EVALUATION PLAN FOR SPACE STATION NETWORK INTERFACE UNITS

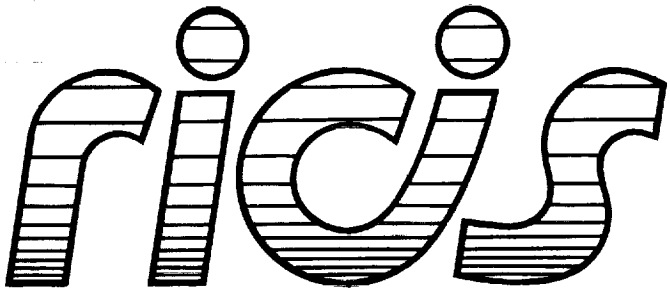
Alfred C. Weaver

Digital Technology

March 1990

Cooperative Agreement NCC 9-16
Research Activity No. SE.31

NASA Johnson Space Center
Engineering Directorate
Flight Data Systems Division



Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the "gateway" concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.

EVALUATION PLAN FOR SPACE STATION NETWORK INTERFACE UNITS

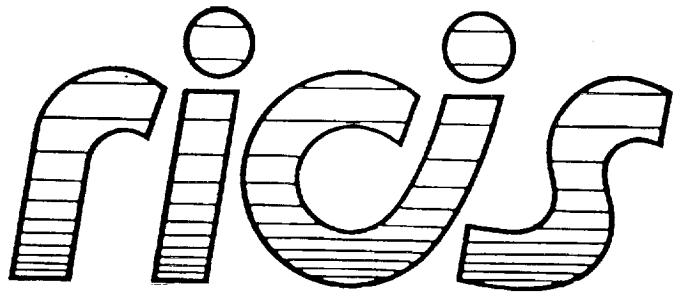
Alfred C. Weaver

Digital Technology

March 1990

Cooperative Agreement NCC 9-16
Research Activity No. SE.31

NASA Johnson Space Center
Engineering Directorate
Flight Data Systems Division



Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Alfred C. Weaver and Digital Technology. Dr. George Collins, Associate Professor of Computer Systems Design, served as RICIS technical representative for this activity.

Funding has been provided by the Engineering Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA technical monitor for this activity was Frank W. Miller, of the Systems Development Branch, Flight Data Systems Division, Engineering Directorate, NASA/JSC.

EVALUATION PLAN FOR SPACE STATION NETWORK INTERFACE UNITS

Alfred C. Weaver
DIGITAL TECHNOLOGY
1400 Ballard Woods Court
Charlottesville, Virginia 22901
(804) 982-2201
Internet: weaver@virginia.edu

This report outlines a procedure for evaluating network interface units (NIUs) produced for the Space Station program. The procedures would be equally applicable to the DMS testbed NIUs produced by Honeywell and IBM.

The evaluation procedures are divided into four areas:

(1) performance measurement tools -- hardware and software which must be developed in order to evaluate NIU performance.

(2) performance tests -- a series of tests, each of which documents some specific characteristic of NIU and/or network performance. In general, these performance tests quantify the speed, capacity, latency, and reliability of message transmission under a wide variety of conditions.

(3) functionality tests -- a series of tests and code inspections that demonstrate the functionality of the particular subset of ISO protocols which have been implemented in a given NIU.

(4) conformance tests -- ideally, this would be a series of tests which prove that the protocols implemented within the NIUs do conform exactly to the ISO standards. Since this is not practical due to the elementary state of the art in protocol testing and validation, the best we can do is to generate a series of tests which would expose whether or not selected features within the ISO protocols are present and interoperable.

1. PERFORMANCE MEASUREMENT TOOLS

1.1 TRAFFIC GENERATOR

Design a traffic generator which imposes a user-defined load on the underlying ISO protocols and the FDDI network. It should be capable of producing messages:

(a) with a data field size of 0 (tests system overhead)
(b) with any data field size up to 4,500 bytes (FDDI frame limit)
(c) of any size up to 10 MB (simulates file transfer)

(d) with any specific destination address
(e) with any group address (multicast)
(f) with the broadcast address

(g) whose lengths are constant
(h) whose lengths vary about a mean length according to an exponential or uniform random message length distribution

(i) whose message arrival rate is constant
(j) whose message arrival rate varies about a mean according to an exponential or uniform random distribution

(k) whose inter-message gaps may be set programmatically from zero (no delay to generate maximum load) to 1 second (long delay to eliminate resource contention)

(l) for a fixed or variable number of messages
(m) for a fixed or variable amount of time
(n) continuously until halted

(o) which can be injected into any layer of the ISO protocols.

Injecting messages into the lower layers of the OSI reference model, and not just at the application layer, is a requirement of Space Station. Testing, documentation, and performance measurement of this feature will require that NASA have access to the source code of each NIU implementation.

The traffic generator serves two basic purposes: (1) when connected to some layer of the protocol stack, it generates the messages which are then measured in accordance with the testing procedures outlined in section 2, and (2) when connected directly to an FDDI network with a MAC-layer interface, it generates traffic on the network which can simulate some degree of background load. Our recommendation is to have two traffic generators, one for each purpose.

The traffic generator which injects messages into the layers of the protocol stack would operate on the NIU hardware itself (i.e., on the Honeywell and/or IBM NIUs). The background traffic generator should operate on a powerful personal computer or engineering workstation, which must in turn be connected to the FDDI network. If the traffic generator is hosted on a PC with a PC/AT bus backplane, then it can be connected to FDDI via an Advanced Micro Devices *FastCard* interface; if using a PC and a microchannel driver, then an IBM FDDI interface will

be required; if operating on an engineering workstation with a VMEbus backplane, then many FDDI interfaces are suitable (e.g., Honeywell, Martin-Marietta, National, CMC, etc.).

1.2 DEVICE SIMULATOR

The *device simulator* is a software package which is programmable to use selected capabilities of the traffic generator. The device simulator can emit messages simulating a specific device whose characteristics are known. The device simulator should support these options:

(a) simultaneously support up to 10 device simulations
(b) emulates any source address
(c) utilizes any destination address (single, group, broadcast)
(d) generates messages of any size
(e) supports inter-message gaps (expressed in milliseconds)
(f) supports probabilistic transmission (transmits with probability *p*)
(g) permits offset between startup and first transmission (avoids in-phase problems)
(h) sends a specific number of messages, or transmits for a given time, or transmits continuously until halted

The traffic generator and device driver must have a user interface. There are two classic options:

(1) a UNIX-style command line which specifies options, such as

```
>generate -S 80AAC01087F2 -D FFFFFFFFFFFF -s 120 -i 10 -p 0.8 -o 0 -N 1000
```

which would run a program named `generate` that would simulate a device with source address 80AAC01087F2 (hexadecimal), broadcasting to all other devices (address FFFFFFFFFFFF hex), sending messages of size 120 bytes, with an inter-message gap of 10 milliseconds, transmitting that message with probability 0.8 at each transmit opportunity, with zero offset between the startup of the program and the first message emitted, and sending a total of 1000 messages. The advantage of the UNIX-style command line is simplicity for the implementor; the disadvantage is its cryptic form and reliance upon the good memory of the user (or a good manual).

(2) a Macintosh-style program using "pop-up" menus to ask questions and interactively construct a device description.

```
Device name (ASCII):              CO2 scrubber
Source address (hex):             80AAC01087F2
Destination address (hex):        FFFFFFFFFFFF
Message size (bytes):             120
Inter-message gap (ms):           10
Probability of transmission:      0.8
Offset after startup (sec):       0
Number of messages:               1000
```

The advantage of the menu-style approach is that it is easy to understand and use; the disadvantage is that it takes more time to implement.

If it is envisioned that the performance tests would be run only a few times, and primarily by the test program authors, then the UNIX command line is preferable due to economy of effort. If the programs are expected to be run many times, or if they are run by people who are not the program authors, then the pop-up menu approach is preferable. We expect that NASA will run the performance evaluation programs many times, and thus we recommend using the pop-up menu approach.

1.3 PERFORMANCE MONITOR

An extremely useful test device would be a passive performance monitor. It would be a station on the FDDI network which would record and analyze traffic as it passed by on the network. It would operate in two modes, *graphic* and *trace*. In graphic mode it would display:

(a) time since network initialization
(b) histogram of frame lengths
(c) scroll graph of network load in frames/sec
(d) scroll graph of network load in bits/sec
(e) mean and maximum network loads
(f) user-selectable scales for graphics

In trace mode it would trap and display, in a user-defined format, the frame headers (and optionally the frame content) of messages which traverse the ring. See Appendix A for a description of a real-time monitor for IEEE 802.5 token ring networks. We recommend that a passive real-time monitor for FDDI be acquired, either by direct purchase or by modifying our IEEE 802.5 monitor.

1.4 PERFORMANCE TESTING PROGRAMS

The performance testing programs would be crafted to measure the items listed in the next section. These programs would use the capabilities of the traffic generator or device simulator to send messages, and in addition would provide timers and counters with which to make performance measurements. The user interface should be menu-style since these programs will be reused many times.

2. PERFORMANCE TESTS

Performance tests are suggested for each OSI layer. An example of how data might be collected and displayed, example graphs are referenced and included in Appendix B. These graphs are drawn from chapter four of a master's thesis entitled *Performance Analysis of the FDDI Token Ring*, by Randall Simonson, University of Virginia, May 1988. Other examples are shown in the published papers included in Appendices C, D, and E.

2.1 DATALINK LAYER PERFORMANCE TESTS

2.1.1 Develop a program to measure the performance of datalink datagrams. Establish a baseline reference configuration by initially fixing the following network parameters. These parameters are then varied in later tests.

CHARACTERISTIC	SUGGESTED VALUE
number of stations	2 or 3
ring circumference	fixed (10-100 m)
priority classes	single synchronous class, no asynchronous class
message arrival rate	varied to achieve offered load target
message arrival distribution	constant
mean packet length	1,024 bytes
message length distribution	constant
restricted token dialogue	disabled
target token rotation time	10 ms
interpacket gap	0

2.1.2 Measure the performance of the reference configuration. Vary the offered load from 5% to 115% of network capacity in steps of 10% (note: it will be difficult to generate high loadings with a small number of stations and/or small message sizes). The measurement program will collect statistics and report the following:

(a) token cycle time (see Appendix B, Figure 4.1)
(b) tokens received at each station (Appendix B, Figure 4.2)
(c) tokens accessed at each station (Appendix B, Figure 4.2)
(d) frames sent (Appendix B, Figure 4.3)

(e) mean and maximum arrival queue length (Appendix B, Figure 4.4)
(f) station throughput (Appendix B, Figure 4.5)
(g) network throughput (similar to Appendix B, Figure 4.5)
(h) user interface latency (from generation program, through OS, into LCC) (Appendix B, Figure 4.6)
(i) queueing delay (at LLC) (Appendix B, Figure 4.6)
(j) network access delay (at MAC) (Appendix B, Figure 4.6)
(k) end-to-end delay (from transmitting program to receiving program) (Appendix B, Figure 4.6)

2.1.3 Present the baseline performance measurements as a series of graphs. The x-axis is offered load from 0 to 120% of capacity. The y-axis should be scaled appropriately for the metric being reported. There are separate graphs for metrics (a)-(k) in 2.1.2. See Appendix B for illustrative graph formats.

Note: Appendix C is a published performance measurement of the Manufacturing Automation Protocol (MAP). An example of measured datalink delay is shown in Figure 4; datalink throughput is shown in Figure 6. Appendix D is a published performance measurement of a different implementation of MAP. Datalink throughput is shown in Figure 7; messaging rate is depicted in Figure 8.

2.1.4 Change the message arrival rate distribution from constant to exponential. Run the tests of 2.1.2 again and report the results graphically as in 2.1.3.

2.1.5 Change the message length distribution to be uniform random. Run the test of 2.1.2 again and report the results as in 2.1.3.

2.1.6 Repeat the performance tests of 2.1.2 while varying the data field size in the message frame across the range: 0, 8, 64, 512, 1024, 2048, 4096, 4500. Produce the graphs of 2.1.3 for each separate message size. Examples of multiple curve plots (one for each message size) for metrics of interest are shown in Appendix B. Plotted against offered load, Figure 4.7 shows token cycle time, Figure 4.8 shows number of tokens accessed, Figure 4.9 is arrival queue length, and Figure 4.10 is network access delay.

Note: In Appendix B, Figure 4.11-4.17 report performance as a function of number of network stations and FDDI ring circumference. These are not likely to be significant variables in the DMS testbed.

2.1.7 Experiment with bi-modal message length distributions. Set up one class of messages with data size 128 bytes and a second class with size 4096 bytes. For a given offered load, let x% of the offered load be derived from short packets and $(100-x)$% be derived from long packets. Vary x from 0 to 100% in steps of 10%. Present the performance results graphically as in 2.1.3.

2.1.8 Experiment with tri-modal message length distributions. Set up three classes of messages with data sizes of 128 bytes, 1024 bytes, and 4096 bytes respectively. For a given offered load, let the percentage of the offered load derived from each class be varied as 10/10/80, 10/80/10,

and 80/10//10, and then again as 25/25/50, 25/50/25, and 50/25/25. Present the performance results graphically as in 2.1.3.

Note: Bi-modal and tri-modal message length distributions are not treated in any of the Appendices. However, all "real" networks show this type of behavior. Real performance data from multi-modal message length distributions would be quite valuable.

2.1.9 Although FDDI permits an unlimited number of asynchronous priority classes, the AMD Supernet chipset implementation supports only a single synchronous class and a single asynchronous class. For a given offered load, divide the load into synchronous and asynchronous priority classes according to the ratios 10/90, 20/80, 30/70, ..., 90/10. Present the performance results graphically as in 2.1.3. Refer to Appendix B, Figure 4.18-4.20, for a discussion of the multiple priority reference configuration.

2.1.10 Holding the TTRT constant, and dividing the offered load between the synchronous and asynchronous class, measure the throughput and delay of those two classes as the asynchronous priority threshold timer setting is reduced from TTRT (initially) to zero. Appendix B, Figures 4.21-4.24 show the expected result — the delay of the asynchronous class rises toward infinity as the token cycle time rises above the priority threshold timer setting.

2.1.11 Repeat 2.1.10 with different settings of the target token rotation time (TTRT). For each repetition, reduce the TTRT from 10 ms to 1 ms in steps of 1 ms. Present the performance results graphically as in 2.1.3. Appendix B, Figures 4.25-4.27 show the expected result — for lower priority traffic, throughput drops to zero and delay rises to infinity as the token cycle time rises to equal or exceed the TTRT value.

2.2 NETWORK LAYER PERFORMANCE TESTS

Identify a core set of tests which capture the performance metrics of interest at the network layer. Due to the explosive growth in the number of tests which could be run, it is suggested for the network and higher layers that the metrics of interest be primarily (1) station throughput, (2) network utilization, and (3) end-to-end delay between the communicating programs.

2.2.1 Identify a reference configuration similar to that in 2.1.1, but specifying the use of the inactive network layer protocol (INLP).

2.2.2 Run a baseline measurement for offered loads varying from 5% to 115% in steps of 10%. Display the results (station throughput and end-to-end delay) graphically.

2.2.3 Repeat 2.2.2 changing the mean message length to be 0, 8, 64, 512, 1024, 2048, 4096, and 4500 bytes. Present the results graphically as a family of curves (one for each message length) on a single plot.

2.2.4 Change the baseline configuration to use a full network layer protocol. Repeat the throughput and end-to-end delay measurements of 2.2.2.

Note: Appendix E is a published performance measurement of the Technical and Office Protocols (TOP). Figure 6 compares the performance cost (in terms of throughput) for having the network protocol active vs. inactive.

2.2.5 Test network layer segmentation by injecting NSDUs of size 4500 bytes and observing the performance results of network layer segmentation. Set the maximum size of the network layer PDU to be 4500 bytes, then 1024 bytes, then 256 bytes. Segmentation is a potentially valuable but expensive protocol service, as discussed in Appendices C, D, and E.

2.3 TRANSPORT LAYER PERFORMANCE TESTS

2.3.1 Identify a reference configuration similar to that in 2.1.1, but specifying the use of a connection-oriented service rather than connectionless. Set the transport layer management parameters appropriately. Initially, (a) select a large value for the transport retransmission timer so that it will not force unnecessary retransmissions, (b) set the initial credit window to be large, and (c) select the option of not calculating transport checksums on the data (checksums are always required on the header).

2.3.2 Using the reference configuration defined above, measure the time required to set up 1, 2, 3, ..., n identical connections.

2.3.3 Using the reference configuration, send all data over a single connection. Vary the offered load from 5 to 115% in steps of 10%. Vary the message size from 0 to 4500 bytes as before. Measure the (a) station throughput, (b) network utilization, and (c) end-to-end delay from transport user to transport user. Plot a family of curves (one for each message size) on each of the three graphs. The y-axis is the metric of interest and the x-axis is offered load.

Note: Transport throughput measurements are shown in Appendix C, Figures 7 and 11, and in Appendix D, Figures 5 and 6. Transport end-to-end delay is shown in Appendix C, Figure 5, and in Appendix E, Figure 8.

2.3.4 Repeat 2.3.3 but vary the offered load across 2, 3, ..., n connections.

2.3.5 Determine the effect of the retransmission timer setting. Plot station throughput, network utilization, and end-to-end delay on graphs whose x-axis is offered load. A family of curves on each graph would show the result of various retransmission time settings in the range of 500 ms down to 1 ms. An example of the effect on throughput of the retransmission timer setting is shown in Appendix C, Figure 8.

2.3.6 Vary the initial size of the transport sliding window (i.e., buffer credit) by powers of two, from 2 to 64. Determine the effect of sliding window size on throughput and delay. An example of the effect on throughput of the maximum sliding window size is shown in Appendix C, Figure 9.

2.3.7 Repeat the experiment of 2.3.3 with transport data checksums turned on. Compare the results graphically with transport data checksums disabled. Since data checksums require

parsing the message a separate time before transmission, data checksums can be an expensive operation.

2.3.8 Divide the offered load into two classes, normal and expedited. Vary the percentage of messages in the expedited class from 1% to 50%. Observe the effect on throughput and delay for both classes of messages.

2.3.9 Repeat 2.3.3 with messages larger than 4500 bytes which will force segmentation. Use messages in the range of 4 KB to 10 MB. Plot the throughput and delay as a function of message size. The cost of segmentation (in terms of throughput and delay) is quite pronounced. Refer to Appendix C, Figures 10 and 11, Appendix D, Figures 5 and 6, and Appendix E, Figure 6-8.

2.3.10 Compare the throughput and delay of transport vs. datalink services. This reveals the cost of the end-to-end reliability provided by the transport layer. While the cost varies with the implementation, Appendices C, D, and E show it to be quite significant.

2.4 SESSION LAYER PERFORMANCE TESTS

The session service provides a means for organized and synchronized exchange of data between cooperating session users. Use of the session layer services will involve a performance penalty upon connection establishment, data exchange, connection release, negotiation for data tokens, and error recovery based on synchronization points. It is expected that the "cost" of the session services would therefore be primarily visible during connection establishment and release, and in error recovery. In an experimental environment in which connections are long-lived and data errors are rare, the session services themselves should be inexpensive.

In terms of performance, session services are best observed as a "delta" in the performance of the transport layer, i.e., use of session services reduces throughput and increases latency by some "delta." In addition, the individual services of the session protocol can be instrumented and timed to determine efficiency.

2.4.1 Select a reference configuration as in 2.3.1 for the transport layer tests. Instead of injecting messages into the transport layer, messages are injected into the session layer.

2.4.2 Establish a connection with a peer session user. After connection establishment, and without introducing major or minor resynchronization points, measure throughput as a function of message size (SPDUs). Plot the throughput results on the same graph as the transport results for the reference configuration. Refer to Appendix E, Figure 7, which shows this comparison of session and transport throughput.

2.4.3 Repeat 2.4.3, but measure end-to-end latency rather than throughput. Refer to Appendix E, Figure 8, which shows end-to-end delay for both session and transport layers.

2.4.4 From the throughout and delay measurements above, calculate the "deltas" -- the average reduction in throughput and average increase in latency which results from using the session

services rather than the transport services.

2.4.5 Instrument the code of the session layer to make the following timings:

(a) establish a connection
(b) release a connection
(c) negotiate for the use of a data token
(d) establish a minor resynchronization point within the dialogue
(e) establish a major resynchronization point within the dialogue
(f) interrupt a dialogue, then restart

2.4.6 Establish a connection and begin a continuous dialogue using fixed length messages. Observe throughput and delay. Now introduce minor resynchronization points between every group of *n* messages (begin with n=10). Observe any reduction in throughput or increase in delay. Plot the change as a function of *n* and message length.

2.4.7 Establish a connection and begin a continuous dialogue using fixed length messages. Observe throughput and delay. Now introduce major (i.e. activity) resynchronization points between every group of *n* messages (begin with n=10). Observe any reduction in throughput or increase in delay. Plot the change as a function of *n* and message length.

2.5 PRESENTATION LAYER PERFORMANCE TESTS

The presentation layer is not included in the definition of MAP 2.1 which was implemented by Honeywell. Presentation layer performance tests are deferred until that layer is present in the DMS testbed.

2.6 APPLICATION LAYER PERFORMANCE TESTS

The application layer currently supports FTAM (File Transfer and Management); other application service elements will be supported in the future. This section is specific to FTAM. Note that it is especially difficult to isolate the performance of FTAM itself since it is bound to the operating system, file manager, and the physical characteristics of the file store (disk).

2.6.1 Establish a file of size *x* bytes. Use FTAM to transfer the file from one system to another. Measure the time required to transfer the file and produce a graph of file transfer time vs. file size. Be aware that this is a gross measurement and encompasses much more than just FTAM; these measurements include the effects of the operating system, file manager, and disk latency (both seek latency and rotational latency). Nevertheless, for a particular environment, the time recorded here is an approximation of what a real user would observe.

2.6.2 Attempt to eliminate the effect of disk latency from the above measurement by copying a file of size *x* bytes onto the same disk drive (i.e., not over the network). Subtract the file copying overhead determined here from the measurements in 2.6.1 to get a better idea of FTAM's contribution to throughput and delay. Note that disk buffer congestion, and the alternating reads and writes to the same disk, still prevent one from isolating FTAM. Plot the results from 2.6.1 and 2.6.2 on the same graph (file transfer time vs. file size).

2.6.3 Copy a file of *x* bytes using an operating system command which does not invoke FTAM (e.g., in UNIX, use the remote copy command *rcp*). Copy the file from one system to another. Plot the results of 2.6.1 and 2.6.3 on the same graph. The difference between them illustrates the FTAM overhead of creating and managing the virtual filestore, and encoding and decoding FTAM PDUs using ASN.1. The copy using *rcp* requires none of these services.

2.6.4 By instrumenting the source code, observe the time required to produce and process each of the following FTAM protocol data units:

F-INITIALIZE
F-TERMINATE
F-U-ABORT
F-P-ABORT
F-SELECT
F-DESELECT
F-CREATE
F-DELETE
F-READ-ATTRIB
F-CHANGE-ATTRIB
F-OPEN
F-CLOSE
F-BEGIN-GROUP
F-END-GROUP
F-RECOVER
F-LOCATE
F-ERASE

3. *FUNCTIONALITY TESTS*

3.1 DATALINK LAYER FUNCTIONALITY TESTS

The datalink functions for FDDI are standardized in the ANSI FDDI MAC protocol definition and implemented in hardware by the chipset. While an LLC protocol such as IEEE 802.2 may be used, it is not required. Thus there are no datalink layer functionality tests to be made other than those related to the layer management entities and station management, neither of which are treated here.

3.2 NETWORK LAYER FUNCTIONALITY TESTS

Inspect the source code of the network layer and observe its operation to verify correct operation of the following:

3.2.1 *PDU Composition* -- Assure that the source address and the destination address are properly derived from the NS-Source-Address and NS-Destination-Address parameters, respectively.

3.2.2 *PDU Decomposition* -- Verify that incoming packets are parsed; NS-Source-Address and NS-Destination-Address parameters are determined; NS-Unitdata is properly constructed from all constituent segments of the received NPDU; and quality of service parameters are recovered from the "options" part of the NPDU header.

3.2.3 *Header Analysis* -- Observe that the network layer protocol properly selects, by examining the Network Layer Protocol Identifier, whether this message has selected the full network protocol, one of its proper subsets, or its inactive subset (INLP).

3.2.4 *Lifetime Control* -- Verify that the network layer decrements the PDU Lifetime count (computed in units of 500 milliseconds) by the maximum of (a) one or (b) the sum of estimated transit delay en route to the node and system processing delay within the node; verify that the network layer discards NPDUs whose lifetime field becomes zero.

3.2.5 *Segmentation* -- When the transmitted NPDU length exceeds the maximum service data unit size supported, then the NPDU must be segmented into two or more derived NPDUs, each with the header of the derived PDU being identical to that of the original PDU, except for the segment length and checksum of the fixed part and the segment offset of the segmentation part.

3.2.6 *Reassembly* -- When a message is segmented at the network layer at transmission, it must be reassembled at reception. Verify that the receipt of all the constituent derived PDUs of a message results in the reassembly of the original message. Determine the value used for the reassembly timeout function (the amount of time a receiver who has received some derived PDUs will wait for all derived PDUs before discarding all derived PDUs and posting an error).

3.2.7 *PDU Discard* -- Verify that the network layer discards packets whenever (a) the checksum fails, (b) local buffers are congested, (c) its destination address is unreachable or unknown, (d) its lifetime expires, or (e) the PDU is involved in a protocol error.

3.2.8 *Error Reporting* -- Verify that the network layer issues an Error Report PDU each time it discards a PDU (except when it discards an Error Report PDU).

3.2.9 *Header Error Detection* -- Verify that network header testing can be suppressed by setting the header checksum to zero. Verify that an incorrect network header checksum causes the PDU to be discarded.

3.2.10 *Padding* -- If the padding option is used, verify that the resulting PDU is properly aligned on computer word boundaries and that octet alignment is still maintained.

3.2.11 *Source Routing* -- Activate this option to observe that named network entities are visited in the exact order specified in the Source Route parameter, with no intermediate vists to other network entities.

3.2.12 *Partial Source Routing* -- Activate this option to observe that named network entities are visited in the order specified in the Source Route parameter, provided that intermediate network entities may be visited between any two entities on the Source Route list.

3.2.13 *Complete Route Recording* -- Activate this option to observe whether the protocol constructs a complete list of all intermediate systems visited by all PDUs and derived PDUs; verify that reassembly at intermediate points is performed only when all derived PDUs took the same route, and that PDUs are discarded when derived PDUs took different routes.

3.2.14 *Partial Route Recording* -- Active this option to record the route of PDUs which visited intermediate systems; verify that reassembly is always permitted at intermediate systems and that the route recorded is the route of any derived PDU.

Note: Tests 3.2.11 through 3.2.14 are only meaningful when two or more network segments are interconnected by a router, and thus are not observable in the initial DMS tested. Still, inspection of the source code should verify that these capabilities exist.

3.2.15 *Quality of Service* -- ISO 8348 Addendum One defines QOS parameters for (a) transit delay, (b) security, (c) cost, (d) priority, and (e) residual error probability. Determine which of these have been implemented. If the priority QOS parameter has been implemented, determine the latency reduction which is available to higher priority messages.

3.2.16 *Primitives* -- By inspection of source code, verify that the network layer implements the main network layer service primitives:

PRIMITIVE	OPERATION
N-CONNECT	set up network connection
N-DATA	send data
N-DATA ACKNOWLEDGE	acknowledge previously sent data
N-EXPEDITED DATA	send higher priority data
N-RESET	reset and resynchronize
N-DISCONNECT	terminate connection

3.3 TRANSPORT LAYER FUNCTIONALITY TESTS

The ISO Transport service definition (ISO 8072) and protocol (ISO 8073) provide for reliable end-to-end transfer of data. ISO 8073 provides five levels of reliability, named class 0 through class 4. Of these, class 4 ("error detection and recovery class") is the most reliable and is the one specified for Space Station. By inspection of the transport layer source code and observation of its operation, the following should be tested:

3.3.1 *General Timers* -- The implementation must continuously support five timers: (a) NSDU lifetime -- discussed previously in section 3.1.4; (b) expected maximum transit delay -- the expected maximum transit delay of remote-to-local and local-to-remote NSDUs, which may be different in each direction, and which is the maximum for "most" transfers; (c) acknowledgement -- a bound on the maximum time between submitting a TPDU to the network layer and receiving an acknowledgement for same; (d) local retransmission time -- a bound on the time to wait before retransmitting a TPDU; and (e) persistence time -- a bound on how long a TPDU may be retransmitted while awaiting an acknowledgement.

3.3.2 *Data Transfer Timers* -- The implementation must support two timers specific to data transfer: (a) inactivity timer -- if this timer expires, the connection has been idle too long and is therefore released; and (b) window timer -- a bound on the maximum interval between updates of the flow control credit parameters.

3.3.3 *Expedited Data* -- Verify that the implementation of expedited TPDUs (ED TPDUs) (a) uses a separate sequence number space from normal TPDUs, (b) requires the first such ED TPDU to have sequence number zero, (c) exempts the ED TPDU from flow control, (d) awaits receipt of an acknowledgement before sending any more data, and (e) follows normal retransmission procedures in case of lost acknowledgements.

3.3.4 *Resequencing* -- Data must be delivered in sequence. Out-of-sequence data outside the transmit window must be discarded. Out-of-sequence data inside the transmit window must be held until the missing packets are replaced.

3.3.5 *Explicit Flow Control* -- The transport entity must provide an initial credit (which may be zero) when the connection is established. Subsequent acknowledgements must be used to extend the edge of the credit window.

3.3.6 *Acknowledgements* -- The transport entity must acknowledge in-sequence TPDUs (a) within the acknowledgement time of 3.3.1 above, (b) whenever an out-of-sequence TPDU is received, and (c) within the duration of the window timer defined in 3.3.2 above.

3.3.7 *Graceful Progress* -- Data flow must resume normally if, after flow control is exerted by an acknowledgement TPDU with a credit of zero, a newer acknowledgement TPDU establishes a non-zero credit.

3.3.8 *Connection Request* -- The Connection Request (CR) TPDU must support these required operations: (a) flow control credit, (b) destination, (c) source, (d) transport class (0 to 4) and options, (e) calling Transport Service Access Point (TSAP) ID, (f) called TSAP-ID, (g) proposed maximum TPDU size, (h) protocol version number, (i) protection parameter (user defined), and

(j) checksum. The CR TPDU may additionally specify options such as alternative protocol classes, acknowledgement time, proposed throughput, proposed transit delay, proposed residual error rate, and proposed priority. Up to 32 bytes of user data may be included in the CR TPDU.

3.3.9 *Connection Confirm* -- The Connection Confirm (CC) TPDU replies to the CR TPDU by specifying the class and service options which it can accept. If a proposed value in the CR TPDU can not be supported, then a less demanding value in the CC TPDU specifies what value can be supported. The connection originator is free to accept the new values in the CC TPDU or, if unacceptable, to release the connection.

3.3.10 *Checksums* -- The implementation must implement checksums in accordance with section 6.17.3 of ISO 8073. Checksums must be calculated on the transport header. Checksums may be calculated on the data (if selected by the checksum parameter during connection establishment as in 3.3.8). If the checksum calculation on a TPDU header or data element fails, then the TPDU must be discarded.

3.3.11 *Segmentation* -- Verify that if the size of the Transport Service Data Unit (TSDU) exceeds the maximum allowable size for the TPDU, then the transport layer segments the TSDU into as many TPDUs as are necessary to transmit the data.

3.3.12 *Transport Service Primitives* -- Verify that the code implements the major transport service primitives:

T-CONNECT	establish and confirm a connection
T-DATA	transfer data
T-EXPEDITED DATA	transfer high priority data
T-DISCONNECT	close the connection

3.4 SESSION LAYER FUNCTIONALITY TESTS

The session service provides the following services:

establish a connection
exchange data in a synchronized manner
release a connection
negotiate for the use of data tokens
select half-duplex or full-duplex data exchange
establish synchronization points within the dialogue
resume the dialogue from an agreed point in case of error
interrupt and later resume a dialogue

These services are provided in three phases: session connection, data transfer, and connection release. By inspection of the source code, verify that the following services are provided.

3.4.1 *Session Connection* -- used to set up session connection and to negotiate tokens and parameters to be used on the connection.

3.4.2 *Data Transfer* -- provides five types of services:

Data transfer services -- normal transfer, expedited transfer, typed data transfer, capability data exchange

Token management services -- give tokens, please tokens, and give control

Synchronization services -- minor resynchronization point, major resynchronization point, resynchronize

Error reporting service -- provider initiated exception reporting, user initiated exception reporting

Activity services -- activity start, activity resume, activity interrupt, activity discard, activity end

3.4.3 *Connection Release* -- provides three services: orderly release, user initiated abort, and provider initiated abort

3.4.4 *Functional Units* -- Observe that session services are divided into twelve functional units:

kernel functional unit
negotiated release functional unit
half-duplex functional unit
duplex functional unit
expedited data functional unit
typed data functional unit
capability data functional unit
minor synchronize functional unit
major synchronize functional unit
resynchronize functional unit
exceptions functional unit
activity management functional unit

3.4.5 *Allowed Subsets* -- Verify that the implementation supports one or more of the allowed subsets of the twelve functional units. The three permissible subsets and their constituent functional units are:

Basic Combined Subset (BCS) -- includes kernel, half-duplex, and duplex

Basic Synchronized Subset (BSS) -- includes kernel, negotiated release, half-duplex, duplex, typed data, minor synchronize, major synchronize, and resynchronize

Basic Activity Subset (BAS) -- includes kernel, half-duplex, typed data, capability data, minor synchronize, exceptions, and activity management

3.4.6 *Quality of Service* -- Verify that the implementation supports these QOS parameters:

session connection protection
session connection priority
residual error rate
throughput (for each direction)
transit delay (for each direction)
optimized dialogue transfer
extended control
session connection establishment delay
session connection establishment failure probability
transfer failure probability
session connection release delay
session connection resilience

3.5 PRESENTATION LAYER FUNCTIONALITY TESTS

The presentation layer is not included in the definition of MAP 2.1 which was implemented by Honeywell. Presentation layer functionality tests are deferred until that layer is present in the DMS testbed.

3.6 APPLICATION LAYER FUNCTIONALITY TESTS

The application layer currently supports FTAM (File Transfer and Management); other application service elements will be supported in the future. This section is specific to FTAM.

FTAM (ISO 8571) is described in four parts: general introduction, virtual filestore description, file service definition, and file protocol definition. With 300+ pages of definition, it is impractical to test all its features. FTAM provides ten major services, each of which is subdivided into sub-services. By inspection of the FTAM source code, verify that the following services and sub-services are implemented.

3.6.1 *FTAM Regime Control* -- consisting of three sub-services: regime establishment, regime termination, regime abort.

3.6.2 *FTAM Filestore Management* -- not yet defined in ISO 8571/3.

3.6.3 *File Selection Regime* -- including file selection, file deselection, file creation, file deletion.

3.6.4 *File Management* -- two sub-services: read attributes and change attributes.

3.6.5 *File Open Regime Control* -- file open and file close.

3.6.6 *Grouping Control* -- beginning-of-grouping service and end-of-grouping service.

3.6.7 *Recovery* -- recovery regime service.

3.6.8 *Access to File Content* -- locate file access data unit service and erase file access data unit

service.

3.6.9 *Bulk Data Transfer* -- read bulk data, write bulk data, data unit transfer, end of data transfer, end of transfer, cancel data transfer.

3.6.10 *Checkpointing and Restarting* -- checkpointing service and restarting data transfer service.

3.6.11 *FTAM Primitives* -- Through visual inspection of the FTAM source code and observation of its operation using a network protocol analyzer, verify that FTAM both produces and accepts these major FTAM protocol data units:

F-INITIALIZE
F-TERMINATE
F-U-ABORT
F-P-ABORT
F-SELECT
F-DESELECT
F-CREATE
F-DELETE
F-READ-ATTRIB
F-CHANGE-ATTRIB
F-OPEN
F-CLOSE
F-BEGIN-GROUP
F-END-GROUP
F-RECOVER
F-LOCATE
F-ERASE

4. CONFORMANCE TESTS

Conformance testing is in its infancy, as evidenced by the slow start on MAP conformance testing at the Corporation for Open Systems. The necessary hardware and software tools for conformance testing, and especially the ability to automatically specify, compile, and then verify protocol operation, are simply not available. This will be a research topic for the entire decade of the 1990s.

As our contribution, we are working on a formal description language for the specification of protocols and a "protocol compiler" which will read the specification and generate an implementation in the programming language 'C'. The system also tests the protocol specification for such important characteristics as deadlock, liveness, reachability, etc. This work is expected to appear as a Ph.D. dissertation in late 1990.

True conformance testing is not possible. In the meantime, perhaps with the help of the Corporation for Open Systems and the National Institutes of Standards and Technology, we can identify individual test sequences which will show whether or not a specific implementation's behavior is consistent with some subset of a standard's requirements.

APPENDIX A

A Real-Time Monitor for Token Ring Networks
Alfred C. Weaver and James F. McNabb

IECON'90
IEEE Industrial Electronics Society Annual Conference
Pacific Grove, California
November 1990

A Real-Time Monitor for Token Ring Networks

Alfred C. Weaver and James F. McNabb
Computer Networks Laboratory
Department of Computer Science
Thornton Hall
University of Virginia
Charlottesville, Virginia 22903

1. Background

In cooperation with our industrial partner, Sperry Marine Inc. of Charlottesville, Virginia, the Computer Networks Laboratory at the University of Virginia designed and built a real-time network for use on ships. Called *SeaNET*, the network interconnects devices on the ship's bridge such as the gyrocompass, autopilot, heading indicator, collision avoidance radar, and voyage management system. The network is based on an IEEE 802.5 token ring LAN and software of our own design. Our real-time messaging system provides the communications primitives needed to open, close, and manage connections, send and receive messages, set options, and report status. The user's application program, hosted on a personal computer with a 25 MHz Intel 80386 processor, can send or receive approximately 600 short (100-byte) messages/second continuously. The end-to-end delay (from user memory to user memory, including all operating system interaction and all network transit delays) for short messages is less than 2 ms. Using longer 2000-byte messages, a single station can transmit approximately 2 Mbit/s continuously.

One of the early difficulties we faced was that we could not directly observe network traffic; correct operation could only be inferred from the action of the transmitters and receivers. Thus we also designed and built a real-time network monitor which we could use to visualize network traffic. The monitor is completely passive so that its use does not affect network loading in any way. The monitor provides a menu-based user interface, real-time displays of network traffic, scrolling graphs of network load, histograms of packet sizes, statistics such as maximum and mean network loading observed, traces of messages observed, and automatic alarms for single events and for event frequency deviations. The monitor will display and analyze all network traffic or any subset of that traffic as specified by user-programmed address filters.

2. Network Diagnostic Tools

Network diagnostic tools fall into two broad categories: protocol analyzers and real-time network monitors. Protocol analyzers, such as Network General's *Sniffer* and Excelan's *LANalyzer*, operate in two modes: capture and display. The analyzer first records a snapshot of network activity and afterwards analyzes and displays the previously observed network traffic (e.g., by disassembling and displaying protocol headers). In contrast, our monitor calculates its statistics and displays its graphs *as the network is operating*, i.e., in real time. Whereas protocol analyzers often use special-purpose hardware to buffer messages and assist in the time-tagging of network events, our monitor utilizes only standard, off-the-shelf IEEE 802.5 token ring interface hardware.

3. Real-Time Network Monitor

Our real-time network monitor operates in either graphic mode or trace mode, and the user can easily toggle between the two. The graphic mode is used to record and display statistical information regarding the performance of the network over a user-defined period of time; the trace mode is used to collect and display packet-specific information. We provide various options for each of these modes which allow the user to customize the monitor's data collection and display.

3.1. The User Interface

Options for the graphic and trace modes are selected through a pop-up menu user interface. The pop-up menu intentionally resembles the interface provided in modern, user-oriented software products (e.g., Borland's Turbo C 2.0). Movement through the menus is accomplished through either the keyboard or, if available, a mouse. If using the mouse, then menu selection is accomplished by an ordinary "point and click" operation. If using the keyboard, arrow keys move up and down through a menu list; the **Return** key descends into lower menus and the **Escape** key ascends. Figure 1 depicts the user interface.

3.2. The Real-Time Displays

Once the user has selected his desired options and filters and has started the monitor in either the graphics or trace mode, the monitor will record and display the desired information in real-time. These displays attempt to keep pace with the network traffic. Figures 2 and 3 show the graphic and trace mode displays, respectively.

3.3. Graphic Mode

Graphic mode displays visual statistics about network utilization. Graphic mode divides the display screen into four areas: the mode information window, a packet length histogram, and two historical scroll graphs. These areas are shown in the following snapshot of the graphic mode display.

3.3.1. Mode Information Window

The *mode information window* in the upper left-hand corner provides four status reports:

Elapsed Time	time spent in graphics mode since last reset operation
Interval	duration of data sample interval over which statistics are calculated
Filter	the user-defined name of the receiver
Destination	destination filter
Source	source filter

3.3.2. Packet Length Histogram

The *packet length histogram* in the upper right-hand corner displays the length distribution of all packets observed since the beginning of graphic mode operation. On a real-time network, packet length distribution tends to be bi-modal — short packets for control messages and long packets for background file transfer. The histogram shows the distribution of packets sizes observed over the range of 1-2000 bytes. The histogram is self-scaling and the current y-axis scale is noted at the top of the graph.

3.3.3. Scroll Graphs

Two scroll graphs, one reporting network loading in units of packets/second and the other in units of bits/second, are shown on the bottom two-thirds of the display screen. These graphs update once every sample interval (selected by the user and reported as "Interval" in the mode information window); as each new measurement is calculated, the screen shifts to the left and the new point is plotted as the rightmost line on the screen. Thus the right side of the screen reflects the network's most recent performance while the left side provides a summary of recent history. The scale of the scroll graphs, along with the current mean values and the maximum value recorded since the last reset, are displayed at the bottom of each scroll graph.

3.4. Trace Mode

Trace mode is used to display the actual contents of network packets. The trace mode display screen is divided into three areas: mode information window, filter and alarm window, and the packet information log. A typical trace mode screen is shown in Figure 3.

3.4.1. Mode Information window

The *mode information window* occupies the upper left-hand corner of the display screen and reports:

Elapsed time	time spend in trace mode since last reset operation.
Packets received	number of packets observed since start of trace.
Packets displayed	number of packets that have been displayed by the monitor.
Trace dump	the name of the file or device (e.g., screen) to which trace information is written.
Screen format	the current display format for the packet information log (see section on "Message Format")

3.4.2. Filter and Alarm window

The *filter and alarm window* occupies the upper right-hand corner of the display screen and reports:

Alarm	current status of alarm option
Filter name	user-defined name of the receive filter
Destination	destination filter
Source	source filter
Message	packet data filter

3.4.3. Packet Information Log

The *packet information log* on the bottom two-thirds of the screen displays the actual contents of packets that match the pattern displayed in the *filter* field of the filter and format window. The time the packet was received, the destination node address, the source node address, and the packet length, respectively, are enclosed in brackets for easy identification. The remaining portion of the packet is displayed on the right-hand side of each line (and on succeeding lines as necessary) as defined by the message *format* specification.

3.5. Message Format

The message format for the packet information log specifies the way in which the actual packet information is displayed in the packet information log — "c" indicates character, "d" means decimal, "x" specifies hexadecimal, "e" is an exclude, and "(n)" is a repetition factor (refer to the examples below). This feature allows the user to tailor the trace display to show precisely what is wanted.

Format examples:

"x x x"	display first, second, and third byte of every packet's data field as hexadecimal
"(3)x"	this is identical to the one above
"(4)c (5)d"	display first four bytes as characters and the next five as decimals
"d e e d"	display first and fourth byte as decimal, skipping the second and third bytes
"(10)e c"	skip first ten bytes and display the next as character

3.6. Message Filtering

The monitor is capable of *filtering* the network traffic so that it "sees" only a particular type of packet. Filtering is accomplished by specifying a filter *pattern* and associating a name with this pattern. Every packet received by the network interface is compared byte-by-byte against the *pattern*. Packets that match the *pattern* are recorded by the monitor and those that do not are ignored.

The address fields of a packet are treated separately from the remaining data field. Figure 4 shows the standard IEEE 802.2 LLC frame format. Source address and destination address patterns are entered separately from the data pattern.

The data pattern to be used as a filter is represented in one or more bytes. The data received must match this pattern byte-by-byte to be recorded. The pattern is specified by a list of values given in decimal (d), character (c or ' '), or hexadecimal (x) format. For repeated values, "(n)" specifies a repetition factor. The asterisk "*" means "don't care" and the ">" symbol represents all the remaining bytes of the message.

Address filter pattern examples:

"[**:**]"	matches all addresses
"[A0:**]"	matches address A0 (hexadecimal by default) and all LSAPs
"[**:02]"	matches all addresses with LSAP of 2
"[A0:02]"	matches address A0 with LSAP of 2

Data filter pattern examples:

"100d 5d"	match if first byte is 100 (decimal) and second byte is 5 (decimal)
"0Ax 'GYRO'"	match if first byte is 0A (hex) and the next 4 bytes are "G", "Y", "R", "O"
"* * (10)0d"	ignores first two bytes and matches if next 10 bytes are zero
"100d 5d >0d"	matches if first byte is 100 (decimal), second byte is 5 (decimal), and all remaining bytes are zero

3.7. Device Naming

Another option is the display of device addresses as names rather than hexadecimal character strings. The monitor allows the user to build a small database which associates known device addresses with names. For example, if the file server has address "80C3FF0089A7", that address can be mapped to the string "File Server". When using this option in trace mode, the monitor will look up each device address and replace it with its mnemonic name as the packet is displayed on the screen. This option greatly enhances the readability of trace data.

3.8. Real-Time Alarm

The *alarm option* applies to both graphic mode and trace mode. It is set "on" or "off" from the *mode* menu, and if "on" can be toggled between "event alarm" and "frequency alarm."

3.8.1. Event Alarm

An *event alarm* notifies the user of the first occurrence of a packet that matches the alarm filter pattern. This can be used to signal that a specific event or action occurred. The notification on the screen consists of the time the packet was received and the destination and source field addresses of the packet.

3.8.2. Frequency Alarm

The *frequency alarm* notifies the user that the frequency of a given packet type is now outside the allowable range. When specifying a frequency alarm filter pattern, the user also specifies a minimum and/or maximum frequency of packets of that type. If the monitor detects that packets matching the alarm filter pattern occur less often than the minimum frequency or more often than the maximum frequency, an alarm is reported. Alarm notification reports the time of the alarm and the number of packets received which match the alarm filter pattern.

4. Conclusions

Our monitor performs a different function from that of protocol analyzers — rather than collecting information for later disassembly and display, the monitor computes network statistics and displays network traffic in real time. This makes it a valuable debugging tool as well as a source of network utilization statistics.

The *graphic* and *trace* modes provide a means of "seeing" network traffic. By using the source and destination address patterns of "*:*" (match all) one can easily generate summaries of all network traffic. By specifying the address and data filters singly or in combination, one can watch any subset of network traffic; for example, all packets from node A, all packets to node B, all packets from A to B, or all packets from A to B which are of a specific type.

The *alarm option* is particularly valuable for debugging real-time networks. The *event alarm* shifts the burden of recognizing rare or unpredictable events from the user to the monitor. The *frequency alarm* provides a simple "watchdog" mechanism to verify that periodic operations continue on schedule.

There are three important attributes of the monitor's real-time behavior. First, the monitor is completely passive; it emits no test data which would alter the network load being measured. Second, the monitor is implemented on a processor which is faster than the processors which drive the token ring transmitters and receivers; this helps reduce overruns. Third, if the network does experience a transient period of high load, priority is given to data capture using a large (1000 event) packet buffer; the packet buffer is then unloaded and displayed (with each event tagged with its actual arrival time) whenever the network traffic subsides.

5. Acknowledgements

The authors gratefully acknowledge the technical direction and support provided by Mr. Ross Bennett at Sperry Marine Inc., and Dr. J.A.N. Lee and Dr. Tor Opsahl at the Virginia Center for Innovative Technology.

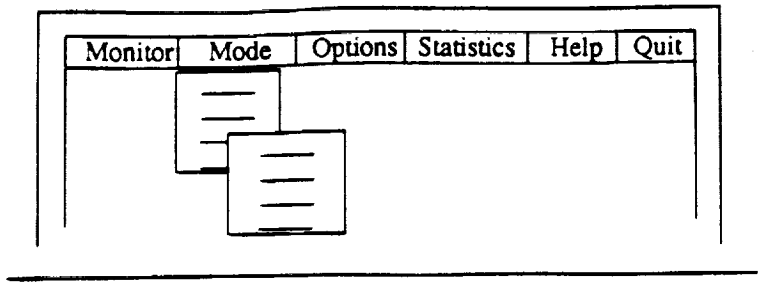


Figure 1.
User Interface

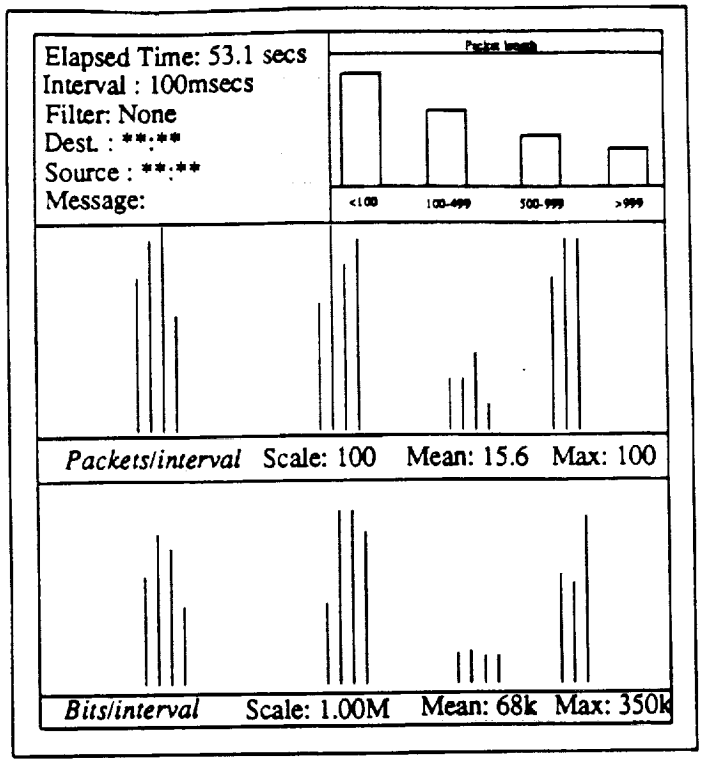


Figure 2.
Graphics Mode

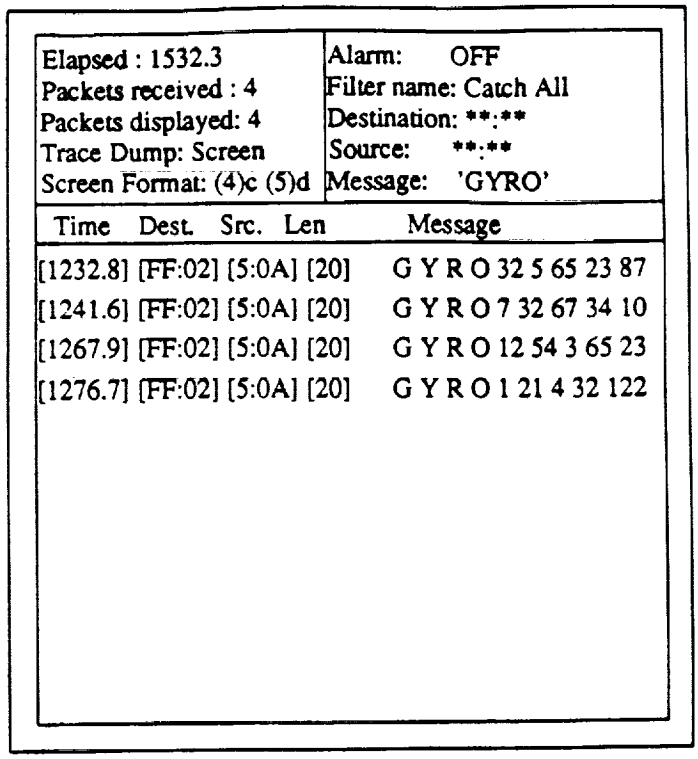


Figure 3.
Trace Mode

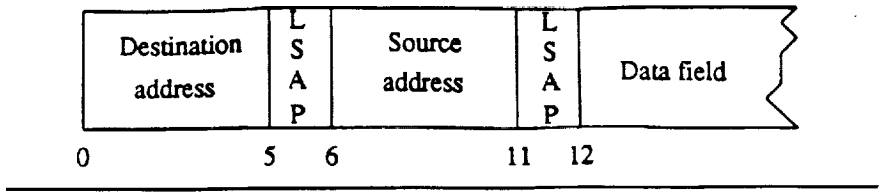


Figure 4.
Frame Format

APPENDIX B

Performance Analysis of the FDDI Token Ring
(Master's thesis, chapter four)

Randall A. Simonson
University of Virginia
May 1988

Chapter 4

Performance of the FDDI Token Ring

4.1. Introduction

This chapter presents a performance analysis of the FDDI token ring. Among the performance metrics presented are token cycle time, throughput, queue lengths, and packet delay. The results in this chapter were obtained via simulation and are presented graphically. Each graph has two independent variables; one is the variable of interest (e.g., mean packet length) and the other is always offered load. On all graphs offered load is varied from 5% to 115%. In our context, offered load is measured at the queue level within each station and then summed over all stations on the network, which accounts for the existence of offered loads in excess of 100%. Each graph is accompanied by a brief discussion to enhance the readers' interpretation of the data.

4.2. Single Priority Operation

Since FDDI provides synchronous and asynchronous service it was necessary to determine whether the single priority performance of each was sufficiently different to warrant separate investigation. Through simulation it was observed that no significant differences existed and separate treatment was therefore not justified.

This observation is reinforced by the protocol's description of synchronous and asynchronous transmission. The protocol states that asynchronous transmission uses that portion of bandwidth unused by synchronous transmission. In single priority operation the implemented priority gets all the available bandwidth whether it be synchronous or asynchronous and therefore no significant differences should exist. Differences in these service types arise only when they are used together in the same network configuration. It was concluded that synchronous transmission alone would provide a good indicator of single priority operation.

4.2.1. Reference Configuration

To begin our analysis we must first establish a baseline (reference) configuration against which variations can be compared. Our reference configuration for single priority operation has the following characteristics:

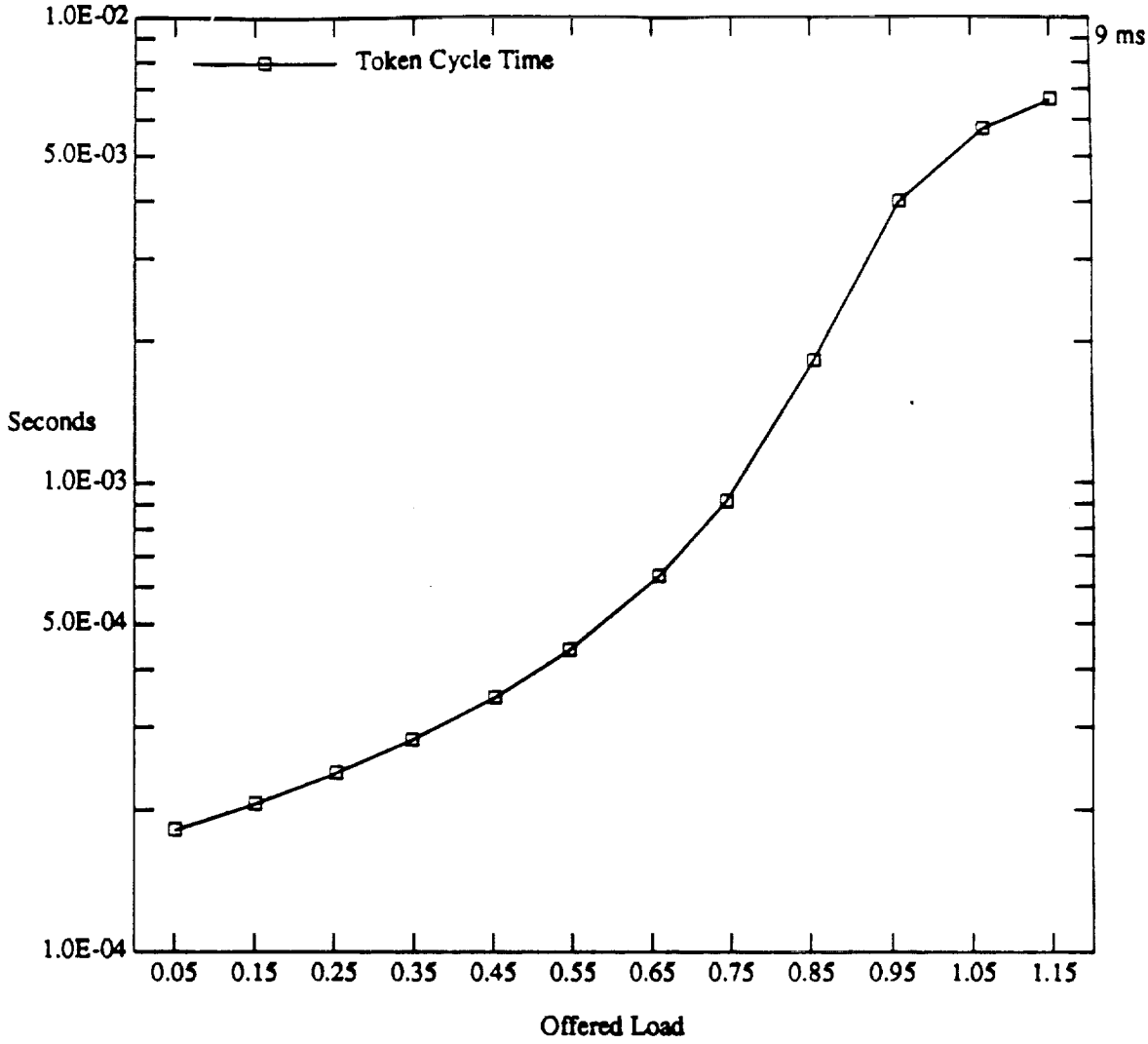
100 stations evenly distributed on the ring
100 Mbps medium rate
10 km ring circumference
32 bit internal station latency
synchronous priority only
9.0 ms target token rotation time
exponential packet arrival distribution
1024 bit mean packet length distributed exponentially
no restricted token dialog
error free transmission

The motivation for this set of network characteristics is simply that it provides a reasonable configuration to which variations can be made and that the target token rotation time is long enough for the network to provide exhaustive service at high offered loads.

4.2.1.1. Token Cycle Time

Token cycle time, also known as token rotation time, is the amount of time necessary for the token to circulate once around the ring. Figure 4.1 shows token cycle time for the single priority reference configuration.

Higher offered loads are achieved by offering more octets per station. Because mean packet length does not vary, generating more octets per station yields more packets per station. Token cycle time increases as offered load increases because there are more packets waiting for transmission per token cycle. Eventually, however, FDDI's "timed token" mechanism (see Section 2.6.2.3) begins to take effect. When this happens, token cycle time stops its steady rise and begins to stabilize as it approaches the target token rotation time (9 ms).



Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.1 — Token Cycle Time

Single Priority Reference Configuration

4.2.1.2. Token and Packet Traffic

As the token circulates around the ring it is only captured by stations having packets to transmit. Figure 4.2 plots the number of opportunities a station has to capture the token (tokens received) against the number of tokens it actually captures (tokens accessed). This graph shows that at low offered loads each station receives many tokens, but few of these are actually accessed. As offered load increases, token cycle time goes up and the number of tokens a station receives in a given amount of time begins to decline. The number of tokens accessed rises at first to accommodate increasing offered load, but since a station can access no more tokens than it receives these metrics begin to converge. At very high offered loads a station accesses almost every token it receives.

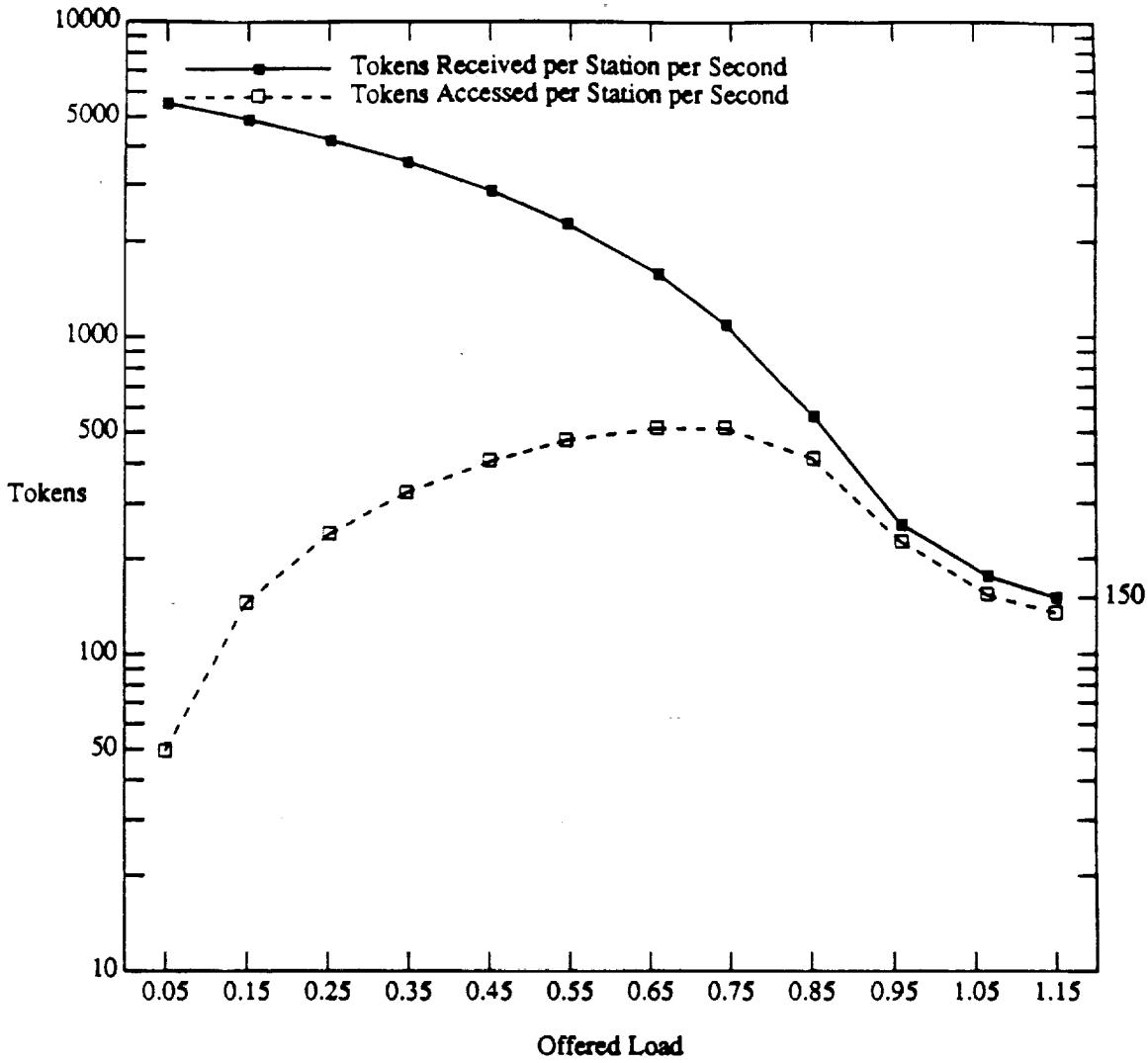
As token cycle time stabilizes, so does the number of tokens each station receives per second. This is why these metrics begin to stabilize at approximately 150 tokens per station per second for the reference configuration.

Figure 4.3 plots the number of tokens accessed per station per second against the number of packets sent per station per second. We have already discussed the tokens accessed curve but notice the difference in the shape of that curve on a linear scale (Figure 4.3) as compared to a logarithmic scale (Figure 4.2).

The number of packets sent per station per second in Figure 4.3 increases linearly until the "timed token"-mechanism begins to limit the number of packets each station may transmit per token access. At this point the "packets sent" metric begins to approach a value of 950 packets per station per second.

4.2.1.3. Arrival Queue Length

A station captures a token when it has packets to transmit. Figure 4.4 depicts the mean number of packets waiting when the token arrives at a station. We have previously established that as offered load increases, more packets arrive at the station between token accesses. This is why arrival queue length increases steadily as offered load increases.

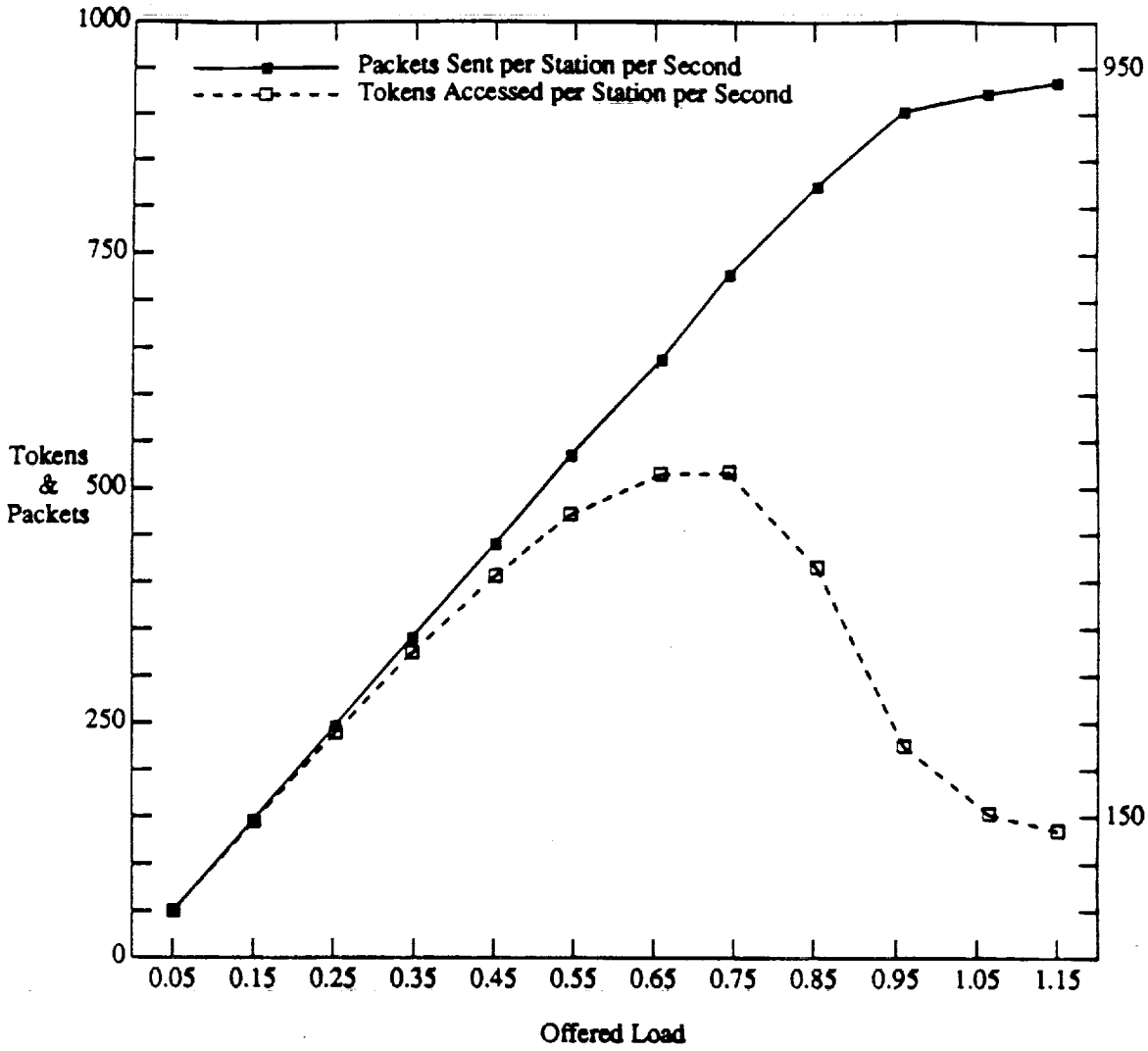


Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.2 — Tokens Received and Tokens Accessed

Single Priority Reference Configuration

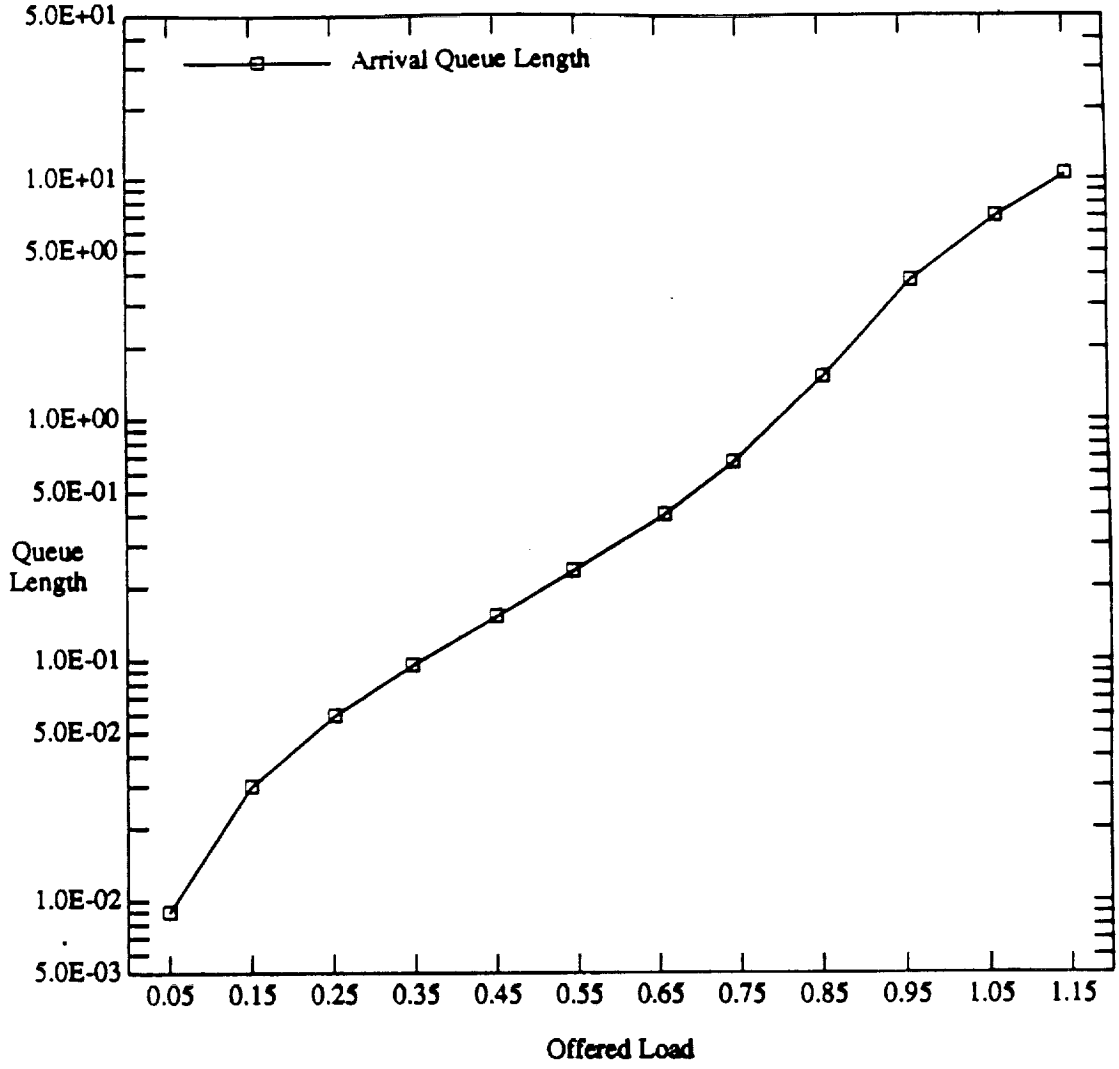


Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.3 — Tokens Accessed and Packets Sent

Single Priority Reference Configuration



Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.4 — Token Arrival Queue Length

Single Priority Reference Configuration

4.2.1.4. Throughput

Throughput is the number of bits (data plus required framing) transmitted in a given amount of time divided by the medium capacity for that same amount of time. Figure 4.5 shows the throughput of the single priority reference configuration. Throughput rises linearly as offered load is increased and stabilizes when the maximum throughput of the network is reached. For this network configuration the maximum throughput is approximately 96%. The lost 4% is due mainly to token transmission time and related overhead.

4.2.1.5. Packet Delay

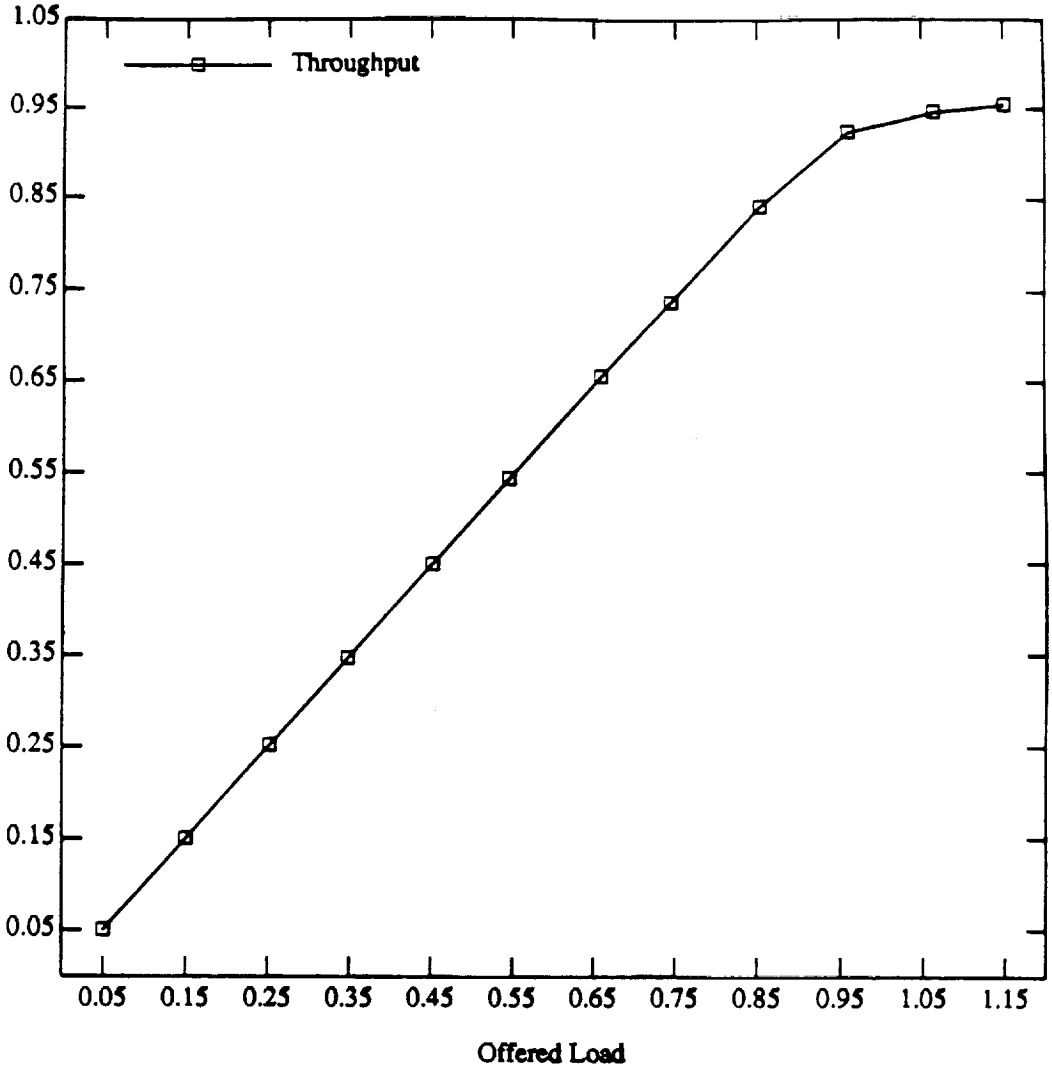
The delay a packet experiences between the time it is enqueued and the time it reaches its destination can be divided into the following four components:

1. Queueing Delay: The time a packet spends in the queue behind other packets.
2. Network Access Delay: The time a packet spends at the head of the queue.
3. Station Delay: The sum of queueing delay and network access delay.
4. Service Delay: The sum of station delay and packet transmission time.

Figure 4.6 depicts these various packet delays. At low offered loads very few packets are generated; therefore, when a packet is enqueued the chances are good that no packets will be ahead of it and network access delay is the major contributor to the service delay of a packet. As offered load increases, packets arrive with greater frequency and this increases the probability that a packet will be enqueued behind packets already in the queue. The result is that as offered load increases, queueing delay contributes more to the service delay of a packet. At approximately 85% offered load queueing delay becomes the major contributor to packet delay. At extremely high offered loads queueing delay accounts for almost all the service delay experienced by a packet.

4.2.2. Effects of Varying Packet Length

The first network parameter of interest is mean packet length. We conducted simulations for mean packet lengths of 32, 64, 128, 256, 512, 1024, and 2048 octets. For these simulations mean packet length is the only network parameter varied.

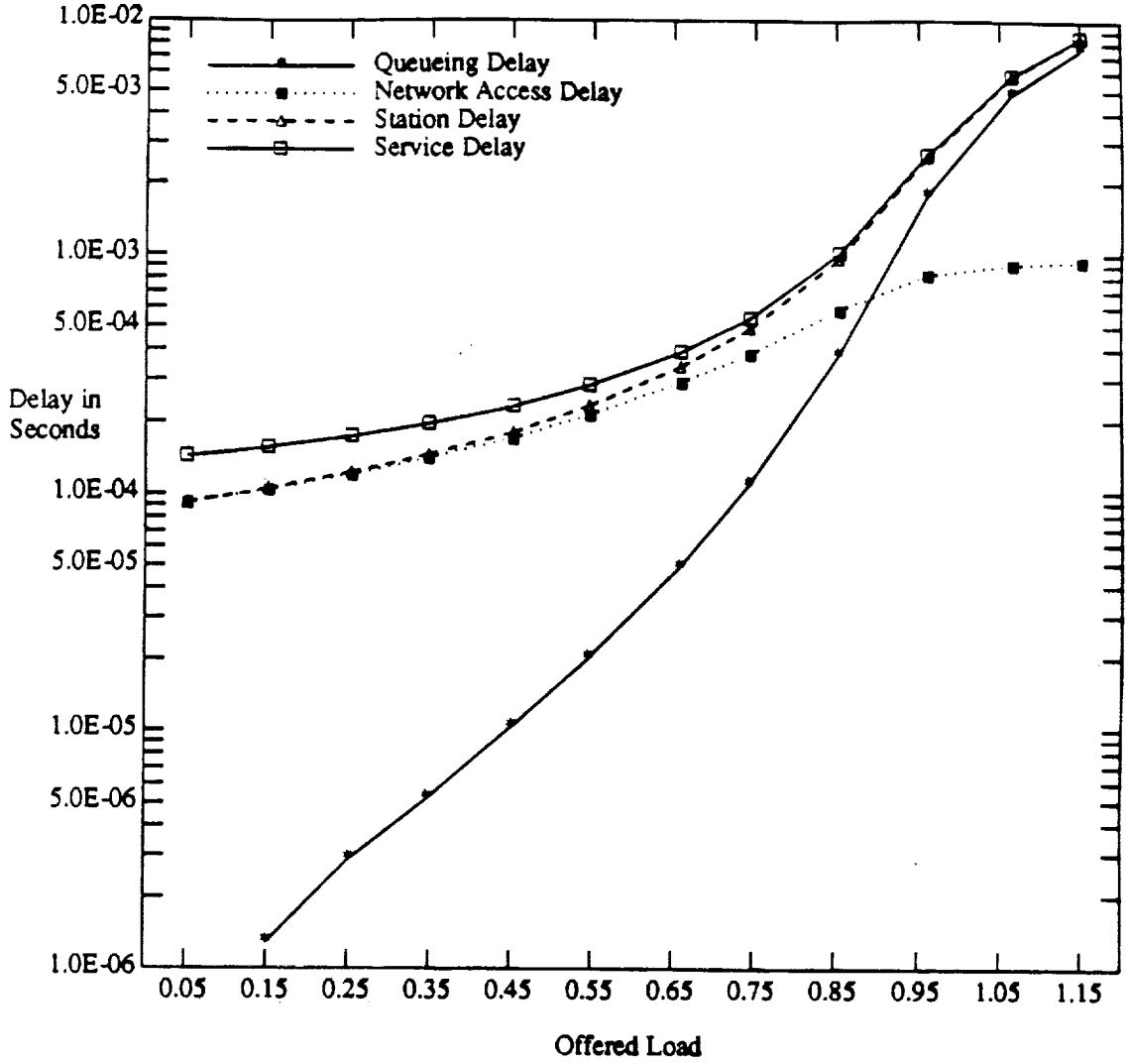


Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.5 — Throughput

Single Priority Reference Configuration



Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.6 — Various Packet Delays

Single Priority Reference Configuration

4.2.2.1. Token Cycle Time

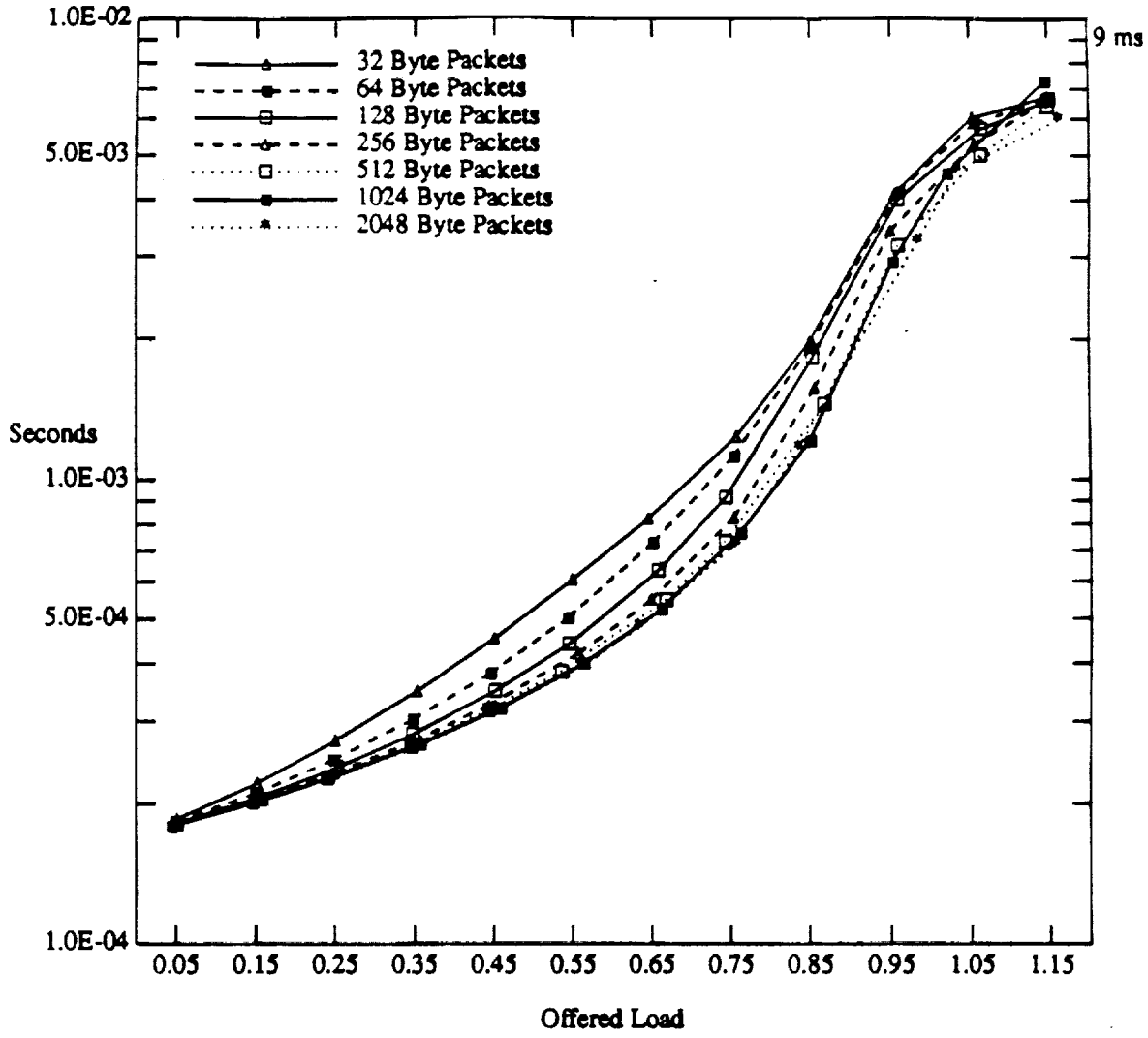
Figure 4.7 shows the effect packet length has on token cycle time. To maintain a certain offered load each station must contribute a certain number of octets to the network traffic. The method used to group these octets is of small consequence. For example, when a token arrives at a station it makes little difference whether two 512-octet packets or sixteen 64-octet packets are enqueued. Each will be serviced in basically the same amount of time and have nearly the same effect on token cycle time.

As we will see in Figure 4.8, small packets over the middle offered loads produce many more token accesses than do large packets. With each token access there is a small amount of overhead. This is why we see small packets having slightly longer token cycle times relative to larger packets over the middle offered loads in Figure 4.7.

4.2.2.2. Token Traffic

Figure 4.8 indicates how variations in packet length affect the number of tokens accessed per station per second. We see from this graph that smaller packets require more token accesses than do larger packets. The reason for this is that as packet size decreases a station must offer more packets to maintain the same offered load. If more packets need to be offered, then the time between packet arrivals will decrease and it follows that the queues will have packets waiting in them more often. If packets are waiting more often, then more tokens will need to be accessed but fewer packets will be serviced per token access.

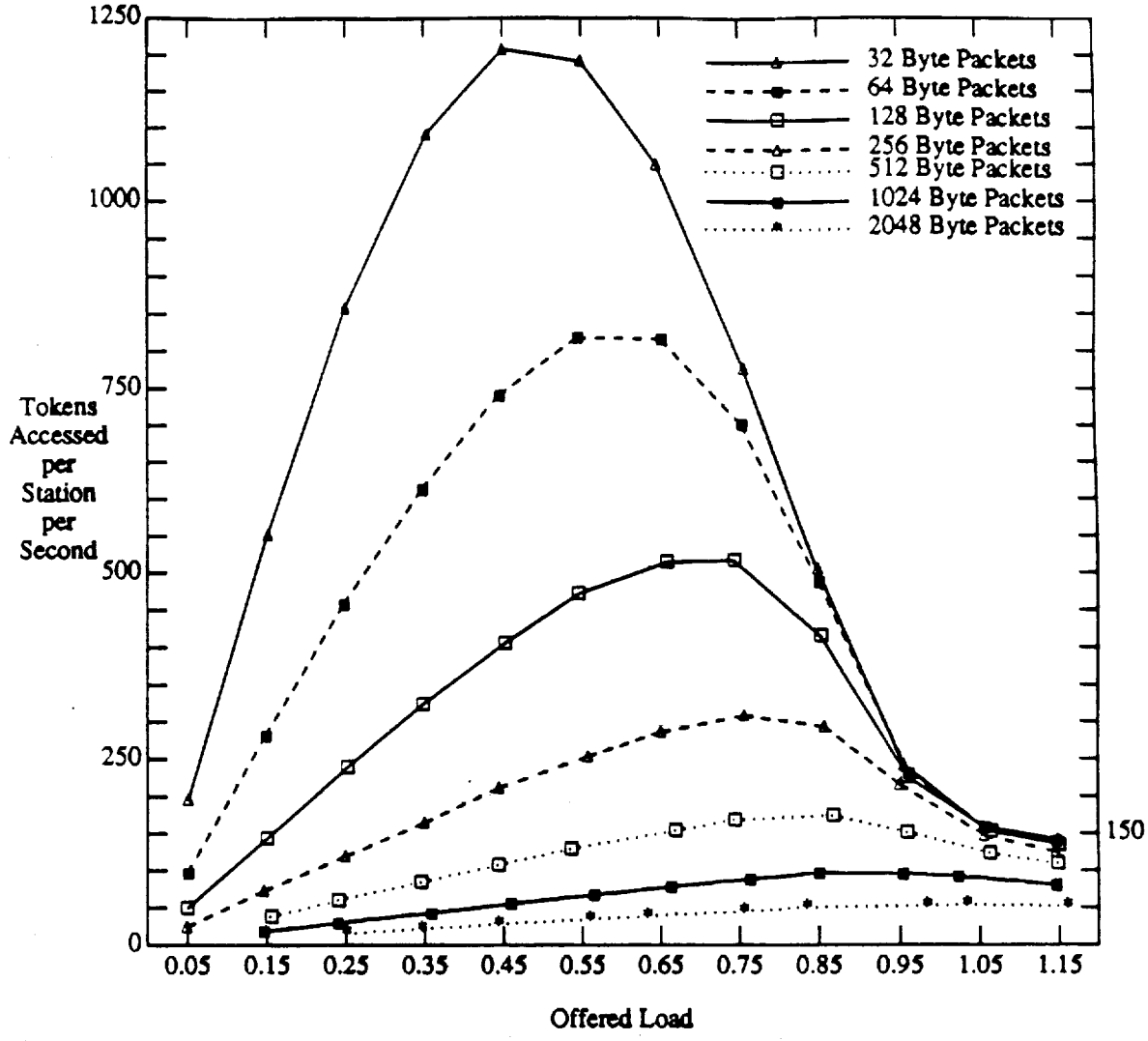
At very high offered loads we again see the number of token accesses begin to stabilize. As before, this occurs because token cycle time has begun to stabilize, and with it, the number of tokens received per station per second has also stabilized. The number of tokens received is the upper bound on the number of tokens accessed and at high offered loads a station accesses almost every token it receives. As a result, these metrics begin to converge.



Configuration:

100 stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
Varying Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.7 — Tokens Cycle Time Varying Packet Length



Configuration:

100 stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
Varying Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.8 — Tokens Accessed Varying Packet Length

4.2.2.3. Arrival Queue Length

Figure 4.9 shows how varying packet lengths affect the number of packets waiting when a token is accessed. As packet size increases the number of packets a station must offer to maintain a certain offered load decreases. As a result, fewer packets arrive between token accesses which is why we see the largest packet lengths producing the shortest arrival queues.

4.2.2.4. Packet Delay

Network access delay is the mean time a packet spends at the head of the arrival queue. Figure 4.10 shows how varying packet length affects this delay metric.

When the token arrives at a station the only packet that has accumulated any network access delay is the first packet in the queue. All packets behind the first packet have no network access delay. Mean network access delay per packet is simply the mean delay incurred by the first packet divided by the mean number of packets serviced per token access. The more packets serviced per token access, the shorter the average network access delay per packet. This is why at high offered loads we see small packets having shorter network access delay relative to large packets.

As offered load increases we eventually reach a point where the maximum number of packets are serviced per token access. When this point is reached network access delay for that particular packet size stabilizes. Figure 4.10 shows that network access delay has stabilized for 32, 64 and 128 octet packets.

4.2.3. Effects of Varying the Number of Stations

The next network parameter of interest is the number of stations on the ring. We conducted simulations for 10, 50, 100, 250, 500, and 1000 stations. One thousand stations is the maximum number of stations FDDI allows. In these simulations the number of stations is the only network parameter varied.

4.2.3.1. Token Cycle Time

In Figure 4.11 we see how variations in the number of stations affect token cycle time. As stations are added to the ring it follows that the token will be accessed more times per token cycle and this will increase token cycle time. This upward trend will continue until token cycle time begins to approach the

target token rotation time.

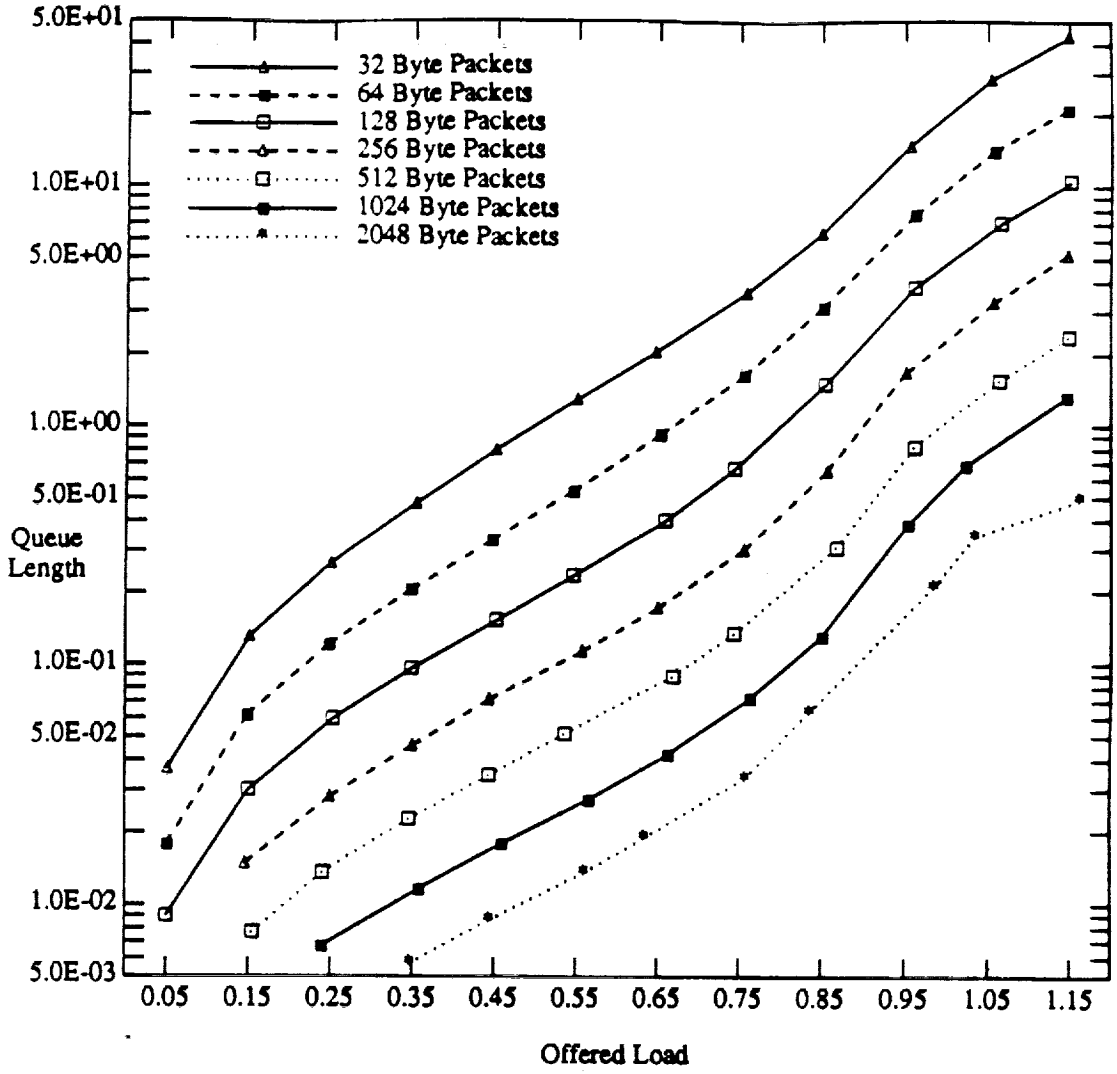
One interesting result presented in this graph is the token cycle time for a ring of 1000 stations. The target token rotation time for this network configuration is 9 ms but token cycle time has slightly exceeded that value. The reason for this is that FDDI allows a station to begin transmission of a packet even if the token hold timer will expire before transmission of that packet is complete. Because of this a station will occasionally release the token late. As the number of stations on the ring increases, it follows that the number of late token releases will also increase. Since all traffic for this configuration is synchronous, there is no asynchronous service to degrade. Consequently, it is difficult to make up for late token releases because at high offered loads each station uses nearly all the bandwidth available to it. An accumulation of late token releases at high offered load results in token cycle time exceeding its target. This accumulated delay is bounded, however. Sevcik and Johnson show in [SEVC85] that the maximum token cycle time is twice the target token rotation time.

4.2.3.2. Throughput

Figure 4.12 shows that increasing the number of stations on the ring results in decreased maximum throughput. The reason this happens is that as the number of stations increases, the token must spend less time at each station per token cycle to maintain the target token rotation time. As a result more bandwidth is consumed by token transmission and related overhead and less is available for packet transmission.

4.2.3.3. Arrival Queue Length

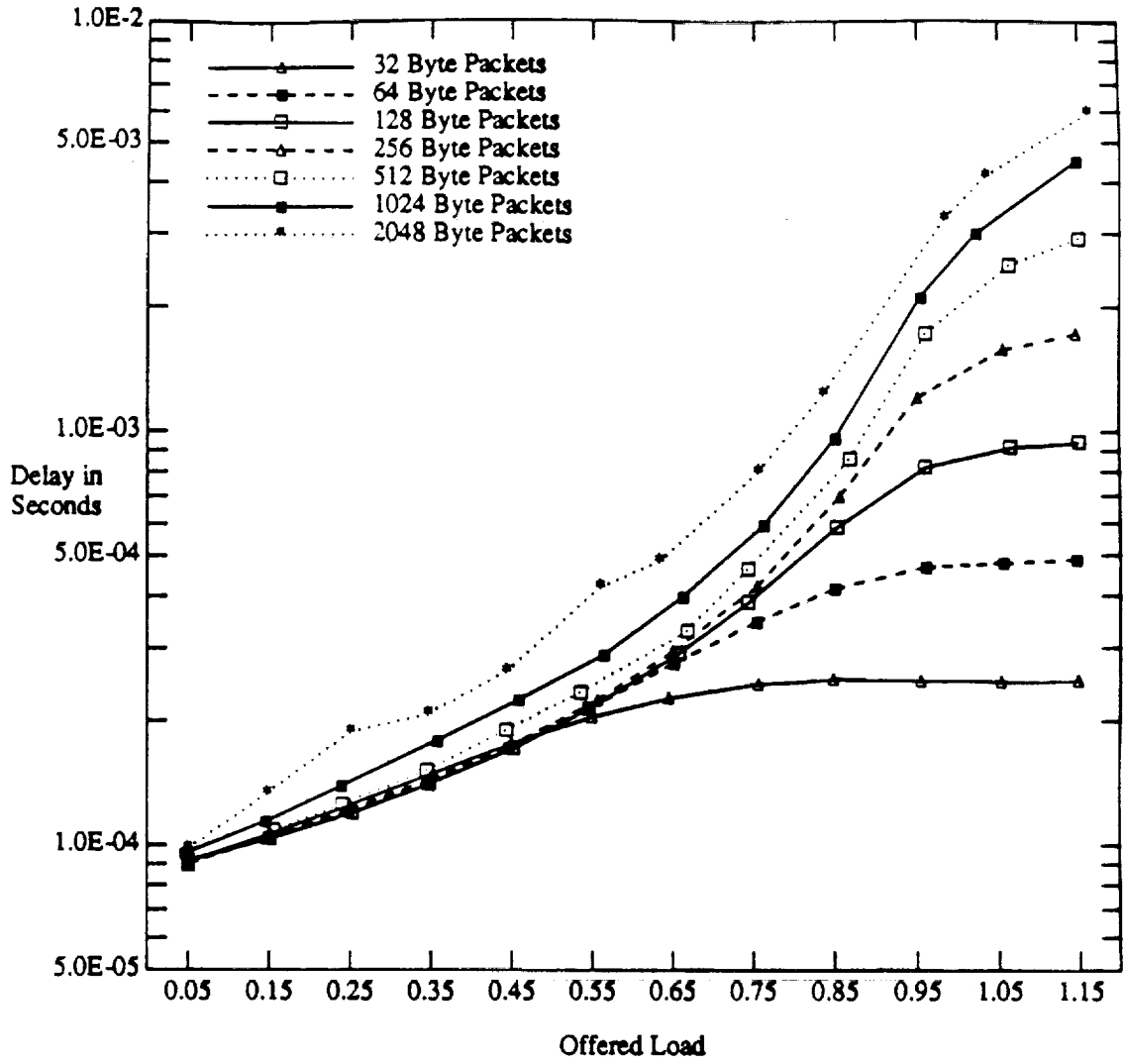
To maintain a certain offered load the network must offer a certain number of packets for transmission. This total number of packets is distributed evenly over all stations on the network. As the number of stations on the network goes down, the number of packets offered by each station must go up. Figure 4.13 shows that having fewer stations results in longer arrival queue lengths. The reason for this is that each station is offering more packets so more packets arrive between token accesses.



Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

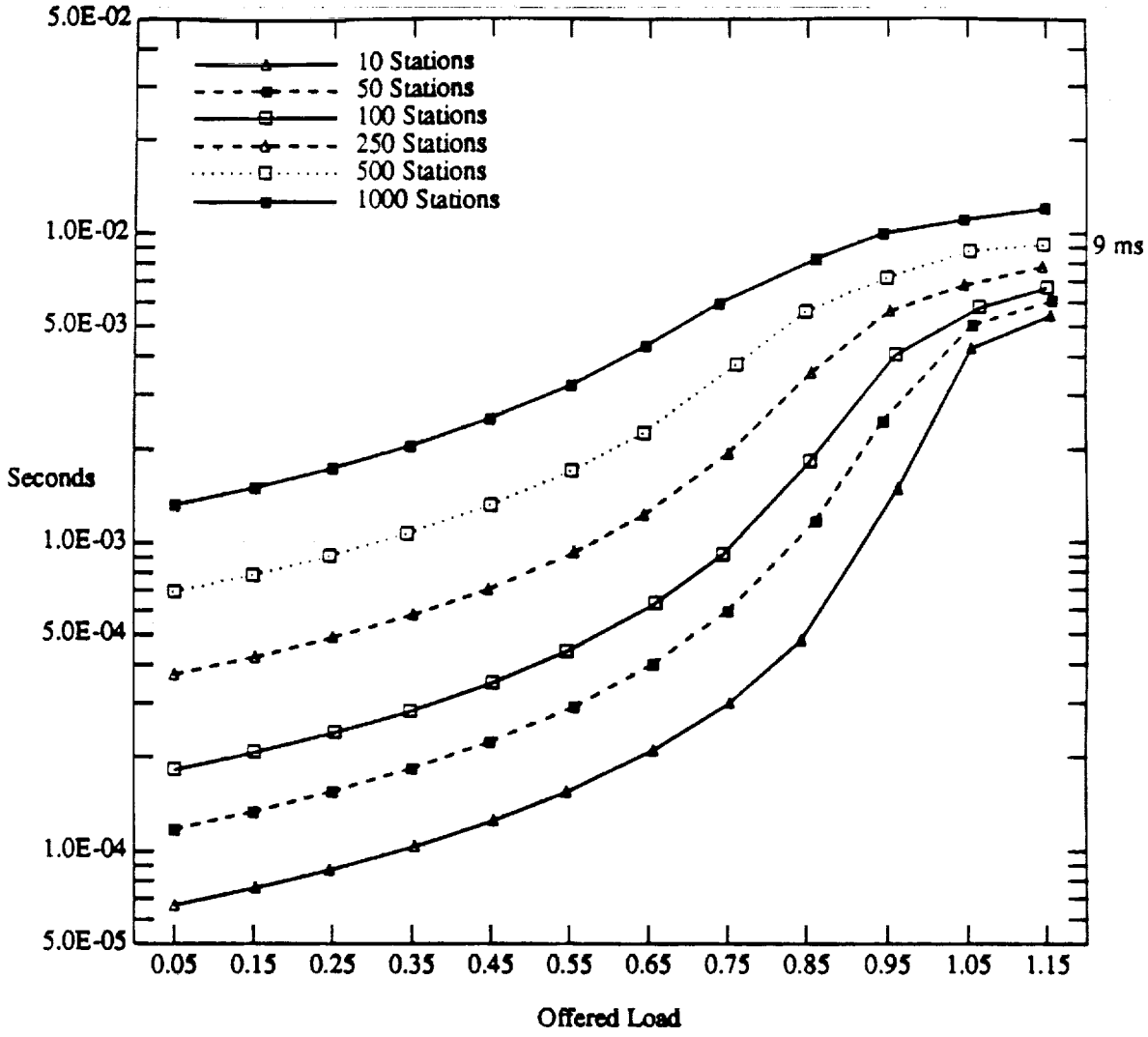
Figure 4.9 — Arrival Queue Lengths Varying Packet Length



Configuration:

100 stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
Varying Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

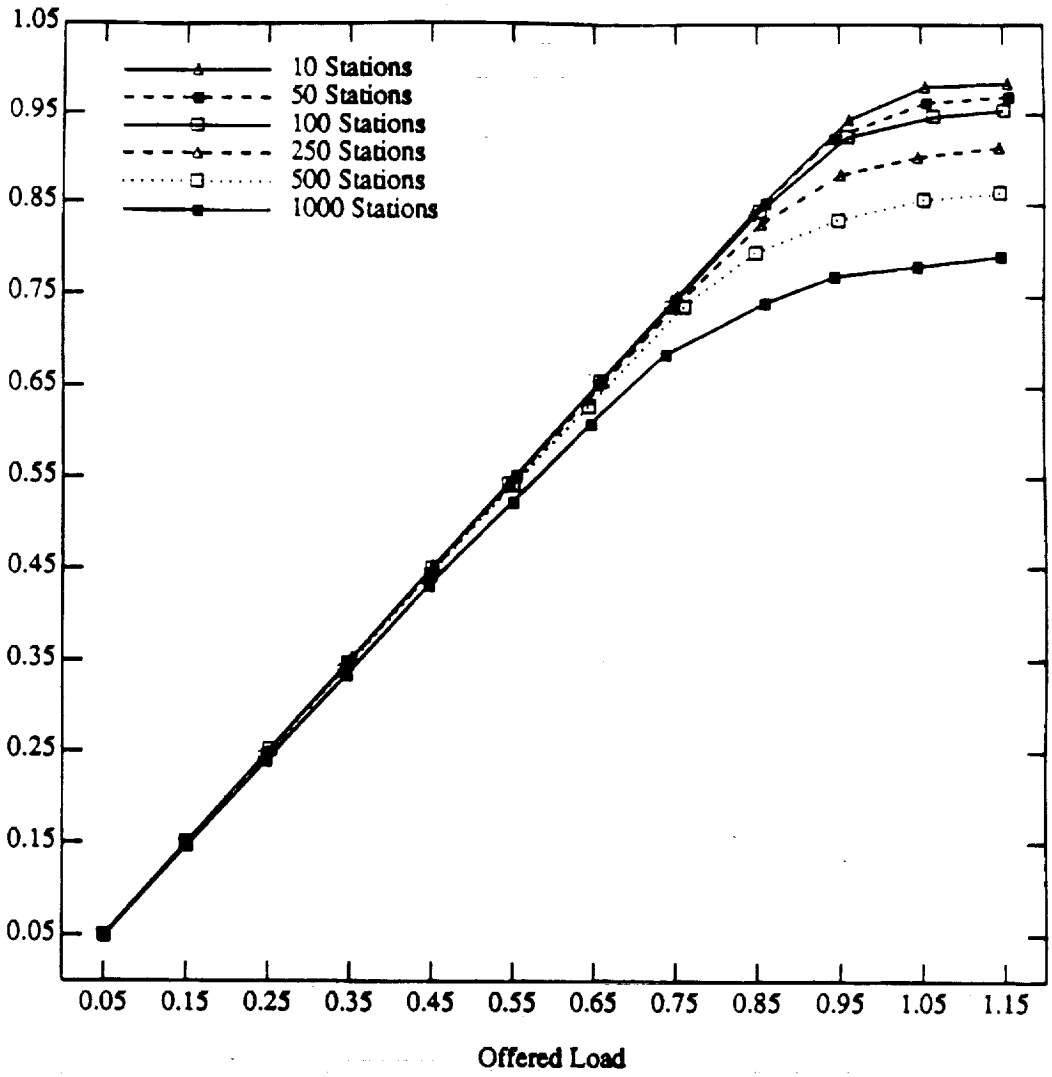
Figure 4.10 — Network Access Delay Varying Packet Length



Configuration:

Varying the Number of Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

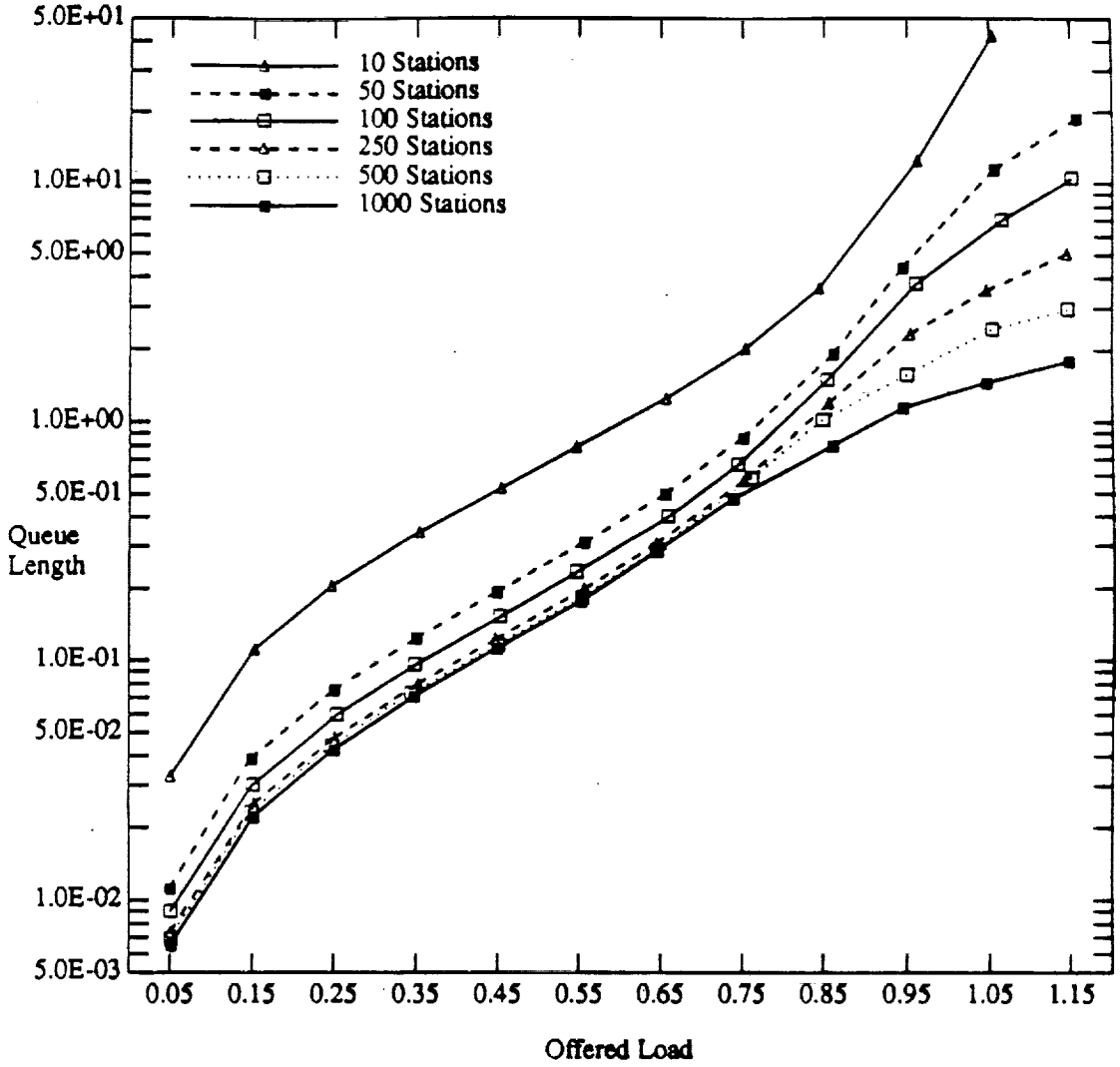
Figure 4.11 — Token Cycle Time Varying the Number of Stations



Configuration:

Varying the Number of Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.12 — Throughput Varying the Number of Stations



Configuration:

Varying the Number of Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.13 — Arrival Queue Lengths Varying the Number of Stations

4.2.3.4. Packet Delay

Figure 4.14 shows how varying the number of stations on the ring affects mean packet service delay. As stations are added to the ring, the mean number of stations a packet must go through to get to its destination (intermediate stations) will also increase. At each intermediate station a packet experiences a small amount of delay in the form of internal station latency. The more intermediate stations there are, the greater the packet's transmission time will be. The result is that service delay increases as the number of stations on the ring increases.

As offered load increases, a packet's queueing delay becomes so large that the delay incurred due to internal station latency becomes negligible. As a result we see the distance between the curves in Figure 4.14 decreases as offered load increases.

4.2.4. Effects of Varying Ring Circumference

The next network parameter of interest is ring circumference. We conducted simulations for 1, 10 and 100 km ring circumferences. Due to the dual counter-rotating ring, the effective ring circumference doubles when a link breaks and the ring goes into loopback mode. To ensure that FDDI's maximum ring circumference (200 km) is not exceeded when a break occurs, the maximum circumference for each of the dual rings is 100 km. For these simulations the only parameter varied is ring circumference.

4.2.4.1. Token Cycle Time and Throughput

Propagation delay is the amount of time it takes a bit to travel from one station to the next. Since stations on the ring are assumed to be equidistant from each other, propagation delay is proportional to ring circumference. The larger the ring, the greater the distance between stations.

Figure 4.15 shows the effect ring circumference has on token cycle time. As expected, the larger the ring, the longer it takes the token to make one complete cycle. The reason for this is that as the ring gets larger, propagation delay grows and it simply takes the token more time to travel between stations.

From Figure 4.16 we see that a larger ring circumference causes maximum throughput to drop. The reason for this is that as the distance between stations grows, the network has more time to wait

between token accesses. This additional waiting time consumes bandwidth that would otherwise be used for packet transmission. The result is a reduction to maximum throughput.

4.2.4.2. Packet Delay

Figure 4.17 shows how varying ring circumference affects mean packet service delay. From this graph we see that a larger ring causes service delay to increase. The reason for this is simply that as the ring grows, packets have farther to travel. As a result, the amount of delay a packet experiences will increase as the ring gets larger. It also follows that as the ring gets larger, packet transmission time will account for a greater portion of total packet delay.

4.3. Multiple Priority Operation

FDDI allows one synchronous priority and up to eight asynchronous priorities. The next section of simulation results presents the performance of FDDI when multiple priority levels are used.

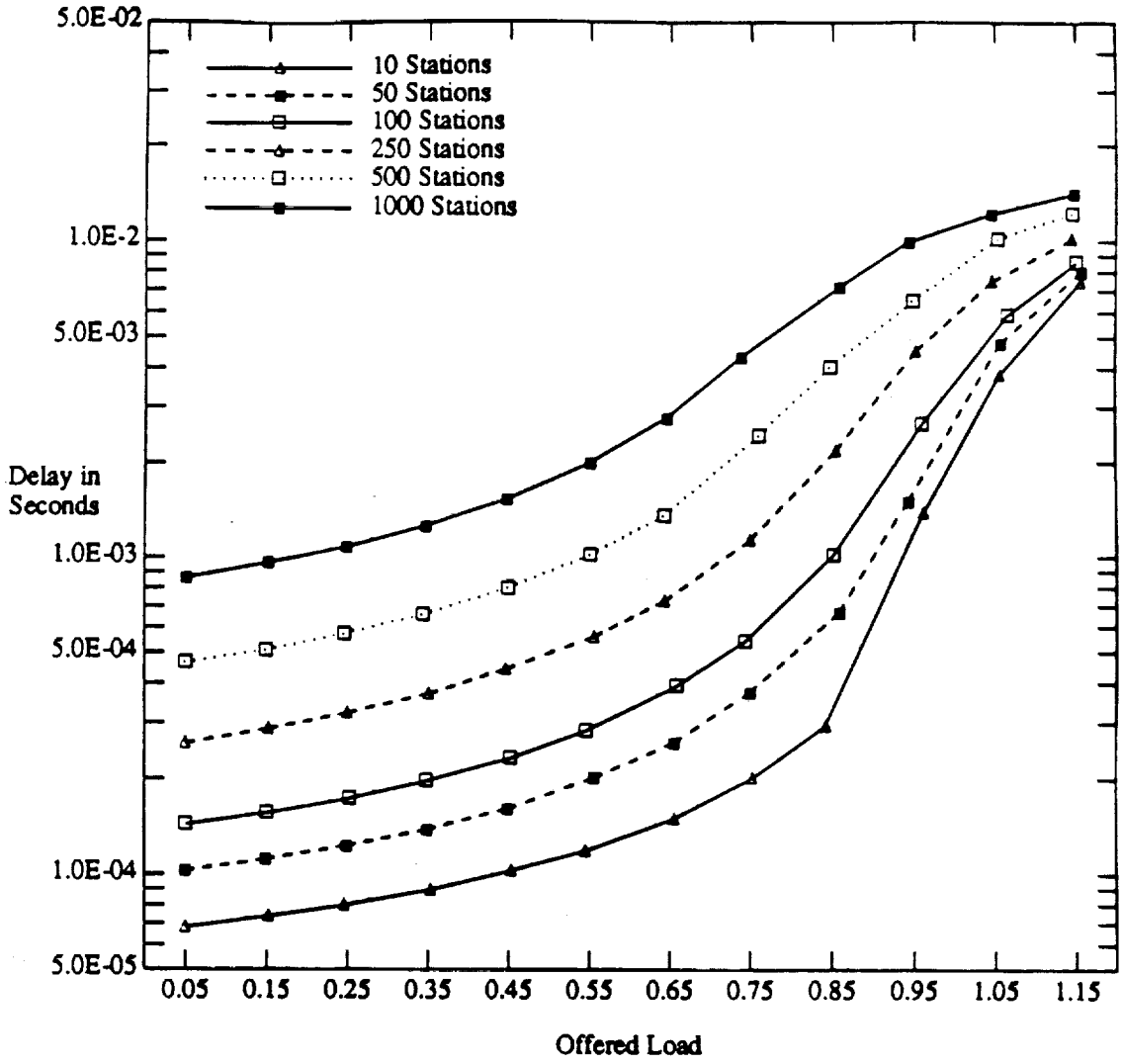
4.3.1. Reference Configuration

The reference configuration for multiple priority operation has the following characteristics:

100 stations evenly distributed on the ring
100 Mbps medium rate
10 km ring circumference
32 bit internal station latency
a synchronous priority and eight asynchronous priorities
9.0 ms target token rotation time
asynchronous priority thresholds held constant at 9.0 ms
exponential packet arrival distribution
1024 bit mean packet length distributed exponentially
no restricted token dialog
error free transmission

As before, the motivation for this configuration was simply that it provide a reasonable configuration to which variations can be made and that target token rotation time is long enough for the network to provide exhaustive service at high offered loads.

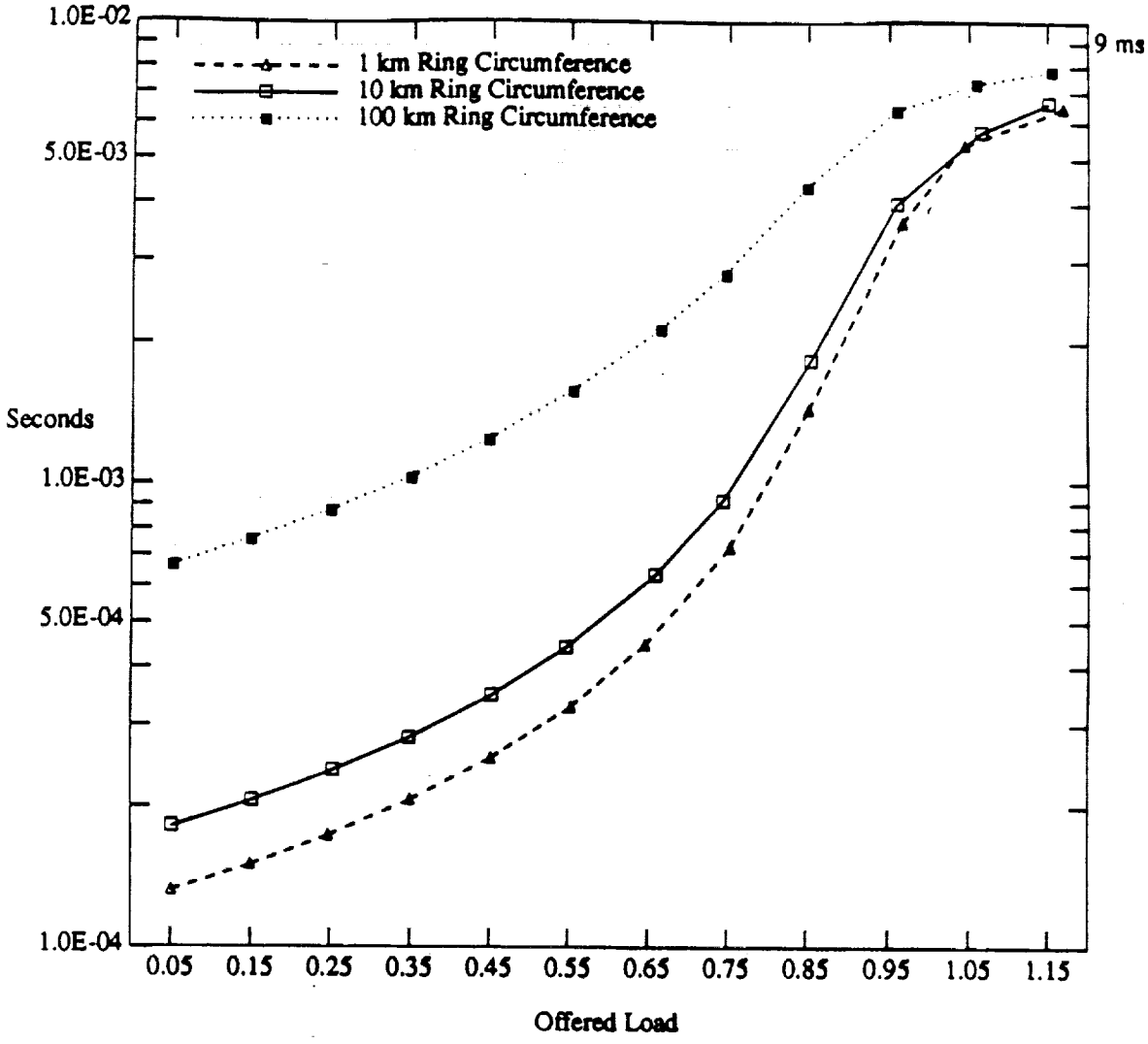
It is important to note that the asynchronous priority threshold values are all set equal to the target token rotation time; by doing this we have placed no artificial limitations on the asynchronous priorities. The only differentiation between the priorities is the order in which they are serviced.



Configuration:

Varying the Number of Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

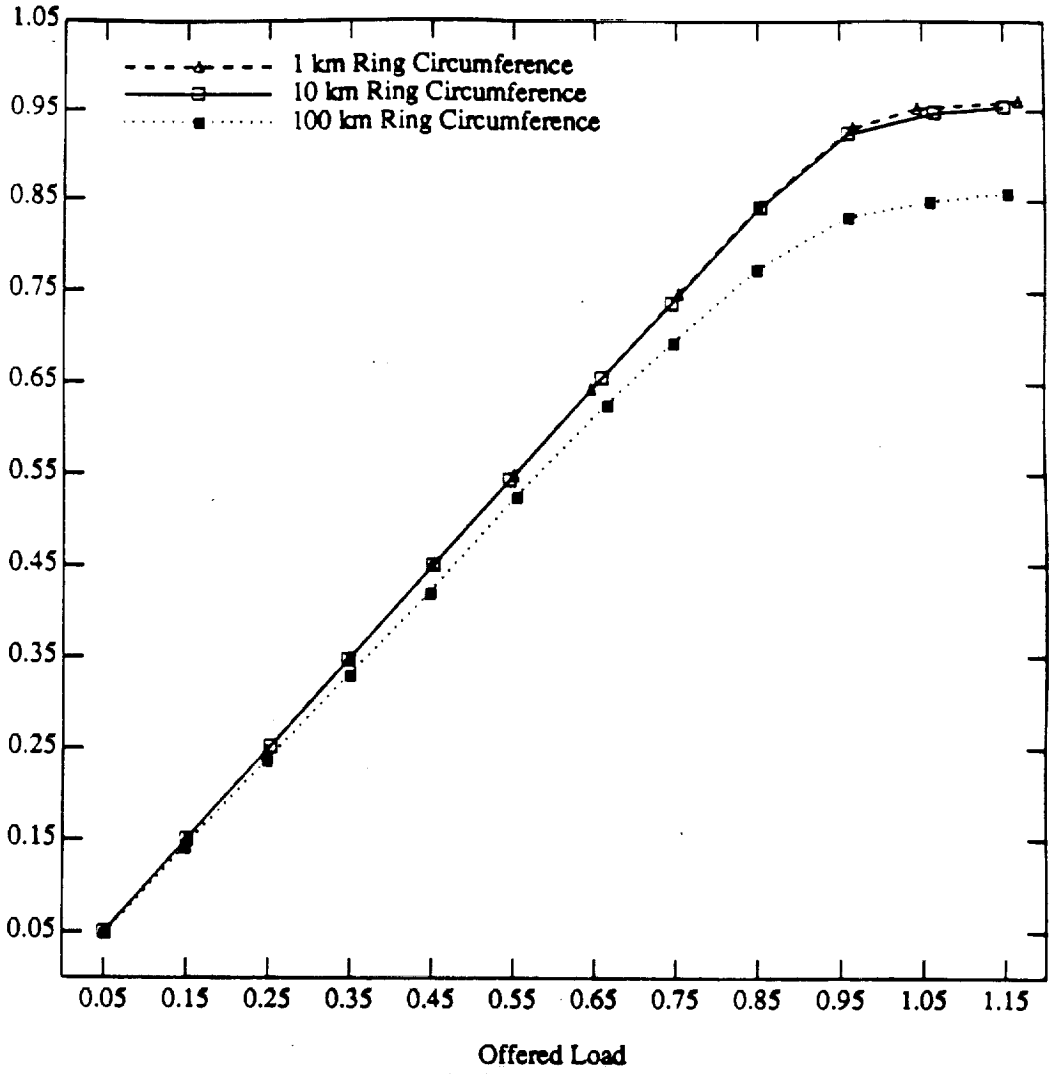
Figure 4.14 — Service Delay Varying the Number of Stations



Configuration:

100 Stations
100 Mbps Medium Rate
Varying Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

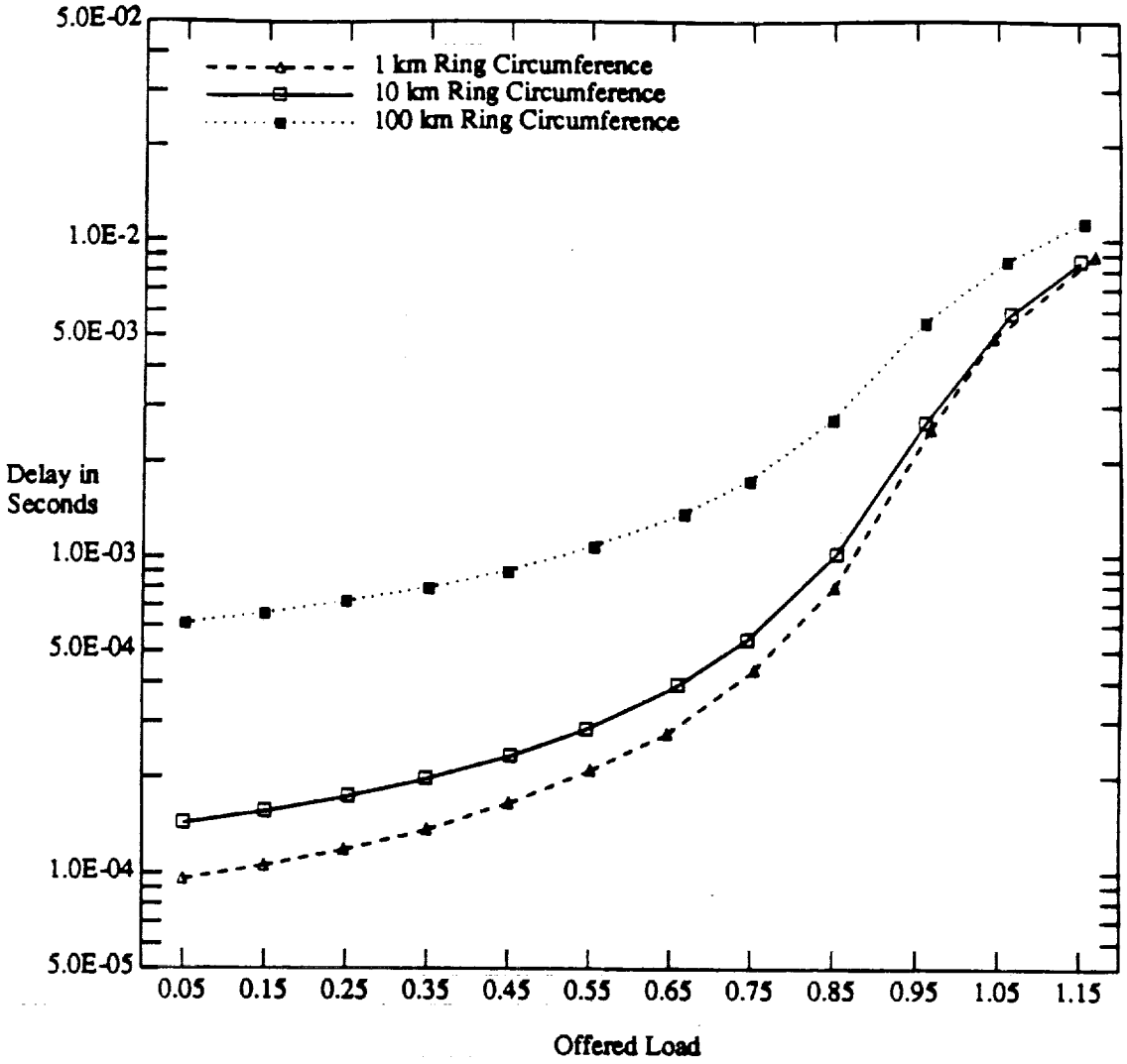
Figure 4.15 — Token Cycle Time Varying Ring Circumference



Configuration:

100 Stations
100 Mbps Medium Rate
Varying Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.16 — Throughput Varying Ring Circumference



Configuration:

100 Stations
100 Mbps Medium Rate
Varying Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority
9.0 ms Target Token Rotation Time
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.17 — Service Delay Varying Ring Circumference

4.3.1.1. Token Cycle Time and Throughput

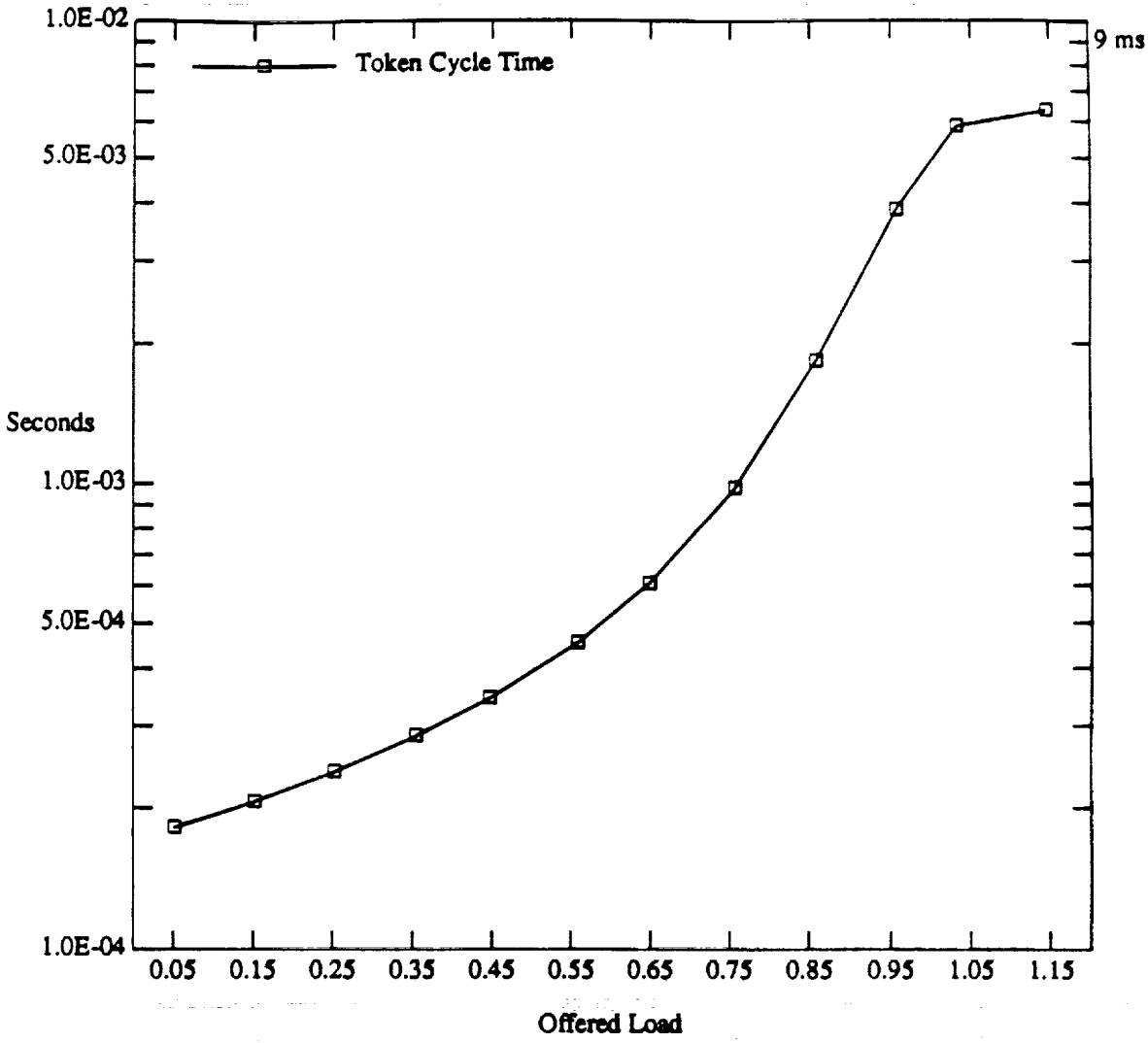
Figures 4.18 and 4.19 depict token cycle time and throughput respectively for the multiple priority reference configuration. These graphs are included to show that multiple priorities yield performance similar to that of a single priority. The graphs produced by the multiple priority reference configuration were so similar to those produced by the single priority reference configuration that inclusion of both sets would be redundant.

The similarity between the reference configurations is due primarily to the fact that no artificial restrictions were placed on the asynchronous priorities. Packet arrivals are assumed to be independent of the priority scheme implemented by the stations. In the first reference configuration all packets were offered at the same priority. In the second reference configuration packet arrivals are evenly distributed among the synchronous and eight asynchronous priorities. Since no artificial restrictions are placed on the asynchronous priorities, all are eligible to transmit packets until the station exhausts its synchronous bandwidth. As a result, asynchronous traffic receives service virtually identical to that of synchronous traffic.

Differences in the performance of the reference configurations would occur if synchronous traffic were heavy enough to consume nearly all the bandwidth available to a station. If this were the case then the need to maintain target token rotation time would result in asynchronous traffic going unserviced.

4.3.1.2. Packet Delay

Figure 4.20 shows service delay for the synchronous and three asynchronous priorities in the multiple priority reference configuration. From this graph we see that all priorities receive virtually the same delay until offered load exceeds the maximum throughput of this configuration. Only then do we see the lower priorities having significantly more delay than the higher priorities. The reason for this is that before offered load reached the maximum throughput there was enough bandwidth available to service all arriving packets. After maximum throughput is reached, there is no longer enough bandwidth available to service all arriving packets, so the packets in the lower priorities have to wait longer.

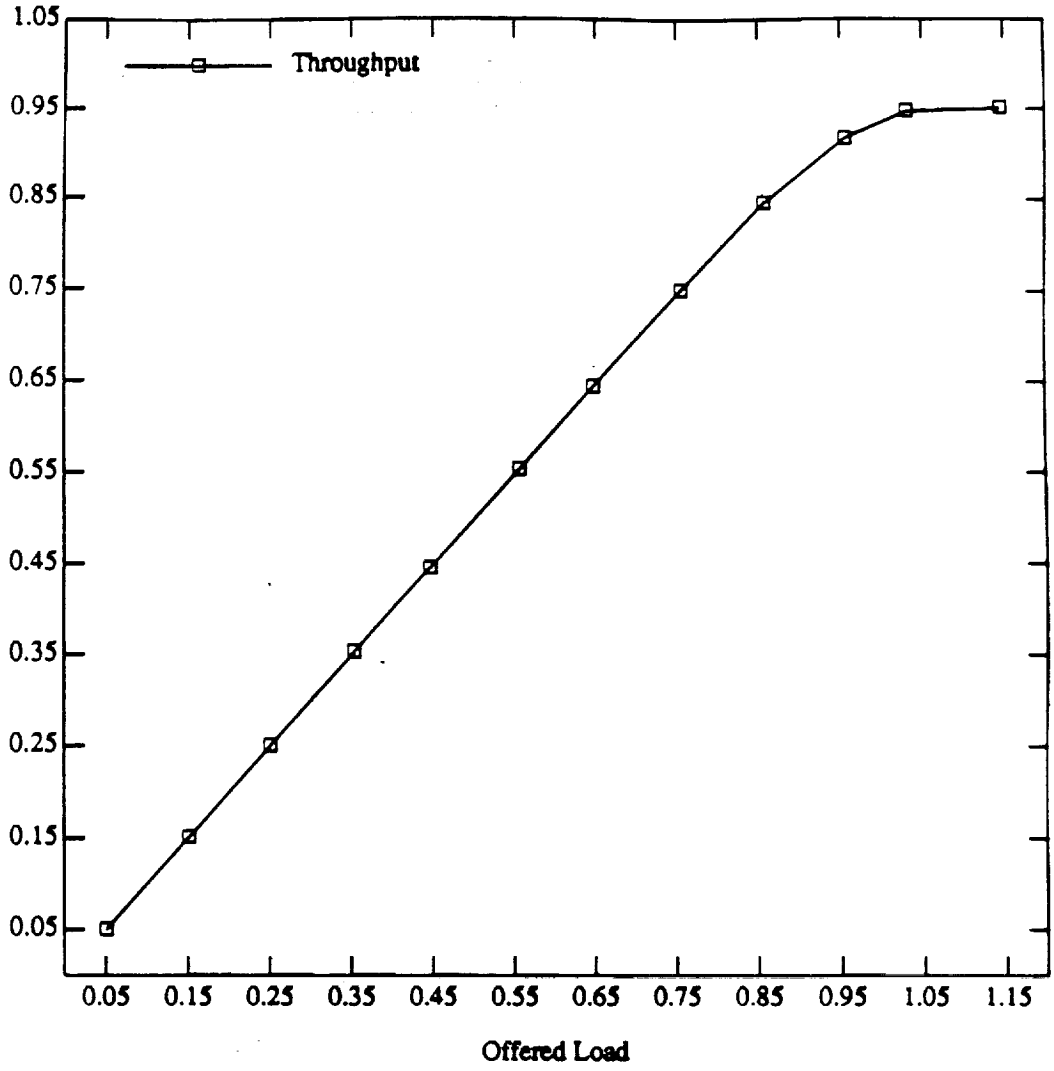


Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority And Eight Asynchronous Priorities
9.0 ms Target Token Rotation Time
Asynchronous Priority Thresholds Held Constant at 9.0 ms
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.18 — Token Cycle Time

Multiple Priority Reference Configuration

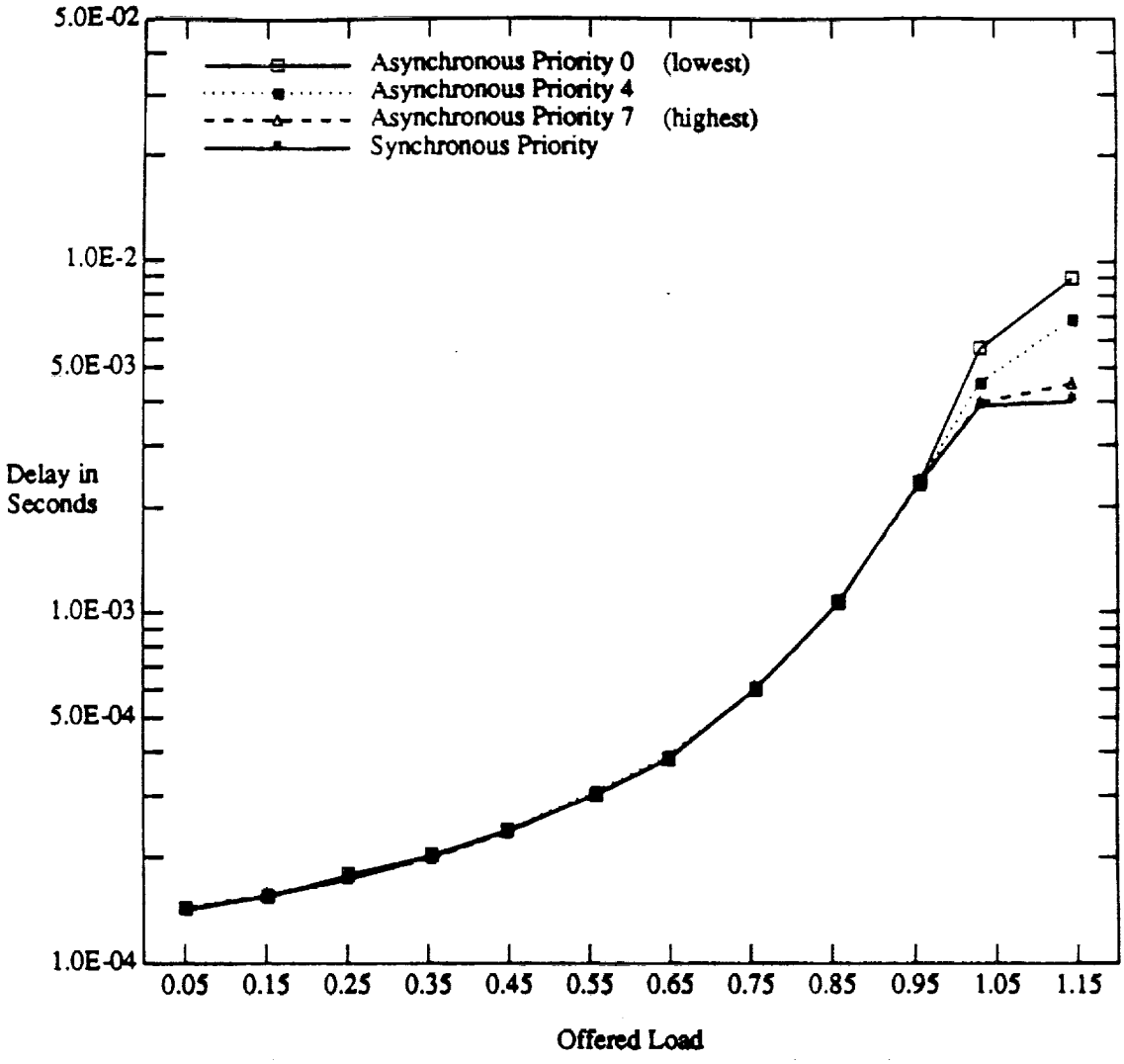


Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority And Eight Asynchronous Priorities
9.0 ms Target Token Rotation Time
Asynchronous Priority Thresholds Held Constant at 9.0 ms
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.19 — Throughput

Multiple Priority Reference Configuration



Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority And Eight Asynchronous Priorities
9.0 ms Target Token Rotation Time
Asynchronous Priority Thresholds Held Constant at 9.0 ms
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.20 — Synchronous And Asynchronous Service Delay

Multiple Priority Reference Configuration

4.3.2. Effects of Varying Asynchronous Priority Thresholds

Asynchronous priority thresholds are the means by which FDDI artificially throttles asynchronous traffic (see Section 2.6.2.3.2). The way they work is that the lowest asynchronous priority gets the lowest priority threshold value and the highest asynchronous priority gets the highest priority threshold value. A packet of a certain priority may only be transmitted if the station's token hold timer is less than the associated priority threshold value. If priority thresholds are not used (as in the multiple priority reference configuration), their default value is the target token rotation time.

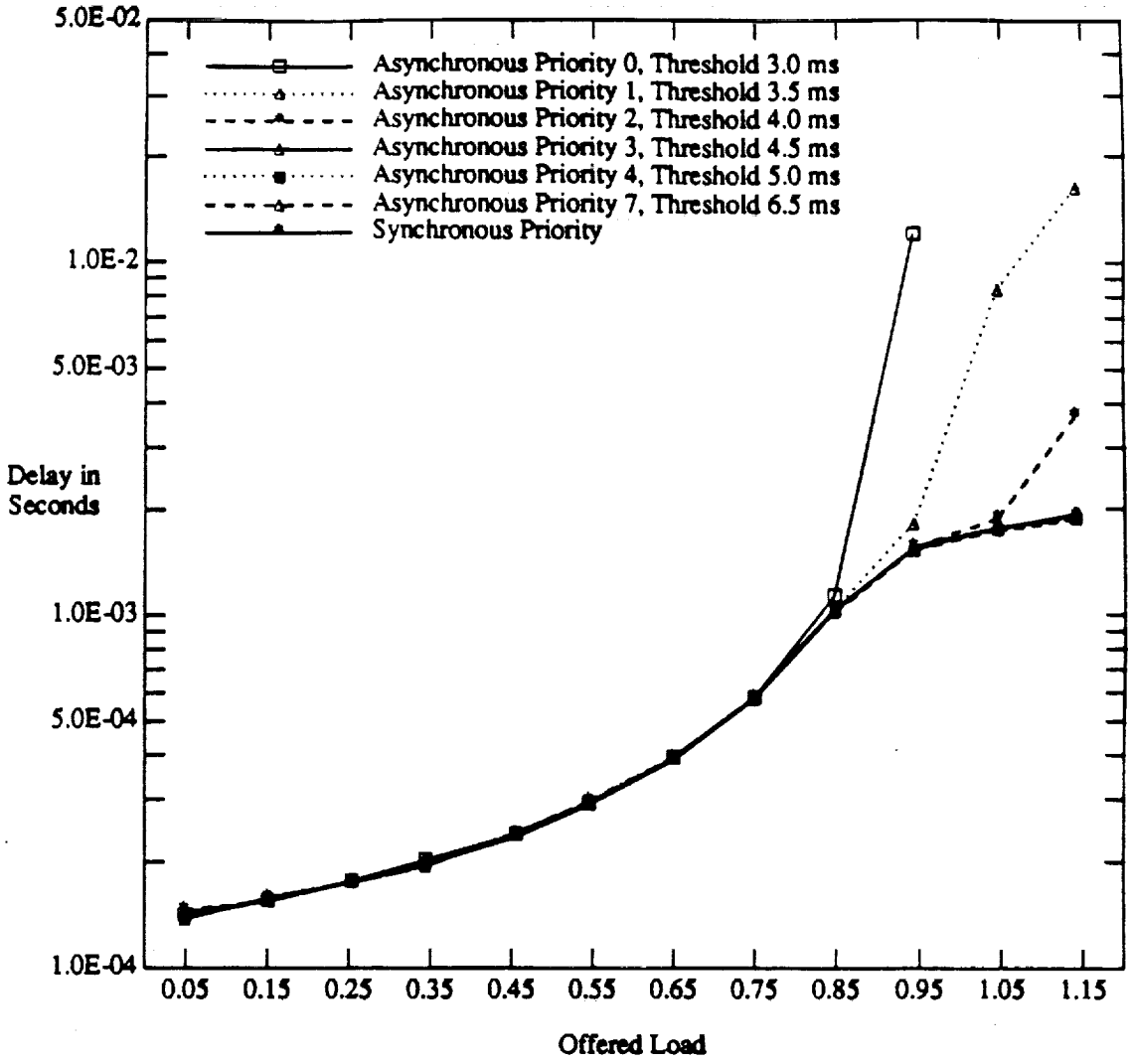
We conducted simulations for priority thresholds held constant at the target token rotation time, varied from 3.0 ms to 6.5 ms by 0.5 ms, and varied from 1.0 ms to 4.5 ms by 0.5 ms.

4.3.2.1. Packet Service Delay

Figures 4.21 and 4.22 show how reducing priority thresholds affects mean packet service delay. Figure 4.21 shows service delays with priority threshold values varied from 3.0 ms to 6.5 ms by 0.5 ms. At low offered loads all priorities receive adequate service. As offered load increases we see service to the lower asynchronous priorities begin to cut off. At 95% offered load the service delay at the lowest asynchronous priority has increased dramatically. The same thing happens to priority one at 105% offered load and to priority two at 115%.

The reason for this is that asynchronous priorities are serviced in order from highest (asynchronous priority 7) to lowest (asynchronous priority 0). By the time packets in asynchronous priority 0 are eligible for service at 95% offered load, the station's token hold timer has advanced to the point where it is greater than priority zero's threshold value. As a result, service to asynchronous priority zero is virtually cut off.

Figure 4.22 shows a similar phenomenon, when the priority thresholds are varied from 1.0 ms to 4.5 ms by 0.5 ms. The difference here is that service cut off to lower priorities begins earlier. On this graph we see service to priorities zero, one and two completely cut off and service to priority three beginning to cut off as offered load increases.

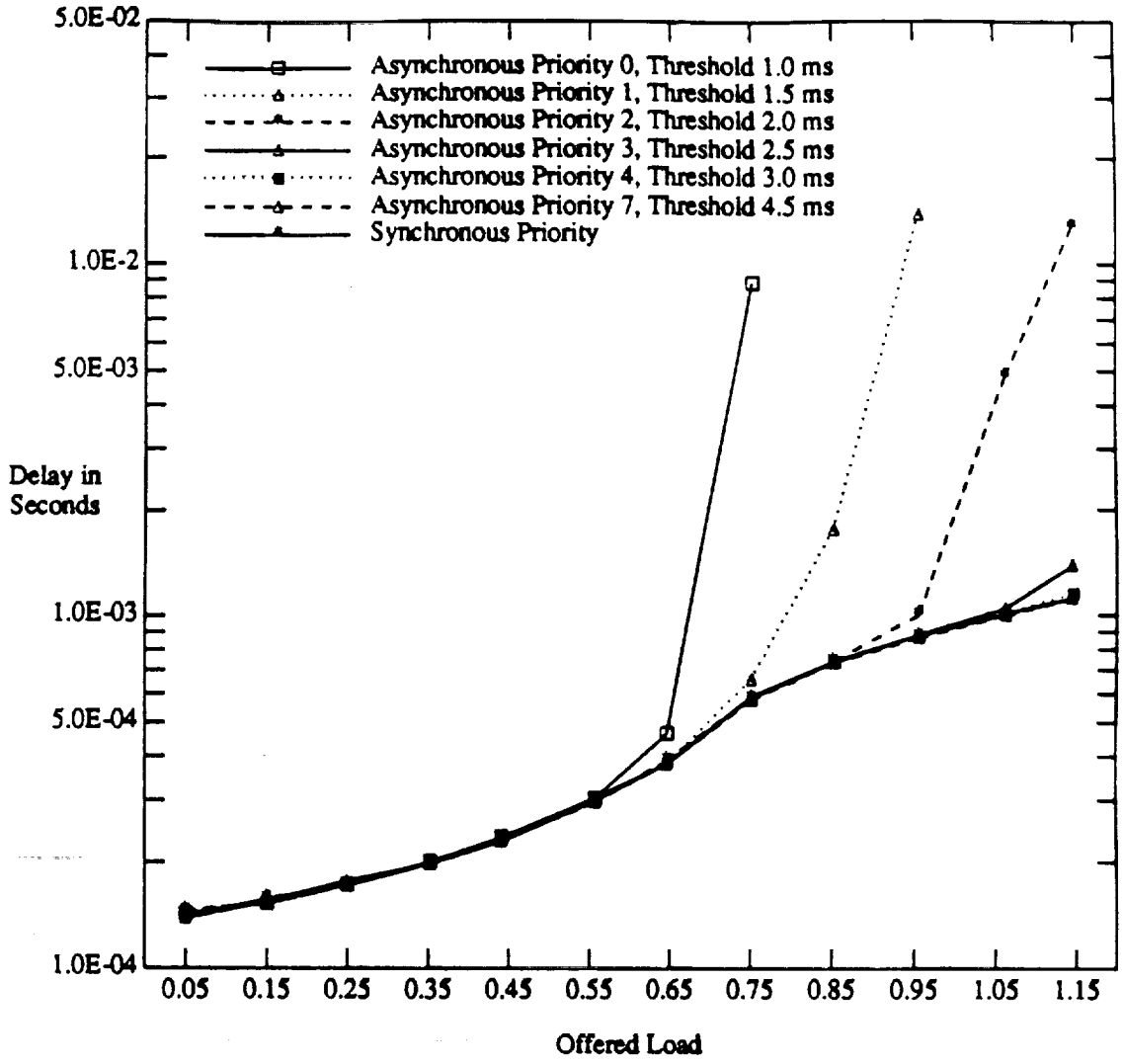


Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority And Eight Asynchronous Priorities
9.0 ms Target Token Rotation Time
Asynchronous Priority Thresholds Varied From 3.0 ms to 6.5 ms by 0.5 ms
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.21 — Synchronous And Asynchronous Service Delay

Priority Thresholds Varied From 3.0 ms to 6.5 ms by 0.5 ms



Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority And Eight Asynchronous Priorities
9.0 ms Target Token Rotation Time
Asynchronous Priority Thresholds Varied From 1.0 ms to 4.5 ms by 0.5 ms
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.22 — Synchronous And Asynchronous Service Delay

Priority Thresholds Varied From 1.0 ms to 4.5 ms by 0.5 ms

4.3.2.2. Token Cycle Time

Figure 4.23 shows how varying priority thresholds affects token cycle time. We saw in Figures 4.21 and 4.22 that as offered load increases, priority thresholds begin to cut off service to the lower asynchronous priorities. When this happens the token finds fewer packets per station that are eligible for transmission, so the station releases the token earlier than it normally would if no priority thresholds were in use. The lower the threshold values are set, the fewer packets the token has to service relative to the number it usually services at that offered load. The result is that lowering priority thresholds causes token cycle time to decrease relative to the reference configuration.

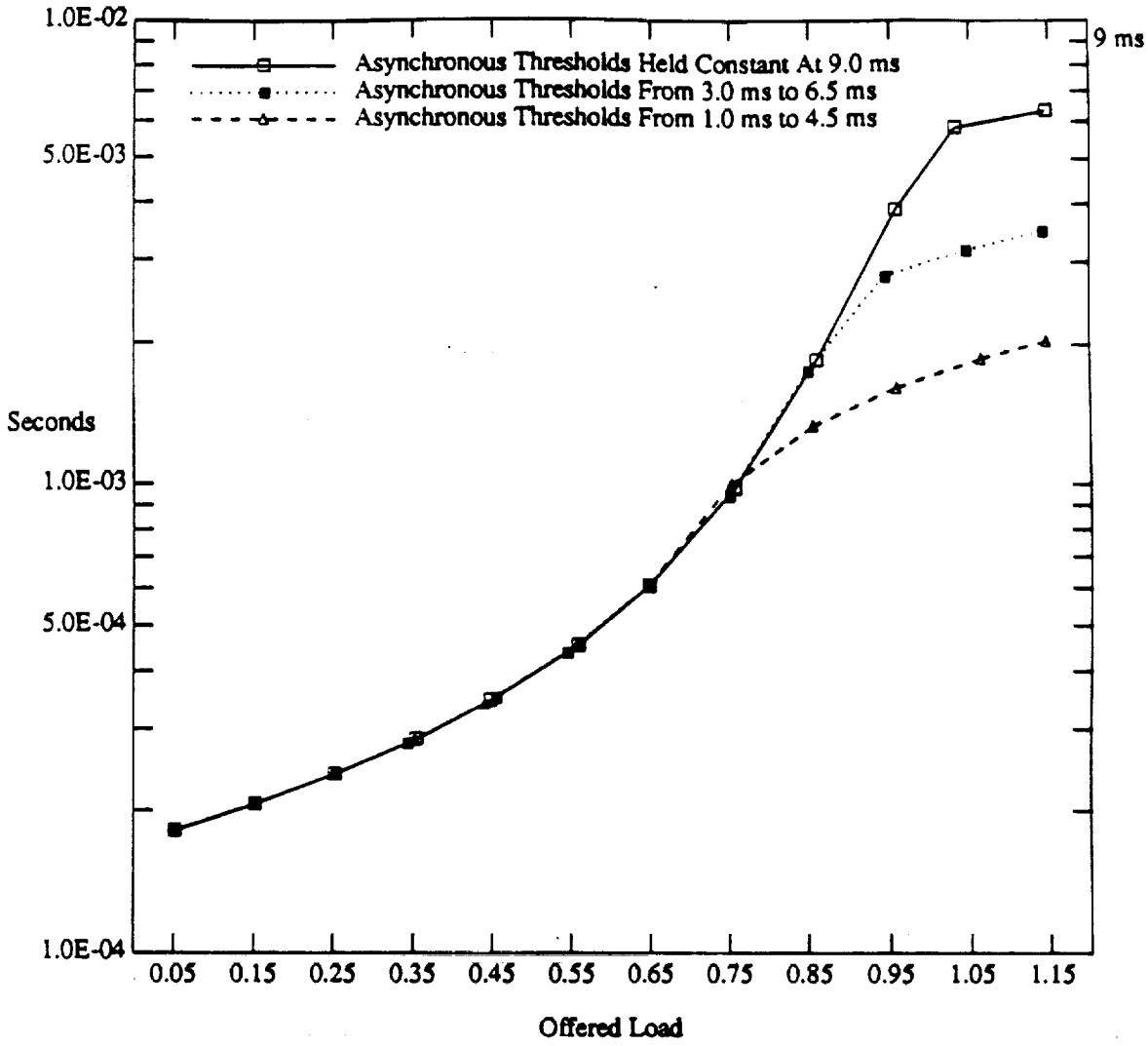
4.3.2.3. Throughput

Lower token cycle time implies that more token transmissions are taking place in a given amount of time. The effect, which can be see in Figure 4.24, is that more bandwidth is consumed by token transmission, resulting in a lower maximum throughput relative to the reference configuration.

4.3.3. Effects of Lowering Target Token Rotation Time

Target token rotation time is a network-wide value negotiated by the stations at ring initilization. It represents the longest token cycle time usable by the station with the most time critical traffic. In our reference configurations, the target token rotation time was set at 9 ms to ensure that the ring would receive exhaustive service even at high offered load. The asynchronous priority thresholds were also set at this value to ensure that the asynchronous priorities were not artificially restricted.

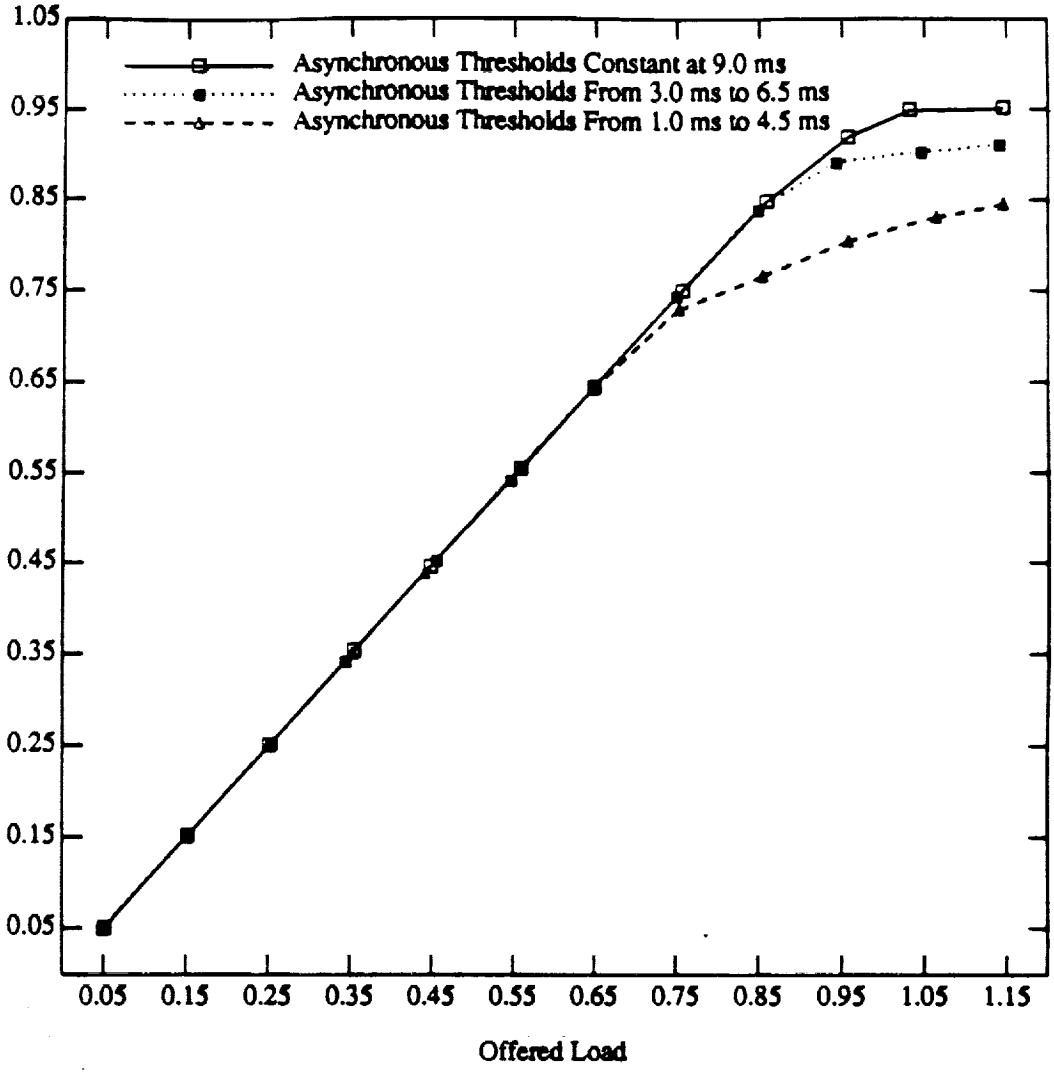
We conducted a series of simulations with the target token rotation time and all asynchronous priority thresholds set to 1 ms. The motivation for this value was that it provided exhaustive service at low offered loads only. In these simulations the target token rotation time and asynchronous priority thresholds were the only parameters varied.



Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority And Eight Asynchronous Priorities
9.0 ms Target Token Rotation Time
Asynchronous Priority Thresholds Varied From 3.0 ms to 6.5 ms
And From 1.0 ms to 4.5 ms by 0.5 ms
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.23 — Token Cycle Time Varying Priority Thresholds



Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority And Eight Asynchronous Priorities
9.0 ms Target Token Rotation Time
Asynchronous Priority Thresholds Varied From 3.0 ms to 6.5 ms
And From 1.0 ms to 4.5 ms by 0.5 ms
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.24 — Throughput Varying Priority Thresholds

4.3.3.1. Token Cycle Time

Figure 4.25 plots the token cycle time of the multiple priority reference configuration against the token cycle time of a network with 1 ms target token rotation time. This graph shows that lowering target token rotation time causes the token cycle time to stabilize at a lower offered load. The reason for this is that lowering the target token rotation time reduces the maximum time the token may spend at each station. The result is that fewer packets need to be waiting when the token arrives to cause it to stay as long as the target token rotation time will allow. As we see in Figure 4.25, the network configuration with a 1 ms target token rotation time reached this limit at approximately 75% offered load.

Because token cycle time is so important to network performance, FDDI monitors the token's progress at each station. When the token arrives late at a given station (i.e., token cycle time is greater than the target token rotation time), that station may not use the token for its asynchronous traffic. In this way, token cycle time is bounded by its target value. As a result, token cycle time rises steadily until it reaches the target token rotation time. The late token arrival mechanism then takes over and causes token cycle time to stabilize quickly.

4.3.3.2. Throughput

Figure 4.26 plots the throughput of the multiple priority reference configuration against the throughput of a network with a 1 ms target token rotation time. This graph shows that lowering the target token rotation time decreases maximum throughput relative to the reference configuration.

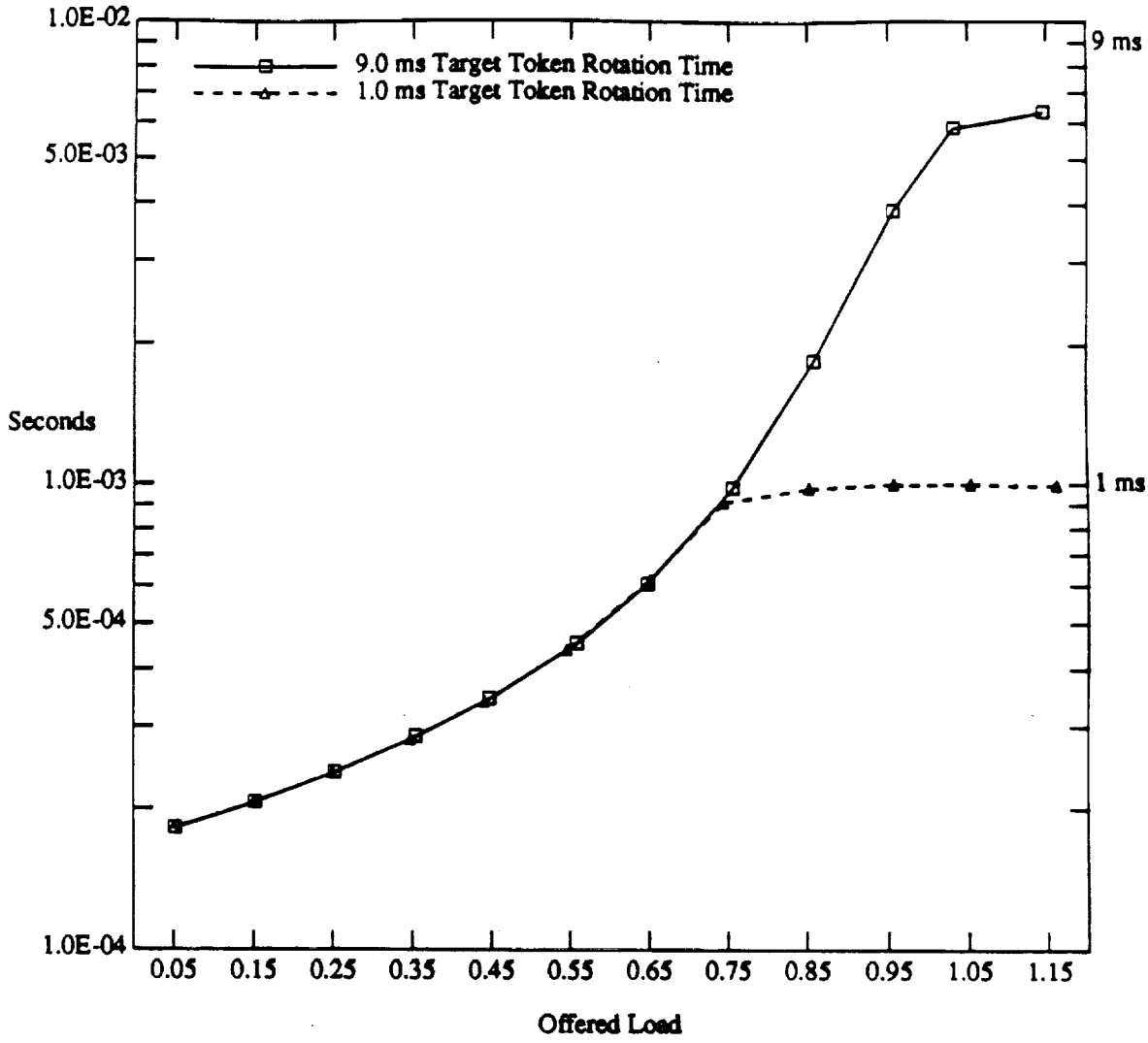
As we saw in Figure 4.25, lowering target token rotation time caused token cycle time to decrease relative to the reference configuration. A lower token cycle time means that more token transmissions are taking place in a given amount of time. Token transmissions and the related overhead consume bandwidth that would otherwise be used for packet transmission. The result is that lowering the target token rotation time will in general lower the maximum throughput.

An effect of FDDI's token cycle time monitoring can be seen by comparing Figures 4.25 and 4.26. At 75% offered load, both network configurations have nearly the same token cycle times, which implies that nearly the same number of token transmissions are taking place. However, throughput for each of the networks is significantly different at 75% offered load. The reason for this is that at 75% offered load, tokens in the network with a 1 ms target token rotation time begin to arrive late. This results in fewer packets being sent per token access, which is why we see a reduction in throughput. At 85% offered load, token cycle time for the network with a 1 ms target token rotation time has risen slightly, resulting in more late tokens and lower throughput as compared to 75% offered load. Throughput eventually stabilizes when token cycle time stabilizes.

4.3.3.3. Packet Delay

Figure 4.27 shows mean packet delay for the synchronous and three asynchronous priorities in a network with a 1 ms target token rotation time. As offered load increases, mean service delay for all priorities remains comparable until the target token rotation time begins to limit token cycle time. At that point (75% offered load), tokens begin to arrive late and mean service delay for all asynchronous traffic is uniformly higher. The reason for the uniform increase is that late token arrivals temporarily suspend service to all asynchronous priorities. The result is a uniform increase in the mean service delay of all asynchronous traffic.

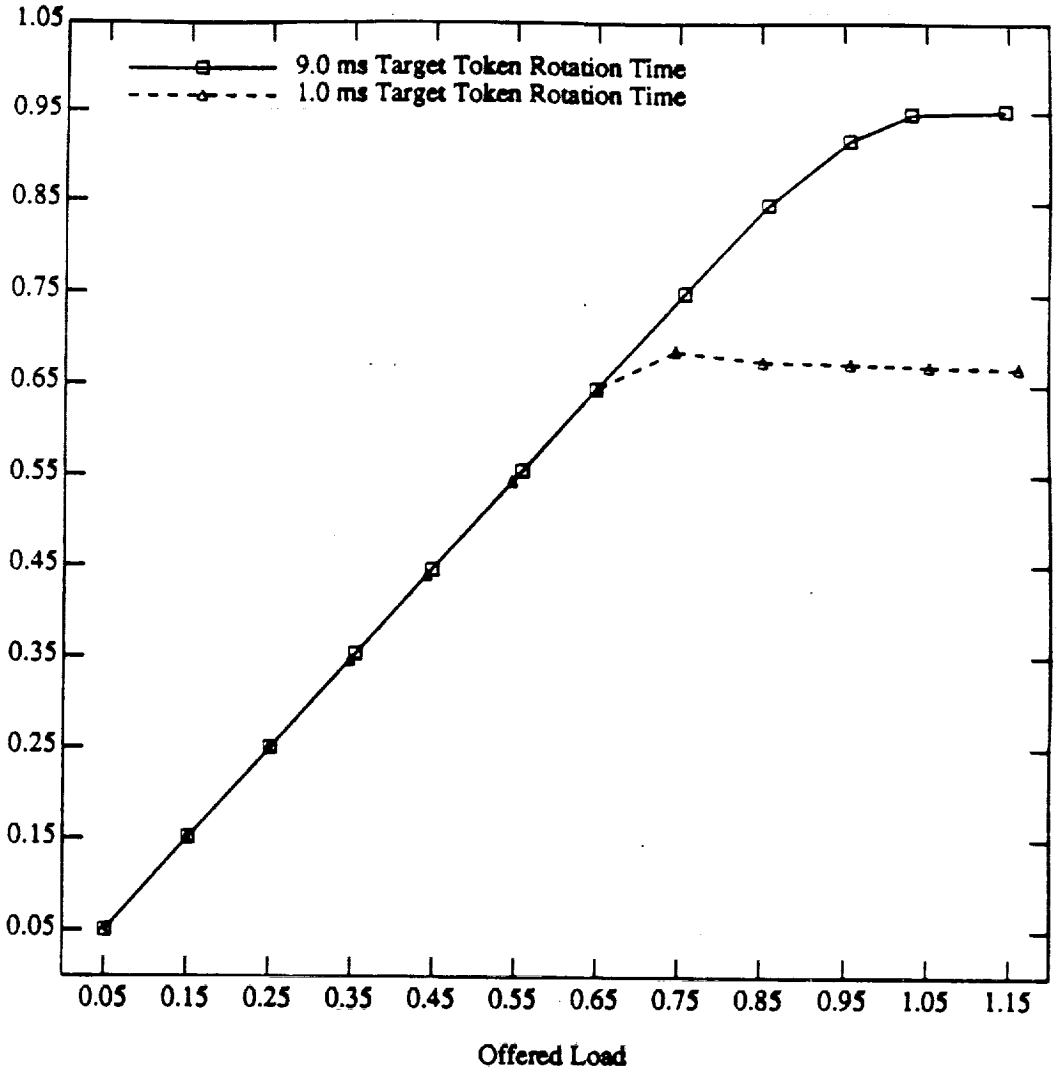
As offered load increases, service to synchronous traffic remains fairly constant, but service to the asynchronous traffic begins to decline. When a station receives a token that is not late, the highest asynchronous priorities are likely to get service, but due to the effect of late tokens, the high asynchronous priorities may have so many packets waiting for transmission that the station's token hold timer may expire before the lower asynchronous priorities get serviced. The result is that lower asynchronous priorities receive even longer mean service delays and eventually get no service at all. Beyond 95% offered load asynchronous priority zero receives service for only a fraction of the packets it offers.



Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority And Eight Asynchronous Priorities
Target Token Rotation Time Varied
Asynchronous Priority Thresholds Held Constant at TTRT
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

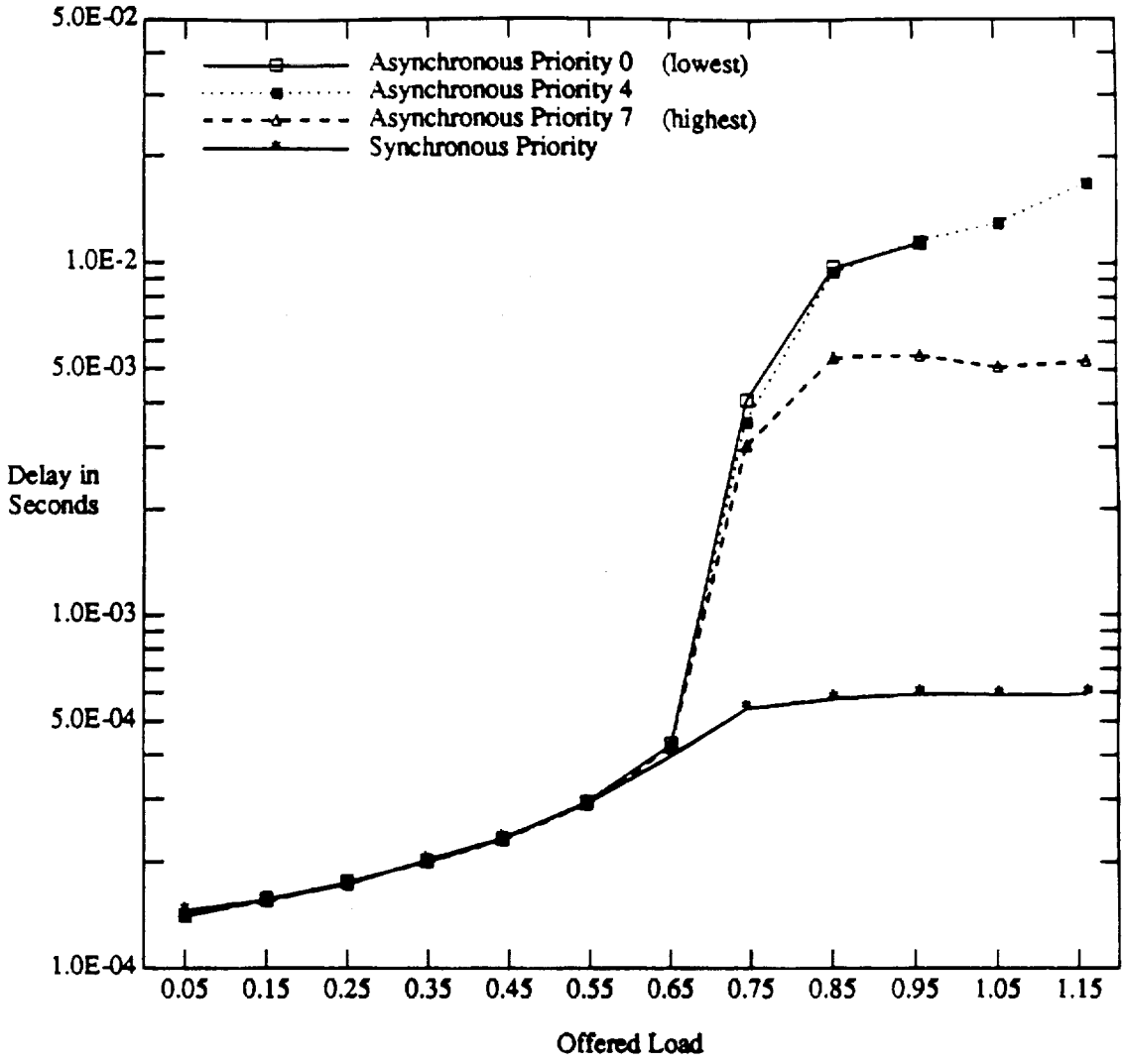
Figure 4.25 — Token Cycle Time Varying Target Token Rotation Time



Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority And Eight Asynchronous Priorities
Target Token Rotation Time Varied
Asynchronous Priority Thresholds Held Constant at TTRT
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.26 — Throughput Varying Target Token Rotation Time



Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority And Eight Asynchronous Priorities
Target Token Rotation Time Varied
Asynchronous Priority Thresholds Held Constant at TTRT
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
No Restricted Token Dialog

Figure 4.27 — Synchronous And Asynchronous Service Delay

1 ms Target Token Rotation Time

4.3.4. Effects of Restricted Token Dialog

Restricted token dialog is used to dedicate asynchronous bandwidth to a single extended dialog between a certain subset of stations (see Section 2.6.2.3.3). To see the effect restricted token dialog has on asynchronous traffic we have altered the multiple priority reference configuration so that 50 stations are eligible for restricted token dialog and 50 stations are not. Also, for this example we make the unrealistic assumption that the simulation will be in a restricted token dialog 60% of the time. We do this simply to make the effect of restricted token communication very clear.

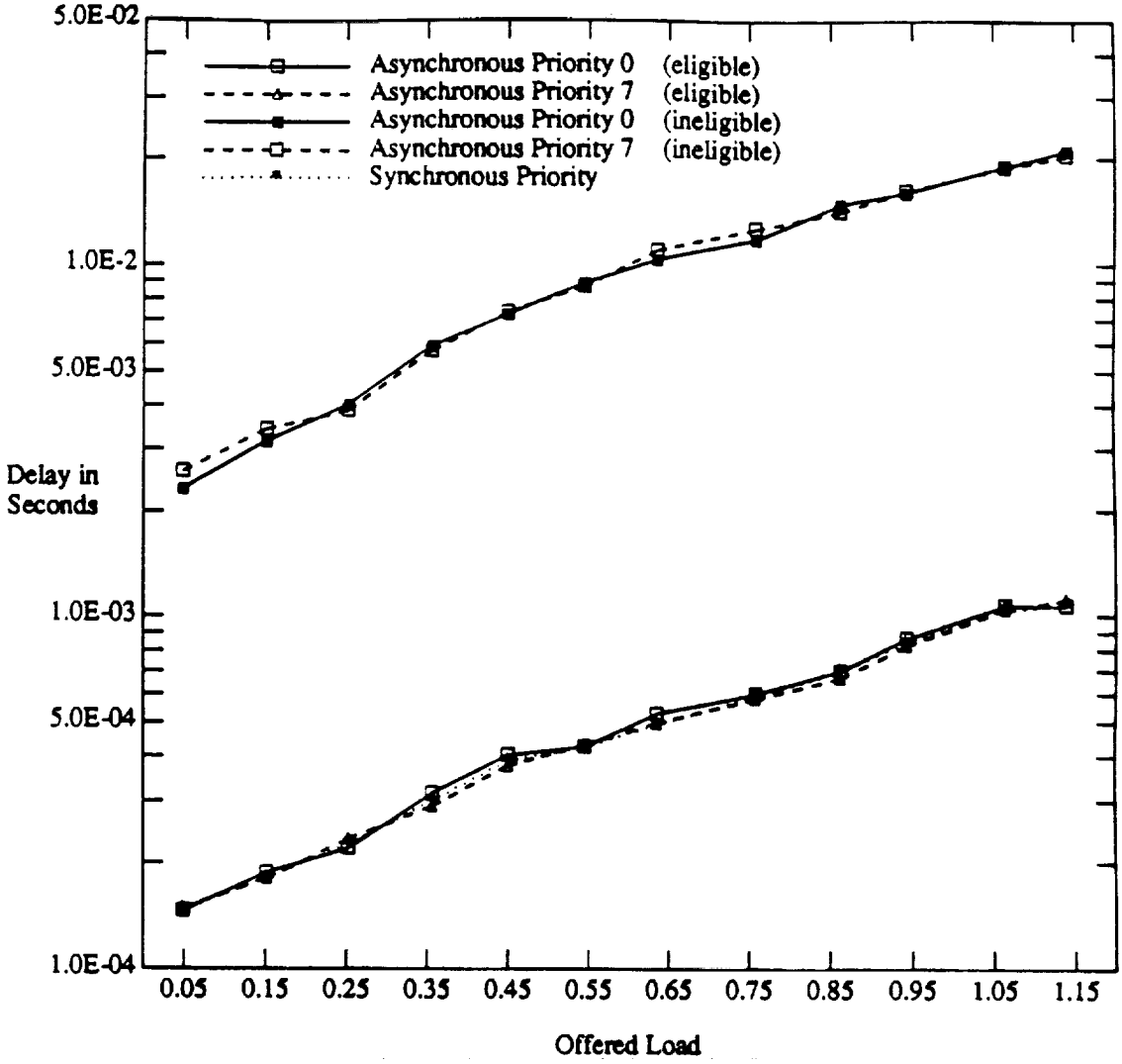
4.3.4.1. Packet Delay

Figure 4.28 shows mean packet service delay for the synchronous priority and the highest and lowest asynchronous priorities of stations that were eligible and ineligible for restricted token dialog. This graph shows that asynchronous priorities for the stations not eligible for a restricted token had uniformly higher service delay.

The reason this occurs is that restricted token dialog occurs in spurts. During these spurts, stations not eligible to participate receive no service for their asynchronous priorities. When the restricted token dialog is finished, service to all stations reverts to normal. The asynchronous packets that were not serviced during the restricted token dialog incurred a certain amount of delay in addition to the delay they would normally receive. This caused the delay of asynchronous priorities in stations not eligible for restricted token dialog to be uniformly higher than the delay incurred by the synchronous priority and the asynchronous priorities in stations eligible for restricted token dialog.

4.3.4.2. Throughput

Figure 4.29 shows that maximum throughput decreases significantly when restricted token dialog is implemented. The reason for this is that during restricted token dialog, stations not participating can only access the token for their synchronous traffic. This implies that ineligible stations hold the token for the short period of time necessary to service their synchronous traffic and then release it. The result is an increase in the number of token transmissions which in turn decreases throughput and the ineligible stations are not generating asynchronous traffic during restricted token dialog.

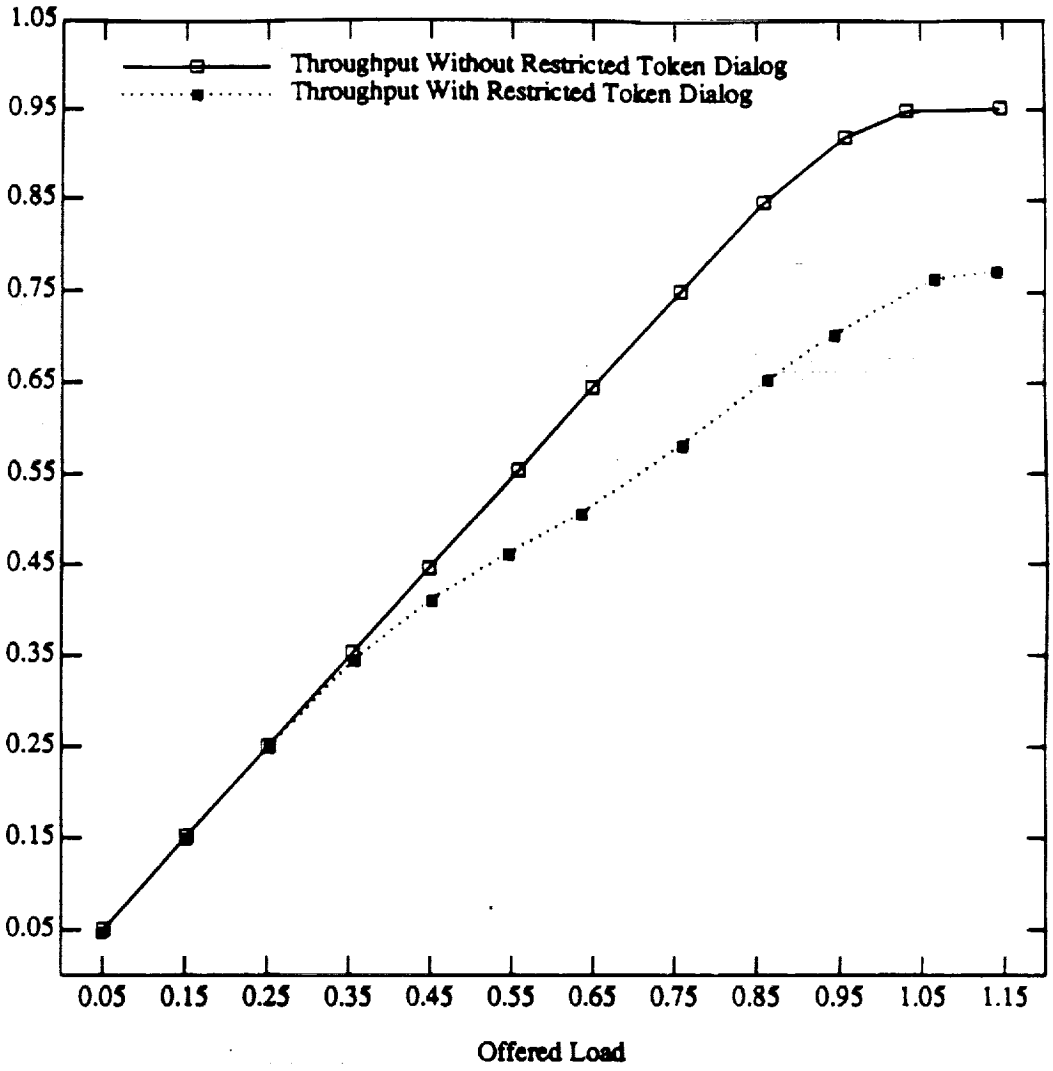


Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority And Eight Asynchronous Priorities
9.0 ms Target Token Rotation Time
Asynchronous Priority Thresholds Held Constant at 9.0 ms
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
Eligibility For Restricted Token Dialog Varied
60% of Simulation Time in Restricted Token Mode

Figure 4.28 — Synchronous And Asynchronous Service Delay

Eligible Station vs. Ineligible Station



Configuration:

100 Stations
100 Mbps Medium Rate
10 km Ring Circumference
32 Bits Internal Station Latency
A Single Synchronous Priority And Eight Asynchronous Priorities
9.0 ms Target Token Rotation Time
Asynchronous Priority Thresholds Held Constant at 9.0 ms
Exponential Packet Arrival Distribution
1024 Bit Mean Packet Length Distributed Exponentially
Eligibility For Restricted Token Dialog Varied
60% of Simulation Time in Restricted Token Mode

Figure 4.29 — Throughput Varying Restricted Token Dialog Usage