_0} itself is part of the cycle (k=0), and that the cycle consists of a single routing (n=1) in which case the algorithm stabilizes at that routing.

In view of the fact that each routing uniquely determines its successor, it follows that the set of all routings $\{R_1, \cdots, R_M\}$ can be partitioned into a collection of cycles, and a collection of transient routings.

If the initial routing is transient it is never repeated by the algorithm, and if it is part of a cycle the algorithm returns to it periodically. More than one cycle may exist. Furthermore, each transient routing leads to some unique cycle as shown in the example of Fig. A.2 which involves three cycles and eleven transient routings.

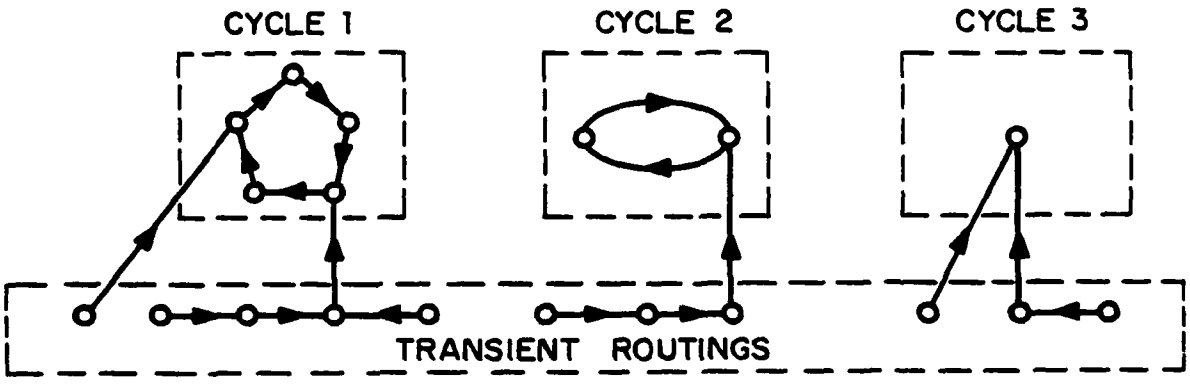


Figure A.2

From a design point of view we are primarily interested in the cycles since routing will eventually be confined to one of them. The design task is to ensure that for at least most host traffic patterns, "good" cycles result in the sense that the corresponding routings yield small average total delay per packet. On the other hand, the formation and nature of cycles depends exclusively on the choice of the link lengths $D_{i\ell}$ for each routing.

The following examples show that *if each link length is chosen to be simply the average delay of that link, then the algorithm will naturally tend to oscillate between very bad routings. If each link length is obtained by adding a sufficiently high bias factor to delay the algorithm can stabilize at a single routing, or perform a small oscillation between two routings, which lie somewhere between an unbiased equilibrium routing and the min-hop routing.* The behavior excibited in these examples is typical as will be demonstrated by the analysis of subsequent sections.

Example 1: Consider the following 16-node ring network where

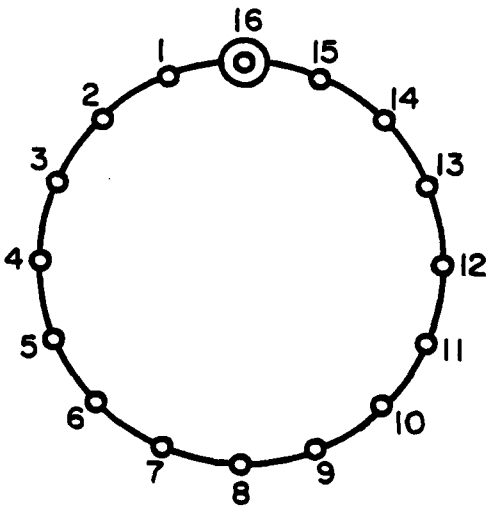


Figure A.3

all messages have node 16 as their destination. All links are bidirectional and, for simplicity, average delay per message on each link will be assumed equal to average flow (in bits/sec) on that link. The results are unaffected if average delay is assumed only proportional to average flow. Oscillatory behavior

is actually aggravated if more realistic average delay-average flow relationships are assumed. The critical assumption is really that link delay is zero when link flow is zero. Ties between equidistant paths are broken in favor of the path with minimum number of hops to the destination, and if this is not decisive the counterclockwise path is chosen by the algorithm.

Notice that under the assumption that average link delay is equal to average link flow, the optimal (single path) routing that minimizes the sum of link delays is the min-hop routing. This is true regardless of the average traffic input pattern. On the other hand, the position of an "equilibrium" routing which splits the flow at a point where clockwise and counterclockwise delay distances to the destination are equalized, strongly depends on the traffic input pattern. This illustrates an important point, namely that routing flow by equalizing clockwise and counterclockwise delay distances does *not* lead to a minimum total delay routing. This is true regardless of the functional relationship between average link flow and average link delay as will be shown and further clarified in the next section.

Case 1:

Choice of Link Length:	Equal to average delay (and flow) on the link.
Average Traffic Inputs to the Destination (Normalized):	Each node sends one unit except for node 8 which sends ε units where $0 \leq \varepsilon << 1$.

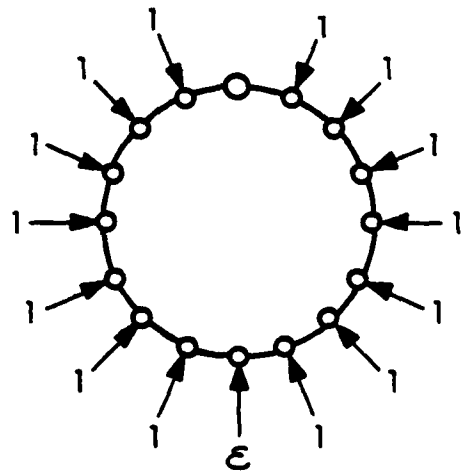


Figure A.4

Every (single path) routing can be characterized by the smallest node index i which sends flow counterclockwise. We call this routing R_i as shown in Fig. A.5.

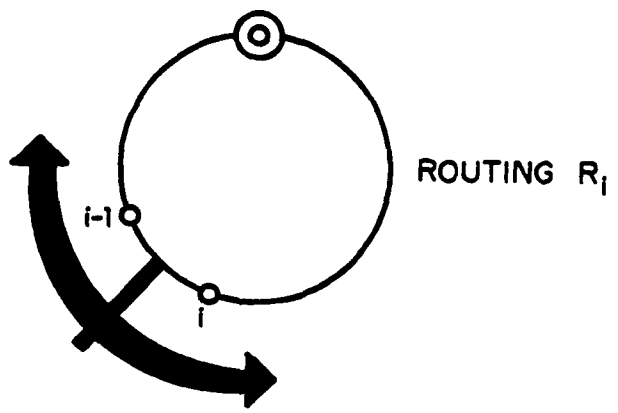


Figure A.5

There are two optimal routings (minimizing the sum of average delays over all links), R_8 and R_9.

Suppose the initial routing is R_8. If $\varepsilon = 0$ then the algorithm will stay at R_8 for all subsequent iterations. However, this is a fragile equilibrium since if $\varepsilon > 0$ the routing sequence and corresponding flows (and delays) generated are R_8, R_{10}, R_3, R_{16}, R_1, R_{16}, R_1,$\cdots$, as shown in Fig. A.6.

Thus for $\varepsilon > 0$ the algorithm when started at an optimum ends up oscillating between the two worst routings R_1 and R_{16}.

When link lengths are taken equal to link delays (i.e., no bias is added), this behavior is typical for ring networks and *does not depend on the initial routing, the traffic input pattern, or the number of nodes.* It is possible to prove that, except for singular cases, *every initial routing will lead to an oscillation between the two extreme routings* (R_1 and R_{16} in the present example). Even in singular cases where other equilibria may form (such as when $\varepsilon = 0$ in the present example), these equilibria can be destabilized by addition of a small traffic input to one or more of the nodes. This fact is proved in Sec. A.6 under a more general hypothesis whereby the link length $D_{i\ell}$ is not necessarily equal to link flow but rather it is any monotonically increasing function of link flow with $D_{i\ell}$ equal to zero whenever link flow is zero.

Case 2:

Choice of Link Length:	Equal to average delay (and flow) on the link plus a positive bias factor α.
Traffic Inputs to the Destination:	Same as Case 1.

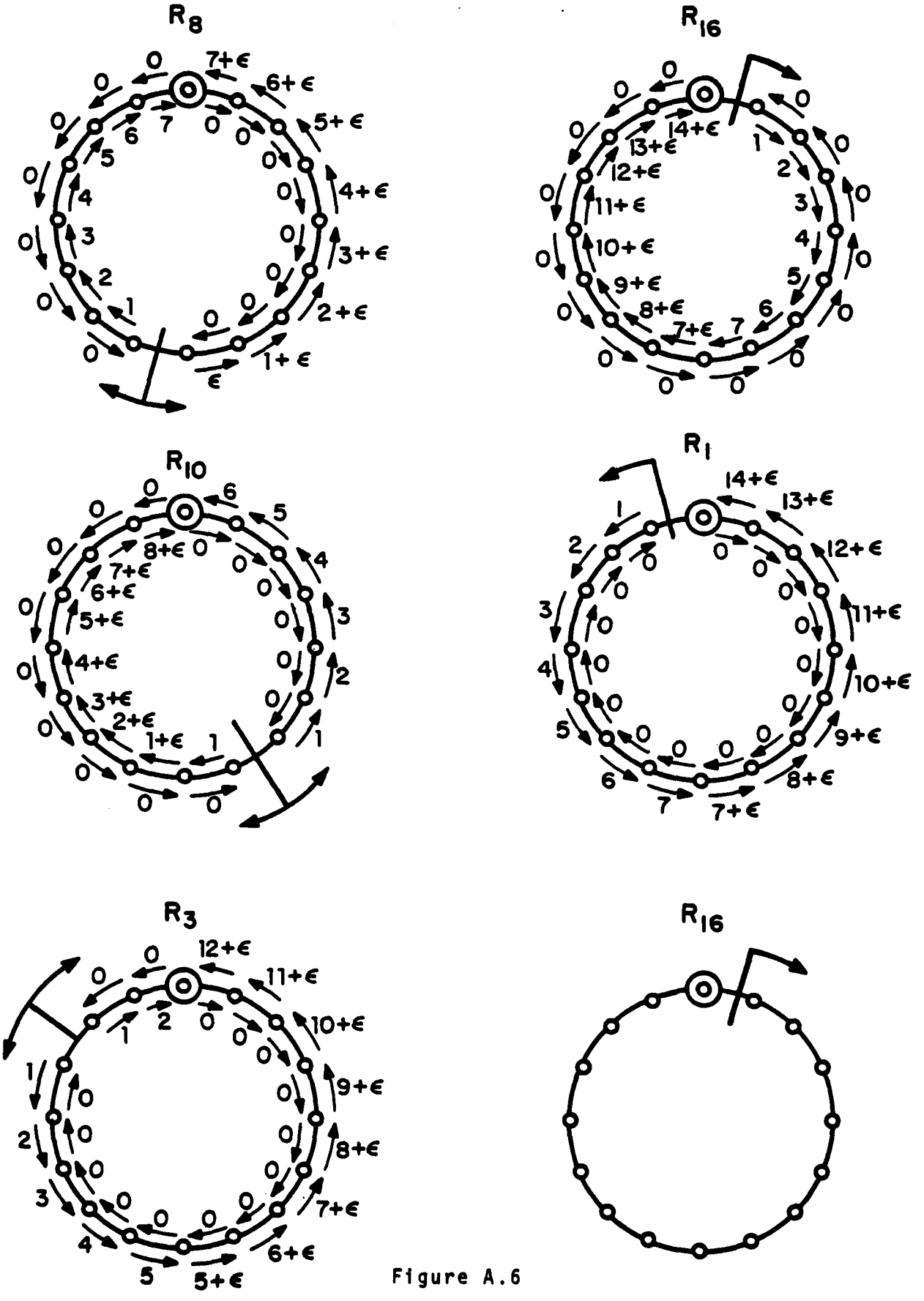


Figure A.6

Suppose the initial routing is R_8. The flow pattern is as shown in Fig. A.7.

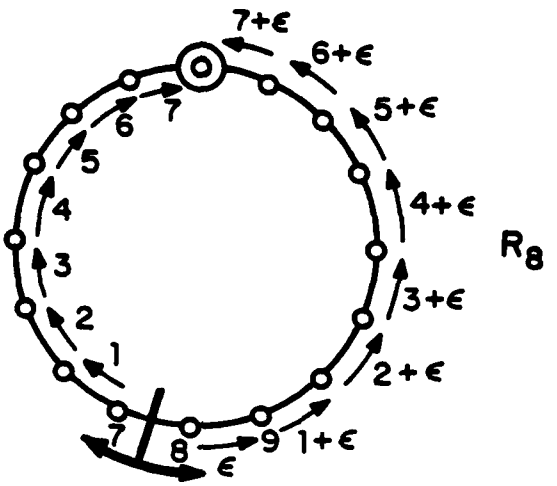


Figure A.7

Clearly, node 8 will switch his traffic in the clockwise direction. Node 9 faces a clockwise distance to the destination equal to $(28+9\alpha)$ and a counterclockwise distance equal to $(28+8\epsilon+7\alpha)$. It follows that node 9 will not switch traffic direction if

$$(28+9\alpha) \geq (28+8\epsilon+7\alpha)$$

or, if

$$\alpha \geq 4\epsilon.$$

In this case the next routing will be R_9. By symmetry the routing subsequent to R_9 is R_8, so that if $\alpha \geq 4\epsilon$ the algorithm stabilizes to an oscillation between the two optimal routings R_8 and R_9.

Notice that the threshold bias value, 4ε, is the product of the input at the "equilibrium" (node 8) times a measure of the equilibrium distance to the destination (this will be interpreted as a measure of marginal delay in the next section) divided by two. This is a special case of a more general expression for the threshold bias value that will be developed in subsequent sections.

It would be misleading however to conclude that a bias value of 4ε would lead to good algorithmic performance. If $\alpha \geq 4\varepsilon$ the only thing that is guaranteed is that the equilibrium is "locally" stable. If the initial routing is not near the equilibrium a much higher value of bias may be needed to bring the routing back near the equilibrium. For example, suppose that due to some unfortunate sequence of events the initial routing is R_1. We see from Fig. A.8 that node 15 faces a

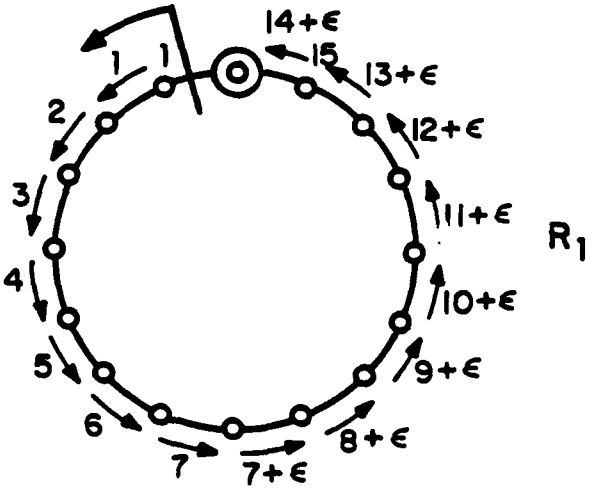


Figure A.8

clockwise distance to the destination of 15α and a counter-clockwise distance of $(14+\varepsilon+\alpha)$. Thus, if $15\alpha<14+\varepsilon+\alpha$ or

$$\alpha < 1 + \frac{\varepsilon}{14}$$

node 15 will switch his traffic, the next routing will be R_{16}, and by symmetry the algorithm will again oscillate between the two worst routings R_1 and R_{16}. In fact, for this network one can calculate that it is necessary to have

$$\alpha \geq 7 + \frac{7\varepsilon}{2}$$

in order to guarantee that for any initial routing the algorithm will eventually oscillate between the two optimal routings R_8 and R_9. For smaller values of α the system will end up oscillating around the equilibrium with the oscillation becoming "larger" as the value of α decreases and the initial routing is farther from the equilibrium.

The two main points that this case illustrates are first that *a value of bias above a certain threshold can stabilize the algorithm around an "equilibrium,"* and second that *a possibly higher value of bias may be necessary in order to guarantee that the algorithm returns to the equilibrium when starting from an arbitrary initial routing.*

The symmetry of the traffic input pattern masks another important feature of the algorithm namely that *the (stable or unstable) "equilibrium" depends on the value of bias. For low values of bias it tends to lie near an unbiased routing equilibrium while for high values of bias it tends to lie near a min-hop routing.* This is illustrated in the next case.

Case 3:

Choice of Link Length: Equal to average delay (and flow) on the link plus a nonnegative bias factor α.

Traffic Inputs to the Destination: Nodes 1, 2, 3, 4 send five units and all other nodes send one unit as shown in Fig. A.9.

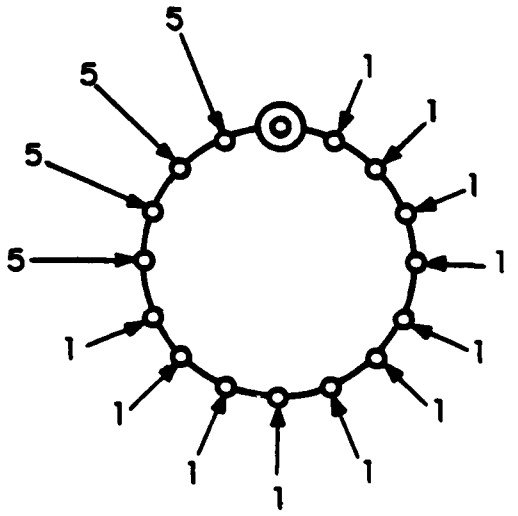


Figure A.9

Routing R_6 shown in Fig. A.10 may be considered as an unbiased equilibrium (α=0) since nodes 5 and 6 face clockwise and counterclockwise delay distances both equal to 55. The equilibrium is actually unstable because, due to our rule of breaking ties in favor of the min-hop distance, node 6 will switch its flow to the clockwise direction in the next routing.

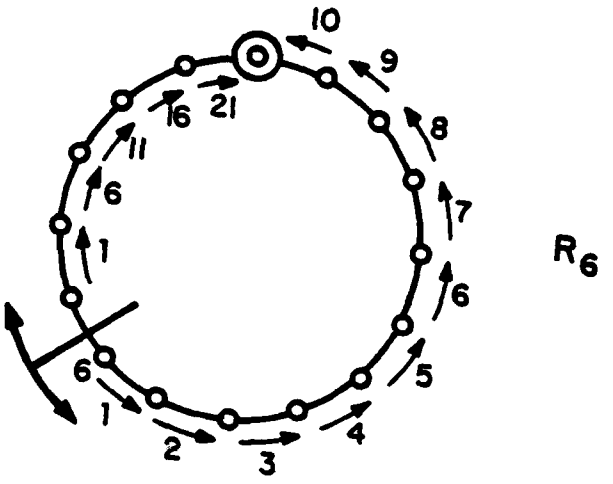


Figure A.10

However, even if that rule were not adopted, it can be shown that addition of a small $\varepsilon>0$ to the input traffic of any one of the nodes will lead the algorithm to an oscillation between the two extreme routings R_1 and R_{16}. On the other hand, a straightforward but lengthy calculation shows that:

(a) *If* $\frac{15}{6} \leq \alpha < 4$, the algorithm will oscillate between R_6 and R_7 if started at R_6 or R_7.

(b) *If* $4 \leq \alpha \leq 8$, the algorithm will eventually stabilize at R_7 if started at R_6, R_7 or R_8.

(c) *If* $8 \leq \alpha < 16$, the algorithm will oscillate between R_7 and R_8 if started at R_7 or R_8.

(d) *If* $16 \leq \alpha$, the algorithm will eventually stabilize at R_7 if started at R_7, R_8, or R_9.

This example reveals that in fact there are two types of stable equilibria to deal with in a ring network, *equilibrium routings* and *equilibrium nodes*. An equilibrium routing is one which the algorithm repeats where started at it, and an equilibrium node is one that switches its traffic whenever the routing is adjacent to it. As the value of the bias factor increases from $\frac{15}{6}$ to ∞ the equilibrium moves from node 6 ($\frac{15}{6} \leq \alpha < 4$), to routing R_7 ($4 \leq \alpha < 8$), to node 7 ($8 \leq \alpha < 16$), to routing R_8 ($16 \leq \alpha$). For values of bias lower than $\frac{15}{6}$ there is no stable equilibrium and the algorithm ends up in an oscillation the magnitude of which depends on the bias and the initial routing. If no bias is added ($\alpha=0$) all initial routings lead to an oscillation between the two extreme routings R_1 and R_{16}. This type of behavior will be further clarified by analysis and examples in the next section.

The unstable behavior of the algorithm for zero or small positive values of bias is not restricted to the case of a ring network but rather it is typical of arbitrary topology networks with a single destination. We present briefly two examples where link delay is taken to be equal to link flow. A more detailed analysis confirming this fact is presented in the sequel.

Example 2: Consider the network shown in Fig. A.11 where each node sends one unit to the destination.

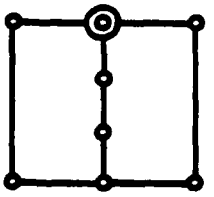


Figure A.11

Let the initial routing be as shown in Fig. A.12.

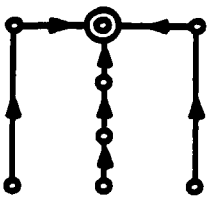


Figure A.12

Then, if bias is zero, the algorithm ends up oscillating between the two routings shown in Fig. A.13.

Figure A.13

Example 3: Consider the network shown in Fig. A.14 where each

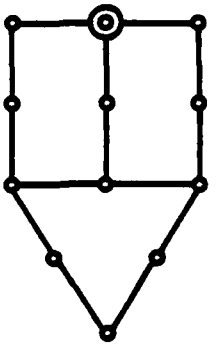


Figure A.14

node sends one unit to the destination.

Let the initial routing be as shown in Fig. A.15.

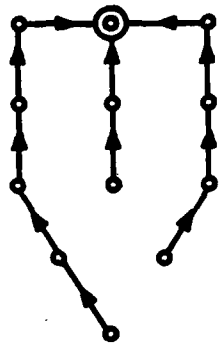


Figure A.15

Then if bias is zero the algorithm ends up oscillating between the two routings shown in Fig. A.16.

Figure A.16

Unstable behavior in the absence of a bias factor can also occur in the case of multiple destinations as the following example shows.

Example 4: Consider the following 8-node ring network. There

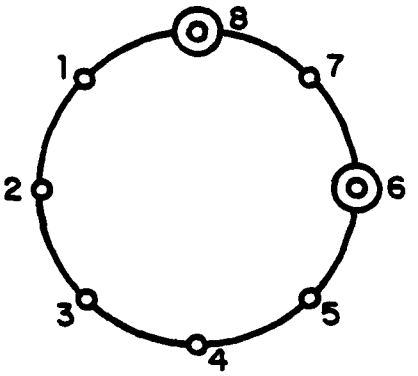


Figure A.17

are two destinations, nodes 6 and 8. Each node sends one unit to each destination and link delay is taken equal to link flow. Let the initial routing and corresponding link flows be as shown in Fig. A.18.

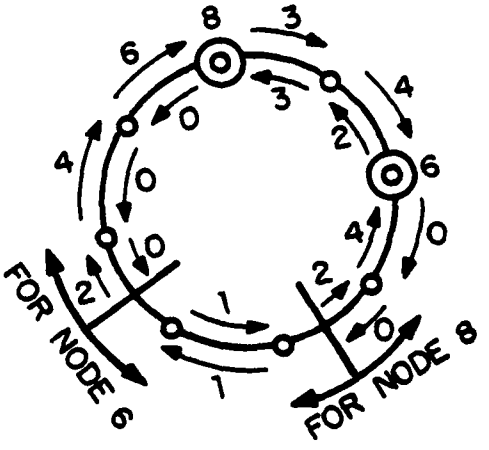
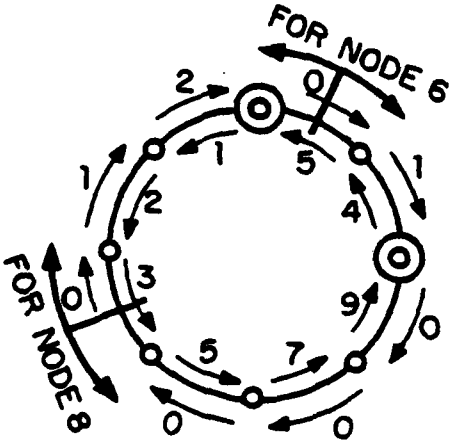
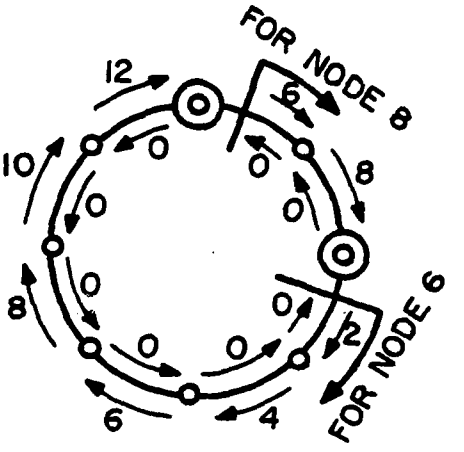


Figure A.18

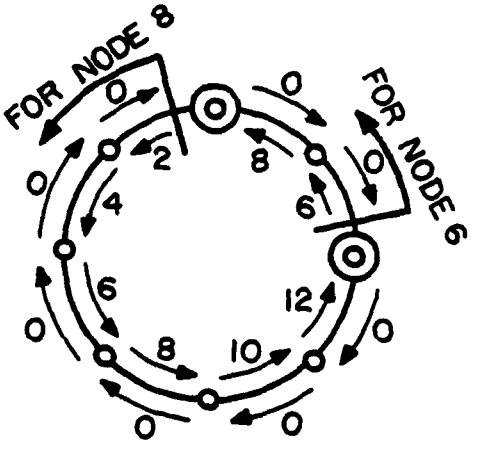
Then if bias is zero the algorithm generates the following sequence of routings and flows shown in Fig. A.19.



SECOND ROUTING



THIRD ROUTING



FOURTH ROUTING

Figure A.19

Thus after two routing updates the algorithm oscillates between the two routings that send all flow clockwise and all flow counterclockwise respectively.

We have considered so far the case where the lengths D_{il} depends exclusively on the preceding routing. A similar conceptual framework can be employed for the case where D_{il} depends on a fixed number of past routings. The details become more complex and their presentation will not be undertaken. We provide instead an example showing that *if D_{il} is obtained by averaging delays of the two past routings then the stability properties of the algorithm are considerably improved.* Subsequent analysis will establish this fact in a more general context.

Example 5: Consider Example 1 for the case where each node sends one unit to the destination except for node 8 which sends nothing. Let each link length be the flow on the link averaged over the preceding two routings. Suppose that the first two initial routings are R_{16} and R_1. Then a straightforward calculation shows that the third, fourth, and fifth routing will be R_8 R_{15} and R_2, and the algorithm settles in the cycle $R_8 \rightarrow R_{15} \rightarrow R_2 \rightarrow R_8 \rightarrow R_{15} \rightarrow R_2 \rightarrow R_8 \rightarrow \cdots$ shown in Fig. A.20. This compares favorably with the cycle $R_{16} \rightarrow R_1 \rightarrow R_{16} \rightarrow \cdots$ obtained in the case without averaging.

The point of view that has been adopted in this section is one whereby the algorithm is viewed as a dynamic system with a finite number of states (the finite collection of possible routings). We studied via examples the manner in which the

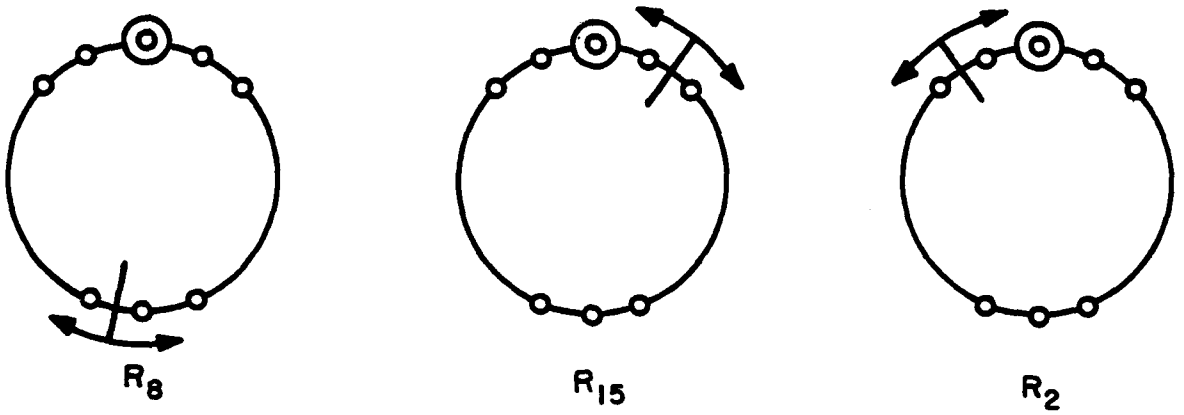


Figure A.20

algorithm moves from state to state, i.e., the dynamic behavior of the corresponding system. Unfortunately, the study of the dynamic behavior and stability properties of systems with a finite number of states is notoriously difficult. To begin with, there is no accepted definition of equilibrium, and in fact, we saw that in the ring network context there are two types of "equilibria" that are of interest – equilibrium routings and equilibrium nodes. Furthermore, there are no established methodological tools that can be of any help in a discrete system framework. This motivates approximation of the discrete system with a continuous system having a continuum of states. For such systems, there is an effective and well developed stability theory that can be used for analysis [*A.2, A.3*]. We take this approach in the following two sections where we introduce a network with a continuum of nodes. Despite the radical nature of this step, it turns out that the analysis not only provides informative results, but also points the way for a corresponding analysis of the algorithm applied to the usual finite node networks.

A.4 THE CONTINUOUS MODEL OF A RING NETWORK WITH A SINGLE DESTINATION

As a first step towards developing and analyzing a more general model, we consider a continuum of nodes arranged in a ring and sending traffic to a single destination.

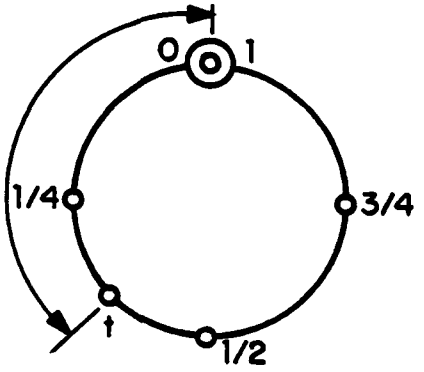


Figure A.21

Points on the ring are identified with their distance t from the destination in the counterclockwise direction, where t is normalized to take values in the interval [0,1]. Traffic can move on the ring in both directions.

For every t in [0,1] we denote by r(t) the *input density* at t. The meaning of r(t) is that for any subinterval $[t_1,t_2]$ of [0,1] the total input traffic originating at the nodes in $[t_1,t_2]$ is

$$\int_{t_1}^{t_2} r(t)dt.$$

We assume that r(t) is piecewise continuous (i.e., continuous everywhere except at a finite number of points). We thus allow the possibility of traffic input densities such as the one shown in Fig. A.22 which approximate the traffic pattern of a

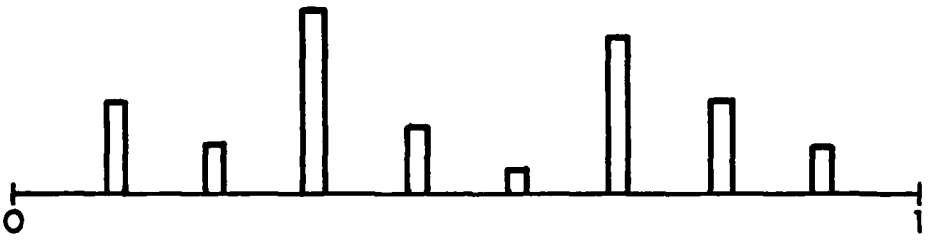


Figure A.22

network with a finite number of nodes. This corresponds to approximating impulses of input by narrow pulses of appropriate height.

We are interested in routings specified by points y in [0,1] where the flow splits, i.e., points larger than y send their flow counterclockwise (or in the positive direction) and points smaller than y send their flow clockwise (or in the negative direction) as shown in Fig. A.23. To a given input

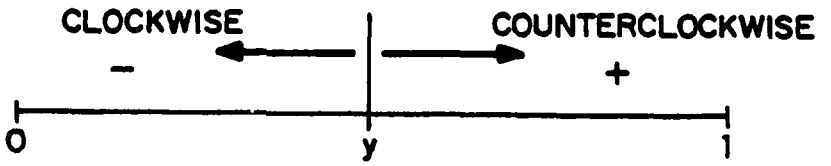


Figure A.23

density function r(t) and routing y, there corresponds at every point t *a flow in the positive direction* $f^+(y,t)$, and *a flow in the negative direction* $f^-(y,t)$ given by

$$f^+(y,t) = \begin{cases} \int_y^t r(\tau)d\tau & \text{if } y \le t \\ 0 & \text{if } t \le y \end{cases} \tag{A.1}$$

$$f^-(y,t) = \begin{cases} 0 & \text{if } y \le t \\ \int_t^y r(\tau)d\tau & \text{if } t \le y. \end{cases} \tag{A.2}$$

In what follows the input density will be considered constant over time, so our notation for flow does not reflect the dependence on r(t).

In order to introduce an algorithm such as (A) in the continuous framework we consider a *delay function* d(f) which for the time being will be assumed to be a function of the flow f only. We later allow dependence of d on y and t as well as on f. The meaning of the delay function d is that given a routing y and any point t, *the distances* D_0 *and* D_1 *from t to 0 and 1* (i.e., to the destination in the clockwise and counterclockwise direction) are given by

$$D_0(y,t) = \int_0^t d[f^-(y,\tau)]d\tau, \tag{A.3}$$

$$D_1(y,t) = \int_t^1 d[f^+(y,\tau)]d\tau. \tag{A.4}$$

These distances depend on the routing y through the flows f^+ and f^-. We will assume that d is a monotonically increasing function of f with everywhere continuous derivative. Furthermore, $d(f) \geq 0$ for all $f \geq 0$.

We consider the following algorithm patterned after algorithm (A). Given a routing y_k at period k the next routing y_{k+1} is determined from the relation

$$D_0(y_k,y_{k+1}) = D_1(y_k,y_{k+1}). \tag{A.5}$$

Since we have

$$D_0(y_k,t) \leq D_0(y_k,y_{k+1}) = D_1(y_k,y_{k+1}) \leq D_1(y_k,t) \text{ if } t \leq y_{k+1}$$

and

$$D_0(y_k,t) \geq D_0(y_k,y_{k+1}) = D_1(y_k,y_{k+1}) \geq D_1(y_k,t) \text{ if } t \geq y_{k+1},$$

it follows that the routing y_{k+1} determined from (A.5) is such that every point t routes its flow in the positive or negative direction according as $D_0(y_k,t) \geq D_1(y_k,t)$ or $D_0(y_k,t) \leq D_1(y_k,t)$, i.e., according to minimum distance to the destination.

We will assume that, for every y_k, equation (A.5) has y_{k+1} as its unique solution. [It is possible to show that this is guaranteed if either $r(t) > 0$ for all t or $d(0) > 0$]. Then we write the relationship between y_k and y_{k+1} as

$$y_{k+1} = g(y_k) \tag{A.6}$$

where g is the function satisfying [cf. (A.5)] for all y

$$D_0[y,g(y)] = D_1[y,g(y)]. \tag{A.7}$$

Thus the algorithm is described compactly by equation (A.6)

Example 1: Let $r(t) = 1$ for all t, and $d(f) = f$ for all f. Then from (A.1) through (A.4)

$$f^+(y,t) = \begin{cases} t-y & \text{if } y \leq t \\ 0 & \text{if } t \leq y \end{cases}$$

$$f^-(y,t) = \begin{cases} 0 & \text{if } y \leq t \\ y-t & \text{if } t \leq y \end{cases}$$

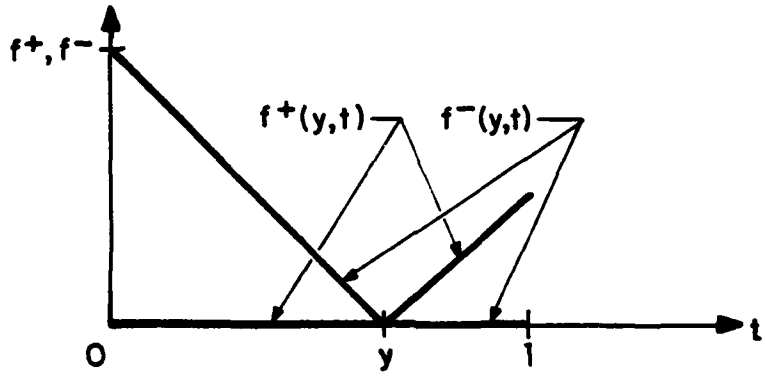


Figure A.24

If $y \leq t$

$$D_0(y,t) = \int_0^y (y-\tau)d\tau = \frac{y^2}{2}$$

$$D_1(y,t) = \int_t^1 (\tau-y)d\tau = \frac{1}{2}(1-t)^2 - y(1-t)$$

If $t \leq y$

$$D_0(y,t) = \int_0^t (y-\tau)d\tau = -\frac{t^2}{2} + yt$$

$$D_1(y,t) = \int_y^1 (\tau-y)d\tau = \frac{1}{2}(1-y)^2$$

The equation $D_0(y_k, y_{k+1}) = D_1(y_k, y_{k+1})$ is quadratic in y_{k+1} and can be solved to yield after some calculation

$$y_{k+1} = \begin{cases} y_k - \sqrt{y_k^2-(1-y_k)^2} & \text{if } y_k \geq \frac{1}{2} \\ \\ y_k + \sqrt{(1-y_k)^2-y_k^2} & \text{if } y_k \leq \frac{1}{2} \end{cases}.$$

This equation is really equation (A.6) specialized to this example. It completely specifies the algorithm.

We say that a routing y* is an *equilibrium* if

$$y^* = g(y^*) , \tag{A.8}$$

or equivalently if

$$D_0(y^*,y^*) = D_1(y^*,y^*). \tag{A.9}$$

Thus the algorithm when started at an equilibrium it remains there. It is possible to show that if r(t) > 0 for all t, or d(0) > 0 then there exists a unique equilibrium. We will assume that one of these two conditions holds in what follows.

The equilibrium has an interesting optimality property which we state as a proposition.

Proposition: The equilibrium y* minimizes over all y in [0,1] the expression

$$J(y) = \int_0^1 p[f^+(y,t)]dt + \int_0^1 p[f^-(y,t)]dt \qquad (A.10)$$

where p is any function having as derivative the delay function d, i.e.,

$$\frac{\partial p(f)}{\partial f} = d(f) \qquad \text{for all } f. \qquad (A.11)$$

Proof: The proof assumes that r(t) is a continuous function but a slight modification proves the result in general. Differentiating J(y) we have

$$\frac{\partial J(y)}{\partial y} = \int_0^1 \frac{\partial p[f^+(y,t)]}{\partial y}\, dt + \int_0^1 \frac{p[f^-(y,t)]}{\partial y}\, dt =$$

$$= \int_0^1 \frac{\partial p[f^+(y,t)]}{\partial f}\, \frac{f^+(y,t)}{\partial y}\, dt + \int_0^1 \frac{\partial p[f^-(y,t)]}{\partial f}\, \frac{\partial f^-(y,t)}{\partial y}\, dt \qquad (A.12)$$

It can be seen from (A.1) and (A.2) that

$$\frac{\partial f^+(y,t)}{\partial y} = \begin{cases} -r(y) & \text{if } y < t \\ 0 & \text{if } t < y \end{cases} \qquad (A.13)$$

$$\frac{\partial f^{-}(y,t)}{\partial y} = \begin{cases} 0 & \text{if } y \leq t \\ \\ r(y) & \text{if } t \leq y. \end{cases} \qquad \text{(A.14)}$$

Combining equations (A.11) through (A.14) we obtain

$$\frac{\partial J(y)}{\partial y} = r(y) \left[-\int_{y}^{1} d[f^{+}(y,t)]dt + \int_{0}^{y} d[f^{-}(y,t)]dt \right]$$

Using (A.3) and (A.4) we finally obtain

$$\frac{\partial J(y)}{\partial y} = r(y) \, [D_0(y,y) - D_1(y,y)].$$

If y^* is the equilibrium it can be seen that we have

$$D_0(y,y) \leq D_1(y,y) \qquad \text{if } y \leq y^*$$

$$D_0(y,y) \geq D_1(y,y) \qquad \text{if } y^* \leq y.$$

Thus

$$\frac{\partial J(y)}{\partial y} \leq 0 \qquad \text{if } y \leq y^*$$

$$\frac{\partial J(y)}{\partial y} \geq 0 \qquad \text{if } y^* \leq y$$

$$\frac{\partial J(y^*)}{\partial y} = 0$$

and it follows that y^* minimizes $J(y)$. A.E.D.

The proposition shows that we can minimize the integral of a certain function over the ring by choosing the length function d to be its derivative *and* guaranteeing that the algorithm $y_{k+1}=g(y_k)$ converges to the equilibrium y*. Thus *if we wish to minimize integral of delay over the ring we should choose as length function d the marginal delay.* Furthermore, *we should ensure that y* is a stable equilibrium,* i.e., the algorithm converges to y* at least when started close to it and, preferably, even when started far from it. This however seems quite impossible for reasons explained in what follows.

We now turn to a discussion of the stability properties of y*. Our starting point is the equation

$$y_{k+1} = g(y_k)$$

specifying the algorithm. Successive iterations of the algorithm can be described as in Fig. A.25.

The local stability properties of y* depend on the derivative $\frac{\partial g(y^*)}{\partial y}$ of g at y*. If $\frac{\partial g}{\partial y}$ exists, is continuous at y* and

$$\left|\frac{\partial g(y^*)}{\partial y}\right| < 1$$

the equilibrium y* is (locally) stable, i.e., there is an interval centered at y* such that when the algorithm is started within this interval it converges to y*. This is illustrated in Fig. A.26.

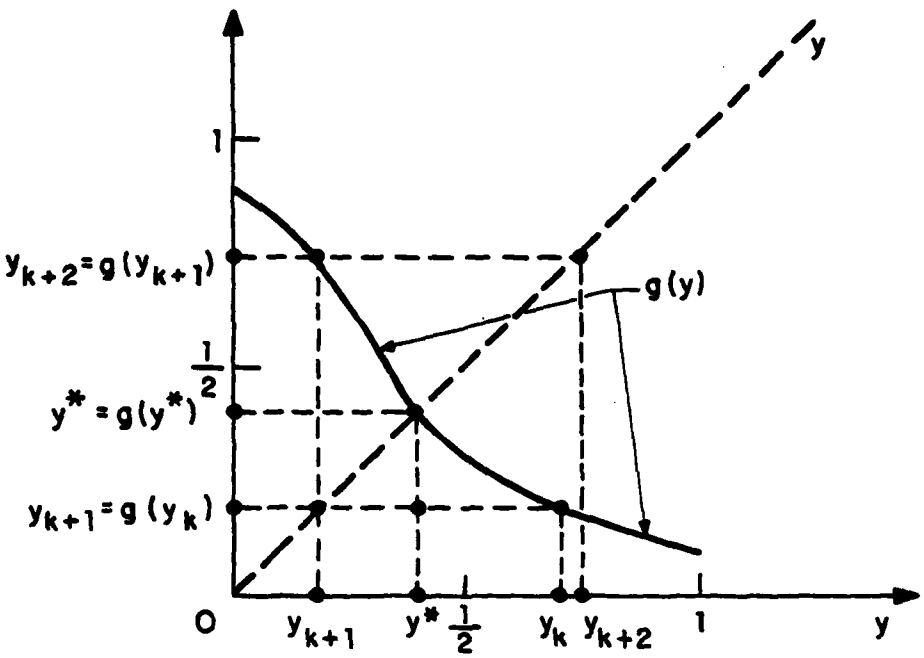


Figure A.25

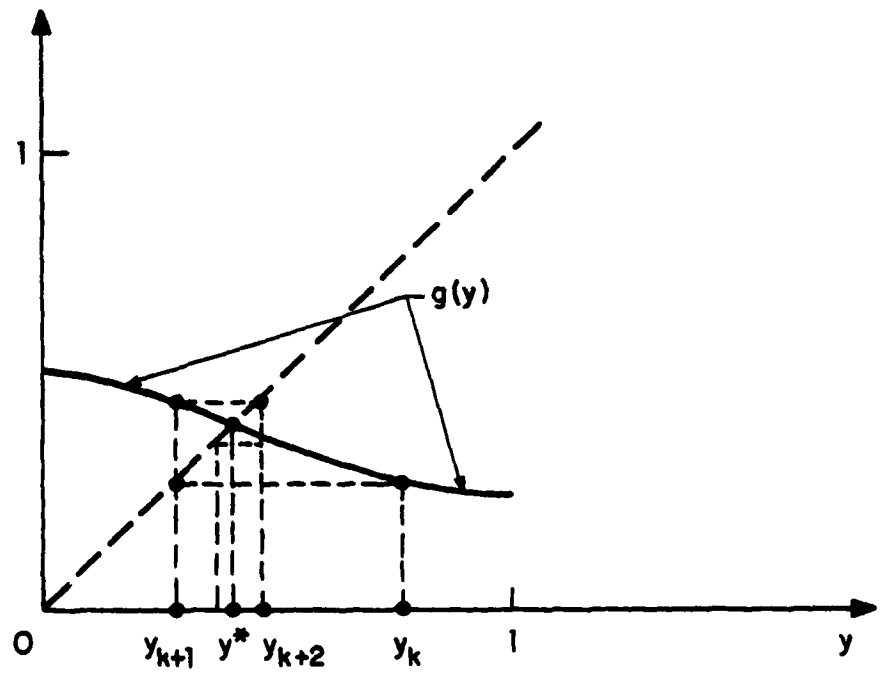


Figure A.26

If $\frac{\partial g}{\partial y}$ is continuous at y* and

$$\left|\frac{\partial g(y^*)}{\partial y}\right| > 1$$

the equilibrium is unstable, i.e., when the algorithm is started close to y* it tends to diverge from it. This is illustrated in Fig. A.27.

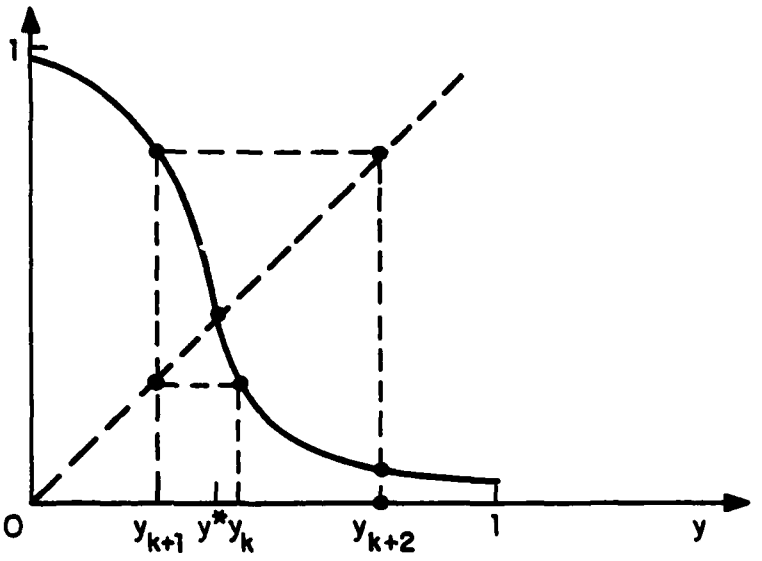


Figure A.27

In the marginal case where $\left|\frac{\partial g(y^*)}{y}\right| = 1$ the situation is unclear and stability of y* depends on the derivative of g at points near y*. If $\frac{\partial g(y^*)}{\partial y}$ does not exist or if it is not continuous, a similar but more careful analysis is possible by taking limit of the derivative from the left and the right.

We now evaluate the derivative $\frac{\partial g(y)}{\partial y}$. Consider the equation that defines g(y)

$$D_0[y,g(y)] = D_1[y,g(y)].$$

Using (A.3) and (A.4) we can write this equation as

$$\int_0^{g(y)} d[f^-(y,\tau)]d\tau = \int_{g(y)}^1 d[f^+(y,\tau)]d\tau. \qquad (A.15)$$

This equation holds for all y, so the derivatives of both sides with respect to y are equal. Differentiation with respect to y yields

$$\int_0^{g(y)} \frac{\partial d[f^-(y,\tau)]}{\partial y} d\tau + d[f^-(y,g(y))] \frac{\partial g(y)}{\partial y}$$

$$= \int_{g(y)}^1 \frac{\partial d[f^+(y,\tau)]}{\partial y} d\tau - d[f^+(y,g(y))] \frac{\partial g(y)}{\partial y},$$

or

$$\{d[f^-(y,g(y))] + d[f^+(y,g(y))]\} \frac{\partial g(y)}{\partial y} = \int_{g(y)}^1 \frac{\partial d[f^+(y,\tau)]}{\partial y} d\tau$$

$$- \int_0^{g(y)} \frac{\partial d[f^-(y,\tau)]}{\partial y} d\tau. \qquad (A.16)$$

We have

$$\frac{\partial d[f^+(y,\tau)]}{\partial y} = \frac{\partial d[f^+(y,\tau)]}{\partial f} \frac{\partial f^+(y,\tau)}{\partial y} \tag{A.17}$$

$$\frac{\partial d[f^-(y,\tau)]}{\partial y} = \frac{\partial d[f^-(y,\tau)]}{\partial f} \frac{\partial f^-(y,\tau)}{\partial y} . \tag{A.18}$$

Using (A.13), (A.14), (A.17), and (A.18), we can write (A.16) as

$$\{d[f^-(y,g(y))] + d[f^+(y,g(y))]\} \frac{\partial g(y)}{\partial y} \tag{A.19}$$

$$= r(y) \left\{ \int_{\max\{y,g(y)\}}^{1} \frac{\partial d[f^+(y,\tau)]}{\partial f} d\tau + \int_{0}^{\min\{y,g(y)\}} \frac{\partial d[f^-(y,\tau)]}{\partial f} \right\}$$

Some caution is necessary in using equation (A.19). It holds only at points y for which $\frac{\partial g}{\partial y}$ exists and is continuous. Because r is assumed only piecewise continuous, the derivative $\frac{\partial g}{\partial y}$ is also only piecewise continuous. However, at points of discontinuity of $\frac{\partial g}{\partial y}$ one can use (A.19) to evaluate the left and right derivatives by taking limit of $\frac{\partial g}{\partial y}$ from the left and the right. As a general rule, whenever we write a derivative we implicitly assume its existence. At the equilibrium y* we have y*=g(y*) and $f^-(y^*,g(y^*)) = f^+(y^*,g(y^*)) = 0$, so we obtain

$$2d(0) \frac{\partial g(y^*)}{\partial y} = -r(y^*) \left\{ \int_{y^*}^{1} \frac{d[f^+(y^*,\tau)]}{\partial f} d\tau + \int_{0}^{y^*} \frac{\partial d[f^-(y^*,\tau)]}{\partial f} d\tau \right\} \tag{A.20}$$

It follows from (A.19) and (A.20) that if $r(y^*) > 0$, y^* is a point of continuity of $r(t)$ and $d(0) = 0$ then $\frac{\partial g(y)}{\partial y} \to -\infty$ as $y \to y^*$ so the equilibrium is unstable.

If $d(0) > 0$ the equilibrium is stable, if $-1 < \frac{\partial g(y^*)}{\partial y}$, or equivalently if

$$d(0) > \frac{r(y^*)\left\{\int_{y^*}^{1} \frac{d[f^+(y^*,\tau)]}{\partial f}\, d\tau + \int_{0}^{y^*} \frac{d[f^-(y^*,\tau)]}{\partial f}\, d\tau\right\}}{2} \qquad \text{(A.21)}$$

The quantity *d(0) represents bias,* i.e., length at zero flow. For equilibrium stability it should be larger than the product of input with the integral of the derivative of d along the ring divided by two. Note that the threshold level of bias strongly depends on the level of input both directly and through the term involving the integral of $\frac{\partial d}{\partial f}$.

Equation (A.21) is hardly the entire story on equilibrium stability. If (A.21) is satisfied one is merely guaranteed that the algorithm tends to the equilibrium y^* when started within some interval centered at y^*. This interval can be very small and in fact it is possible that the algorithm diverges from y^* when started outside the interval. A case in point is when the input density r happens to take a very small value at y^* and much larger values at other points. Then (A.21) indicates that very small values of bias can stabilize the equilibrium which is, of course, true but a little thought based on the examples given so far suggests that the algorithm can oscillate violently when started far away from the equilibrium. This leads to investigation of circumstances under which the equilibrium is *globally stable,* i.e., the algorithm converges to it regardless of the starting point.

A sufficient condition for global stability of y^* is that for every y we have

$$|g(y)-y^*| \leq \rho|y-y^*| \tag{A.22}$$

where ρ is same scalar with $0 < \rho < 1$ (i.e., g is a contraction mapping). When (A.22) holds we are guaranteed that the distance to the equilibrium is reduced with every iteration. Equation (A.22) can be guaranteed to hold if we have for some ρ with $0 < \rho < 1$ and *every* y

$$\left|\frac{\partial g(y)}{\partial y}\right| \leq \rho.$$

(This follows from the mean value theorem whereby we have

$$g(y) = g(y^*) + (y-y^*)\,\frac{\partial g(\tilde{y})}{\partial y}$$

where $\tilde{y}$ is a point lying between y and y^*. Hence

$$|g(y)-y^*| = |g(y)-g(y^*)| = \left|(y-y^*)\,\frac{\partial g(\tilde{y})}{\partial y}\right| \leq \rho|y-y^*|.)$$

Now using (A.19) we have

$$\left|\frac{\partial g(y)}{\partial y}\right| = \frac{r(y)\left\{\int_{\max\{y,g(y)\}}^{1} \frac{\partial d[f^+(y,\tau)]}{\partial}d\tau + \int_{0}^{\min\{y,g(y)\}} \frac{\partial d[f^-(y,\tau)]}{\partial}d\tau\right\}}{d[f^-(y,g(y))] + d[f^+(y,g(y))]}.$$

We have since d is monotonically increasing

$$d[f^-(y,g(y))] \geq d(0)$$

$$d[f^+(y,g(y))] \geq d(0).$$

Let r^* be the maximum of the input density

$$r^* = \max_{0 \leq t \leq 1} r(t)$$

Let b be an upper bound to $\frac{\partial d}{\partial f}$ in the range of operation. Since r^* is an upper bound to both $f^+(y,\tau)$ and $f^-(y,\tau)$ a possible (somewhat conservative) upper bound is

$$b = \max_{0 \leq f \leq r^*} \frac{\partial d(f)}{\partial f} \quad .$$

Then we have

$$\left|\frac{\partial g(y)}{\partial y}\right| \leq \frac{r^* b}{2d(0)} \cdot$$

Thus if

$$\frac{r^* b}{2d(0)} < 1$$

or, equivalently,

$$d(0) > \frac{r^* b}{2}$$

the equilibrium y^* is globally stable and the algorithm converges to it from an arbitrary starting point.

Thus, we confirm that *the threshold level of bias for stability of the equilibrium is proportional to a measure of level of traffic input and a measure of level of the derivative of d along the ring.* For local stability of y^* only the level of input at y^* matters but in order to guarantee globally stable behavior of the algorithm the inputs at other points must be taken into account.

Example 1 (Continued): Consider again the case where r(t) = 1 for all t, and d(f) = f for all f. Here

$$y^* = \frac{1}{2}$$

is an (unbiased) equilibrium. From the stability criterion (A.21) we expect that the equilibrium is unstable. Indeed we have already calculated that

$$y_{k+1} = g(y_k) = \begin{cases} y_k^2 - \sqrt{y_k^2-(1-y_k)^2} & \text{if } y_k \geq \frac{1}{2} \\ y_k + \sqrt{(1-y_k)^2-y_k^2} & \text{if } y_k \leq \frac{1}{2} \end{cases}$$

from which it follows that

$$\frac{\partial g(y^*)}{\partial y} = -\infty$$

confirming the instability of the equilibrium. The graph of g together with a typical iteration sequence is shown in Fig. A.28.

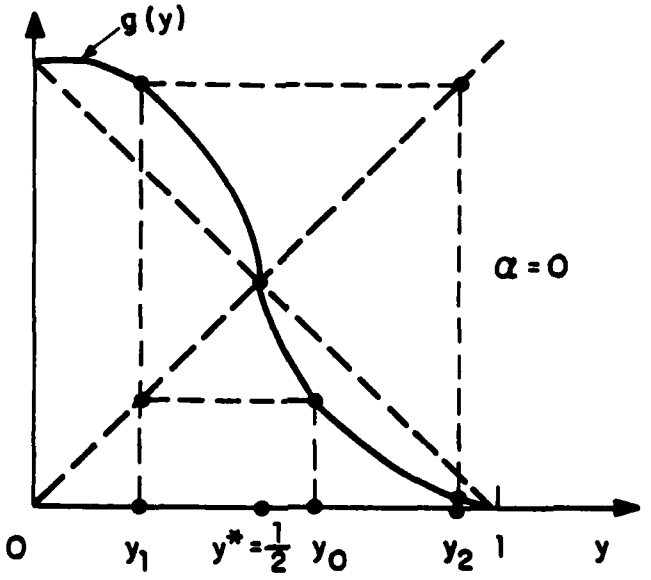


Figure A.28

It can be seen that the algorithm tends to an oscillation between the extreme routings 0 and 1 for every starting point $y_0 \neq y^*$.

Consider now the case

$$d(f) = \alpha + f$$

where α is a positive bias factor. The routing

$$y^* = \frac{1}{2}$$

is again the equilibrium for every $\alpha > 0$ because of symmetry of the input density. A straightforward but lengthy calculation shows that

$$y_{k+1} = g(y_k) = \begin{cases} y_k + 2\alpha - \sqrt{(y_k+2\alpha)^2 - y_k^2 + 2y_k - 1 - 2\alpha} & \text{if } y_k \geq \frac{1}{2} \\ y_k - 2\alpha + \sqrt{(y_k-2\alpha)^2 - y_k^2 - 2y_k + 1 + 2\alpha} & \text{if } y_k \leq \frac{1}{2} \end{cases}$$

From (A.20) or from direct differentiation of the equation above we obtain

$$\frac{\partial g(y^*)}{\partial y} = -\frac{1}{2\alpha} \quad .$$

Thus, the equilibrium is stable for

$$\alpha > \frac{1}{2} \quad .$$

The graph of g in this case together with a typical iteration sequence is given in Fig. A.29. It is in fact possible to show that in this case the equilibrium is globally stable.

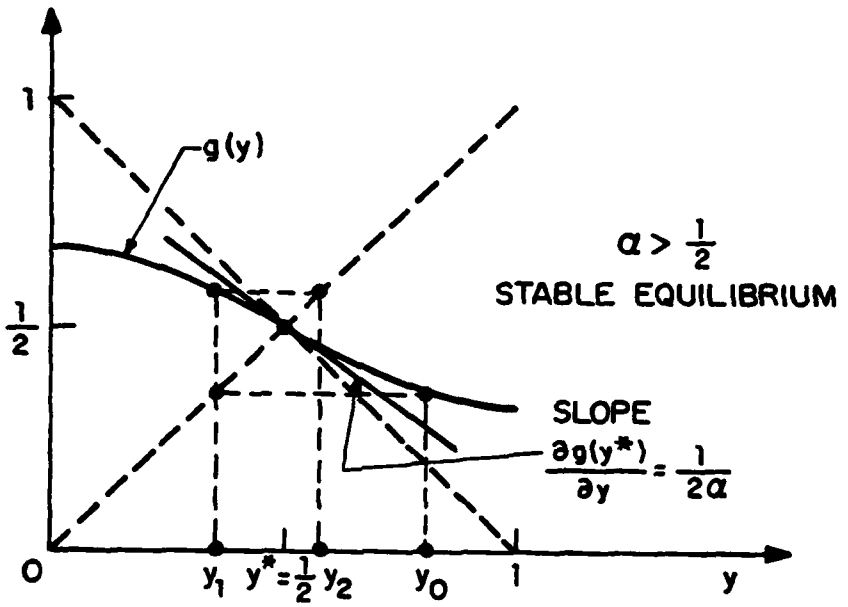


Figure A.29

For

$$0 < \alpha < \frac{1}{2}$$

the equilibrium is unstable. The graph of g together with a typical iteration is shown in Fig. A.30. It can be seen that for every starting point except y* the algorithm will tend to an oscillation between α and $(1-\alpha)$.

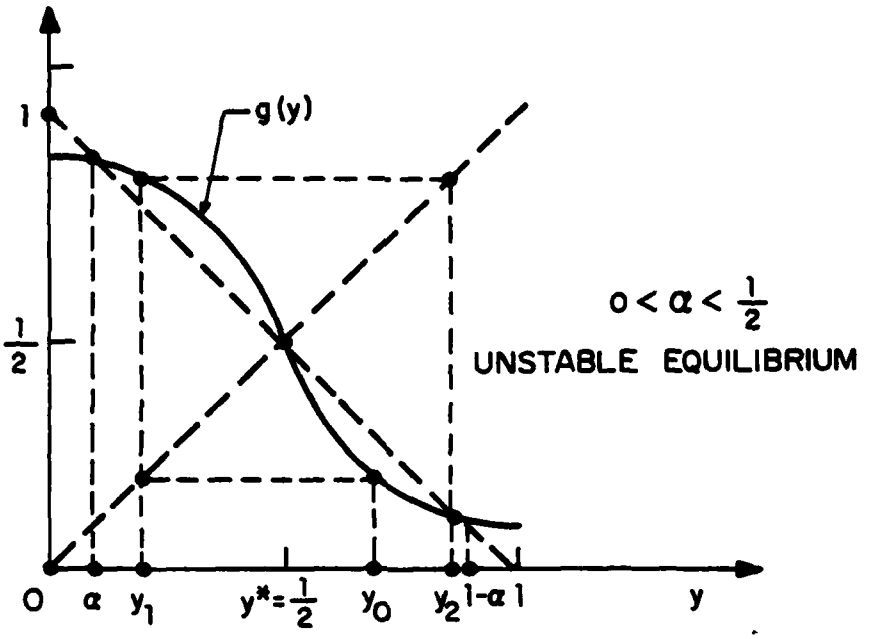


Figure A.30

We now consider an example where the traffic input is not symmetric.

Example 2: Take for $\lambda > 0$ and $\alpha \geq 0$

$$r(t) = \begin{cases} \lambda & \text{for } 0 \leq t \leq \frac{1}{2} \\ 0 & \text{for } \frac{1}{2} < t \leq 0 \end{cases}$$

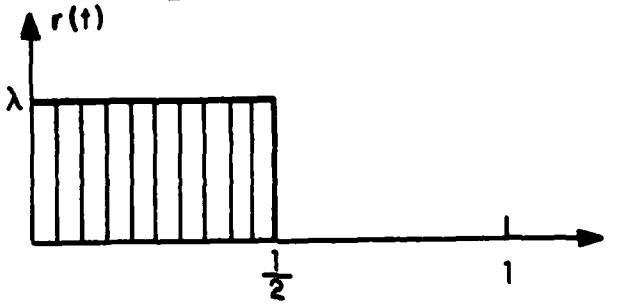


Figure A.31

Take also

$$d(f) = \alpha + f.$$

The equilibrium y^* corresponding to any α is defined from the equation

$$D_0(y^*,y^*) = D_1(y^*,y^*).$$

We have by straightforward calculation

$$D_0(y^*,y^*) = \alpha y^* + \frac{\lambda}{2} y^{*2}$$

$$D_1(y^*,y^*) = \frac{\lambda}{2} y^{*2} - (\alpha+\lambda)y^* + \alpha + \frac{3\lambda}{8} .$$

By equating $D_0(y^*,y^*)$ and $D_1(y^*,y^*)$ we obtain

$$y^* = \frac{8\alpha+3\lambda}{16\alpha+8\lambda} .$$

The threshold for stability can be computed from equation (A.21) to be

$$\alpha > \frac{\lambda}{2} .$$

and the equilibria which are stable are only those corresponding to levels of α above the threshold. Thus while equilibria range from $\frac{3}{8}$ ($\alpha = \frac{\lambda}{2}$) to $\frac{1}{2}$ ($\alpha = \infty$), stable equilibria are only those from $\frac{7}{16}$ ($\alpha = \frac{\lambda}{2}$) to $\frac{1}{2}$ ($\alpha = \infty$).

In the preceding example we see that there is a set of stable equilibria parameterized by the bias level α which in turn ranges from some threshold value to ∞. We also see that while the threshold level of bias depends on the level of traffic λ, the set of stable equilibria does *not* depend on λ. We explore this situation further.

Suppose that an input density $r(t)$ is given and consider input densities of the form

$$r_\lambda(t) = \lambda r(t)$$

where λ is a positive parameter. We would like to investigate the dependence of the set of stable equilibria on the parameter λ (i.e., the traffic level), for the case where the length density is of the form

$$d(f) = \alpha + \hat{d}(f)$$

where $\hat{d}(0) = 0$ and α is a nonnegative bias parameter.

Suppose that $\hat{d}$ is of the form

$$\hat{d}(f) = \beta f^n$$

where β and n are positive scalars. The equation for equilibrium is

$$\alpha y^* + \int_0^{y^*} \hat{d}[\lambda f^-(y^*,\tau)]d\tau = \alpha(1-y^*) + \int_{y^*}^1 \hat{d}[\lambda f^+(y^*,\tau)]d\tau. \qquad (A.24)$$

It is easy to see that if y^* is an equilibrium corresponding to input $r(t)$ and bias α, then y^* is also an equilibrium corresponding

to input $\lambda r(t)$ and bias $\lambda^n \alpha$. On the other hand, the threshold level of bias of (A.21) is also multiplied by λ^n when input changes from $r(t)$ to $\lambda r(t)$. Thus we see that *when $\hat{d}$ is of the form $\hat{d}(f) = \beta f^n$ the set of all equilibria as well as the set of all stable equilibria is independent of the traffic input level λ.*

A careful examination of the case where $\hat{d}(f) = \beta f^n$ shows that the reason for independence of the set of equilibria is that when the input is multiplied uniformly by λ, then $\hat{d}$ is multiplied by λ^n, and $\frac{\partial \hat{d}}{\partial f}$ is multiplied by λ^{n-1}. In other words, the factor of increase of $\hat{d}$ which affects the level of bias necessary to produce a given equilibrium y^* [cf. (A.24)], equals the factor of increase of input times the factor of increase of $\frac{\partial d}{\partial f}$, i.e., the factor of increase of threshold bias level [cf. (A.21)].

Carrying this reasoning one step further, consider a fixed y^* and consider for each $\lambda > 0$ the value of $\alpha_{y^*}(\lambda)$ of bias which produces y^* as equilibrium. The function $\alpha_{y^*}(\lambda)$ is of the general form indicated in Fig. A.32 and is obtained by solving (A.24) for α.

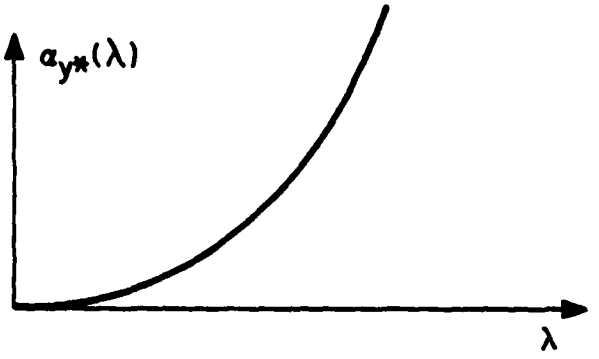


Figure A.32

Consider also the expression

$$T_{y^*}(\lambda) = \frac{\lambda r(y^*) \int_{y^*}^{1} \frac{\partial d[\lambda f^+(y^*,\tau)]}{\partial f} d\tau + \int_{0}^{y^*} \frac{\partial d[\lambda f^-(y^*,\tau)]}{\partial f} d\tau}{2} \quad (A.25)$$

where $f^+(y^*,\tau)$ and $f^-(y^*,\tau)$ are the flows in the positive and negative direction corresponding to y^* and $\lambda = 1$.

If

$$\alpha_{y^*}(\lambda) > T_{y^*}(\lambda),$$

then y^* is a stable equilibrium at input level λ. If

$$\alpha_{y^*}(\lambda) < T_{y^*}(\lambda),$$

then y^* is an unstable equilibrium at input level λ. Note however, that $\alpha_{y^*}(\lambda)$ grows roughly as $\hat{d}(\lambda f)$ grows with λ. On the other hand, $T_{y^*}(\lambda)$ grows roughly as $\lambda \frac{\partial \hat{d}(\lambda f)}{\partial f}$ grows with λ. Thus, *if* $\lambda \frac{\partial \hat{d}(\lambda f)}{\partial f}$ *grows with* λ *faster than* $\hat{d}(\lambda f)$ *any given equilibrium* y^* *may become unstable for sufficiently high input level* λ. For example, this is true if

$$\hat{d}(f) = e^f$$

when

$$\hat{d}(\lambda f) = e^{\lambda f}, \quad \lambda \frac{\partial \hat{d}(\lambda f)}{\partial f} = \lambda^2 e^{\lambda f} .$$

This observation is particularly relevant if one contemplates to take the actual average delay as the function $\hat{d}$. As the level of input increases every candidate equilibrium may be destabilized if the marginal delay increases much faster than delay as link utilization is increased. To put the same argument in other words, for large levels of input the level of bias necessary to stabilize the algorithm may become very large due to large marginal delay with a routing very close to min-hop resulting. Thus, in order to stabilize the algorithm one may have to relinquish adaptation when it is most needed (i.e., when the network is congested), and this suggests that some thought should be given to choices of $\hat{d}$ other than average delay.

A.4.1 Choosing the Bias as a Function of the Current Routing

We have dealt with the case

$$d(f) = \alpha + \hat{d}(f)$$

where α is a constant bias. Since values of α which bring the algorithm closer to an equilibrium depend strongly on the traffic input as well as the current routing it is of interest to consider the case where bias is selected on the basis of current information, for example, flows, delays, etc. We are thus led to consider a length density function of the form

$$d(f,y) = \alpha(y) + \hat{d}(f)$$

where $\hat{d}(0) = 0$. An analysis that closely parallels the earlier one shows that given an input density function $r(t)$ and routing y_k, the next routing y_{k+1} is given by an equation of the form

$$y_{k+1} = g(y_k)$$

where g(y) is a function defined from the equation [cf. (A.3) through (A.5)]

$$\alpha(y)g(y) + \int_0^{g(y)} \hat{d}[f^-(y,\tau)]d\tau = \alpha(y)[1-g(y)] + \int_{g(y)}^1 \hat{d}[f^+(y,\tau)]d\tau$$

where the positive and negative flows f^+, f^- are given by (A.1) and (A.2). Differentiation with respect to y yields the analog of (A.19) which is

$$\{2\alpha(y)+\hat{d}[f^-(y,g(y))]+\hat{d}[f^+(y,g(y)]\} \frac{\partial g(y)}{\partial y} =$$

$$= -r(y)\left\{ \int_{\max\{y,g(y)\}}^1 \frac{\partial d[f^+(y,\tau)]}{\partial f} d\tau + \int_0^{\min\{y,g(y)\}} \frac{\partial d[f^-(y,\tau)]}{\partial f} d\tau \right\}$$

$$+ [1-2g(y)] \frac{\partial \alpha(y)}{\partial y} . \qquad \text{(A.27)}$$

At an equilibrium we have $\hat{d}[f^-(y^*,g(y^*))] = \hat{d}[f^+(y^*,g(y^*))] = 0$ so that for equilibrium stability we must have [cf. (A.20)]

$$\alpha(y^*) > \frac{r(y^*)\left\{ \int_{y^*}^1 \frac{\partial \hat{d}[f^+(y^*,\tau)]}{\partial f} d\tau + \int_0^{y^*} \frac{\partial \hat{d}[f^-(y^*,\tau)]}{\partial f} d\tau \right\} - (1-2y^*) \frac{\partial \alpha(y^*)}{\partial y}}{2}$$

It follows from these formulas that stability of the algorithm will be improved by selecting $\alpha(y)$ so that it increases with level of input and level of marginal delay. It is also improved by selecting $\alpha(y)$ so that

$$[1-2g(y)]\,\frac{\partial\alpha(y)}{\partial y} > 0.$$

A reasonable choice of $\alpha(y)$ is to take it to be some monotonically increasing function of

$$D_T(y) = \int_0^1 \hat{d}[f^+(y,\tau)]d\tau + \int_0^1 \hat{d}[f^-(y,\tau)]d\tau$$

for example, a quadratic function of the form

$$\alpha(y) = \lambda_0 + \lambda_1 D_T(y) + \lambda_2 [D_T(y)]^2$$

with coefficients λ_0, λ_1, λ_2 determined experimentally. In the context of the SPF algorithm $D_T(y)$ represents the current sum of all reported link delays, so that the quadratic rule or a piecewise constant approximation of it for selecting $\alpha(y)$ can be fairly easily implemented.

We note that it is possible to make bias different for each t (or for each link in the communication network context). It is possible to analyze this case along the preceding lines but it appears that no useful design guidelines can be deduced. A procedure that is definitely inappropriate is to make the bias of highly loaded links large, and the bias of lightly loaded links small. This results in aggravating oscillations of traffic from the highly loaded links to the lightly loaded links. The reverse procedure whereby heavily loaded links receive small bias and lightly loaded links a large bias tends to stabilize the algorithm in a congested state. While it may be possible to device good schemes for assigning a different bias to each link, such schemes are not obvious and simple-minded schemes don't seem to be appropriate.

A.4.2 Basing the Routing Decision on More than one Past Routings

In the algorithm considered so far the routing y_{k+1} depends exclusively on the earlier routing y_k and we saw that it is necessary to introduce damping into the algorithm in the form of bias in order to avoid instability. We also saw that high values of bias cause an undesirable equilibrium tendency towards the min-hop routing while at the same time such high values may be necessary to prevent instability. This motivates a search for ways to reduce the level of bias necessary to stabilize the algorithm. This can be accomplished by various forms of *averaging the effects of past routing decisions*, some of which we describe shortly. In the context of practical routing algorithms, such averaging techniques are easy to implement. In fact, one of the averaging techniques is based on *asynchronous and even random delay reporting of nodes and corresponding routing updating*. Such operation is easier to implement than the one that requires synchronous delay reporting and routing updating of nodes.

For simplicity, we write

$$d(f) = \alpha + \hat{d}(f) \qquad \text{(A.30)}$$

where $\hat{d}(0) = 0$, and we refer to $\hat{d}$ as delay.

Averaging Algorithm 1 (Synchronous Delay Reporting):

This algorithm is the same as the earlier one except that distances to the destination in the clockwise and counterclockwise direction are obtained by averaging delays over several routings. Specifically, for a fixed positive integer n, and given a sequence of past routings y_k, y_{k-1}, $\cdots$, we define for any t in [0,1] "averaged" distances to 0 and 1 by

$$\tilde{D}_0(y_k,y_{k-1},\cdots,y_{k-n},t) = \frac{1}{n+1}\sum_{i=0}^{n} D_0(y_{k-i},t)$$

$$\tilde{D}_1(y_k,y_{k-1},\cdots,y_{k-n},t) = \frac{1}{n+1}\sum_{i=0}^{n} D_1(y_{k-i},t).$$

We can also write using (A.3), (A.4), and (A.30)

$$\tilde{D}_0(y_k,y_{k-1},\cdots,y_{k-n},t) = \alpha t + \int_0^t \frac{1}{n+1}\sum_{i=0}^{n} \hat{d}[f^-(y_{k-i},\tau)]d\tau \qquad (A.31)$$

$$\tilde{D}_1(y_k,y_{k-1},\cdots,y_{k-n},t) = \alpha(1-t) + \int_t^1 \frac{1}{n+1}\sum_{i=0}^{n} \hat{d}[f^+(y_{k-i},\tau)]d\tau. \qquad (A.32)$$

Thus distances are calculated by integrating $\frac{1}{n+1}\sum_{i=0}^{n}\hat{d}[f(y_{k-i},\tau)]$, which is an averaged delay over the routings $y_k,\cdots,y_{k-n}$, in place of $\hat{d}[f(y_k,\tau)]$ which is the delay corresponding to the last routing.

The new routing y_{k+1} is obtained from the equation

$$\tilde{D}_0(y_k,y_{k-1},\cdots,y_{k-n},y_{k+1}) = \tilde{D}_1(y_k,y_{k-1},\cdots,y_{k-n},y_{k+1}). \qquad (A.33)$$

This defines uniquely y_{k+1} in terms of $y_k,y_{k-1},\cdots,y_{k-n}$. As earlier we write the corresponding equation as

$$y_{k+1} = g(y_k,y_{k-1},\cdots,y_{k-n}).$$

A routing y^* is said to be an *equilibrium* if

$$y^* = g(y^*,y^*,\cdots,y^*).$$

It is clear that *y^* is an equilibrium in this sense for a given bias level α, if and only if it is an equilibrium for the same α in the sense given earlier in this section.*

The equilibrium y^* is stable if it is also a stable equilibrium of equation (A.34) linearized around y^*. It is a known fact that this is true if all roots of the characteristic polynomial

$$C(p) = p^{n+1} - \frac{\partial g(y^*)}{\partial y_k} p^n - \frac{\partial g(y^*)}{\partial y_{k-1}} p^{n-1} - \cdots - \frac{\partial g(y^*)}{\partial y_{k-n+1}} p - \frac{\partial g(y^*)}{\partial y_{k-n}}$$

lie inside the unit circle (i.e., have modulus less than unity). We calculate the derivatives $\frac{\partial g}{\partial y_{k-i}}$.

We have for $\alpha > 0$ similarly as earlier for every i

$$\frac{\partial g(y^*)}{\partial y_{k-i}} = - \frac{r(y^*)}{2\alpha} \frac{1}{n+1} \left\{ \int_{y^*}^{1} \frac{\partial d[f^+(y^*,\tau)]}{\partial f} d\tau + \int_{0}^{y^*} \frac{\partial \hat{d}[f^-(y^*,\tau)]}{\partial f} d\tau \right\} .$$

Define

$$\mu = \frac{r(y^*)}{2\alpha} \left\{ \int_{y^*}^{1} \frac{\partial \hat{d}[f^+(y^*,\tau)]}{\partial f} d\tau + \int_{0}^{y^*} \frac{\partial \hat{d}[f^-(y^*,\tau)]}{\partial f} d\tau \right\} . \qquad (A.35)$$

The characteristic polynomial can be written as

$$C(p) = p^{n+1} + \frac{\mu}{n+1} p^n + \frac{\mu}{n+1} p^{n-1} + \cdots + \frac{\mu}{n+1} p + \frac{\mu}{n+1} .$$

We now use the following fact:

Lemma: Let α be a positive scalar and n be a positive integer. The roots of the polynomial

$$p^{n+1} + \alpha p^n + \alpha p^{n-1} + \cdots + \alpha p + \alpha$$

lie inside the unit circle if and only if $\alpha < 1$.

Proof: This result is probably well known, so we only sketch a proof. Assume first that $\alpha < 1$. Then the statement that the polynomial has all roots inside the unit circle is equivalent to the system

$$y_{k+1} = -\alpha y_k - \alpha y_{k-1} - \cdots - \alpha y_{k-n} \tag{A.36}$$

being asymptotically stable. Thus it will suffice to show that every sequence $\{y_k\}$ generated by the system converges to zero. The system (A.36) can also be written as

$$y_{k+1} = (1-\alpha)y_k + \alpha y_{k-n-1}. \tag{A.37}$$

Since $0 < \alpha < 1$, we have for all k

$$|y_{k+1}| \leq (1-\alpha)\,|y_k| + \alpha |y_{k-n-1}|. \tag{A.38}$$

Let

$$\delta_k = \max\{|y_k|, |y_{k-1}|, \cdots, |y_{k-n-1}|\}.$$

From (A.38) we have $|y_{k+1}| \leq \delta_k$. It follows that $\{\delta_k\}$ is a non-increasing sequence, and hence converges to a limit δ. If $\delta = 0$ we are done, so assume $\delta > 0$. For an $\varepsilon > 0$, let $\bar{k}$ be such that

$|y_k| \leq \delta + \varepsilon$ for all $k \geq \bar{k} - n$. Take ε sufficiently small so that from (A.37) it follows that if $\delta - \varepsilon \leq |y_{k+1}| \leq \delta + \varepsilon$ then y_k and y_{k-n-1} have the same sign as y_{k+1} and choose $\tilde{k} \geq \bar{k}$ such that $\delta - \varepsilon \leq |y_{\tilde{k}+1}| \leq \delta + \varepsilon$. It follows from (A.37) that

$$|y_{\tilde{k}+1}| \leq \max\{|y_{\tilde{k}}|, |y_{\tilde{k}-n-1}|\} \leq \delta + \varepsilon$$

while

$$\min\{|y_{\tilde{k}}|, |y_{\tilde{k}-n-1}|\} \geq \frac{1}{1-\alpha} |y_{\tilde{k}+1}| - \frac{\alpha}{1-\alpha} \max\{|y_{\tilde{k}}|, |y_{\tilde{k}-n-1}|\}$$

$$\geq \frac{\delta-\varepsilon}{1-\alpha} - \frac{\alpha}{1-\alpha} (\delta+\varepsilon) = \delta - \frac{1+\alpha}{1-\alpha} \varepsilon.$$

so finally

$$\delta - \frac{1+\alpha}{1-\alpha} \varepsilon \leq |y_{\tilde{k}}| \leq \delta + \varepsilon,$$

and $y_{\tilde{k}}$ has the same sign as $y_{\tilde{k}+1}$. Repeating the argument it follows that, for ε sufficiently small, $y_{\tilde{k}}$, $y_{\tilde{k}-1}$, $y_{\tilde{k}-2}$, $\cdots$, y_{k-n} have the same sign as $y_{\tilde{k}+1}$. This contradicts (A.36) and the result is proved.

The reverse statement of the proposition follows by showing that for $\alpha \geq 1$ the system (A.36) is not asymptotically stable via a similar argument Q.E.D.

We now apply the result of the lemma to our problem. We have that the equilibrium y^* will be stable if and only if

$$\mu < n + 1.$$

It follows using (A.35) that in the averaged algorithm the bias level must satisfy

$$\alpha > \frac{r(y^*)\left\{\int_{y^*}^{1} \frac{\partial \hat{d}[f^+(y^*,\tau)]}{\partial f}\, d\tau + \int_{0}^{y^*} \frac{\partial \hat{d}[f^-(y^*,\tau)]}{\partial f}\, d\tau\right\}}{2(n+1)}$$

in order for the corresponding equilibrium y^* to be stable. If we compare this with the earlier algorithm [cf. (A.21)], we see that *in the averaged algorithm the bias threshold level for stability is reduced by the factor* $\frac{1}{n+1}$ *over the nonaveraged algorithm.* Thus for n = 1 the reduction is 50%, for n = 2 the reduction is 66.6%, for n = 3 it is 75%, etc. For a given traffic input, and any given bias level, the corresponding equilibrium can be made stable by averaging delays over a sufficiently large number of periods. Note, however, that the threshold level for bias, even if reduced, is still proportional to level of traffic input and level of marginal delay.

In conclusion, it appears that averaging over past delays offers significant advantages over the nonaveraged algorithm. If the traffic input statistics are stationary over long periods, then the averaging scheme given here or some other similar scheme (such as a fading memory scheme) is worth adopting. The only disadvantage of averaging is reduced sensitivity to sudden traffic input changes (see the discussion of Sec. A.4.3).

Averaging Algorithm 2 (Asynchronous Delay Reporting):

Here we consider the following algorithm patterned after an envisioned asynchronous operation of the SPF algorithm. The set of "nodes" [0,1] is partitioned into n subsets which we call S_1, S_2, $\cdots$, S_n. At some time, say 0, the nodes in S_1 report their delays averaged over the preceding n routings and a routing update takes place. Then at time $\sigma_1 > 0$ the nodes in S_2 do the same thing. Similarly, for $i = 1, \cdots, n-1$, at time $(\sigma_1+\sigma_2+\cdots+\sigma_i)$ the nodes in S_{i+1} do the same thing. At time $(\sigma_1+\sigma_2+\cdots+\sigma_n)$ the nodes in S_1 again report their delays, an updating takes place and the process is repeated.

The modeling of this routing updating process can best be understood for the case where $n = 2$. For normalization purposes we assume that

$$\sigma_1 + \sigma_2 = 1.$$

Assume that odd-indexed routings y_{2k+1} correspond to delay reports of nodes in S_1, and even-indexed routings y_{2k+2} correspond to delay reports of nodes in S_2 as shown in Fig. A.33.

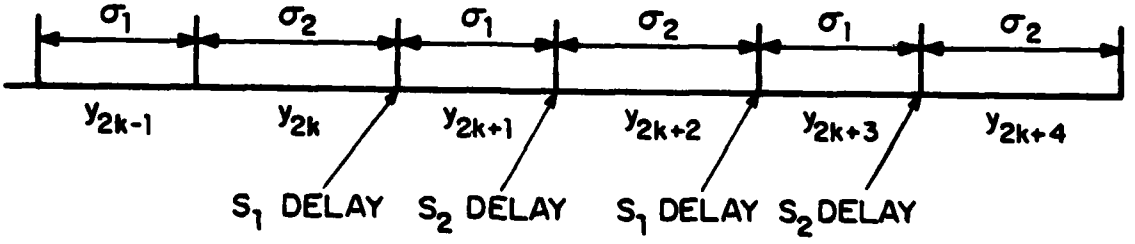


Figure A.33

The routing y_{2k+1} is generated (after the nodes in S_1 report their delays) according to the equation

$$\alpha y_{2k+1} + \int_0^{y_{2k+1}} \{\sigma_1 \hat{d}[f^-(y_{2k-1},\tau)] + \sigma_2 \beta(t) \hat{d}[f^-(y_{2k},\tau)] + \quad (A.35)$$

$$+ \sigma_2 [1-\beta(t)] \hat{d}[f^-(y_{2k-2},\tau)]\} d\tau =$$

$$= \alpha(1-y_{2k+1}) + \int_{y_{2k+1}}^{1} \{\sigma_1 \hat{d}[f^+(y_{2k-1},\tau)] + \sigma_2 \beta(t) \hat{d}[f^+(y_{2k},\tau)] +$$

$$+ \sigma_2 [1-\beta(t)] \hat{d}[f^+(y_{2k-2},\tau)]\} d\tau$$

where

$$\beta(t) = \begin{cases} 1 & \text{if } \tau \text{ belongs to } S_1 \\ 0 & \text{if } \tau \text{ belongs to } S_2. \end{cases}$$

This equation is obtained by observing that for every node τ in S_1 the "reported" delays are

$$\sigma_2 \hat{d}[f^-(y_{2k},\tau)] + \sigma_1 \hat{d}[f^-(y_{2k-1},\tau)]$$

$$\sigma_2 \hat{d}[f^+(y_{2k},\tau)] + \sigma_1 \hat{d}[f^+(y_{2k},\tau)]$$

while for every node in S_2 the current delays (reported at the preceding period) are

$$\sigma_1\hat{d}[f^-(y_{2k-1},\tau)] + \sigma_2\hat{d}[f^-(y_{2k-2},\tau)]$$

$$\sigma_1\hat{d}[f^+(y_{2k-1},\tau)] + \sigma_2\hat{d}[f^+(y_{2k-2},\tau)].$$

Similarly, the routing y_{2k+2} is generated (after the nodes in S_2 report their delays) according to the equation

$$\alpha y_{2k+2} + \int_0^{y_{2k+2}} \{\sigma_2\hat{d}[f^-(y_{2k},\tau)] + \sigma_1[1-\beta(\tau)]\hat{d}[f^-(y_{2k+1},\tau)] + \sigma_1\beta(\tau)\hat{d}[f^-(y_{2k-1},\tau)]\}d\tau = \quad (A.36)$$

$$= \alpha(1-y_{2k+2}) + \int_{y_{2k+2}}^1 \{\sigma_2\hat{d}[f^+(y_{2k},\tau)] + \sigma_1[1-\beta(\tau)]\hat{d}[f^+(y_{2k+1},\tau)] + \sigma_1\beta(\tau)\hat{d}[f^+(y_{2k-1},\tau)]\}d\tau.$$

Let $g_1(y_{2k}, y_{2k-1}, y_{2k-2})$ be the solution for y_{2k+1} of (A.35) and $g_2(y_{2k+1}, y_{2k}, y_{2k-1})$ be the solution for y_{2k+2} of (A.36). Then the algorithm can be described compactly as

$$y_{2k+2} = g_2(y_{2k+1}, y_{2k}, y_{2k-1}),$$

$$y_{2k+1} = g_1(y_{2k}, y_{2k-1}, y_{2k-2}).$$

The equations above represent a discrete-time nonlinear system which is periodic with period two. It can be converted into a stationary system by writing it in the form

$$y_{2k+2} = g_2[g_1(y_{2k}, y_{2k-1}, y_{2k-2}), y_{2k}, y_{2k-1}]$$

$$y_{2k+1} = g_1(y_{2k}, y_{2k-1}, y_{2k-2}) \qquad \text{(A.37)}$$

$$y_{2k} = y_{2k}$$

$$y_{2k-1} = y_{2k-1}.$$

A routing y^* is said to be an equilibrium if

$$y^* = g_2(y^*, y^*, y^*) = g_1(y^*, y^*, y^*).$$

Note that the equilibria in the sense of the synchronous algorithm are also the equilibria of the asynchronous algorithm. It is a known fact that in order for an equilibrium y^* to be locally stable, it is sufficient that the system (A.37) linearized at y^* be stable. Equivalently the matrix

$$M = \begin{bmatrix} \dfrac{\partial g_2}{\partial y_{2k+1}} \dfrac{\partial g_1}{\partial y_{2k}} + \dfrac{\partial g_2}{\partial y_{2k}} & \dfrac{\partial g_2}{\partial y_{2k+1}} \dfrac{\partial g_1}{\partial y_{2k-1}} + \dfrac{\partial g_2}{\partial y_{2k-1}} & \dfrac{\partial g_2}{\partial y_{2k+1}} \dfrac{\partial g_1}{\partial y_{2k-2}} & 0 \\ \dfrac{\partial g_1}{\partial y_{2k}} & \dfrac{\partial g_1}{\partial y_{2k-1}} & \dfrac{\partial g_1}{\partial y_{2k-2}} & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

must have all its eigenvalues within the unit circle.

We now compute the derivatives in the matrix above. We have similarly as in earlier derivations

$$\frac{\partial g_1}{\partial y_{2k}} = \frac{r(y^*)}{2\alpha}\left\{\int_{y^*}^{1} \sigma_2\beta(\tau)\ \frac{\partial\hat{d}[f^+(y^*,\tau)]}{\partial f}\,d\tau + \int_{0}^{y^*} \sigma_2\beta(\ )\ \frac{\partial\hat{d}[f^-(y^*,\tau)]}{\partial f}\,d\tau\right\}$$

$$\frac{\partial g_1}{\partial y_{2k-1}} = -\frac{r(y^*)}{2\alpha}\left\{\int_{y^*}^{1} \sigma_1\ \frac{\partial\hat{d}[f^+(y^*,\tau)]}{f}\,d\tau + \int_{0}^{y^*} \sigma_1\ \frac{\partial\hat{d}[f^-(y^*,\tau)]}{\partial f}\,d\tau\right\}$$

$$\frac{\partial g_1}{\partial y_{2k-2}} = -\frac{r(y^*)}{2\alpha}\left\{\int_{y^*}^{1} \sigma_2[1-\beta(\tau)]\ \frac{\partial\hat{d}[f^+(y^*,\tau)]}{\partial f} + \right.$$

$$\left. + \int_{0}^{y^*} \sigma_2[1-\beta(\tau)]\ \frac{\partial\hat{d}[f^-(y^*,\tau)]}{\partial f}\,d\tau\right\}$$

$$\frac{\partial g_2}{\partial y_{2k+1}} = -\frac{r(y^*)}{2\alpha}\left\{\int_{y^*}^{1} \sigma_1[1-\beta(\tau)]\ \frac{\partial\hat{d}[f^+(y^*,\tau)]}{\partial f}\,d\tau + \right.$$

$$\left. + \int_{0}^{y^*} \sigma_1[1-\beta(\tau)]\ \frac{\partial\hat{d}[f^-(y^*,\tau)]}{\partial f}\,d\tau\right\}$$

$$\frac{\partial g_2}{\partial y_{2k}} = -\frac{r(y^*)}{2\alpha}\left\{\int_{y^*}^{1} \sigma_2\ \frac{\partial\hat{d}[f^+(y^*,\tau)]}{\partial f}\,d\tau + \int_{0}^{y^*} \sigma_2\ \frac{\partial\hat{d}[f^-(y^*,\tau)]}{\partial f}\,d\tau\right\}$$

$$\frac{\partial g_2}{\partial y_{2k-1}} = -\frac{r(y^*)}{2\alpha}\left\{\int_{y^*}^{1} \sigma_1\beta(\tau)\ \frac{\partial\hat{d}[f^+(y^*,\tau)]}{\partial f}\,d\tau + \right.$$

$$\left. + \int_{0}^{y^*} \sigma_1\beta(\tau)\ \frac{\partial\hat{d}[f^-(y^*,\tau)]}{\partial f}\,d\tau\right\}$$

Define

$$\mu = \frac{r(y^*)}{2\alpha}\left\{\int_{y^*}^{1} \frac{\partial \hat{d}[f^+(y^*,\tau)]}{\partial f}\, d\tau + \int_{0}^{y^*} \frac{\partial \hat{d}[f^-(y^*,\tau)]}{\partial f}\, d\tau\right\}$$

$$\gamma = \frac{r(y^*)}{2\alpha\mu}\left\{\int_{y^*}^{1} \beta(\tau)\, \frac{\partial \hat{d}[f^+(y^*,\tau)]}{\partial f}\, d\tau + \int_{0}^{y^*} \beta(\tau)\, \frac{\partial \hat{d}[f^-(y^*,\tau)]}{\partial f}\, d\tau\right\}$$

and denote

$$\sigma = \sigma_1.$$

Then the matrix M can be written as

$$M=\begin{bmatrix} \sigma(1-\sigma)\gamma(1-\gamma)\mu^2-(1-\sigma)\mu & \sigma^2\gamma(1-\gamma)\mu^2-\sigma\gamma\mu & \sigma(1-\sigma)(1-\gamma)^2\mu^2 & 0 \\ -(1-\sigma)\gamma\mu & -\sigma\mu & -(1-\sigma)(1-\gamma)\mu & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

It is possible to show that if $\mu < 1$ then this matrix has all its eigenvalues within the unit circle so that if an equilibrium within the unit circle so that if an equilibrium is stable in the synchronous algorithm, then it is also stable in the asynchronous algorithm, so that *in the asynchronous algorithm the threshold level of bias for stability is reduced.* This fact can also be proved in the general case where $n > 2$. The extent to which asynchronous operation improves the stability properties of the algorithm is not clear as yet. From worked out examples, however, it appears that *the reduction in threshold*

bias level is substantial and even when the equilibrium is unstable, the associated oscillatory behavior is not nearly as violent as in the synchronous algorithm.

The conclusion is that asynchronous operation involves a form of averaging that has a pronounced beneficial effect on the dynamic behavior of the algorithm. Since it is also easier to mechanize than synchronous node broadcasting, it should clearly be adopted in the routing algorithm. Asynchronous operation can also be combined with averaging over several past routings by using a *fading memory scheme* for measuring average delay whereby the delays of most recent packets are weighted more heavily over previcus packets. A scheme of this form is easy to implement and from preliminary analysis along the lines of this section, as well as computational examples it appears to be most suitable in terms of improving the dynamic behavior of the algorithm.

A.4.3. Time-Varying Input Statistics and Speed of Convergence

The preceding analysis was based on the assumption that the traffic input statistics are stationary and concentrated on investigation of conditions under which the algorithm converges or diverges to an equilibrium. When input statistics change with time, the performance of the algorithm depends strongly on its speed of convergence. Roughly speaking, the algorithm should converge faster than the input statistics change in order to be able to track well the time-varying equilibrium. The speed of convergence of the algorithm depends on the value of $\left|\frac{\partial g}{\partial y}\right|$ for the first case examined in this section, and on the value of the maximum eigenvalue modulus of the corresponding matrix in the other cases. The smaller this value is,

the faster the algorithm converges. Thus it can be seen that large values of bias and greater degrees of averaging not only improve the stability properties of the algorithm but also enhance its speed of convergence. On the other hand, an algorithm with large degree of averaging utilizes heavily delay information that is obsolete in a time-varying input statistics context. These two factors tend to counteract each other and the performance of such algorithms in the presence of variations of input statistics is as yet unclear. Computational results provided in a subsequent section indicate that by increasing the degree of averaging one does not necessarily diminish the speed of response to a sudden change in input traffic.

A.5 EXTENSIONS OF THE CONTINUOUS MODEL

The extension of the continuous node model of the preceding section to the case of an arbitrary network is quite straightforward and will only be sketched in an informal manner. Consider first the case of an undirected network with a single destination, and let r be the input density function mapping points on the undirected links of the network to the nonnegative real numbers. The meaning of r again is that given any interval I on a link the input originating at this interval is the integral of r over I. One may view r as L separate functions, where L is the number of undirected links, one for each link. We assume that each of these functions is piecewise continuous. In order to consider notions of length we associate with each undirected link (i,l) two directions $i \to l$ and $l \to i$. A length function δ is a function which assigns to each point on an undirected link (i,l) two nonnegative numbers, one associated with the direction $i \to l$ and the other associated with the direction $l \to i$. We assume that δ is piecewise continuous on every link in each direction. The meaning of δ is that given any two points on a link (i,l) their distance in the direction $i \to l$ is the integral between the two points of δ as defined in that direction. The distance in the opposite direction $l \to i$ is defined analogously. Similarly, we can consider paths between points on possibly different links and define their length in one or the other direction.

We now associate to a given length function δ a shortest path of every point, and an associated routing. To simplify matters, we assume that δ is everywhere positive. This is actually a very mild assumption as will become evident from

what follows. Given any point we consider the collection of paths to the destination and their associated distances specified by the function δ. A path of minimum distance is referred to as a shortest path from the point to the destination and the corresponding distance is referred to as the shortest distance of the point to the destination. The *routing* corresponding to δ is the set of points for which there are more than one equidistant paths to the destination. A routing is said to be *regular* if it does not contain any nodes of the network (in the ordinary sense), otherwise, it is said to be *singular*.

Given the function δ, a shortest path of each point and the corresponding routing can be constructed in a simple manner along similar lines as for usual networks. We first construct a shortest path tree for the network in the usual manner by using as (directed) link lengths those specified by the length function δ. (The length of the directed link (i,l) is the integral of δ along (i,l) in the direction $i \to l$). This gives us a shortest path and the associated shortest distance for every point on the shortest path tree including all the nodes of the network. A shortest path for points on links that are not part of the shortest path tree can be obtained as follows:

Let (i,l) be a link that is not on the tree. Let D_i and D_l be the shortest distances of nodes i and l. The shortest distance of a point t on (i,l) is

$$D(t) = \min\left\{ D_i + \int_t^i \delta_{il}(\tau)d\tau,\ D_l + \int_t^l \delta_{il}(\tau)d\tau \right\} \qquad (A.38)$$

where δ_{1i} is δ in the direction $1 \to i$ and δ_{i1} is δ in the direction $i \to 1$. It can be seen that the routing corresponding to δ is regular if and only if each (ordinary) node of the network has only one shortest path associated with it. If a routing is regular than every one of its points lies in the "interior" of some link. In what follows, we restrict ourselves to regular routings thereby considerably simplifying the analysis. Notice that the preceding construction shows that *a routing (regular or not) consists of a finite number of points*. The number is equal to the number of undirected links which are not on the shortest path tree.

Given a shortest path "tree" in the generalized sense described earlier we can define the flow corresponding to it. At each point, say t, of a link (i,1) there are two flows to consider (one of which is zero in the single destination case); the flow in the direction $i \to 1$ and the flow in the direction $1 \to i$. Each is defined in the natural way by integrating the input density function r over the portion of the network that lies "upstream" from the point t, i.e., over the set of points the shortest paths of which meet t on their way to the destination. Notice that at the finite number of points which form the routing, the flow is zero in either direction.

Suppose now we are given a monotonically increasing, continuously differentiable function d mapping flow into the positive numbers. Given a shortest path tree corresponding to a length function δ with routing Y we can define a new length $\bar{\delta}$ which assigns to points t in any one of the two possible directions the length $\bar{\delta}(t) = d[f(t)]$ where f(t) is the flow at t corresponding to δ in the appropriate direction. The corresponding routing is denoted $\bar{Y}$.

We are now in a position to define an algorithm similar to the one of the previous section. Given a length function δ_0 and a corresponding routing Y_0, the next length function is $\delta_1 = \overline{\delta}_0$ with corresponding routing $Y_1 = \overline{Y}_0$. Similarly, we define δ_2 and Y_2, and, for all k, δ_k and Y_k.

We say that a routing Y* corresponding to a length function δ^* is an *equilibrium routing* if $\overline{\delta}^* = \delta^*$ and $\overline{Y}^* = Y^*$.

Given a regular equilibrium routing $Y^* = \{y_1^*, y_2^*, \cdots, y_n^*\}$ consider for $j = 1,2,\cdots,n$ the undirected link (i_j, l_j) containing y_j^* and the two shortest paths from y_j^* to the destination. A simple but fundamental observation is that *these two paths meet at some point thereby forming a ring of the type considered in the previous section.* For $j = 1,2,\cdots,n$ we parameterize points on the ring containing y_j^* by the number in [0,1] going from smaller to larger numbers as we traverse the ring in the counterclockwise direction similarly as in the previous section. Thus points v_κ on the link (i_j, l_j) can and will be identified by the number in [0,1] specifying their position on the ring corresponding to y_j^*. It is easy to see now that, given Y*, any collection $Y = \{y_1, y_2, \cdots, y_n\}$ such that y_j lies in the interior of (i_j, l_j) specifies a flow f_Y through each point in the network that follows the (ordinary) shortest path tree corresponding to δ^* and Y*, and on each link (i_j, l_j) it separates in the two opposite directions at the point y_j. This flow defines a length function δ_Y via the relation $\delta_Y(t) = d[f_Y(t)]$ which in turn yields in the manner described earlier a shortest path tree and a routing denoted by g(Y). It is easy to show (using the regularity of Y*) that if Y is sufficiently close to Y* then the (ordinary) shortest

path tree corresponding to δ_Y is the same as the one corresponding to Y* and that the elements of the routing g(Y) lie on the links (i_j,l_j).

The algorithm described earlier can now be redefined as

$$Y_{k+1} = g(Y_k). \tag{A.39}$$

The definition is local within a sufficiently small neighborhood of Y* and is associated with the (ordinary) shortest path tree corresponding to Y* and the associated parameterization of the ring subnetworks containing the links (i_j,l_j).

Similarly as in the preceding section, we say that the equilibrium Y* is (locally) *stable* if there is a neighborhood of Y* (defined in terms of the parameterization of the rings associated with Y* as discussed earlier), such that the sequence $\{Y_k\}$ generated by (A.39) is well defined and converges to Y* for every choice of Y_0 within this neighborhood.

In order for Y* to be stable it is sufficient that the n×n matrix $\frac{\partial g(Y^*)}{\partial Y}$ be defined and have all its eigenvalues within the unit circle. The computation of $\frac{\partial g(Y^*)}{\partial Y}$ is straightforward along the lines of the preceding section. Each diagonal term $\frac{\partial g_j(Y^*)}{\partial y_j}$ is given by the expression

$$\frac{\partial g_j(Y^*)}{\partial y_j} = \frac{-r(y_j^*)}{2d(0)} \left\{ \int_{y_j^*}^{1} \frac{\partial d[f^+(Y^*,\tau)]}{\partial f}\, d\tau + \int_{0}^{y_j^*} \frac{\partial d[f^-(Y^*,\tau)]}{\partial f}\, d\tau \right\} \tag{A.40}$$

Each nondiagonal term $\frac{\partial g_i(Y^*)}{\partial y_j}$ if given by the same expression except that the integral of $\frac{\partial d}{\partial f}$ is taken over the portion of the ring corresponding to y_j that overlaps with the ring corresponding to y_i. If the rings corresponding to y_i and y_j do not overlap then

$$\frac{\partial g_i(Y^*)}{\partial y_j} = 0.$$

It is interesting to note that again it is necessary that the bias level exceeds a generically positive threshold level in order for Y* to be a stable equilibrium. This level equals the maximum modulus of a matrix having as entries the inputs at Y* times a global measure of $\frac{\partial d}{\partial f}$ divided by two similarly as in the case of Sec. A.4. We note that a similar analysis can be carried out for averaged versions of the algorithm, and, as in the ring case, it appears that averaging can result in substantial improvement in dynamic behavior.

It should be clear that the multidestination case admits a very similar treatment. The algorithm and associated equilibria are defined in the same manner as in the single destination case by associating each destination with a routing. Again, the stability criterion can be expressed in terms of the eigenvalues of a matrix with entries similar to those of the single destination case. We will omit the details. An interesting feature of the multidestination case is that the flow at an equilibrium or any routing for that matter is not necessarily zero so that in place of 2d(0) in the denominator of (A.40) we have the sum

$$d[f^{+}(y_j^*)] + d[f^{-}(y_j^*)]$$

where $f^{+}(y_j^*)$, $f^{-}(y_j^*)$ are the flows in the two directions at y_i^*. This indicates that a stable equilibrium is possible even at zero bias level.

A.6 FINITE NODE RING NETWORKS

Consider a ring network with N nodes arranged in a ring as shown in Fig. A.34. We denote

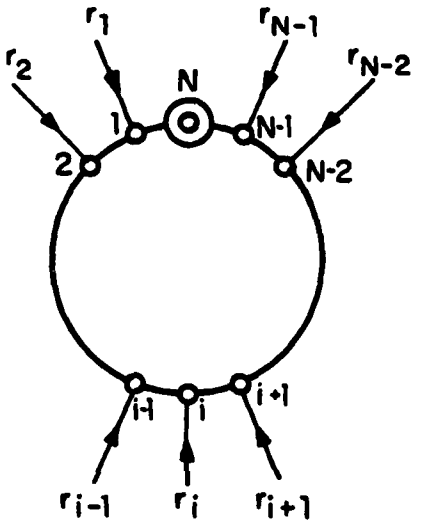


Figure A.34

the nodes by $1,2,\cdots,N$ and consider the case where N is the only destination. The traffic *input* originating at node i (and destined for N) is denoted by r_i. Similarly, as in Sec. A.3, the *routing* R_i, $i = 1,\cdots,N$ is the one for which all nodes $j < i$ route their traffic in the clockwise direction and all nodes $j \geq i$ route their traffic in the counterclockwise direction.

Given a routing R_i, the flows on each undirected link $(j-1,j)$ in the clockwise and counterclockwise direction are denoted by $f_j^-(i)$ and $f_j^+(i)$ respectively and are given by

$$f_j^-(i) = \begin{cases} 0 & \text{if } i \leq j \\ r_{i-1}+r_{i-2}+\cdots+r_j & \text{if } j < i \end{cases}$$

$$f_j^+(i) = \begin{cases} r_i+r_{i+1}+\cdots+r_{j-1} & \text{if } i < j \\ 0 & \text{if } j \leq i. \end{cases}$$

We are given a *delay function* $d(f)$ mapping flows to the nonnegative real numbers which is assumed monotonically increasing and continuously differentiable. Given a routing R_i we define the distances $D^-(i,j)$, $D^+(i,j)$ of node j to the destination in the counterclockwise and clockwise direction respectively by

$$D^-(i,j) = \sum_{\ell=1}^{j} d[f_\ell^-(i)]$$

$$D^+(i,j) = \sum_{\ell=j+1}^{N} d[f_1^+(i)].$$

We consider a shortest path algorithm whereby given a routing R_i the next routing is R_n where the node n is such that

$$D^-(i,j) \geq D^+(i,j) \qquad \text{for } j \geq n$$

$$D^-(i,j) < D^+(i,j) \qquad \text{for } j < n,$$

except if $D^-(i,i) = D^+(i,i)$ in which case the next routing is R_i.

Thus given an initial routing R^0 the algorithm generates successive routings $R^1, R^2, \cdots, R^k, R^{k+1}, \cdots$, via the procedure just described.

The following proposition shows that if $d(0) = 0$ and the first two routings are different, i.e., $R^0 \neq R^1$ then the algorithm ends up oscillating between the two extreme routings R_1 and R_N as described in Example 1 of Sec. A.3.

Proposition: Let $d(0) = 0$ and assume that $R^0 \neq R^1$. Then there exists an index $\bar{k}$ such that for all $k \geq \bar{k}$ either $R^k = R_1$ and $R^{k+1} = R_N$ or $R^k = R_N$ and $R^{k+1} = R_1$.

Proof: Let R_i be a routing and assume that the routing subsequent to R_i is R_n with $n \neq i$. For concreteness assume that $n < i$. We will show that either $i = N$ or else the routing subsequent to R_n is R_j with $j > i$.

If $i \neq N$, then since R_n is the routing subsequent to R_i we have

$$D^-(i,n-1) < D^+(i,n-1) = D^+(i,i). \tag{A.41}$$

We also have

$$D^+(i,i) \leq D^+(n,i) \tag{A.42}$$

with equality only if $r_n = r_{n+1} = \cdots = r_{i-1} = 0$, and

$$D^-(n,i) \leq D^-(i,n-1) \tag{A.43}$$

with equality only if $r_n = r_{n+1} = \cdots = r_{i-1} = 0$.

From (A.41), (A.42), and (A.43) we have

$$D^+(n,i) \geq D^+(i,i) > D^-(i,n-1) \geq D^-(n,i)$$

so finally

$$D^+(n,i) > D^-(n,i).$$

It follows that in the routing R_j which is subsequent to R_n, node i will switch his traffic to the clockwise direction so that $j > i$.

We can show using a very similar argument that if $n > i$ then either $i = 1$ or else the routing subsequent to R_n is R_j with $j < i$.

Thus we have that the number of nodes that lie between two successive routings is increasing at each iteration if none of these routings is R_1 or R_N. On the other hand, if the current routing is R_1 or R_N, then the next routing will clearly be R_N or R_1 respectively. This proves the proposition. Q.E.D.

Notice that, if $d(0) = 0$, the situation $R^0 = R^1$ can only occur if $D^-(i,i) = D^+(i,i)$ where i is the node for which $R^0 = R_i$. Thus, if we add any $\varepsilon > 0$ to any one of the node inputs we will have $R^0 \neq R^1$ and the algorithm will again end up oscillating between R_1 and R_N.

We now turn to consideration of various notions of equilibria and stability. We say that R_i is an *equilibrium routing* if

$$d(r_{i-1})+d(r_{i-1}+r_{i-2})+\cdots+d(r_{i-1}+\cdots+r_1)<d(0)+d(r_i)+d(r_i+r_{i+1})+\cdots+$$
$$+d(r_i+\cdots+r_{N-1}) \quad (A.44)$$

and

$$d(r_i)+d(r_i+r_{i+1})+\cdots+d(r_i+\cdots+r_{N-1})\leq d(0)+d(r_{i-1})+\cdots+d(r_{i-1}+\cdots+r_1) \tag{A.45}$$

or equivalently if

$$D^-(i,i-1) < D^+(i,i-1)$$

and

$$D^+(i,i) \leq D^-(i,i).$$

It follows from this definition that R_i is an equilibrium routing if and only if it repeats itself via the shortest path algorithm.

We say that a node i is an *equilibrium node* if

$$d(0)+d(r_{i-1})+\cdots+d(r_{i-1}+\cdots+r_1)<d(r_i)+d(r_i+r_{i+1})+\cdots+d(r_i+\cdots+r_{N-1}) \tag{A.46}$$

and

$$d(0)+d(r_{i+1})+d(r_{i+1}+r_{i+2})+\cdots+d(r_{i+1}+\cdots+r_{N-1})\leq d(r_i)+d(r_i+r_{i-1})+\cdots +d(r_i+\cdots+r_1), \tag{A.47}$$

or equivalently

$$D^-(i,i) < D^+(i,i)$$

and

$$D^+(i+1,i) \leq D^-(i+1,i).$$

In words, a node i is an equilibrium node if he switches his traffic in both cases where the routing is R_i and R_{i+1}.

The examples of Sec. A.3 show that both equilibrium routings and equilibrium nodes are of interest from the point of view of dynamic behavior.

We say that *an equilibrium routing R_i is locally stable* if routing R_{i+1} generates either R_i or R_{i-1} through the algorithm, and routing R_{i-1} generates either R_i or R_{i+1}. We say that *an equilibrium node i is locally stable* if routing R_i generates R_{i+1} via the algorithm, and routing R_{i+1} generates R_i. The definition of local stability is based on the idea that when the algorithm starts "close enough to equilibrium" it should not lead to a "growing" oscillation.

Consider an equilibrium routing R_i. In order that it be stable we must have that node (i-2) will not switch its traffic when the routing is R_{i+1}, and node (i+1) will not switch when the routing is R_{i-1}. The first condition can be expressed as

$$3d(0)+d(r_{i+1})+\cdots+d(r_{i+1}+\cdots+r_{N-1})>d(r_i+r_{i-1}+r_{i-2})+\cdots$$
$$+d(r_i+r_{i-1}+\cdots+r_1) \qquad (A.48)$$

while the second condition can be expressed as

$$3d(0)+d(r_{i-2})+\cdots+d(r_{i-2}+\cdots+r_1)\geq d(r_{i-1}+r_i+r_{i+1})+\cdots$$
$$+d(r_{i-1}+r_i+\cdots+r_{N-1}). \qquad (A.49)$$

Inequality (A.48) can be written by using the mean value theorem as

$$2d(0)+d(r_i)+d(r_i+r_{i+1})+\cdots+d(r_i+r_{i+1}+\cdots+r_{N-1})$$

$$-r_i\left[\frac{\partial d(\tilde{f}_i)}{\partial f}+\frac{\partial d(\tilde{f}_{i+1})}{\partial f}+\cdots+\frac{\partial d(\tilde{f}_{N-1})}{\partial f}\right]$$

$$> d(r_{i-1})+d(r_{i-1}+r_{i-2})+\cdots+d(r_{i-1}+\cdots r_1)-d(r_{i-1})$$

$$+r_i\left[\frac{\partial d(\tilde{f}_{i-2})}{\partial f}+\cdots+\frac{\partial d(\tilde{f}_1)}{\partial f}\right] \qquad (A.50)$$

where $\tilde{f}_1,\cdots,\tilde{f}_{N-1}$ are some flows satisfying

$$r_{i-1}+\cdots+r_1 \le \tilde{f}_1 \le r_i+r_{i-1}+\cdots+r_1$$
$$\vdots$$
$$r_{i-1}+r_{i-2} \le \tilde{f}_{i-2} \le r_i+r_{i-1}+r_{i-2}$$
$$0 \le \tilde{f}_i \le r_i$$
$$\vdots$$
$$r_{i+1}+\cdots+r_{N-1} \le \tilde{f}_{N-1} \le r_i+r_{i+1}+\cdots+r_{N-1}.$$

Using (A.44) and the fact $d(r_{i-1}) \ge d(0)$, it follows that (A.50), and hence also (A.48) will be satisfied if

$$d(0) \ge \frac{r_i}{2}\sum_{\substack{\ell=1\\ \ell\neq i-1}}^{N-1}\frac{\partial d(\tilde{f}_\ell)}{\partial f}. \qquad (A.51)$$

Similarly, inequality (A.49) can be written as

$$2d(0)+d(r_{i-1})+d(r_{i-1}+r_{i-2})+\cdots+d(r_{i-1}+\cdots+r_1)$$

$$-r_{i-1}\left[\frac{\partial d(\hat{f}_{i-1})}{\partial f}+\frac{\partial d(\hat{f}_{i-2})}{\partial f}+\cdots+\frac{\partial d(\hat{f}_1)}{\partial f}\right] \ge$$

$$\ge d(r_i)+d(r_i+r_{i+1})+\cdots+d(r_i+\cdots+r_{N-1})-d(r_i)$$

$$+r_{i-1}\left[\frac{\partial d(\hat{f}_{i+1})}{\partial f}+\cdots+\frac{\partial d(\hat{f}_{N-1})}{\partial f}\right] \qquad (A.52)$$

where $\hat{f}_1,\cdots,\hat{f}_{N-1}$ are some flows satisfying

$$r_{i-2}+\cdots+r_1 \le \hat{f}_1 \le r_{i-1}+r_{i-2}+\cdots+r_1$$

$$\vdots$$

$$0 \le \hat{f}_{i-1} \le r_{i-1}$$

$$r_i+r_{i+1} \le \hat{f}_{i+1} \le r_{i-1}+r_i+r_{i+1}$$

$$\vdots$$

$$r_i+\cdots+r_{N-1} \le \hat{f}_{N-1} \le r_{i-1}+r_i+\cdots+r_{N-1}.$$

Using (A.45) and the fact $d(r_i) \ge d(0)$ it follows that (A.52), and hence also (A.48) will be satisfied if

$$d(0) \ge \frac{r_{i-1}}{2} \sum_{\substack{\ell=1 \\ \ell\neq i}}^{N-1} \frac{\partial d(f_\ell)}{\partial f} \tag{A.53}$$

Conditions (A.51) and (A.53) represent sufficient conditions for local stability of the equilibrium routing R_i. Notice that they have a very similar form to the stability criterion (A.21) for the continuous ring model.

Consider now an equilibrium node i. In order that it be stable, we must have that node (i-1) will not switch its traffic when the routing is R_{i+1}, and node (i+1) will not switch when the routing is R_{i-1}. The first condition can be expressed as

$$2d(0) + d(r_{i+1}) +\cdots+d(r_{i+1}+\cdots+r_1) > d(r_i+r_{i-1})+\cdots+d(r_i+\cdots+r_2) \tag{A.54}$$

while the second condition can be expressed as

$$2d(0) + d(r_{i-1})+\cdots+d(r_{i-1}+\cdots+r_1) \geq d(r_i+r_{i+1})+\cdots+d(r_i+\cdots+r_{N-1}).$$

(A.55)

Inequality (A.54) can be written as

$$d(0) + d(r_i)+d(r_i+r_{i+1})+\cdots+d(r_i+\cdots+r_{N-1}) -$$

$$-r_i\left[\frac{\partial d(\tilde{f}_i)}{\partial f} + \frac{\partial d(\tilde{f}_{i+1})}{\partial f} +\cdots+ \frac{\partial d(\tilde{f}_{N-1})}{\partial f}\right] >$$

$$> d(r_{i-1}) +\cdots+ d(r_{i-1}+\cdots+r_1)+r_i\left[\frac{\partial d(\tilde{f}_{i-1})}{\partial f} +\cdots+ \frac{\partial d(\tilde{f}_1)}{\partial f}\right]$$

(A.56)

where $\tilde{f}_1,\cdots,\tilde{f}_{N-1}$ are some flows satisfying

$$r_{i-1}+\cdots+r_1 \leq \tilde{f}_1 \leq r_i+r_{i-1}+\cdots+r_1$$

$$- - - - - - - - - - -$$

$$r_{i-1} \leq \tilde{f}_{i-1} \leq r_i+r_{i-1}$$

$$0 \leq \tilde{f}_i \leq r_i$$

$$- - - - - - - - - - -$$

$$r_{i+1}+\cdots+r_{N-1} \leq \tilde{f}_{N-1} \leq r_i+r_{i+1}+\cdots+r_{N-1}.$$

Using (A.46) it follows that (A.56), and hence also (A.54) will be satisfied if

$$d(0) \geq \frac{r_i}{2} \sum_{\ell=1}^{N-1} \frac{\partial d(\tilde{f}_\ell)}{\partial f} \quad . \qquad \text{(A.57)}$$

Similarly, inequality (A.55) can be written as

$$d(0) + d(r_i)+d(r_i+r_{i-1})+\cdots+d(r_i+\cdots+r_1) -$$

$$-r_i\left[\frac{\partial d(\tilde{f}_i)}{\partial f} + \frac{\partial d(\tilde{f}_{i-1})}{\partial f} +\cdots+ \frac{\partial d(\tilde{f}_1)}{\partial f}\right] \geq$$

$$\geq d(r_{i+1})+\cdots+d(r_{i+1}+\cdots+r_{N-1})+r_i\left[\frac{\partial d(\tilde{f}_{i+1})}{\partial f} +\cdots+ \frac{\partial d(\tilde{f}_{N-1})}{\partial f}\right] \quad (A.58)$$

Using (A.47) it follows that (A.58), and hence also (A.55) will be satisfied if (A.57) holds.

Thus, local stability of an equilibrium node is guaranteed if (A.57) holds. Notice again the similarity with the stability criterion (A.21) for the continuous ring. This similarity provides confidence in employing results for the continuous model to predict behavior in the discrete model.

It is possible to obtain at least some results relating to averaging algorithms for the finite node ring model. The analysis can become quite complex, and this is even more so when an effort is made to extend results to the arbitrary topology and multiple destination case. It appears, however, that the conclusions reached regarding averaging algorithms for the continuous model are at least qualitatively valid for the finite node model. This conjecture is reinforced by available computational experience, some of which we now present.

A.6.1 Computational Results

We experimented with a 30-node ring network having node 30 as the single destination. We considered a synchronous and an asynchronous algorithm with evenly spaced delay broadcasts. A fading memory scheme was used to average delays corresponding to the present and past routings. In this scheme "delays" are computed at each iteration by means of the formula

$$[\text{New Delay of Link } (i,\ell)] = \beta[\text{Old Delay of Link } (i,\ell)] + (1-\beta)(0.05)^3[\text{Current flow of } (i,\ell)]^4. \quad \text{(A.59)}$$

The scalar β is referred to as the *decay factor* and takes values in the interval $[0,1)$. Large values of β imply a greater degree of averaging with delays corresponding to past routings. The expression $(0.05)^3[\text{Current flow of } (i,\ell)]^4$ represents our choice for delay function. It rises rapidly as flow becomes larger than 20.

In the synchronous algorithm all nodes "report" their link "delays" simultaneously at each iteration and all of these delays are used in the shortest path computation that determines the new routing.

In the asynchronous algorithm nodes compute their "delays" at every iteration by using equation (A.59). However, they "report" their link delays only every 29th iteration, with node 1 reporting at the 1st iteration, node 2 at the 2nd iteration and so on. Thus, for example, at the 30th iteration node 1 will report his link "delays" and the new routing will

be computed on the basis of the delays reported by node 2 at the 2nd iteration, by node 3 at the second iteration, and so on. This procedure is patterned after the envisioned asynchronous operation of the algorithm in the ARPANET.

The shortest path computation and corresponding routing updating is performed by using link lengths $D_{i\ell}$ given by

$$D_{i\ell} = \text{Bias} + [\text{Most recently reported "delay" of } (i,\ell)].$$

The bias has been chosen to be either constant or to depend on most recently reported delays by means of the formula

$$\text{Bias} = 0.02 \times (\text{Sum of most recently reported "delays" over all links}).$$

(For the polynomial expression used to represent delay a linear expression for bias seem appropriate. For more realistic delay curves a quadratic expression for bias may prove more suitable).

The initial routing in all runs was taken to be R_{1s} and all initial reported "delays" were taken to be zero. These initial conditions correspond to the situation where there is no traffic input to the network for a long period of time and there is a sudden change to a nonzero traffic input pattern while the algorithm is at the min-hop routing R_{1s}. Thus, the computational results do not only show the stability properties of the algorithm but also provide an indication on how fast the algorithm responds to a sudden change in input traffic conditions. It is interesting to note that the results indicate that the algorithm with high degree of averaging (asynchronous

operation $\beta = 0.99$, $\beta = 0.995$) responds as fast or even faster to the input traffic change than the algorithm with low degree of averaging ($\beta = 0$, $\beta = 0.98$) despite the fact that it utilizes more heavily delay information from the preceding input pattern. This can be attributed to the faster speed of convergence resulting from high degrees of averaging.

Case 1 (Constant Bias): Here the inputs are

$$r_1 = \cdots = r_5 = 5,\ r_6 = \cdots = r_{29} = 1.$$

We have calculated that for the synchronous algorithm with no averaging ($\beta=0$) the threshold level of bias for stability is approximately 24. We have operated the algorithms with levels of bias of 1 and 8. Tables A.1 and A.2 provide sequences of routings generated for a variety of decay factors. A number such as, for example, 15 means routing R_{15}. We have grouped 29 iterations of the asynchronous algorithm in a single entry by providing the "minimum" and the "maximum" routing during each group of 29 iterations. Thus, for example, an entry 10/18 in the table means that during the 29 iterations represented by the entry the routing was between R_{10} and R_{18}. In most cases, the routing in the asynchronous algorithm changes slowly so that this method of reporting routings provides an adequate measure of algorithmic performance.

The computational results confirm the theoretical predictions based on the continuous model analysis. The best synchronous algorithm from the point of view of stability is the one with decay factor 0.9. The synchronous algorithm with

TABLE A.1. BIAS = 1

Mode of Operation	Synchronous			Asynchronous			
Decay Factor	0	0.5	0.9	0	0.98	0.99	0.995
Successive Routings	15 1 30 1 30 1 30 1	15 1 30 23 7 12 4 28 2 29 14 5 24 2 29 15 4	15 3 29 15 9 9 9 9 9 9	1/16 2/9 2/24 3/27 4/27 1/7 2/30	11/22 1/29 20/28 3/22 17/25 3/20 15/25 3/21	13/17 1/28 23/28 4/17 15/21 4/10 11/20 4/19 15/22	14/16 2/27 23/26 4/17 16/19 6/11 9/12 7/10 7/10 7/10 7/10 8/10 8/10

TABLE A.2. BIAS = 8

Mode of Operation	Synchronous			Asynchronous			
Decay Factor	0	0.5	0.9	0	0.98	0.99	0.995
Successive Routings	15 2 30 1 30 1 30 1	15 6 15 6 15 6 15 6	15 14 12 11 11 11 10 11 10 11 10 10 11	5/12 7/15 9/13 10/12 9/11 10/11 10/11	15/15 7/11 11/12 10/10 10/11 10/11	15/15 11/13 10/11 10/11	15/15 13/14 11/12 11/11 11/11 10/11 10/11

decay factor $\beta = 0.5$ is inferior to the asynchronous schemes in which oscillatory behavior is considerably less pronounced for a wide variety of choices of decay factor. Notice that the decay factor in the asynchronous algorithm is applied at each iteration so that a decay factor β in this algorithm is comparable to a factor β^{29} in the synchronous algorithm. Notice that the choice of decay factor (and hence the degree of averaging) affects much more the synchronous algorithm than the asynchronous.

Case 2 (Variable Bias): Here the bias was chosen according to the formula

Bias = 0.02 × (Sum of most recently reported "delays" over all links).

The results for a variety of network input patterns appear in Tables A.3 through A.5. Notice that *identical routing sequences would be generated by the algorithm if all node inputs were multiplied by some positive constant* λ. This is true because both node "delays" and bias would be multiplied by the same constant (λ^4) leaving the shortest path computations unaffected. In the asynchronous algorithm with $\beta = 0$ occassionally there appeared a single routing in a 29-iteration group which was significantly different than the others in the same group. These iterations are marked by a * in Tables A.4 and A.5. It appears that the variable bias scheme combined with asynchronous operation and averaging gives stable and congestion sensitive algorithmic performance for a very broad spectrum of traffic input patterns.

TABLE A.3. BIAS = 0.02 × (TOTAL DELAY), INPUT: $r_1=\cdots=r_5=5$, $r_6=\cdots=r_{29}=1$

Mode of Operation	Synchronous			Asynchronous			
Decay Factor	0	0.5	0.9	0	0.98	0.99	0.995
Successive Routings	15 2 26 2 26 2 26	15 2 26 18 14 10 10 9 12 6 17 3 25 14 11 10 10 9	15 2 25 22 19 17 15 14 13 13 12 11 11 10 10 10 10	1/13 6/15 8/14 8/13 9/12 8/11 9/11 9/10 9/11 9/11 9/10 9/10	1/25 12/21 7/11 9/11 9/11 9/10 10/10 9/10 10/10 9/10	1/26 16/22 9/13 9/10 9/10 9/10 10/10 9/10 10/10 9/10 10/10	1/26 17/22 11/14 10/11 10/10 10/10 10/10 10/10 10/10 9/10 10/10 10/10 9/10

TABLE A.4. BIAS = 0.02 × (TOTAL DELAY), INPUT: $r_1=\cdots=r_{10}=10, r_{11}=\cdots=r_{27}=1, r_{28}=r_{29}=5$

Mode of Operation	Synchronous			Asynchronous			
Decay Factor	0	0.5	0.9	0	0.98	0.99	0.995
Successive Routings	15 2 25 2 25 2 25	15 2 24 19 12 7 13 4 22 13 6 17 6 18 6 18	15 2 23 20 18 16 15 14 13 12 11 10 9 9 9 9 9 8 9	1/10 5/13 6/12 7/12 7/11 7/11 8/13* 8/15* 8/13* 8/11 8/10 7/11	1/24 12/21 9/12 6/10 9/14 5/11 9/14 6/10 9/14 6/10 9/14 9/14	1/24 15/21 8/13 8/8 8/10 8/8 8/10 7/9 8/11 10/12 7/9 9/11 7/8 9/11 7/8 9/11	1/24 6/22 9/14 7/9 8/9 8/9 8/9 8/9 8/9 9/9 8/9 9/9 8/9 9/9 8/9 9/9 8/9 9/9 8/9 9/9

TABLE A.5. BIAS = 0.02 × (TOTAL DELAY), INPUT: $r_1=\cdots=r_4=1$, $r_5=\cdots=r_9=5$ $r_{10}=\cdots=r_{20}=1$, $r_{21}=\cdots=r_{25}=5$, $r_{26}=\cdots=r_{29}=1$

Mode of Operation	Synchronous			Asynchronous			
Decay Factor	0	0.5	0.9	0	0.98	0.99	0.995
Successive Routings	15 17 11 26 4 27 4 27 4 27	15 17 13 19 11 21 10 21 10	15 17 14 16 15 16 15 16 15 16	1/23 6/18 10/24* 11/25* 11/25* 11/25* 12/25* 12/24* 14/18 13/17 14/16 15/16 15/16	1/27 12/26 12/15 14/16 15/16 15/16 15/16 15/16 15/16 15/16	1/27 15/26 14/15 15/15 15/16 15/16 15/16	1/27 16/26 15/16 15/16 15/16 15/16 15/16

7. CONCLUSIONS

The analysis conducted thus far suggests the following conclusions for the SPF algorithm with link lengths chosen as

$D_{i\ell}$ = Bias + Average Delay per packet in link (i,ℓ)

where the bias value is the same for every link.

a. To each average input traffic pattern and each bias value there corresponds an "equilibrium".

b. For a fixed input traffic pattern, a high level of bias yields an equilibrium near the min-hop routing while a low level of bias leads to an "unbiased" equilibrium where delay distances to the corresponding destination are nearly equal along two paths. The unbiased equilibrium is sensitive to congestion even though it does not minimize total delay.

c. For a fixed input traffic pattern, there is a threshold value of bias (generically positive) above which the corresponding equilibria are stable and below which the corresponding equilibria are unstable.

d. The threshold bias value increases sharply as the level of traffic input rises and congestion develops.

e. Averaging delays over several updating periods and updating link delays asynchronously does not affect equilibria but significantly reduces the stability threshold level of bias thereby stabilizing equilibria which otherwise would be unstable. Even when the algorithm is unstable, its dynamic behavior seems to be dramatically improved by averaging and asynchronous operation.

f. The length of the updating period does not affect significantly the dynamic behavior of the algorithm. It does affect the accuracy of measured average delay and more importantly it affects the speed of algorithmic response to a sudden change in traffic input pattern.

g. There are no obvious ways to choose a different bias for every link in an effort to improve the performance of the algorithm.

The preceding conclusions represent the main trends in dynamic behavior. In an actual operating environment, random effects and deviations due to model simplifications are to be expected.

Based on the conclusions reached we suggest the following design guidelines as a basis for experimentation.

a. It appears that asynchronous updating of link delays and some form of averaging of the effects of several successive routings is definitely better than synchronous updating without averaging. A fading memory scheme together with asynchronous and evenly spaced node broadcasts appear most suitable.

b. A bias value should be added to each link delay. It should probably be the same or nearly the same for every link. One simple possibility is to keep it constant at all times at a fairly high value which should be experimentally determined. Another more complex possibility is to make bias a function of current *total* delay (defined as the sum of current delays over all links). The function can be taken to be piecewise constant with parameters determined experimentally. Adding a high bias

to links with high delay and low bias to links with low delay is probably a bad policy and tends to aggravate oscillatory behavior.

c. The length of the updating period and the level of threshold change for delay updating do not seem to significantly affect dynamic behavior. Thus they should probably be chosen on the basis of other considerations.

There are several areas where further analysis should be fruitful. It is clear that a damping mechanism is necessary to stabilize the SPF algorithm such as provided by addition of bias and employment of averaging. The extent of damping has not as yet been adequately quantified and further analysis can clarify the situation. It is also possible that other effective damping schemes will be discovered by further analysis, and these may improve dynamic behavior and sensitivity to congestion.

It is to be expected that on-line simulation of the algorithm on the ARPANET will reveal interesting new phenomena clarification of which will be enhanced by further analysis. The present report is the first attempt to quantitatively analyze the dynamic behavior of shortest path routing algorithms and should provide a starting point for considerable further research in this area.

REFERENCES

1. J.M. McQuillan, I. Richer, E.C. Rosen, "ARPANET Routing Algorithm Improvements," BBN Report No. 3803, prepared for ARPA and DCA, April 1978.

2. E.I. Jury, *Inners and Stability of Dynamic Systems*, J. Wiley, N.Y., 1974.

3. J.M. Ortega and W.C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, N.Y., 1970.