A Feasibility Study of Digitized Voice Distribution via the Xpress Transfer Protocol

Submitted to:

E-Systems, Inc.
7700 Arlington Boulevard
Falls Church, VA 22046-1572

Attention:

Mr. Stephen P. Johnson
Software Engineering Manager
Mailcode A402

Submitted by:

Alfred C. Weaver
Associate Professor and Director
Computer Networks Laboratory
Department of Computer Science
University of Virginia
Charlottesville, VA 22903

7 December 1991

A Feasibility Study of
Digitized Voice Distribution
via the Xpress Transfer Protocol

1. BACKGROUND

1.1. Telephone Systems

Voice transmission in telephone systems has traditionally been accomplished using analog techniques. However, in the 1980s, digital transmission began to replace analog transmission because:

(1) Analog signals attenuate with distance, and there is a limit to the number of amplifiers which can be used before the signal-to-noise ratio becomes unacceptably low. In contrast, digital regenerators restore the original signal exactly. Telephone links implemented with analog signaling on copper media typically achieve bit error rates of 10^{-5}, whereas digital transmission over fiber optic media can achieve bit error rates of 10^{-12} or better.

(2) Digital signaling allows voice, computer data, music, television, facsimile, video telephone, etc., to be multiplexed together over the same circuits.

(3) Digital equipment could reuse the existing media of the long distance carriers—copper, microwave, and fiber optic cable.

A digital telephone system uses a *codec* (coder-decoder) to sample the analog voice signal every 125 microseconds; each sample produces a 7-bit (in the United States) or 8-bit (in Europe) sample. The sampling frequency is dictated by Nyquist sampling theory which says that 8000 samples per second are sufficient to capture all the information contained in a 4 KHz (e.g., telephone) channel. The codec transforms the analog voice channel into a 56 Kbit/sec (using 7-bit samples) or 64 Kbit/sec (using 8-bit samples) digital data stream using Pulse Code Modulation (PCM) techniques.

In the Bell Telephone system, 24 voice channels are multiplexed with control and framing information onto a single T1 channel operating at 1.544 Mbits/sec. Higher speed channels such as T2, T3, and T4 (operating at 6.132, 44.736, and 274.176 Mbits/sec, respectively) represent aggregations of some number of lower speed channels. Similar strategies, but with different channel speeds, are utilized in Europe.

1.2. Computer Systems

In the context of computer systems, there is much to be gained from treating digital voice as simply a special case of digital data—special because it has timing requirements as well as accuracy requirements. If digital voice can be carried over computer networks *and still meet system timing requirements,* then it may be possible to collapse two separate systems (telephone system and computer system) into one. In fact, if the communications subsystem provides sufficient bandwidth, digital video could also be carried along with voice and data to form a truly integrated, multimedia, all-digital communications service. Five advantages of such a system are:

(1) lower bit error rate
(2) higher system reliability
(3) integration of voice, video, and data
(4) reuse of existing network components (e.g., bridges, routers)
(5) shared use of a single cable plant

The last three items may be of significant value to a security-conscious system, since they reduce security certification and validation complexity by dealing with one integrated system rather than three independent systems.

2. DIGITIZED VOICE DISTRIBUTION

While digitized voice is easy to acquire (there are many commercial vendors of analog-to-digital and digital-to-analog converters), digitized voice can not be transmitted on a sample-by-sample basis. To make distribution practical, multiple voice samples must first be "packetized"— that is, multiple contiguous samples must be accumulated and processed as a single data packet. This is necessary because there is considerable overhead associated with transmitting, receiving, and processing each data packet.

In a digital voice distribution system, some number (typically 1-8) of analog voice channels are processed by an analog-to-digital converter. Each voice channel produces a 64 Kbit/sec (8 Kbyte/sec) digital data stream. The A/D converter continuously samples the analog voice channel and outputs one digital sample (one byte) every 125 microseconds. These digital samples accumulate in a FIFO queue (one queue per voice channel). Digital voice processing is a special (and simpler) case of digital signal processing, which uses more accurate digital samples (typically 12 or 16 bits) and a higher sampling frequency to recover information in the incoming data stream.

A microprocessor periodically services the queue for each voice channel and removes n bytes of data. (Frequency of service and the number of data samples available are inversely related.) The n bytes of data become the *payload* of a message sent from the transmitting host to the destination host over an intervening computer network. At the destination host, a microprocessor processes the incoming message, extracts the payload, and delivers n bytes of data to a FIFO queue. A digital-to-analog converter extracts one byte of data every 125 microseconds and replays the resulting analog signal through a speaker or telephone handset.

A block diagram of a single-channel voice system is shown in Figure 1. On the transmitting side, the A/D circuitry monitors the analog input (e.g., microphone) and delivers one sample to the first-in-first-out (FIFO) queue every 125 microseconds. The user application program, running on the host, removes data samples from the FIFO and submits them to the communications subsystem for processing. The communications protocol, shown here as the Xpress Transfer Protocol, builds a Transport Protocol Data Unit (TPDU) which contains the application's data. XTP submits the packet to the FDDI LAN, which frames the packet and physically transmits it over the ring. On the receiving side, FDDI hardware extracts the packet from the ring, deframes it, and delivers it to XTP. XTP performs its reliability functions, transparently recovers from any data errors, and delivers the payload to the destination application. This user application writes the data into a FIFO queue. The D/A circuitry removes one sample from the FIFO every 125 microseconds and directs the corresponding analog signal to a playback device such as a speaker.

Note that Figure 1 is a simplification of a real system. Practical systems would operate with multiple channels simultaneously, channels would be bi-directional, and the single segment token ring shown could in reality be any number of network segments (either local area or wide area), and could be joined by either bridges or routers.

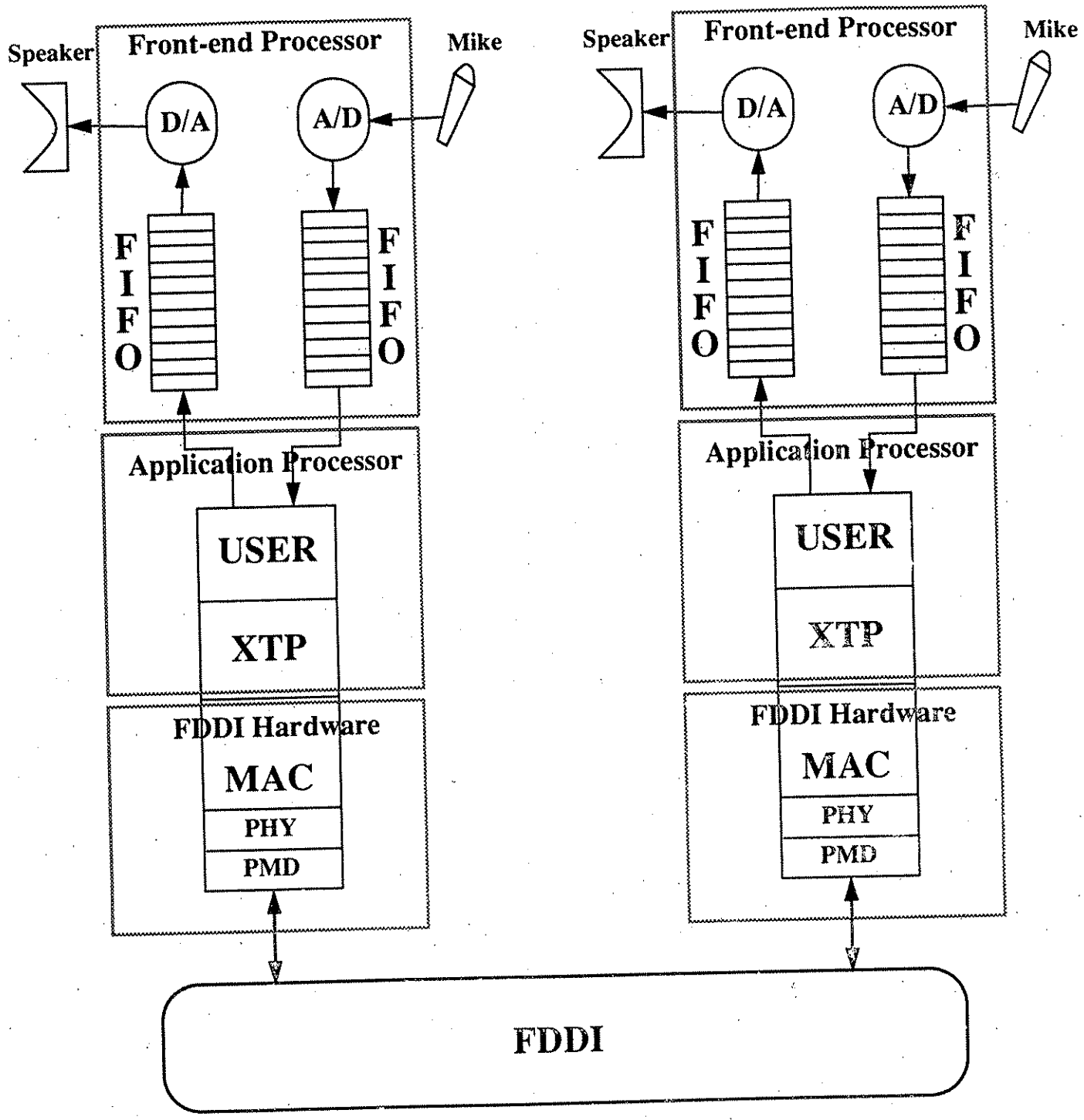


Figure 1.

Architectural Concept

The performance goal of such a system is simply stated for a single voice channel: empty the A/D converter's output FIFO sufficiently often that it never overflows (fills up with data), and deliver that content to the D/A converter's input FIFO sufficiently often that it never underflows (runs out of data). In addition, this process must be performed with a sufficiently small end-to-end latency such that it does not affect the quality of bi-directional communications.

Using modern computing hardware, software, and communications protocols, all the above can be easily achieved for a single voice channel—but it would make the world's most expensive telephone! Such a system is only practical if it can handle multiple voice channels simultaneously. It is highly desirable that such a system utilize common, commercial (as opposed to proprietary) components, that it operate over standard computer networks, and that it operate with standard computer communications protocols.

3. COMMUNICATIONS PROTOCOLS

The A/D converter can be considered a continuous source of data and the D/A converter a continuous sink for data. The purpose of the communications protocol is to provide the logical connection between the A/D's FIFO at the data's source and the D/A's FIFO at the data's destination. In addition, the protocol must support simultaneous use of the transport connection by multiple voice channels.

For each voice channel, a user process at the source collects n bytes of data, representing n contiguous voice samples, and delivers it (along with addressing information sufficient to identify the recipient) to a buffer which connects the user process to the communications subsystem. In a real system, each voice channel could produce n data bytes every n* 125 microseconds; in

our system, we assumed that data was available whenever it was needed. Thus we simulated a system with an unlimited number of potential voice channels to transport, and it was a goal of the experiments to determine how many voice channels could be transported under given circumstances.

On the transmitting side, the role of the transport protocol was to collect whatever data had accumulated in its buffers at the moment of protocol invocation, to package the data as a Transport Protocol Data Unit (TPDU), to encapsulate the TPDU as a legal FDDI frame, and finally to transmit the frame. On the receiving side a symmetric process occurred which accepted an FDDI frame, extracted a TPDU, delivered its payload to the buffers which connect the receiving transport protocol to the user process, and signal the destination user process that it has received data. The user process would then extract its data and deliver it. In a real system, the user process would deliver its data to a particular D/A converter's input FIFO queue; in our system we just delivered it to dedicated memory locations.

One of the advantages of using a transport protocol is that, at each invocation, it moves whatever amount of data has accumulated since the last invocation. This means that if, say, two contiguous data sample groups, each of size n bytes, accumulate in the user's transmit buffers for a single voice channel, then the protocol will packetize and deliver one packet with a payload of $2n$ bytes rather than two packets with a payload of n bytes each. This is an additional source of efficiency in the event that the system experiences transient loading of the processor or the network.

An outstanding question is whether the overall system goals are best met by a datalink protocol or a transport protocol. This question is discussed further in a later section. However, in summary, a datalink protocol is limited to a single segment network; transport protocols permit

system operation over multiple network segments connected by routers. In addition, transport protocols transparently recover from data errors of all types (e.g., bit errors on the medium, buffer overflow in the receiver, packet loss in the router). Another way of looking at the question is this: if a system can have all the advantages of a transport protocol and still meet its timing requirements, then it is better to use a transport protocol and thereby enable multi-segment networks, routers, transparent error recovery, etc. All our efforts in this study were directed at documenting the cost (in terms of latency) of the transport protocol; we did not study datalink protocols at all.

Given the decision to evaluate only a transport protocol approach, the requirements of the transport protocol were that it provide sequenced, in-order delivery without duplicates, with transparent error recovery, and that it do so with latency and jitter characteristics which did not degrade the quality of the voice channels being carried. This suggested that efficiency of implementation was a major concern, and for that reason we chose the Xpress Transfer Protocol (XTP) to supply this service. Our previous testing of XTP, reported in [Hart91], had shown XTP to be more efficient than either TCP or TP4 for this type of transport task. XTP provides all the services found in the classic transport protocols such as TCP and TP4, and in addition provides several useful new features such as a priority subsystem, intra-protocol scheduling, selective acknowledgement, selective retransmission, and transport layer reliable multicast. Each of these functions, unique to XTP, is a potentially valuable feature for a voice delivery service.

priority subsystem—XTP allows a user to mark messages (and hence packets) with a transport layer priority. At every moment in time XTP will then work on its most important (highest priority) packet. Thus, voice packets could be given priority over other message types in the transport protocol (as well as on the FDDI ring).

intra-protocol scheduling—the priority subsystem is active end-to-end, which means that high priority packets are treated expeditiously in the transmitter, in the receiver, and in all interior routers.

selective acknowledgement—rather than require an explicit acknowledgement for each voice packet (as in TCP and TP4), selective acknowledgement allows the transmitter to ask for acknowledgements when and if desired.

selective retransmission—rather than handling retransmissions with a *go-back-n* scheme as in TCP and TP4, XTP can retransmit only the gaps in a data stream.

multicast—multicast allows identical information to be reliably sent to any number of receivers with only a single transmission.

The scope of this study restricted us to experimentation with a single protocol, so all results reported herein reflect the use of XTP. However, the suitability of other transport protocols is discussed in a later section.

4. EXPERIMENT DESIGN

4.1. Hardware and Software

We used two "stations," each consisting of a Motorola 133XT processor board (25 MHz Motorola 68020 with 4MB memory), VMEbus backplane, pSOS real-time operating system, and Martin Marietta FDDI network. Our software included user applications and the Xpress Transfer Protocol (version 3.6, July 1991), both written in Microtec C. While replacing selected system components could have increased overall performance (e.g., replacing the 68020 with a 30 MHz 68030, or choosing a faster FDDI board), we did not attempt such "heroic" measures to

increase performance. The hardware suite is simple, commercial, standard, easily available, and already well-known to E-Systems; thus our point of view was merely to document its performance, rather than to optimize it to produce some predefined level of performance. Throughout the experiments reported here, there was no explicit attempt to "tune" the system for better performance.

Each station was both a transmitter and a receiver in the experiments discussed later, thereby enabling true bi-directional communication. Actual A/D and D/A converters were not used, nor were physical microphones or speakers. The lack of physical A/D and D/A devices should have minimal effect on the validity of our results, since we believe that read/write operations on the FIFO queues should be equivalent to the read/write operations we performed on user memory.

For some experiments, background traffic was generated in FDDI's synchronous class by a data load generator (described in detail in experiment 2, section 5.3.1). LAN throughput measurements were made with *FiberTap*, our real-time FDDI network monitor, described in Appendix H.

FDDI supports both *synchronous* and *asynchronous* data classes. Synchronous data is served preferentially at each station at each token arrival, and asynchronous data is served if and only if the token rotation time is less than a user-defined limit. Throughput all our experiments, we used the default value in the AMD SuperNet FDDI chip set for negotiated token rotation time (T_neg = 10.1 ms). Since voice traffic was deemed most important in these experiments, all voice traffic was assigned to the FDDI synchronous class.

Different FDDI implementations use different default values for T_neg; although AMD chose 10.1 ms, National Semiconductor chose 40 ms. In accordance with the definition of

FDDI, the network chooses the smallest of these values as its operational goal token rotation time. Thus, even though later experiments with multicast introduce a second manufacturer's FDDI chips, the AMD parameters prevail.

Even so, the selection of the goal token rotation time was not critical in these experiments. Normally, FDDI's Station Management (SMT) protocol is used to negotiate a portion of the network's synchronous bandwidth for use by each station with synchronous data; however, SMT was not functional on the Martin Marietta FDDI equipment, so this feature was not used.

4.2. Experiment Overview

The focus of the experimental work was on moving the data through the end-to-end communications system as fast as possible, and observing the effect of various system parameters on system throughput, latency, and jitter.

To that end, we developed a basic experiment in which the two stations described above continuously exchanged "voice packets." The basic system measurements were (a) network throughput, (b) end-to-end latency, and (c) jitter (i.e., variation in packet delivery time). The throughput was important because that measure, divided by 64 Kbits/sec, indicated how many voice channels could be simultaneously active. The latency measurements documented how long it took to deliver a packet, and we could compare that to the interpacket arrival rate required by the hardware for any given packet size. The jitter measurements demonstrated the variability in packet delivery times, especially when there was contention for system resources (processor and network).

The basic experiment was performed for voice sample sizes (i.e., values of n) ranging from 8 to 4096, and the throughput, latency, and jitter characteristics were measured. Throughput and

latency measurements were presented in tables; jitter measurements were presented as scatter plots of the latency times of 1000 individual packets. Latency was calculated by timing the round trip of a voice packet and dividing by 2 to yield the one-way latency. From the jitter data we calculated the average end-to-end latency of a data packet, and we calculated the 99.9% threshold point—that is, the latency value such that 99.9% of all measured latency values were equal to or smaller than that value.

The basic experiment was then extended to introduce background load on the FDDI network. Since background asynchronous load was expected (and verified) to have little effect on the performance of the synchronous data, all background loads were in the synchronous class so that it truly competed with the voice traffic for system resources. The experiments above (which had no background FDDI load) were then repeated with background loads of 25, 50, and 75 Mbits/sec. While doing these experiments we noted that system performance was sensitive to the *type* of background load as well as its intensity. Thus background load was generated two ways, one using a single packet per token service discipline (SPPT) in which at most one data frame is emitter per token reception, and the other using a multiple packet per token scheme (MPPT) in which multiple data frames are chained together and transmitted after token reception. The former is the most common FDDI service discipline, although the latter might be utilized by a system designer who was trying to optimize throughput of a highly loaded synchronous server. FDDI certainly permits MPPT service, but using it is awkward since it means withholding packets which are otherwise ready for transmission just so they can be chained with other; even if this service type was desirable, it is more likely to be used in an asynchronous service class (e.g., for file servers) than in the synchronous class. Although we think that the former (SPPT service) is the more likely, we ran experiments with both cases to gauge its impact.

Another set of experiments documented the performance if the system under conditions of packet loss. Although the actual system was normally error-free, we artificially introduced packet loss rates of 1%, 5%, and 10% to observe its effect on throughput, latency, and jitter. These experiments exercised the reliability and error repair mechanisms of the underlying transport protocol.

The experiments were further extended by adding background load to the processor. Whereas in the basic experiments the processor (used here as a communications engine) was used to carry only voice traffic, in these experiments it was also required to process auxiliary (non-voice) asynchronous communications. Using asynchronous traffic patterns defined by E-Systems, we documented the effect on voice traffic of having the processor handle non-voice traffic simultaneously.

The above experiments assumed a traditional unicast connection between transmitter and receiver; however, one of the unique features of XTP is that it can support 1-to-m, or *multicast*, connections. Using multicast, a single transmission can be reliably received by *m* receivers. We repeated selected experiments using multicast to determine the potential performance cost of this new service type.

For all of their variations, the above experiments still assumed a single segment LAN topology. While that is adequate today, we believe that future distributed applications will call for voice distribution over multi-segment networks. In another experimental section we discuss the measured performance of various network routers, and explore the expected impact of using routers in a voice distribution system.

Although our experiments were restricted to using a single transport and network protocol (XTP), we can generalize our results to other protocols. We first discuss the advantages and

disadvantages of using a transport protocol versus a link-layer protocol, and then discuss the relative merits of XTP versus two other well-known transport protocols, TCP and TP4.

In the final section of this report we use our experimental evidence to draw conclusions concerning the feasibility of voice distribution using a transport protocol and FDDI, and the suitability of using higher layer protocols in this environment. We discuss the architecture, feasibility, and expected performance of a prototype implementation.

5. EXPERIMENTAL RESULTS

5.1. Timing

The performance measurements reported here all depend upon maintaining an accurate time base throughout the duration of the experiment. Our Motorola 133XT processor boards were equipped with programmable timers with a resolution of 81.3 microseconds; that is, the timer provided a register which could be read via software, and the register was incremented by one every 81.3 microseconds.

Figures 2 and 3 illustrate the resolution of the timer. In this experiment the timer was first reset and then, in a loop which operated 1000 times, the value of the timer was read and recorded. After all 1000 samples had been recorded they were converted to units of milliseconds and displayed in Figure 2. The horizontal axis represents the 1000 successive timer samples; the vertical axis is the time value, in milliseconds, recorded for each sample. The result appears to be a straight line, but in fact an enlargement of the vertical scale for the first 80 samples shows that the timer values are actually discrete. The first sample returns a timer value of 0, the second and third samples return a value of 0.0813 ms, samples 4, 5, 6, and 7 each show a value of 2*0.0813=0.1626 ms, samples 8, 9, and 10 return a value of 3*0.0813=0.2439 ms, etc.

Successive samples continue to show this "quantum effect" whereby the only times reported are multiples of 0.0813 ms.

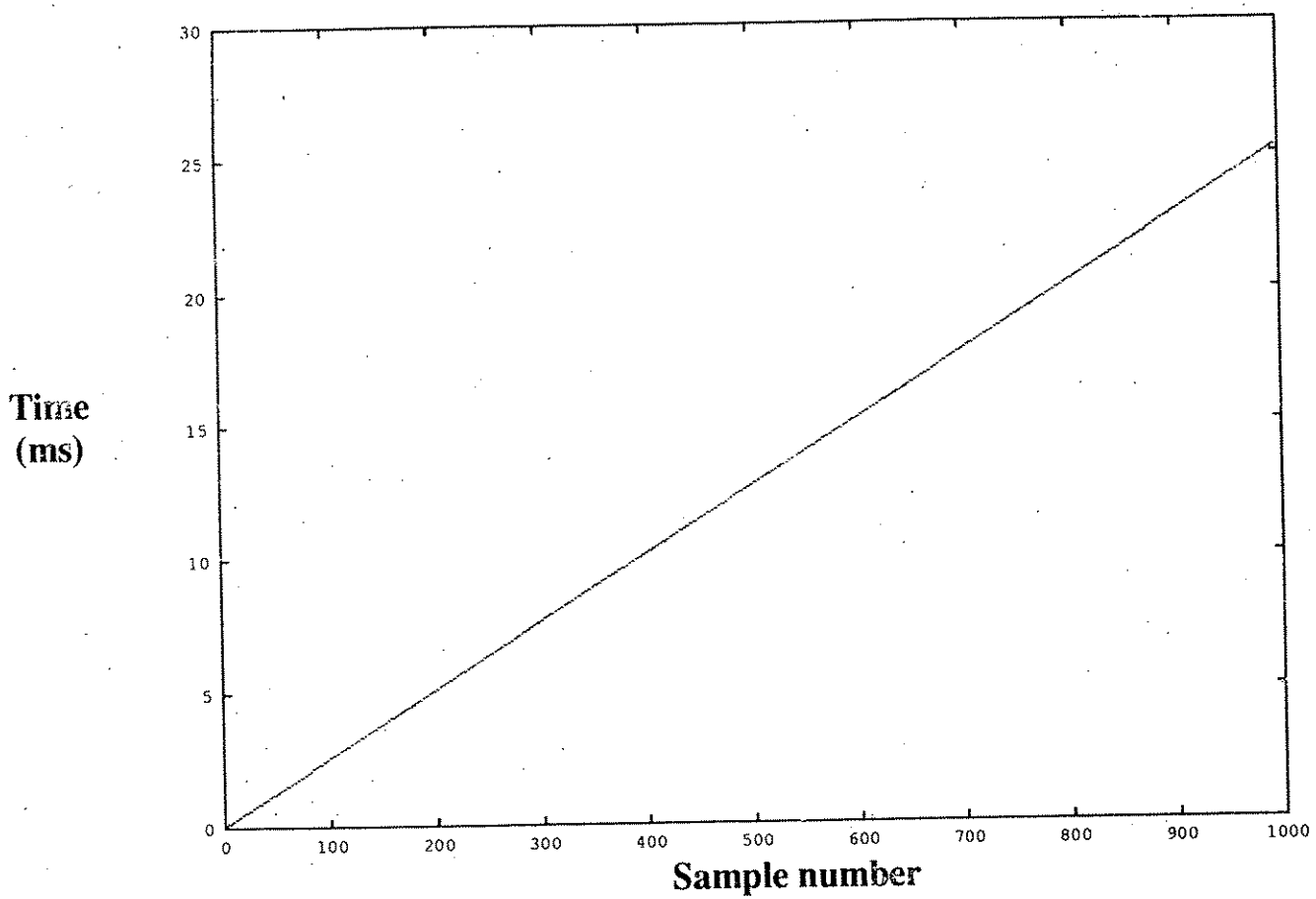


Figure 2.

1000 Timer Samples

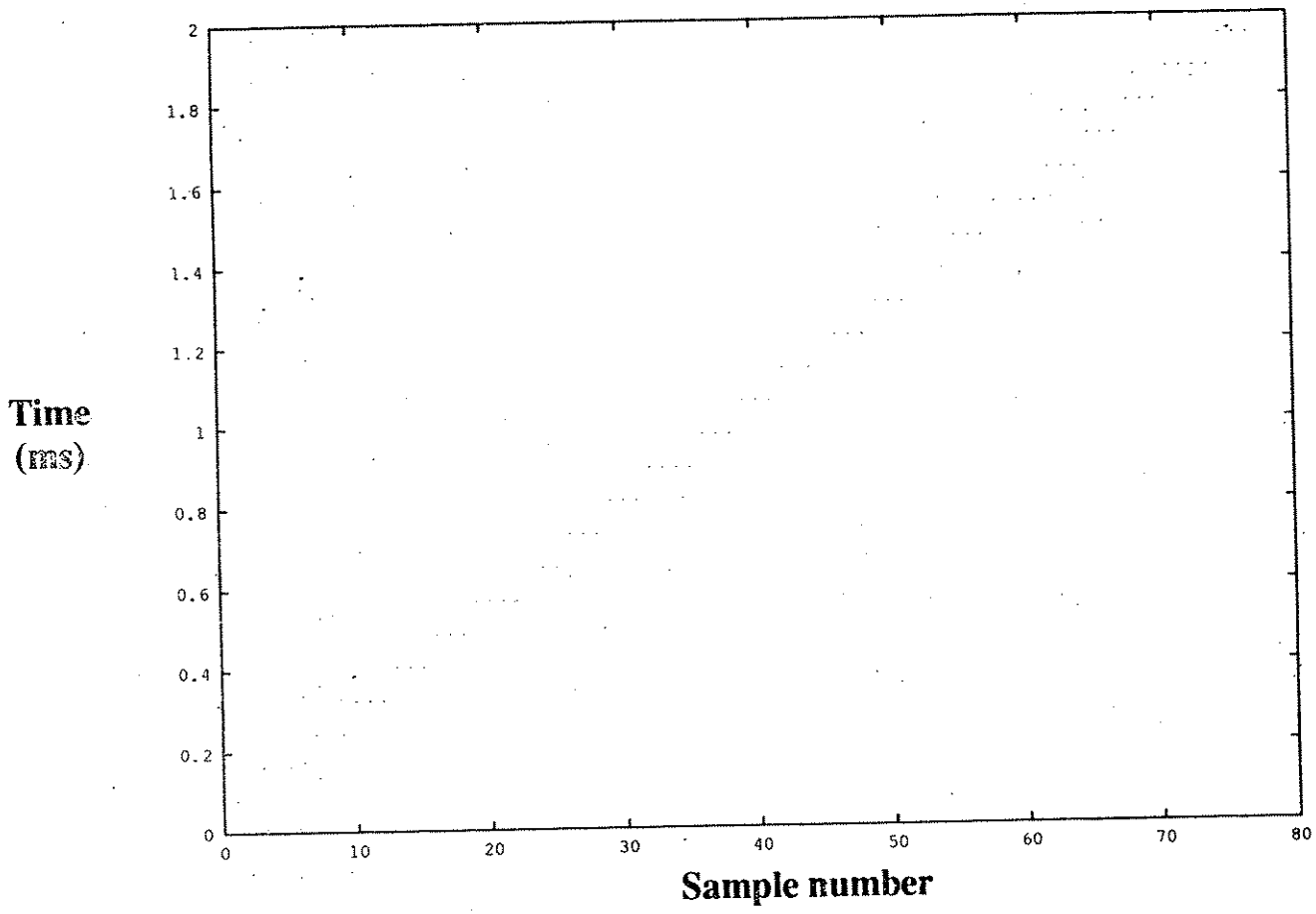


Figure 3.

80 Timer Samples

This effect is explained by the operation of the underlying hardware—since the timer register is incremented by a fixed amount at a fixed frequency, time appears to be discrete rather than continuous. Note in Figure 3 that the processor can sample the timer three or four times in any given 0.0813 ms period. For our purposes, this degree of timer resolution is sufficient. All our measured events (e.g., end-to-end delivery of the shortest packet) have a duration in excess of 2 ms, so there are at least 25 timer ticks in even our shortest event.

In the plots of experimental results which appear in the Appendices, please note that the apparent "quantum effect" regarding latency values is an artifact of the timing hardware, and does not actually occur in fact. However, since the quantum is so small (0.0813 ms), it does not significantly affect any of the measurements generated herein.

5.2. Experiment 1: The Basic Configuration

5.2.1. Experiment 1: Design

The basic experiment provided the transport protocol with a continuous stream of n byte packets. On the transmitting side, the processor's only job was to collect n bytes of data from a pseudo-FIFO (dedicated memory), deliver it to XTP, run the transport protocol, and transmit the data over FDDI. On the receiving side the processor performed the symmetric operations: receive an FDDI packet, run XTP, deliver data to the user, and have the user deliver the data to a pseudo-FIFO (again, dedicated memory). There was no background load on the FDDI network, and no other computational or communications load on the processor. The basic experiment represents a "best case" (for the given system architecture) for data throughput and latency, thus we use it as an upper bound for determining how many voice channels could be simultaneously supported by this particular system architecture.

All real-world considerations such as contention for the network (background traffic on the LAN), contention for the processor (background computational load or non-voice communications load on the processor), packet loss in the network (bit errors on the medium, buffer overruns in the receiver, packet loss in network routers), and less efficient transport protocols (TCP, TP4) will result in decreased system performance from that reported for the basic experiment. The impact of these additional influences are considered in the experiments which follow.

The basic experiment transmitted one megabyte (1,048,576 bytes) of pseudo-voice data from one station to the other, utilizing the full reliability features of XTP. Data to be transmitted was delivered to XTP in units of 8, 16, 32, 64, 128, 256, 512, 1024, 2048, or 4096 bytes; this number represents the value of n and is called "voice data size." When data of size n bytes is delivered to XTP, it is placed inside an XTP frame which consists of a 40-byte header and a 4-byte trailer. The header contains a *key* field which uniquely identifies the receiving process; the trailer contains a 4-byte transport checksum which provides an additional integrity check for the data. The XTP frame (44 bytes plus user data) is in turn encapsulated in a 25-byte FDDI frame (8 bytes for the LLC header, 6 bytes for the MAC destination address, 6 bytes for the MAC source address, 4 bytes for the cyclic redundancy code, 1 byte for the frame status field).

XTP allows the user to specify programmatically the degree of reliability to be imposed on the transmission. If error checking is elected (thereby making this transmission connection-oriented, as opposed to datagram or connectionless), the user may further specify how often XTP should solicit acknowledgements for the data stream. Acknowledgement frequency can vary from never to always. In these experiments we have chosen the most robust error-checking possible, namely that an acknowledgement packet is required from the receiver for every data packet transmitted by the sender. These acknowledgements are provided by an XTP "control

packet" of size 117 bytes. Thus, every transmitted packet of size n+69 bytes forces a 117 byte acknowledgement packet to be transmitted on the reverse channel. Since both stations are transmitting data and receiving data simultaneously, data and acknowledgements are flowing in both directions at all times. Even though each transmitted packet requires an acknowledgement, the transmitter does not enter into a stop-and-wait protocol; the transmitter proceeds as fast as it can (it is paced primarily by the amount of buffer space available in the receiver), using the acknowledgements only for error control (not flow control).

As an optimization, we could reduce the reverse channel traffic by decreasing the frequency of acknowledgements, or even eliminating them altogether by using a transport datagram (i.e., unacknowledged) service. However, we chose to document the cost of the most robust service type, knowing that optimizations such as these were available if we needed to build a higher performance system. For example, throughput would improve if we reduced the frequency of acknowledgements, thereby reducing both network load and acknowledgement processing.

5.2.2. Experiment 1: Analysis

Table 1 shows a number of results for the basic experiment. As identified in the first column, each row represents the measured and calculated results for an experiment in which the user provides data to XTP in units of n bytes (labeled "voice data size"), where n varies from 8 to 4096 by powers of two. The second column of each row, labeled "frame size," shows the actual amount of data which has to be sent to carry n bytes of user data; this number reflects the 69 bytes of overhead added to each packet by XTP (44 bytes) and by FDDI (25 bytes). The third entry shows how many packets, each with a payload of n bytes, were sent in order to transmit one megabyte.

Column four, labeled "user throughput," documents the end-to-end, single-channel throughput in units of megabits/sec, and includes only the voice data in the computation of throughput (i.e., XTP and FDDI framing are not counted as user throughput). The "network throughput" figure in column five (megabits/sec) does account for the required XTP and FDDI framing; looking at row one, sending voice data in 8-byte groups results in 30 Kbits/sec of user throughput but, because of framing overhead, requires 288 Kbits/sec of network throughput to accomplish it.

"Total time" in column six records the duration of the experiment in seconds. The "packet rate" in column seven indicates how many XTP packets were transmitted per second. Column eight ("average latency") records the average one-way, end-to-end latency (in milliseconds) for a packet with payload *n* bytes. Finally, column nine ("99.9% threshold") identifies that latency, in milliseconds, for which 99.9% of all packets have a latency which is equal to or less than this latency.

BASIC EXPERIMENT

voice data size (bytes)	frame size (bytes)	packets sent	user throughput (Mbits/sec)	network throughput (Mbits/sec)	total time (sec)	voice packet rate (packets/sec)	average latency (ms)	99.9% threshold (ms)
8	77	131,072	0.030	0.288	278.890	470	2.728	2.764
16	85	65,536	0.060	0.319	139.440	470	2.732	2.846
32	102	32,768	0.120	0.383	69.890	469	2.751	2.846
64	133	16,384	0.237	0.493	35.330	464	2.791	2.927
128	197	8,192	0.463	0.713	18.100	453	2.890	3.008
256	325	4,096	0.890	1.130	9.360	438	3.062	3.171
512	581	2,048	1.651	1.873	5.080	403	3.417	3.496
1,024	1,093	1,024	3.039	3.244	2.760	371	4.015	4.146
2,048	2,117	512	5.053	5.223	1.667	307	5.242	5.366
4,096	4,165	256	7.557	7.684	1.110	231	7.699	7.805

69 bytes framing overhead (44 bytes from XTP, 25 from FDDI)
no background processor load
no background FDDI load

Table 1
Basic Experiment: Throughput and Latency

Overhead. Framing overhead ranges from 90% when using small voice data sizes (8 bytes of payload in a 77 byte frame) to 1.5% when using large voice data sizes (4096 bytes of payload in a 4165 byte frame). As expected, large frames are much more efficient than small frames, and, all other variables being equal, this fact alone would argue for using the largest possible voice data size. (Other factors make this less attractive.) User throughput and network throughput measurements further verify the efficiency of large packets.

Packet rate. This is a fundamental measure of the overhead found in the entire architecture. Using a voice data size of 8 bytes, the fastest possible transmission rate is 470 packets per second. In other words, this architecture is incapable of producing any kind of packet faster than once every 2.1 milliseconds. This is a fundamental system limit.

Theoretically, packet generation rate is orthogonal to packet latency; that is, if an application can generate x packets/sec, the packet arrival rate at the destination will reflect the generation rate, regardless of the packets' latencies. The side-effect, of course, is that as latency increases, it becomes more and more difficult to maintain a true bi-directional conversation. Thus, we have chosen a more rigorous standard of performance: not only must the transmitting FIFO be serviced at a minimum rate (dependent upon voice data size), but the resulting packets must be delivered with a latency which avoids creating a pipeline of packets in transit through the communications protocol (the pipeline effect is unavoidable in the two FIFOs due to the nature of the hardware). From this point on, we focus on latency as being the primary indicator of system performance.

Latency. Average latency varies from 2.7 ms for a voice data size of 8 bytes to 7.7 ms for a voice data size of 4096 bytes. The 99.9% threshold numbers are very close to the average latencies, varying by at most 0.1 ms; this indicates very low variance in the delivery time of packets.

However, the real meaning of the end-to-end latencies can only be understood by contrasting them with what a practical system would require, and then looking at what margin of safety these measures provide.

In a real system, fixing the size of *n* places an upper bound on the acceptable delivery time of a packet of that size. Any group of *n* voice data samples represents *n** 125 microseconds of voice. If *n* voice samples are collected and transmitted repeatedly, then each such group must be delivered within *n** 125 microseconds, or else the destination FIFO will underflow due to lack of data. To appreciate the impact of this delivery deadline, examine Table 2.

BASIC EXPERIMENT

voice data size (bytes)	new data needed every (ms)	new data generated every (ms)	packet generation rate (packets/sec)
8	1.000	2.127	470
16	2.000	2.127	470
32	4.000	2.132	469
64	8.000	2.155	464
128	16.000	2.208	453
256	32.000	2.283	438
512	64.000	2.481	403
1,024	128.000	2.695	371
2,048	256.000	3.257	307
4,096	512.000	4.329	231

69 bytes framing overhead
no background processor load
no background FDDI load

Table 2
Basic Experiment: Required vs. Observed Arrival Periods

A packet with a payload of n bytes must be delivered within n* 125 microseconds of the packet before it. Table 2 shows, for each value of n, the maximum time which may elapse before the output FIFO underflows (labeled "new data needed every") and the observed amount of time which was required to generate a packet with n bytes of payload (labeled "new data generated every"). The "new data generated" column is computed from the packet rate in column four.

For example, if packets contain 8 bytes of payload, then they must arrive once per millisecond to avoid a FIFO underflow; however, they can only be generated every 2.1 ms, so this value of n is not practical. The table reveals that 16-byte payloads are likewise impractical for the same reason. At the other end of the scale, if there are 4096 voice samples per packet, then they are needed once every 512 ms, yet they can be generated every 4.3 ms. So large values of n are very practical from the standpoint of the delivery schedule (other factors make them less attractive). The results in Table 2 show that as the difference between the values in columns 2 and 3 increases, we have a greater margin of safety in assuring timely delivery of packets.

Note that the results in Table 2 are for a single voice channel, and that a practical system must simultaneously manage multiple voice channels. Thus the "margin of safety" discussed above represents time which could be used for handling additional voice channels.

Table 3 shows how many voice channels could be carried in this particular experiment.

BASIC EXPERIMENT

voice data size (bytes)	user throughput (Kbits/sec)	equivalent voice channels (64 Kbits/sec each)
8	30	0
16	60	0
32	120	1
64	238	3
128	468	7
256	897	14
512	1661	25
1024	3050	47
2048	5115	79
4096	7294	113

69 bytes framing overhead
no background processor load
no background FDDI load

Table 3
Basic Experiment: Voice Channels Available

As shown in Table 3, dividing the user throughput measurement (in Kbits/sec) by 64 Kbits/sec yields the number of voice channels which could be carried using this architecture. For a system supporting, say, 60 voice channels, a voice data size of nearly 2 kilobytes is required.

This experiment suggests that channel efficiency and capacity increase as payload size increases, and indeed this is true. But an opposing factor is the startup time of the pipeline which is being constructed between the transmitter and receiver. As shown in Table 2, a voice data sample of n bytes takes n* 125 microseconds to collect before it even begins its journey through the network. Thus there is a startup delay upon connection establishment of n* 125 microseconds plus the end-to-end latency, and the same delay whenever speech begins anew after a period of silence. As n becomes large, so does the startup latency; at n=4096, for example, the startup delay is in excess of half a second.

Jitter. Our jitter measurements are intended to illustrate the variance inherent in any communications subsystem. Appendix A contains a set of ten jitter measurements, one each for the ten values of n which are investigated throughout this study. The purpose of these measurements is to calculate the average one-way end-to-end latency, its 99.9% threshold value, and to then plot the individual latencies of 1000 successive packets. On the jitter plot we can visually observe the distribution of end-to-end latency values.

For the first jitter plot in Appendix A (voice data size equals 8 bytes), all 1000 measurements of one-way end-to-end latency report one of two numbers—2.6 or 2.8 ms. Recall that this "quantum effect" is an artifact of the timing hardware and that, in reality, latencies are distributed between these two values. Nevertheless, since the quantum is so small (81.3 microseconds), this has no significant effect on our measurements or our conclusions.

Table 4 summarizes the average latency and 99.9% threshold values for the basic experiment.

BASIC EXPERIMENT

voice data size (bytes)	average latency (ms)	99.9% threshold (ms)
8	2.728	2.764
16	2.732	2.846
32	2.751	2.846
64	2.791	2.927
128	2.890	3.008
256	3.062	3.171
512	3.417	3.496
1,024	4.015	4.146
2,048	5.242	5.366
4,096	7.699	7.805

69 bytes framing overhead
no background processor load
no background FDDI load

Table 4
Basic Experiment: Average and Threshold Latency

From Table 4 we can make two observations:

(1) Latency is strongly dependent upon the voice data size, but even in the case of n=4096 bytes we get reliable delivery in 7.8 ms. This gives us a considerable margin of safety with regard to the delivery time requirement for all the medium-to-large voice data sizes.

(2) In all ten jitter plots, variance was low. The worst case difference in delivery times between any two packets of the same voice data size was 0.16 ms, which is negligible.

5.2.3. Experiment 1: Conclusions

1. System efficiency increases with increasing voice data size.

2. The side effect which argues against making the voice data size arbitrarily large is that the connection startup time is proportional to voice data size, and in fact is equal to n* 125 microseconds plus the end-to-end latency.

3. Payloads of 32 bytes or larger are practical for a single voice data channel.

4. The number of simultaneous voice channels which can be supported likewise increases with voice data size. Voice data sizes of 1K, 2K, and 4K bytes would support 47, 79, and 113 voice channels (64 Kbits/sec each) respectively.

5. Jitter plots confirm that there is very little variance in the end-to-end delivery latency.

6. All the throughput, latency, and jitter data reported for the basic experiment represent a "best case" scenario for the chosen architecture. Additional experiments are required to determine the impact of background FDDI load, background processor load, etc.

5.3. Experiment 2: Synchronous SPPT Background FDDI Load

5.3.1. Experiment 2: Design

Our second experiment was designed to illustrate the influence of background load on the FDDI network. Since background traffic in the asynchronous class can reasonably be expected to have little effect on the voice traffic in the asynchronous class, we restricted our experiments to background *synchronous* traffic.

The basic experiment was repeated three times, except that a background load of differing intensity was imposed on the FDDI network for each trial. This load was created by attaching a separate station to the FDDI ring, and having it generate 25, 50, or 75 Mbits/sec of data which was addressed to a non-existent station. The sole purpose of this "traffic generator" was to make the FDDI ring look busy.

The traffic generator was a high-performance personal computer (25 MHz Intel 80386 processor) equipped with an AMD Fastcard FDDI interface. It created FDDI frames of length 4167 bytes (33,336 bits); at FDDI's 100 Mbits/sec transmission rate, each frame required 33 microseconds to transmit. An important decision was whether each frame should be processed individually (i.e., transmit at most one data frame per token arrival), which we call "single-packet-per-token" or "SPPT" service, or whether multiple packets should be chained together to make a larger burst of transmitted data (i.e., transmit more than one packet per token arrival), which we call "multiple-packet-per-token" or "MPPT" service. Obviously, MPPT service is more efficient for the network since it reduces overhead, but is more injurious to the variance of voice data latency since it permits non-voice stations to capture the token for longer periods of time.

In a previous section we observed that SPPT service is the more natural selection, since MPPT service requires withholding packets that are otherwise ready for transmission until multiple packets can be chained together for bulk transmission. Not only is this awkward, but such a low level decision is normally made by the network interface software (i.e., by the low level drivers connecting the communications protocol to the network hardware); the user would normally never be aware that such a decision existed, and thus would normally have no control over it. However, since the outcome of this decision obviously affects performance, we have modeled it both ways. In experiment two, the background load generator produced SPPT traffic; in experiment three it produced MPPT traffic.

For experiment two the load generator produced traffic according to Table 5.

Packet size (bytes)	Packet size (bits)	Packet generation rate (packets/sec)	Total offered load (Mbits/sec)
4167	33336	750	25
4167	33336	1500	50
4167	33336	2250	75

Table 5
SPPT Background Load Generation Parameters

5.3.2. Experiment 2: Analysis

Experiment two, "Synchronous SPPT Background Load," augments the basic experiment by adding a background load of 25, 50, or 75 Mbits/sec. All of the background load is carried in FDDI's synchronous class (and thus competes for channel bandwidth with the voice traffic), and all of it is generated using a SPPT service discipline.

Appendix B contains thirty scatter plots, ten each for the three background load intensities. The effect of the background load is shown in Table 6 which compares the 99.9% threshold points for the basic experiment (with zero background load) against these three background loads.

Table 6A shows the number of voice channels which could be carried for a given voice data size and given background synchronous SPPT load on FDDI.

SYNCHRONOUS SPPT BACKGROUND LOAD

voice data size (bytes)	99.9% threshold latencies (ms)			
	0 Mbits/sec	25 Mbits/sec	50 Mbits/sec	75 Mbits/sec
8	2.764	2.927	2.927	3.171
16	2.846	2.927	2.927	3.171
32	2.846	2.927	3.089	3.171
64	2.927	3.008	3.008	3.252
128	3.008	3.089	3.089	3.496
256	3.171	3.333	3.659	3.659
512	3.496	3.740	3.821	3.821
1,024	4.146	4.228	4.390	4.634
2,048	5.366	5.528	5.610	5.772
4,096	7.805	7.967	8.049	8.221

Table 6
99.9% Threshold Latency with Synchronous SPPT Background FDDI Load

BACKGROUND SPPT LOAD ON FDDI

voice data size (bytes)	0 Mbits/sec (Kbits/sec)	channels	25 Mbits/sec (Kbits/sec)	channels	50 Mbits/sec (Kbits/sec)	channels	75 Mbits/sec (Kbits/sec)	channels
8	26	0	27	0	27	0	27	0
16	51	0	54	0	54	0	55	0
32	104	1	108	1	108	1	110	1
64	217	3	216	3	214	3	216	3
128	424	6	422	6	420	6	421	6
256	827	12	821	12	818	12	816	12
512	1546	24	1540	24	1530	23	1530	23
1024	2792	43	2750	42	2764	43	2735	42
2048	4732	73	4732	73	4755	74	4691	73
4096	7162	111	7080	110	7095	110	7431	116

Table 6A
Number of Voice Channels with Background FDDI Load

In Table 6 we display 99.9% threshold values, rather than average values, so that we get an even better picture of the effect of background synchronous SPPT load on end-to-end latency. As expected, the addition of background load does increase the overall delay, but not by a significant amount.

Adding 25 Mbits/sec of background load increases the 99.9% threshold point by about 0.1 ms; 50 Mbits/sec of background load increases it by about 0.3 ms; 75 Mbits/sec of background load increases it by about 0.5 ms. None of these increases are significant. Thus we conclude that synchronous SPPT background load on FDDI, even at a rate of 75 Mbits/sec, has no serious effect on overall voice data latency.

We must point out that a major reason that the effect is so minor is that, even though the load generator can produce lots of traffic (2250 packets/sec for the highest loading case), each packet is served using SPPT service. So in an FDDI ring of three nodes (two voice stations and the traffic generator), the voice traffic of any one station is interrupted by the transmissions of at most two other stations. As will be seen from the next experiment using MPPT service, SPPT service is highly desirable because it smooths the flow of synchronous traffic among stations, allowing each voice station a timely opportunity to transmit.

5.3.3. Experiment 2: Conclusions

1. Heavy background synchronous FDDI traffic had little effect on latency in this configuration. The worst case increase in the 99.9% threshold was less than 0.5 ms, which is insignificant.

2. The reason that background load had so little effect was because (1) the ring was small (three stations), and (2) service was SPPT. Thus each station was always given a timely opportunity to transmit.

3. SPPT service is highly desirable. Since this is the normal service discipline of FDDI hardware, and since users are not normally aware that there is a choice between SPPT and MPPT service, the default situation of SPPT service is the correct choice for voice traffic in the synchronous class.

5.4. Experiment 3: Synchronous MPPT Background FDDI Load

5.4.1. Experiment 3: Design

Experiment three differed from experiment two in that the synchronous background load on FDDI was generated using a multiple-packet-per-token (MPPT) service discipline rather than SPPT. MPPT correctly models the situation in which a particular station is capable of high, sustained loads in the synchronous class, and furthermore that network efficiency is of such paramount importance that the network interface routines have been written to hold packets for transmission until a prescribed number of them have been accumulated and chained together; when the prescribed number has been accumulated, then that group of packets is transmitted upon the next token arrival.

MPPT traffic is generated by reprogramming the load generator. Packets are generated internally using the same scheme described previously; however, 15 packets are accumulated before physical transmission is attempted. Packets are 4167 bytes (33,336 bits) each, so a group of 15 of them represents a transmission of 500,040 bits behind a single token. At the 100 Mbits/sec rate of FDDI, each MMPT transmission requires 5 ms.

The MPPT experiment generates "bursty" traffic at the rates shown in Table 7.

Packet size (bytes)	MPPT parameter (packets/burst)	Burst rate (bursts/second)	Burst period (ms between bursts)	Total offered load (Mbits/sec)
4167	15	50	20	25
4167	15	100	10	50
4167	15	150	6.66	75

Table 7
MPPT Background Load Generation

Given that a single burst will consume 5 ms of network transmission time, and that voice packets are being generated continuously, some voice packets will be delayed the full 5 ms while other, generated after the burst begins, will suffer a smaller delay.

Appendix C contains thirty scatter plots, ten each for the background loads of 25, 50, and 75 Mbits/sec, all in FDDI's synchronous class, and all using the "bursty" MPPT service discipline. Table 8 summarizes the results for the average latencies.

SYNCHRONOUS MPPT BACKGROUND LOAD

voice data size (bytes)	average latency (ms)			
	0 Mbits/sec	25 Mbits/sec	50 Mbits/sec	75 Mbits/sec
8	2.728	3.526	4.958	6.501
16	2.732	3.522	4.959	6.501
32	2.751	3.525	4.958	6.504
64	2.791	3.522	4.956	6.501
128	2.890	3.816	4.957	6.503
256	3.062	3.884	4.957	6.504
512	3.417	4.319	4.958	6.505
1,024	4.015	4.861	5.397	6.502
2,048	5.242	6.498	9.932	6.503
4,096	7.699	9.765	9.940	9.919

Table 8
Average Latency with Synchronous MPPT Background FDDI Load

The average latencies resulting from MPPT service show the expected result—average latency generally increases with increasing background load. From looking at the jitter plots we observe that variance has increased substantially when compared to SPPT service. For example, compare the plots of SPPT service vs. MPPT service for 25 Mbits/sec background load (the eighth plot of Appendix B vs. the eighth plot of Appendix C). For SPPT service, latency is tightly grouped between 4.0 and 4.2 ms; for MPPT service, latency falls into two bands, one around 4 ms and another around 5.7 ms. Latencies in the higher band resulted from packets which were generated while the token was being held by the load generator, and thus their delivery was partly delayed by the burst created by MPPT service in the load generator. Latencies in the lower band resulted from packets which were generated and then transmitted between bursts from the load generator. Note that the lower band around 4 ms corresponds closely to the results from the basic experiment in which there was no background load (refer to the eighth plot of Appendix A).

Table 9 shows the 99.9% threshold values from these same experiments. These results are not as consistent as the averages since they are reporting the second-highest latency observed in a group of 1000 packets. An important point is that, in all of Table 9, the worst case 99.9% threshold value was 10 ms.

SYNCHRONOUS MPPT BACKGROUND LOAD

voice data size (bytes)	99.9% threshold latencies (ms)		
	25 Mbits/sec	50 Mbits/sec	75 Mbits/sec
8	5.122	4.959	6.585
16	5.122	4.959	6.585
32	5.122	5.041	6.585
64	5.041	4.959	6.585
128	5.447	4.959	6.992
256	5.203	5.041	6.585
512	5.854	5.041	6.585
1,024	5.772	7.967	6.585
2,048	7.236	10.000	6.667
4,096	9.837	10.000	9.919

Table 9
99.9% Threshold Latency with Synchronous MPPT Background FDDI Load

Once again, we must point out that the latencies recorded from the MPPT experiment are directly related to the architecture of having two stations and one load generator. Each voice station is given an opportunity to transmit with a delay which is at most one 15-packet burst from the load generator and one voice packet from the other voice station. Had the background load been generated by multiple load generators, each of them could have held the token for some amount of time (here 5 ms) and thus could have dramatically increased end-to-end delay. Note that the potential increase in token rotation time is bounded only by the operation of FDDI Station Management which limits the amount of synchronous data which any one station can emit on any one token cycle. Although SMT was not operational on our hardware, it should be operational in a production system to avoid starvation of the voice servers.

However, the whole issue is avoided if SPPT service is used in preference to MPPT service. SPPT is the natural service type for voice stations which emit voice packets periodically rather than continuously. Requiring SPPT rather than MPPT service on the whole network would at most impact the efficiency of a station sending non-voice traffic in the synchronous class. Thus we recommend the user of SPPT service everywhere.

5.4.2. Experiment 3: Conclusions

1. MPPT service caused end-to-end latencies to depart from a smooth distribution and instead to fall into groups.

2. Latencies in the lower group resulted from packets generated and transmitted between bursts from the load generator; their latencies are not significantly different from those observed in the basic experiment with no background load.

3. Latencies in the higher group resulted from packets generated while a burst was in progress; their latency was affected by the duration of the burst.

4. The increase in latency due to MPPT service was bounded here by the fact that there was only one load generator; had their been more than one then the effect would have been more dramatic.

5. MPPT service marginally increases the efficiency of a station sending non-voice traffic in its synchronous class while increasing the variance of all the voice traffic. Thus MPPT service is not recommended.

5.5. Experiment 4: Retransmission

5.5.1. Experiment 4: Design

Modern fiber optic local area networks rarely lose data, but when they do it is the responsibility of the transport protocol to retransmit it without intervention by the user. As a result, the user had the illusion that the end-to-end bitpipe was error-free.

All transport protocols add sequence numbers to their transmitted packets so that a receiver can detect out-of-sequence data. TCP and XTP use byte-oriented sequence numbers, whereas TP4 uses packet-oriented sequence numbers. Upon detecting out-of-sequence data, TCP and TP4 both revert to a go-back-n strategy in which the first missing data element (a byte in TCP and a packet in TP4) are retransmitted, *along with all following information*. Thus TCP and TP4 may resend data already correctly buffered by the receiver.

XTP uses a selective retransmission scheme in which the receiver notifies the sender of exactly which *spans* of data were received correctly. From that list of acknowledged data XTP

creates a list of *gaps* (contiguous bytes) which should be retransmitted. XTP then retransmits only the missing gaps.

While this feature is active in XTP, it is difficult to see error repair in action because errors are so rare on good FDDI equipment. Thus, to observe any errors at all, we had to modify XTP such that it failed to acknowledge *x*% of the packets received, thereby making them appear to be lost in transit. When a receiver received the next (out-of-order) voice packet, that caused the receiver to notify the sender that data was missing. XTP then retransmitted the lost data.

In this experiment we introduced data loss rates of 1%, 5%, and 10% to investigate its effect on data latency.

5.5.2. Experiment 4: Analysis

Table 10 records average latencies when the system is subject to random losses of 1%, 5%, and 10% of total voice packets. For comparison, we include in the second column the results from the basic experiment in which the loss rate was zero.

RETRANSMISSIONS

voice data size (bytes)	average latency (ms)			
	0% loss	1% loss	5% loss	10% loss
8	2.728	2.838	2.837	2.838
16	2.732	2.838	2.837	2.838
32	2.751	2.940	2.838	2.838
64	2.791	2.980	2.979	2.985
128	2.890	2.989	2.988	2.988
256	3.062	3.122	3.117	3.120
512	3.417	3.492	3.487	3.491
1,024	4.015	4.068	4.070	4.071
2,048	5.242	5.316	5.317	5.313
4,096	7.699	7.779	7.776	7.776

Table 10
Average Latency with 1%, 5%, and 10% Packet Loss

Table 10 makes a strong case for the power and utility of XTP as a transport protocol. Error repair is extremely effective and efficient. Comparing the columns for 0% and 10% loss, we see that the worst case increase in average delivery time was less than 0.1 ms, even for the largest voice data size. An operational network with a loss rate of 10% would be extremely unusual, and yet even in that abnormal situation the expected increase in latency was insignificant.

Appendix D contains thirty jitter plots, ten each for 1%, 5%, and 10% loss rates. Comparing these with the jitter plots of the basic experiment in Appendix A, we find only small differences—a slight increase in the value of the average delay, and a slight increase in the variability of that delay. Still, none of these increases was significant.

5.5.3. Experiment 4: Conclusions

1. Modern fiber optic networks such as FDDI have a very low packet loss rate.

2. Even so, when errors do occur, repair was swift and efficient. When comparing a loss rate of 0% to a loss rate of 10% in this architecture, the worst case increase in average end-to-end delay was less than 0.1 ms.

3. XTP, using selective retransmission, provided a very effective mechanism for end-to-end reliability.

4. Retransmission had no significant effect on jitter.

5.6. Experiment 5: Background Asynchronous Processor Load

5.6.1. Experiment 5: Design

In all the experiments up to this point, the CPU has had the single task of processing voice data. In experiment five we investigated the effect of background asynchronous processor load.

If the CPU is making calculations or processing non-voice messages, that activity will reduce the amount of processing power available to run the transport protocol, which will in turn reduce the throughput and increase the latency of the voice traffic. The degree of interference with the voice traffic is controlled entirely by the nature of this background load on the processor.

E-Systems defined two explicit loading cases. The first of these, the "average" case, required a voice station to also process 10 messages/sec and transmit them in FDDI's asynchronous class. The second of these, a "worst case" scenario for a particular application, required the voice station to process 120 messages/sec and transmit them in FDDI's asynchronous class. In both cases the distribution of message lengths was then same: half of the messages were 40 bytes in length and the other half were 512 bytes in length (lengths refer to payload size).

Referring to Table 1, a single station is able to process between 470 (for voice data size equals 8) and 231 (for voice data size equals 4096) messages/sec. We concluded that the addition of 10 messages/sec to the processor's workload would have a negligible effect, and subsequent testing showed this to be true, so experiment five was concerned only with the "worst case" load of 120 asynchronous messages/sec. We repeated the basic experiment but added the background message processing load; then we repeated it again with 25 and finally 50 Mbits/sec of background synchronous FDDI load (using SPPT service).

5.6.2. Experiment 5: Analysis

Appendix E contains thirty jitter plots for the case of 120 messages/sec of non-voice, asynchronous load competing with the voice traffic, ten each for the cases of 0, 25, and 50 Mbits/sec of background synchronous load on the FDDI network. The load generator was configured to use SPPT service. Table 11 summarizes the results.

ASYNCHRONOUS PROCESSOR LOAD
120 messages/sec
50% of length 40 bytes, 50% of length 512 bytes

voice data size (bytes)	average latency (ms)		
	0 Mbits/sec background load	25 Mbits/sec background load	50 Mbits/sec background load
8	3.689	3.740	3.775
16	3.688	3.740	3.776
32	3.693	3.743	3.778
64	3.709	3.743	3.776
128	3.745	3.781	3.859
256	3.850	3.890	3.939
512	4.024	4.045	4.108
1,024	4.324	4.350	4.443
2,048	5.241	5.272	5.278
4,096	7.704	7.746	7.907

Table 11
Average Latency for Asynchronous Processor Load
with Synchronous Background FDDI Load

As expected, the asynchronous processor load had little effect on the latency of the synchronous voice data. Compare the data in Table 11 with that of the basic experiment in Table 1. The basic experiment (Table 1) had no asynchronous processor load and no background FDDI traffic. When compared to the asynchronous processor load with no background FDDI traffic (Table 11, column "0 Mbits/sec background load"), the worst case increase of about 1 ms was incurred only by the small voice data sizes; there was almost no increase for the larger sizes.

When the basic experiment is compared to the combination case of having 120 messages/sec of asynchronous processor load and 50 Mbits/sec of background synchronous FDDI load, there is still some increase but it remains negligible. For small voice data sizes the increase is less that 1.1 ms, and for large voice data sizes it increased only 0.2 ms.

Thus we observe that the asynchronous message processing (even the so-called "worst case" of 120 messages/sec) did not significantly interfere with the voice traffic. Even the addition of background synchronous FDDI traffic made little difference.

The reason, of course, is that the total message traffic imposed by the asynchronous processor loading is small. The split message length distribution results in an average message length of 276 bytes, and 120 messages/sec of that size generate just over 0.25 Mbits/sec. This asynchronous load is negligible when compared to the 7+ Mbits/sec of voice traffic (for voice data size equal 4096) or the 50 Mbits/sec of background FDDI traffic.

5.6.3. Experiment 5: Conclusions

1. The addition of a "worst case" asynchronous processing load (defined to be 120 messages/sec, half of length 40 bytes and half of length 512 bytes) did not significantly increase the latency of the voice traffic.

2. The voice traffic in FDDI's synchronous class was effectively insulated from additional communications load in the asynchronous class.

5.7. Experiment 6: Multicast

5.7.1. Experiment 6: Design

A unique feature of XTP is its capability for supporting a 1-to-many connection called *multicast*. Other transport protocols such as TCP and TP4 have no such capability. It is thought that multicast will have natural application to situations such as conference calls in which multiple destinations should receive an identical data stream. Using multicast, this is accomplished with a single transmission; for lack of any multicast capability, a user of TCP or TP4 could at best simulate the feature using some number of serial unicasts. Even so, management of such a multi-peer connection would then become a user responsibility, whereas in XTP the management of a multicast connection is inherent to the communications protocol.

In experiment six we enlarged our network to include not only the two voice stations and the background traffic generator, but also an additional voice station which was a member of the multicast voice group. Due to lack of identical equipment, the additional receiver was not the same as the two Motorola 68020/VMEbus/pSOS-based voice stations; it was an ALR FlexCache running XTP on a 25 MHz Intel 386 processor. Another difference was that the FlexCache operated a Network Peripherals FDDI interface (which used the National Semiconductor FDDI chip set), which proved to be interoperable with the Martin Marietta equipment and its AMD SuperNet FDDI.

In this experiment each voice message was assigned to a multicast group and transmitted using a transport multicast group address. Any number of stations could have been listening on

that group address, but in this experiment only two receivers were active. Message delivery to the set of multicast receivers was entirely reliable, that is, XTP assured that all data was reliably transmitted to all active receivers. Error repair, if any, was completely transparent as befits a transport protocol. The experiments conducted included a basic multicast experiment, followed by its repetition with the addition of 25, 50, and 75 Mbits/sec of synchronous background traffic using MPPT service.

5.7.2. Experiment 6: Analysis

Table 12 shows the average latencies and table 13 shows the 99.9% threshold latencies for experiment six.

MULTICAST
two receivers

voice data size (bytes)	average latency (ms)			
	0 Mbits/sec background load	25 Mbits/sec background load	50 Mbits/sec background load	75 Mbits/sec background load
8	5.266	6.311	6.209	7.559
16	5.273	6.318	6.213	7.564
32	5.277	6.293	6.240	7.573
64	5.336	6.330	6.247	7.588
128	5.431	6.375	6.290	7.608
256	5.603	6.412	6.609	7.673
512	5.894	6.461	8.136	7.770
1,024	6.712	9.515	10.460	10.804
2,048	7.997	11.174	13.761	11.608
4,096	10.402	14.134	15.247	14.575

Table 12
Average Latency for Multicast
with Synchronous Background MPPT FDDI Load

MULTICAST
two receivers

voice data size (bytes)	99.9% threshold (ms)			
	0 Mbits/sec background load	25 Mbits/sec background load	50 Mbits/sec background load	75 Mbits/sec background load
8	5.366	7.724	7.724	7.805
16	5.447	7.724	7.724	8.537
32	5.336	7.724	9.756	8.347
64	5.447	7.724	7.480	8.293
128	5.528	7.805	7.805	8.211
256	5.772	7.886	10.488	9.431
512	6.016	7.967	10.163	8.293
1,024	6.829	13.171	17.967	19.512
2,048	8.211	19.837	20.081	21.463
4,096	10.569	24.878	24.146	33.984

Table 13
99.9% Threshold for Multicast
with Synchronous Background MPPT FDDI Load

Multicast is inherently a more "expensive" operation than unicast since the protocol must handle multiple receivers; still, multicast is often less expensive than serial unicast for receiver groups even as small as two or three.

For the case of no background load on FDDI, the average latency using a two receiver multicast was less than twice the latency of a serial unicast in all cases. For small voice data sizes the increase was from about 2.7 ms using unicast to about 5.2 ms using multicast. Since the multicast replaced two unicasts, the proper comparison would be 2*2.7=5.4 ms for unicast vs. 5.2 ms for multicast, which makes multicast slightly faster.

For the largest voice data size, the increase was from about 7.7 ms using unicast to 10.4 ms using multicast. Again, the proper comparison is the delay of two unicasts (2*7.7=15.4 ms) vs. 10.4 ms for multicast. So, in this architecture, multicast is more effective that two serial unicasts, even for a receiver group size of two, for all voice data sizes.

As background load increased, the spread in latencies became more pronounced. When compared to multicast with no background load, a 75 Mbits/sec MPPT load increased average latency for small voice data sizes from about 5.2 to about 7.5 ms. At the largest voice data size, average latency increased from 10.4 to 14.5 ms.

Appendix F contains forty jitter plots, ten each for the cases of 0, 25, 50, and 75 Mbits/sec of background synchronous FDDI load. As load increased, the variance of the data increased markedly. Table 13 reports the 99.9% threshold values and they reflect this dramatic increase. Whereas at 75 Mbits/sec background load the average latency differed from the 99.9% threshold value by about 1 ms for small voice data sizes, for the largest voice data size the average was 14.6 ms but the 99.9% threshold was 34.0 ms, a 133% increase. Looking at the worst case, the last plot in Appendix F for a data size of 4096 bytes, we see a distinct "banding" of data. These

discrete bands are caused by the influence of the MPPT service. Had the background load been generated using SPPT, the latencies would have been more uniform. As previously suggested by experiment three, MPPT service should be abandoned in favor of SPPT service.

In spite of its favorable performance in this experiment, the multicast results would have been better had we used a better FDDI chipset. The AMD SuperNet interface does not support multicast link layer addresses (i.e., MAC group addresses), so all data in this experiment was actually transmitted using a link layer broadcast and then filtered at the transport layer to recognize the transport multicast address. Newer FDDI chips, such as those from National Semiconductor, do support link layer multicast addresses; that in turn would reduce processing in the destination hosts which would increase throughput and decrease latency.

5.7.3. Experiment 6: Conclusions

1. For the given system configuration, a multicast to two receivers was accomplished with lower total latency than two serial unicasts for all voice data sizes.

2. Multicast is an effective technique for distributing identical data to multiple receivers. The transport multicast capability is unique to XTP.

3. Multicast would have been even more effective if it had been supported by a link level multicast (i.e., MAC group address), rather than the link layer broadcast provided by the AMD SuperNet chips.

5.8. Experiment 7: Routers

One of the significant advantages of using a transport protocol instead of a datalink layer protocol is that data can be routed over multiple network segments. Experiment seven investigated whether latency considerations permitted routing of digitized voice.

The ideal situation would have been to measure the performance of an XTP router, but no such device exists. Therefore, we examined instead published data on the performance of three contemporary commercial Internet (IP) routers. The three routers considered were the Wellfleet Link Node, the Cisco AGS, and the Proteon p4200.

5.8.1. Experiment 7: Analysis

Appendix G contains both performance and delay graphs for these three routers. Each of these will be discussed in turn.

First, we should note that these are Ethernet routers, not FDDI routers. This means that the maximum packet size examined was 1518 bytes, of which 1500 are payload. Thus this data is indicative, but not necessarily representative, of the performance of modern FDDI routers. FDDI router information was simply not available.

Wellfleet. As shown in the first graph in Appendix G, the Wellfleet Link Node moved small packets rapidly between a pair of Ethernets, but performance decreased rapidly with increasing packet size. We considered three packet sizes: 512, 1024, and 1518 bytes. Wellfleet's performance was approximately 2000, 1500, and 1000 packets/sec for these three packet sizes, respectively. In terms of delay (second graph in Appendix G), the Wellfleet Link Node was measured at 1.4, 1.5, and 1.4 ms for packets of length 64, 512, and 1024 bytes, respectively.

Cisco. The Cisco AGS router's packet processing rate was faster than the Wellfleet for smaller packets but slower for larger packets. Refer to the third and fourth graphs in Appendix G. Performance measurements for packet sizes of 512, 1024, and 1518 bytes were about 1500, 1000, and 700 packets/sec, respectively. Delay through the Cisco router was significantly better than Wellfleet, and was measured to be 0.1, 0.5, and 1.0 ms for 64, 512, and 1024 byte packets respectively.

Proteon. The Proteon p4200 was the slowest of the three routers. Referring to the fifth and sixth graphs in Appendix G, its performance for packets of length 512, 1024, and 1518 bytes was approximately 800, 600, and 400 packets/sec respectively. Likewise, delay through the router was the longest of the three. The delay for 64, 512, and 1024 byte packets was 1.0, 2.8, and 3.3 ms, respectively. Discussions with Proteon revealed that the model p4200 was known to be slow and was due for a replacement with a new model which is to have significantly improved (but as yet undocumented) performance.

The router data was inconclusive. Since the three units benchmarked were Ethernet routers rather than FDDI routers, it was not clear whether these performance numbers were valid for our environment. Furthermore, the range of packet sizes over which they were tested falls short of the range of interest for FDDI; the maximum size Ethernet packet is 1518 bytes whereas the maximum FDDI packet is 4500 bytes. The Cisco and Proteon routers both showed a delay curve which was dependent upon packet length, indicating that packet copying was a significant fraction of the total delay. It is simply unknown whether a maximum length FDDI packet (which is three times longer than a maximum length Ethernet packet) would have a delay three times as long as a maximum length Ethernet packet. Likewise, it is unknown whether advances in router technology (which are proceeding at a rapid pace) will significantly improve the throughput and

reduce the delay through a FDDI router.

The router issue clearly needs more examination. Since we were using XTP packets, and the routers for which we had performance data were Ethernet routers routing Internet packets, it is very difficult to estimate the performance of an XTP FDDI router. Our best attempt at doing so assumes that XTP routing is no slower than IP routing (due to the design of the protocol it should actually be faster), that FDDI routers are no slower than Ethernet routers, and that advances in router technology will eliminate the present disparity between copying 1500 byte packets and 4500 byte packets. Under those assumptions, we estimate that an FDDI XTP router will introduce an additional delay of less than 1 ms for a 1024 byte packet.

Since only a fraction of the total voice distribution traffic would be routed over multiple networks, the added delay of the router would apply to only a portion of the network traffic. Even so, if the router adds a delay in the neighborhood of 1 ms, we think it is of small significance to any single voice channel. We think that the ability to route digitized voice information over arbitrary network topologies is a significant benefit. If in fact it can be accomplished at a cost of an additional delay on the order of 1 ms per one kilobyte packet, then it is well worth the cost.

5.8.2. Experiment 7: Conclusions

1. It is difficult to estimate the delay introduced by an XTP router, since none exist. Using existing Ethernet and Internet routers as a guide, we estimate a delay of approximately 1 ms through an XTP and FDDI router for packets of size 1024 bytes.

2. The ability to route digitized voice over an arbitrary network topology is a very significant benefit. If it can be done with an additional delay on the order of 1 ms per one kilobyte packet, it

is well worth the investment.

6. DATALINK VS. TRANSPORT PROTOCOLS

Using the Xpress Transfer Protocol (a combined transport plus network layer protocol) is clearly more costly than using a datalink layer protocol. Is it worth it?

A survey of the literature on using local area networks for the distribution of voice data ([Arth79], [Frie89], [Gait89], [Gait90], [Gehl91], [Suda89]) suggests the following:

(1) the acceptable jitter between successive voice packets is about 20 ms

(2) an acceptable loss rate for voice packets is 1-2%

(3) historically, voice packets have carried a modest amount of information (20 to 50 ms) so that the loss of any one packet had little impact on voice understandability

(4) the total delay between voice nodes should not be greater than 250 ms to avoid the start/stop effect (common to satellite voice channels)

These criteria are easily met by a datalink protocol running over FDDI, as evidenced by E-Systems' existing voice distribution system, but, with the proper choice of voice data size, XTP can meet those goals as well. Thus we can not discard either approach based on these historical criteria.

Datalink protocols have the advantage of simplicity; since they are less powerful than transport protocols, they therefore consume less of the CPU. If we restrict the comparison to an environment in which a datalink protocol will work (e.g., a single segment LAN), then in a head-to-head competition against a transport protocol a datalink protocol would probably support a larger number of voice channels. At least on a single segment LAN, the packet loss rate of a datalink protocol operating over FDDI should be perfectly adequate. With a datagram protocol, steering a voice channel to a particular destination process would require the user to

manage process addresses, since the protocol manages only host addresses (MAC addresses). In contrast, the transport protocol manages peer addresses automatically when it accomplishes connection setup. In the context of a single segment LAN, both approaches work. The datalink layer protocol could reasonably be expected to be somewhat more efficient, whereas the transport protocol is probably easier to use because it provides a greater variety of services to the user.

The advantages of a transport protocol emerge when we leave the environment of a single segment network. Datalink protocols do not operate over routers, and at a minimum a network layer protocol is required. If a network protocol is added to accommodate routing, then a transport protocol is a natural addition as well. A transport protocol provides guaranteed, in-order, sequential delivery without duplicates, which is generally good for data even though not strictly required for voice. If XTP is chosen as the transport protocol, then there is the unique advantage of multicast, which in these experiments was shown to have lower latency than serial multicast even for receiver group sizes as small as two.

So the original question now divides into sub-questions.

(1) Must the system operate over multiple, interconnected LANs? If so, then the LANs must be interconnected by bridges or routers. If the choice is bridges, datalink and transport protocols are both feasible; if the choice is routers, only a transport plus network protocol is feasible.

(2) Must the system interconnect with wide area networks? If so, only a transport plus network protocol is feasible.

(3) Would the system benefit from a transport multicast capability? If so, only a transport protocol, and in fact only XTP, is feasible.

(4) Can a given system "afford" the power and convenience of a transport protocol? The answer is obviously application dependent, but the data we gathered is encouraging. Consider the case when voice data size equals 1024 bytes. For a single voice channel, delivery must occur every 128 ms. Yet if we examine all the data we collected (excluding multicast which is special purpose), the worst case latency observed was 10 ms (this was the 99.9% threshold with heavy background synchronous load). Thus, the performance of a transport protocol for any single voice channel is clearly well within bounds, and it is a matter of further experimentation to determine how many voice channels can be carried simultaneously.

7. XTP VS. OTHER TRANSPORT PROTOCOLS

We used XTP for this study because (1) it was readily available, (2) we had intimate knowledge of its construction and operation, and (3) it alone provided a multicast capability. While we believe that XTP worked well in the given environment, we can not conclude that TCP or TP4 would not work, only that they would not work as well.

An examination of XTP vs. TCP in [Hart91] showed that XTP had all the features of TCP and more, and was somewhat more efficient than the Wollongong TCP against which it was compared. A notable difference was the time spent in establishing a new connection—XTP (which optimizes connection establishment) could set up a new connection and begin sending data in 2-3 ms, whereas TCP needed 14-17 ms for the same task.

If we substituted TCP for XTP, and compared only pre-established connections, we would expect to see a slight decrease in throughput, but no significant change in latency. Thus we project that TCP would work in this environment, although it might carry fewer voice channels simultaneously. An advantage of TCP, however, is that IP routers are commercially available

for FDDI.

Substituting TP4 for XTP would, we believe, cause a more serious degradation. We are confident that a TP4 implementation would have lower throughput and higher latency than either TCP or XTP executing on identical hardware. Exactly what its performance would be is, of course, impossible to predict without more experimentation.

Since it was not a goal of this feasibility study to compare the actual performance of XTP, TCP, and TP4, our comments here are qualitative rather than quantitative. We believe that all three protocols would work for a small number of voice channels, and that the distinctions among them would emerge as the number of voice channels increased.

Although we think that XTP has a small speed advantage over TCP (and a larger one over TP4) when implemented in software, the real performance gains will occur when XTP is implemented in hardware as the *Protocol Engine*. Protocol Engines Inc. is currently designing a four-piece VLSI chipset which will mate with one or more commercial FDDI chipsets. The design goal of the Protocol Engine is to realize transport services at FDDI rates, i.e., 100+ Mbits/sec through the Protocol Engine. If that goal is realized then XTP in hardware would be vastly superior to TCP or TP4 for digitized voice distribution.

TCP and TP4 each have at least one advantage over XTP—they are standards. TCP is an extant military standard and TP4 is an extant international standard, whereas XTP has just recently begun the standardization process through ANSI. Standardization is a political process, so it is currently unknown whether and when XTP will emerge as an ANSI standard. All we can say with assurity is that ANSI has authorized the necessary study projects for a "High Speed Protocol" (HSP) and that XTP is being considered by the HSP committee.

8. CONCLUSIONS

Basic Experiment

1. System efficiency increases with increasing voice data size (*n* 8-bit voice samples). The side effect which prohibits arbitrarily large voice data size is the connection startup time for a channel, which is *n** 125 microseconds plus end-to-end latency.

2. In the basic configuration, the number of voice channels which could be supported simultaneously is a function of voice data size as follows:

voice data size (bytes)	voice data throughput Kbits/sec	equivalent voice channels (64 Kbits/sec/channel)
128	468	7
256	897	14
512	1661	25
1024	3050	47
2048	5115	79
4096	7294	113

3. For a voice data size of *n*, data must be delivered within *n** 125 microseconds to avoid FIFO underflow in the receiver. For all *n*>=128, the average delivery latency and the 99.9% threshold latency were well below the required delivery latency.

voice data size (bytes)	required delivery latency (ms)	average latency (ms)	99.9% threshold (ms)
128	16	2.890	3.008
256	32	3.062	3.171
512	64	3.417	3.496
1,024	128	4.015	4.146
2,048	256	5.242	5.366
4,096	512	7.699	7.805

4. In the basic configuration, jitter was extremely low. The worst deviation of measured latency from average latency was 0.2 ms.

5. The basic configuration was a "best case" scenario for the given architecture, so the impact of additional background FDDI load, additional processor load, packet loss and retransmission, etc., had to be confirmed by experimentation.

Synchronous SPPT Background FDDI Load

6. In this configuration the impact of a heavy (75 Mbits/sec) background load in FDDI's synchronous class was insignificant; in the worst case, the 99.9% threshold latency increased by less than 0.5 ms. The reason that the effect was so small was that the ring was small (three stations) and the background load was generated using a single-packet-per-token (SPPT) service discipline.

7. SPPT service is highly desirable; it provides frequent opportunities for the voice stations to transmit.

Synchronous MPPT Background FDDI Load

8. A multiple-packet-per-token (MPPT) service discipline is permitted by FDDI, and its use increases network efficiency by reducing per-packet overhead. However, its use is injurious to voice traffic, since synchronous voice can then be delayed by non-voice synchronous data.

9. The delays which might be encountered when using MPPT service are bounded only by the correct operation of FDDI's Station Management (SMT) protocol, which limits the amount of synchronous data which any one station can send on any one token cycle. If MPPT service is

used, SMT should be operational.

10. MPPT service marginally increases the efficiency of a station sending non-voice synchronous traffic, while increasing the variance of all voice traffic. Thus MPPT service is not recommended; use SPPT instead.

Packet Loss and Retransmission

11. Modern fiber optic networks such as FDDI have a very low packet loss rate. Even so, when errors do occur, XTP repairs them swiftly and efficiently. When comparing a packet loss rate of 0% to a loss rate of 10% in this configuration, the worst case increase in average end-to-end delay was less that 0.1 ms.

12. XTP's selective retransmission algorithm is very effective for error repair.

13. One advantage of using a transport protocol is that the residual end-to-end packet loss rate is zero; that is, every packet transmitted will be received.

Asynchronous Processor Load

14. The addition of an asynchronous processing load (defined to be 120 messages/sec, half of length 40 bytes and half of length 512 bytes) did not significantly increase the latency of voice traffic.

15. Voice traffic in FDDI's synchronous class was effectively insulated from any additional communications load in the asynchronous class.

Multicast

16. Multicast is an effective technique for delivering identical data to multiple receivers with a single transmission. For the given system configuration, and for all voice data sizes, a multicast to two receivers was accomplished with lower overall latency than two serial unicasts.

17. Multicast would have been even more effective if it had the support of MAC group addresses (as with the National Semiconductor FDDI), rather than the more limited link layer broadcast supported by the AMD SuperNet FDDI.

18. XTP is the only protocol which supports a transport layer reliable multicast; there is no such concept in TCP or TP4.

Routers

19. It is difficult to predict the performance of an XTP router over FDDI, since none exist. However, judging from the performance of Internet routers over Ethernet, it seems reasonable to assume a router delay of less than 1 ms for 1024 byte packets.

20. If an XTP router can be built which yields sub-millisecond delays, then voice distribution over multiple segment networks becomes truly feasible.

Datalink vs. Transport Protocols

21. A datalink protocol over FDDI and XTP over FDDI both satisfy the requirements for voice distribution over a single segment network. The datalink protocol could be reasonably expected to be somewhat more efficient (since it is less powerful), but the transport protocol would probably be easier to use since it provides a variety of services to the user.

22. If a network is multi-segment, then it must be connected via bridges or routers. If routers are chosen, then a transport plus network protocol is the natural choice.

XTP vs. Other Transport Protocols

23. No formal comparison was made between XTP and either TCP or TP4. Experience with all three suggests that XTP would have slightly higher throughput than TCP and significantly better throughput than TP4 in this environment. XTP could therefore be expected to carry more voice channels simultaneously than the others.

24. Performance will improve markedly when XTP is available in hardware as the *Protocol Engine*. The goal of the Protocol Engine is to provide transport services at the 100+ Mbits/sec rate of FDDI or similar media.

25. TCP is a military standard and TP4 is an international standard. Standardization efforts for XTP have just recently begun through ANSI. It is impossible to predict whether and when XTP might become a recognized ANSI or ISO standard.

9. REFERENCES

[Arth79] E. Arthurs and B.W. Stuck, "A Theoretical Traffic Performance Analysis of an Integrated Voice-Data Virtual Circuit Packet Switch," *IEEE Transactions on Communications,* Vol. 27, No. 7, July 1979.

[Frie89] E. Friedman and C. Ziegler, "Packet Voice Communications Over PC-Based Local Area Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 7, No. 2, February 1989.

[Gait89] S.S. Gaitonde, D.W. Jacobson, A.V. Pohm, "Bounding Delay on a Token Ring Network with Voice, Data, and Facsimile Applications: A Simulation Study," *Proc. of the Eighth Phoenix Conference on Computer Communications,* 1989.

[Gait90] S.S. Gaitonde, D.W. Jacobson, A.V. Pohm, "Bounded Delay on a Multifarious Token Ring Network," *Communications of the ACM,* Vol. 33, No. 1, January 1990.

[Gehl91] T.L. Gehl, "Packetized Voice for Simulated Command, Control, and Communications," *Interservice/Industry Training Systems Conference*, to be presented.

[Hart91] Timothy W. Hartrick, *Performance Evaluation of a Software Implementation of the Xpress Transfer Protocol*, Master of Science thesis, Department of Computer Science, University of Virginia, January 1991.

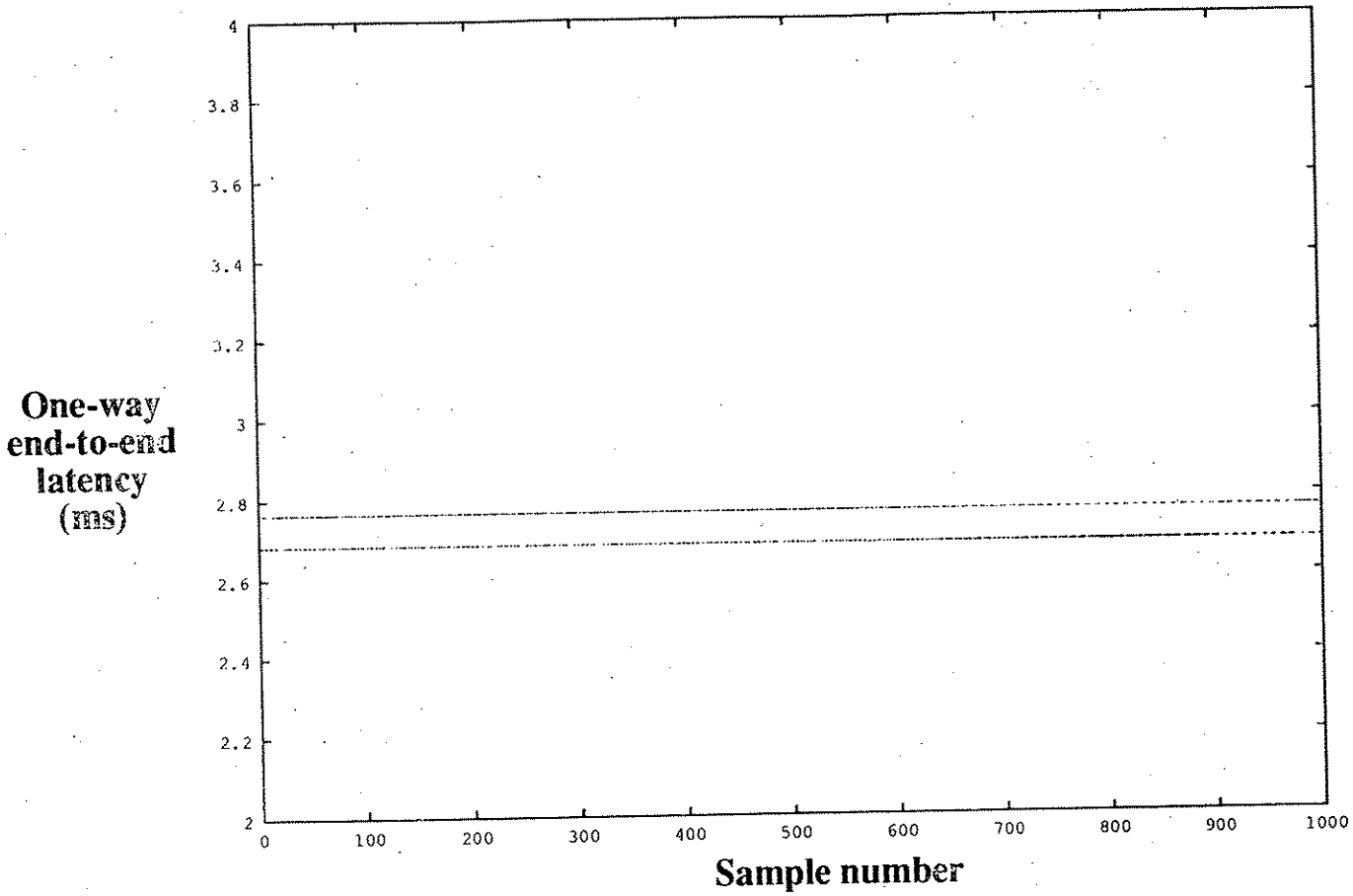
[Suda89] T. Suda and T.T. Bradley, "Packetized Voice/Data Integrated Transmission on a Token Passing Ring Local Area Network," *IEEE Transactions on Communications," Vol. 32, No. 3, March 1989.*

Appendix A

Experiment 1: Basic Experiment

JITTER MEASUREMENT

Voice Data Size: 8 bytes



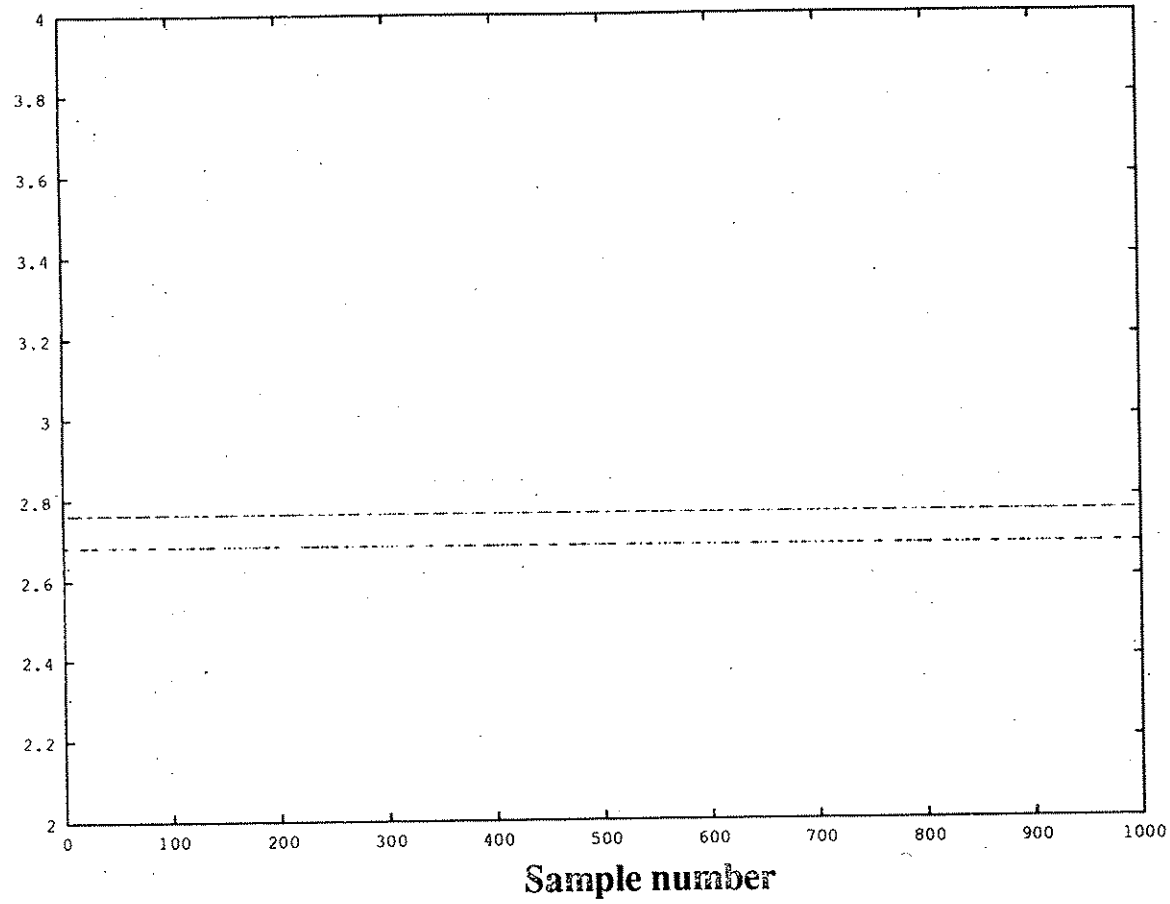
1000 samples
No asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 2.728 ms
99.9% threshold: 2.764 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 16 bytes



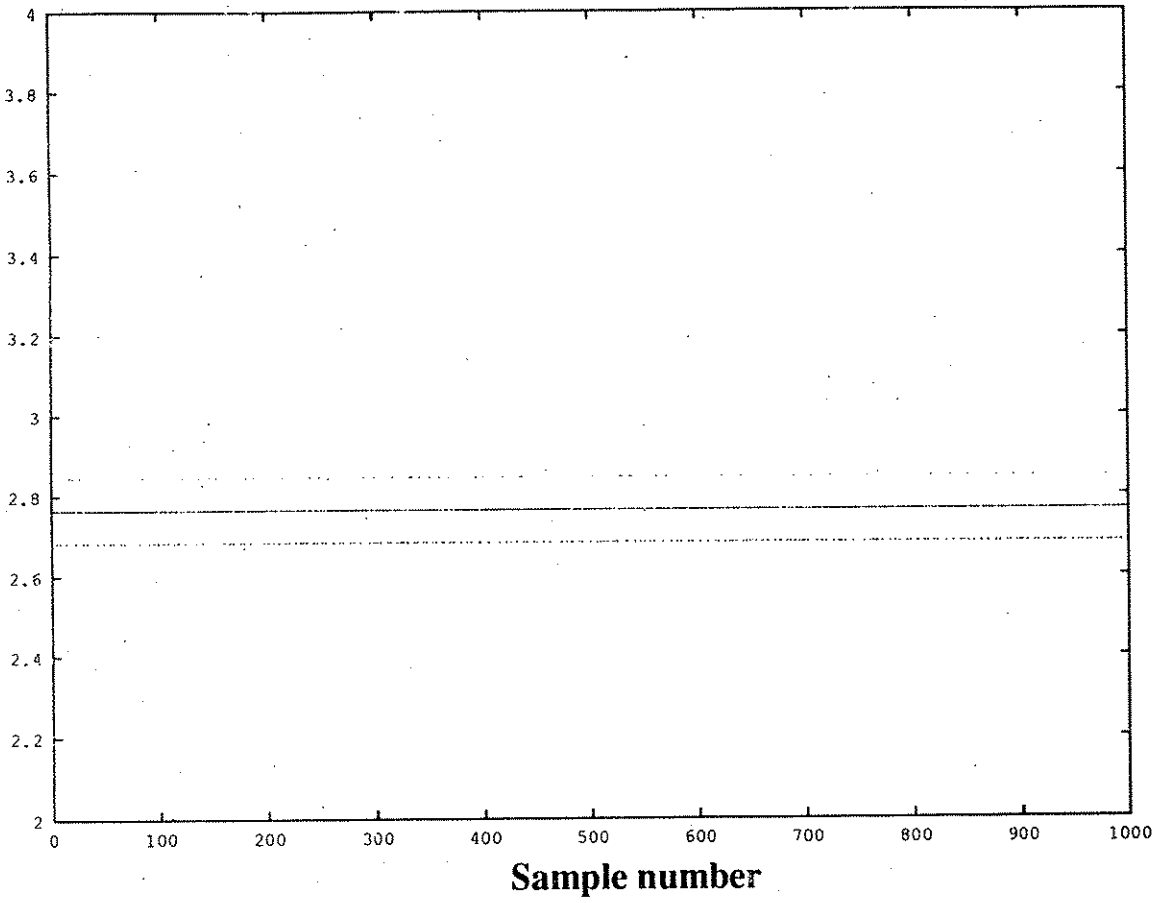
1000 samples
No asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 2.732 ms
99.9% threshold: 2.846 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 32 bytes



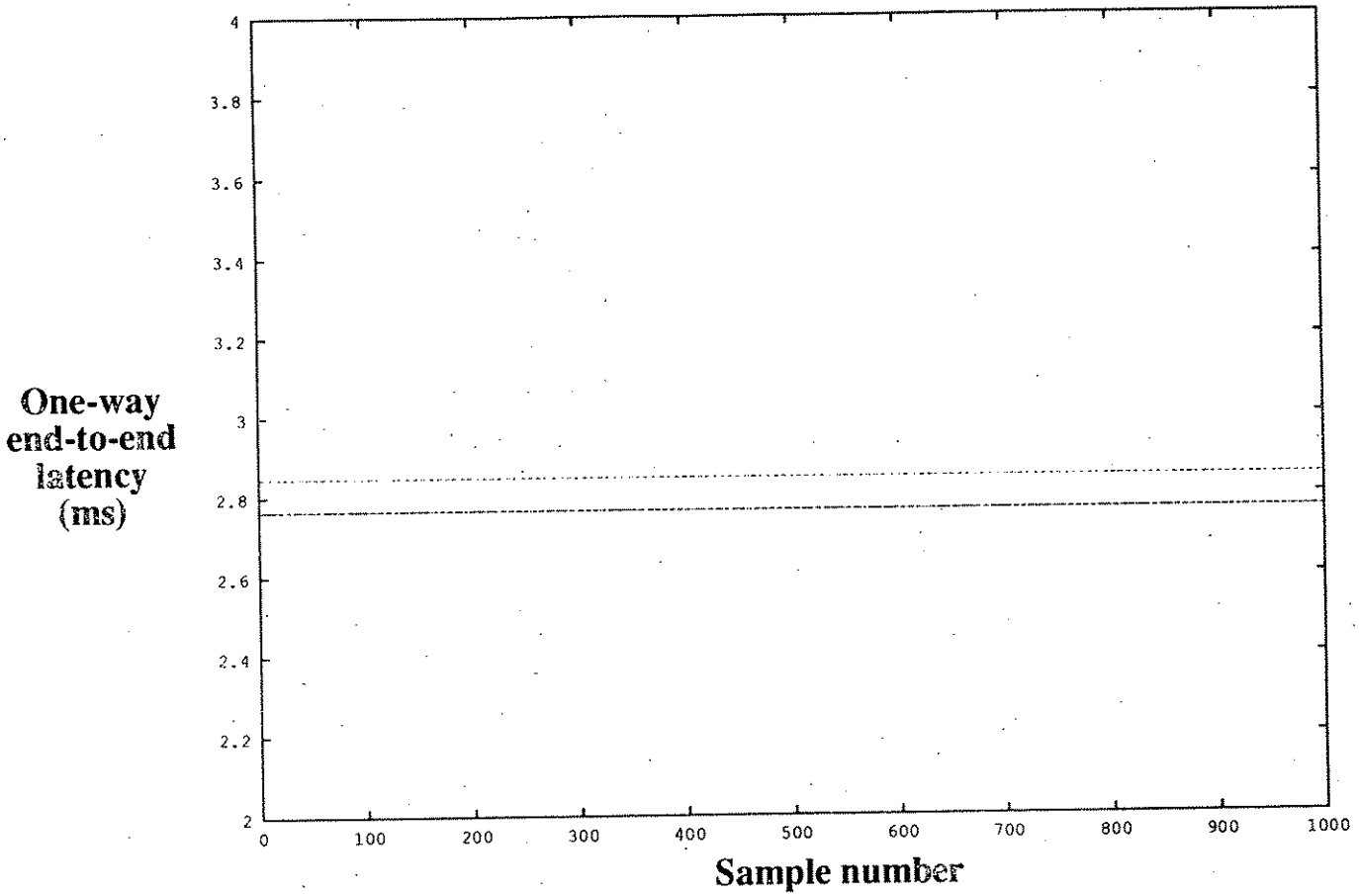
1000 samples
No asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 2.751 ms
99.9% threshold: 2.846 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 64 bytes



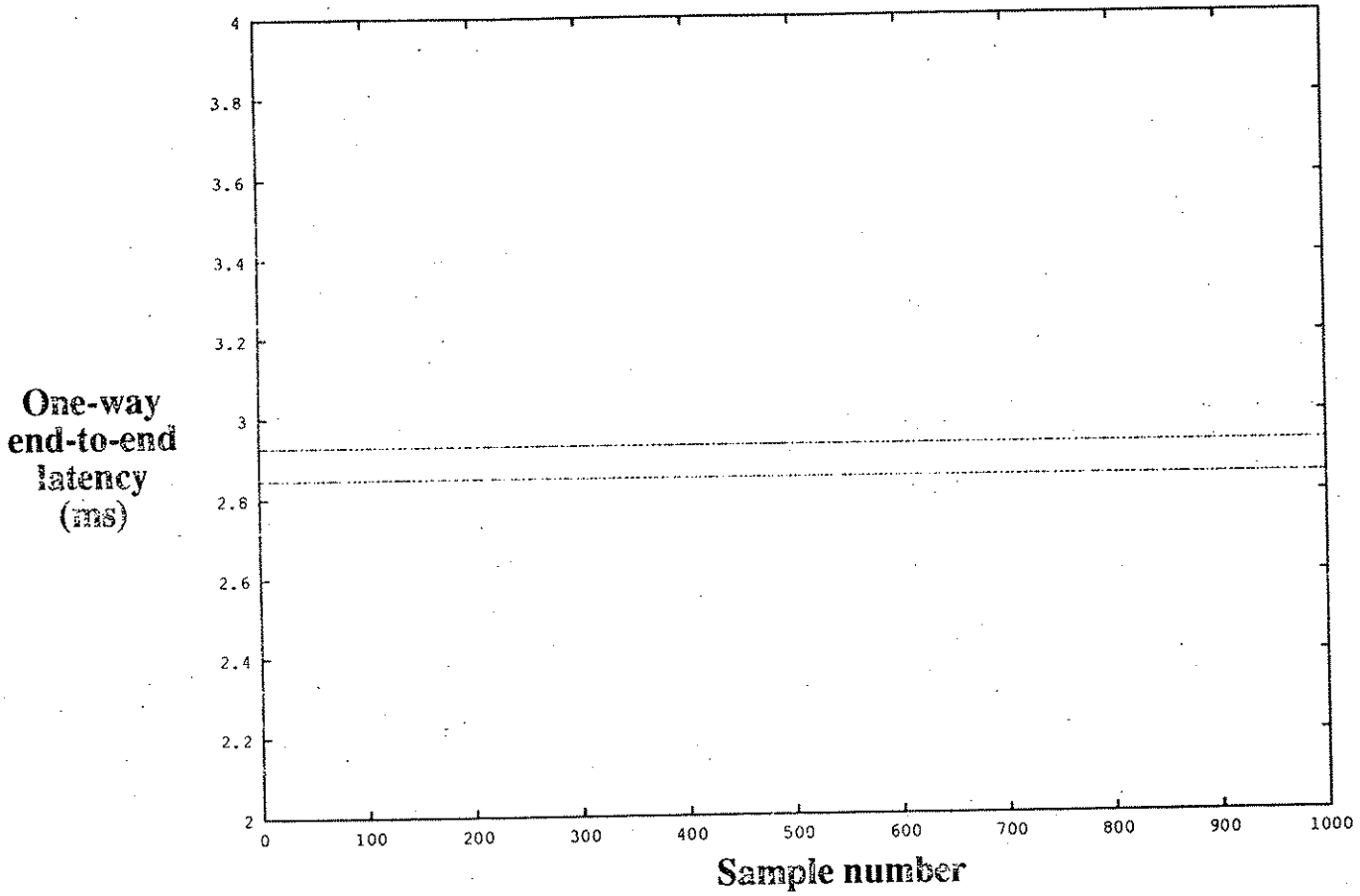
1000 samples
No asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 2.791 ms
99.9% threshold: 2.927 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 128 bytes



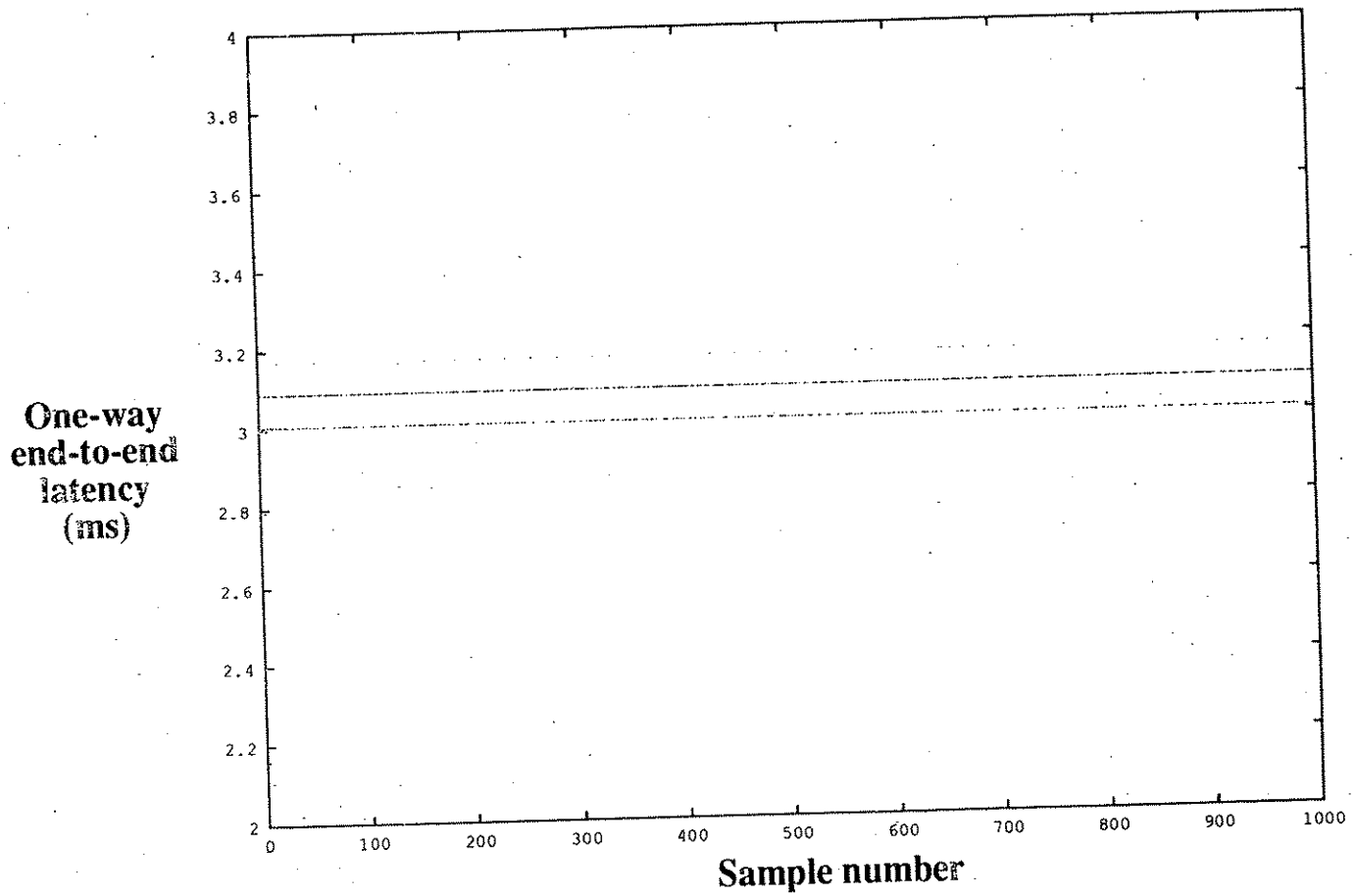
1000 samples
No asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 2.890 ms
99.9% threshold: 3.008 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 256 bytes



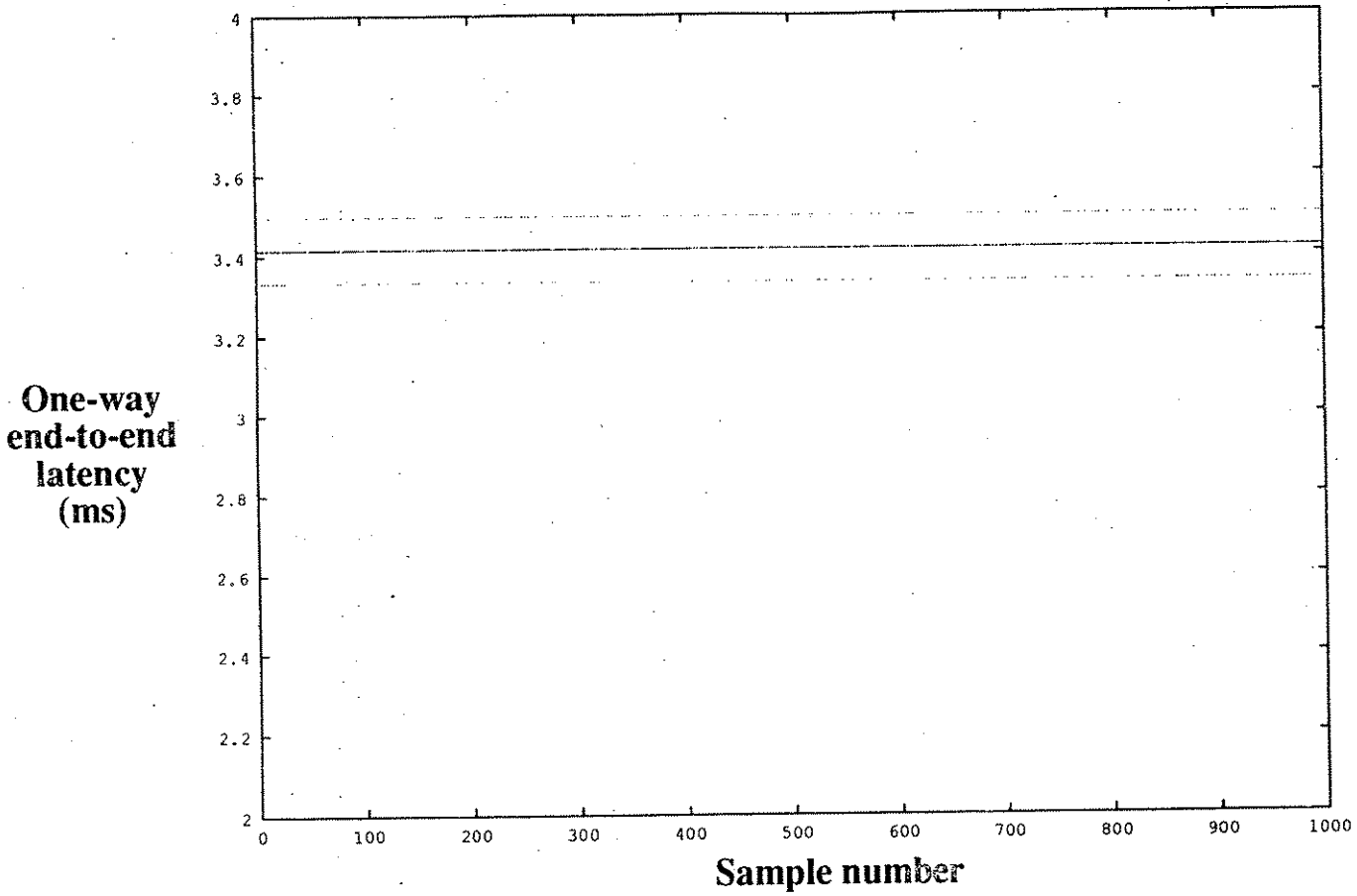
1000 samples
No asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 3.062 ms
99.9% threshold: 3.171 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 512 bytes



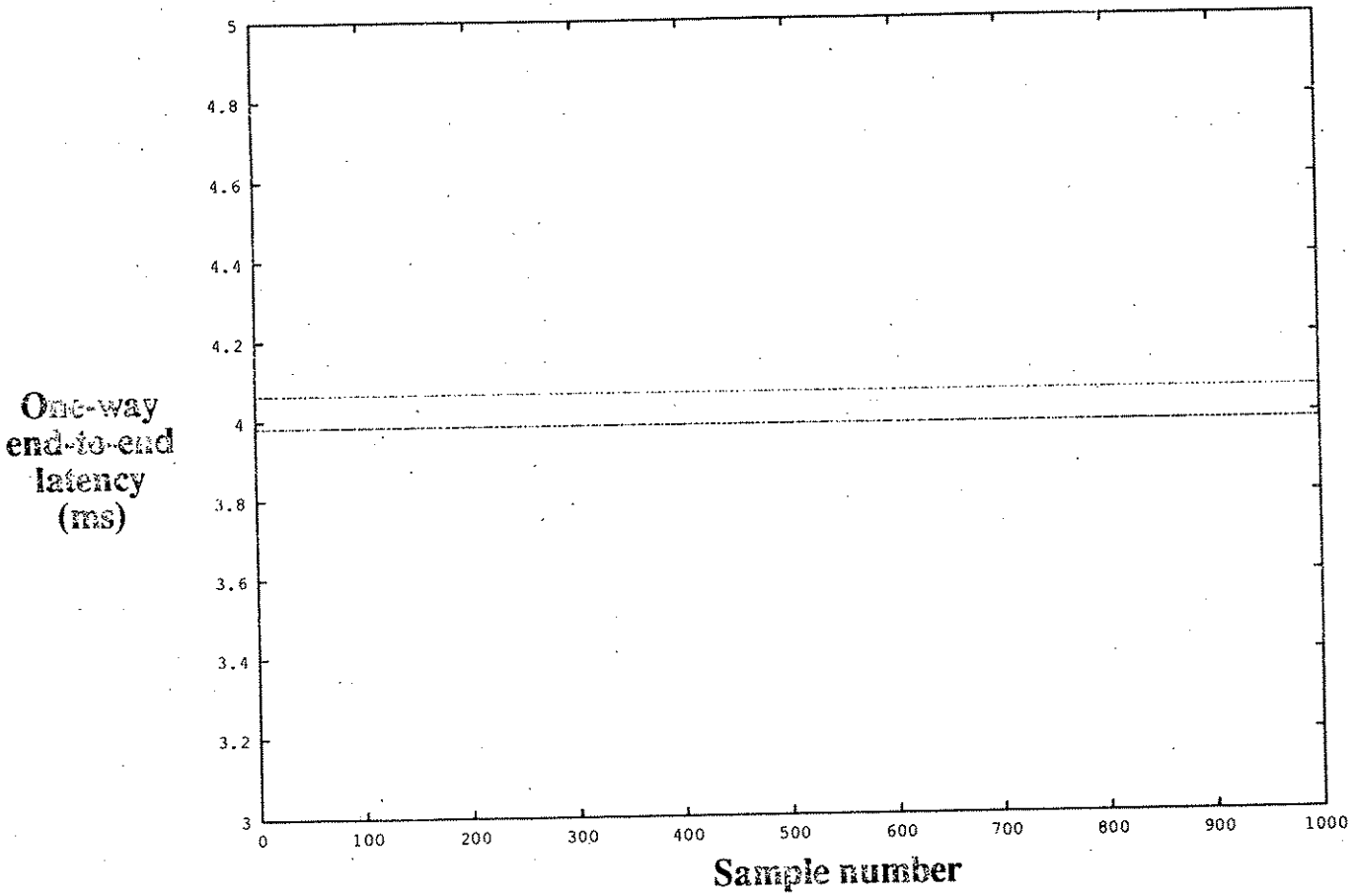
1000 samples
No asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 3.417 ms
99.9% threshold: 3.496 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



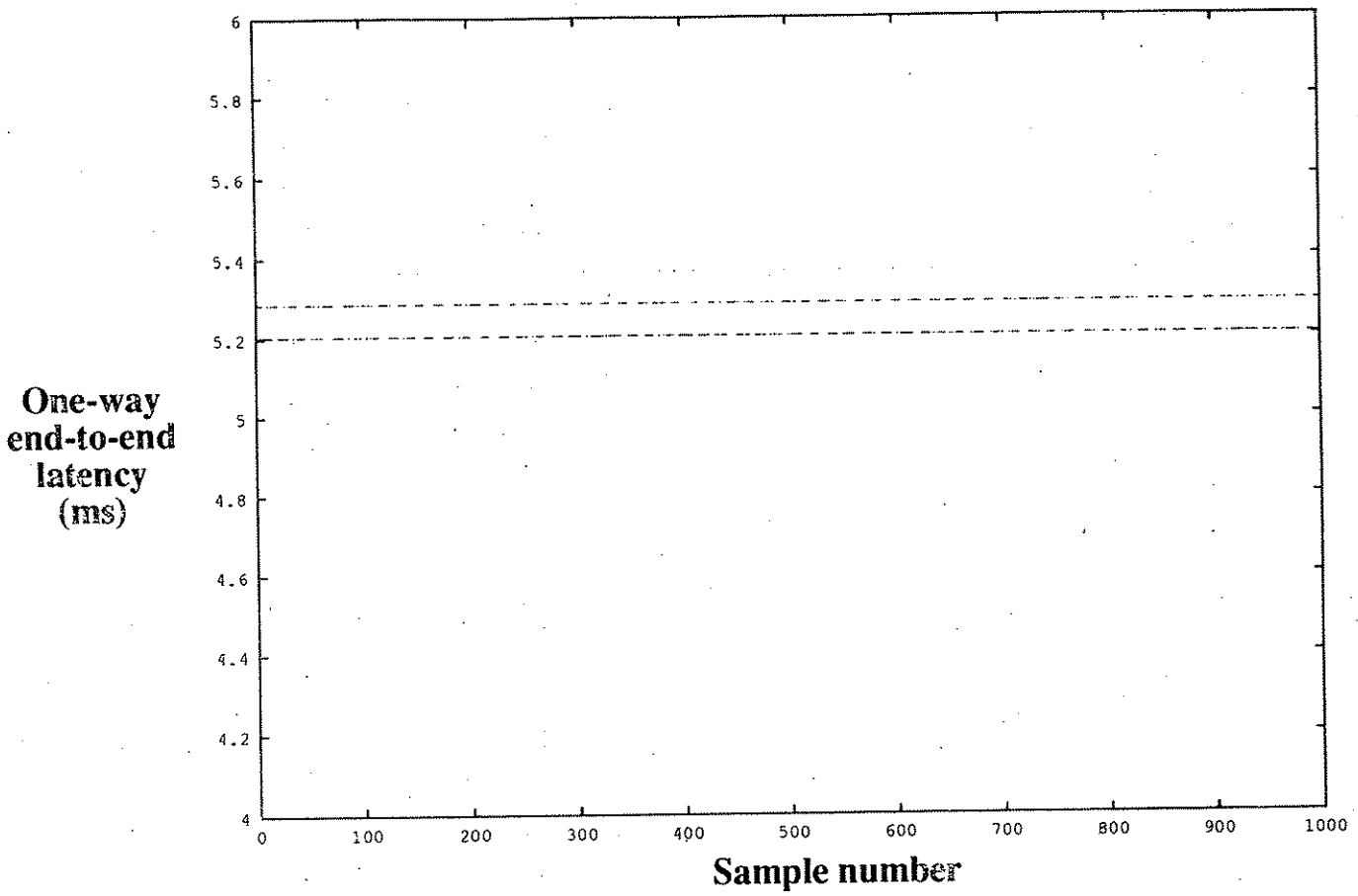
1000 samples
No asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 4.015 ms
99.9% threshold: 4.146 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



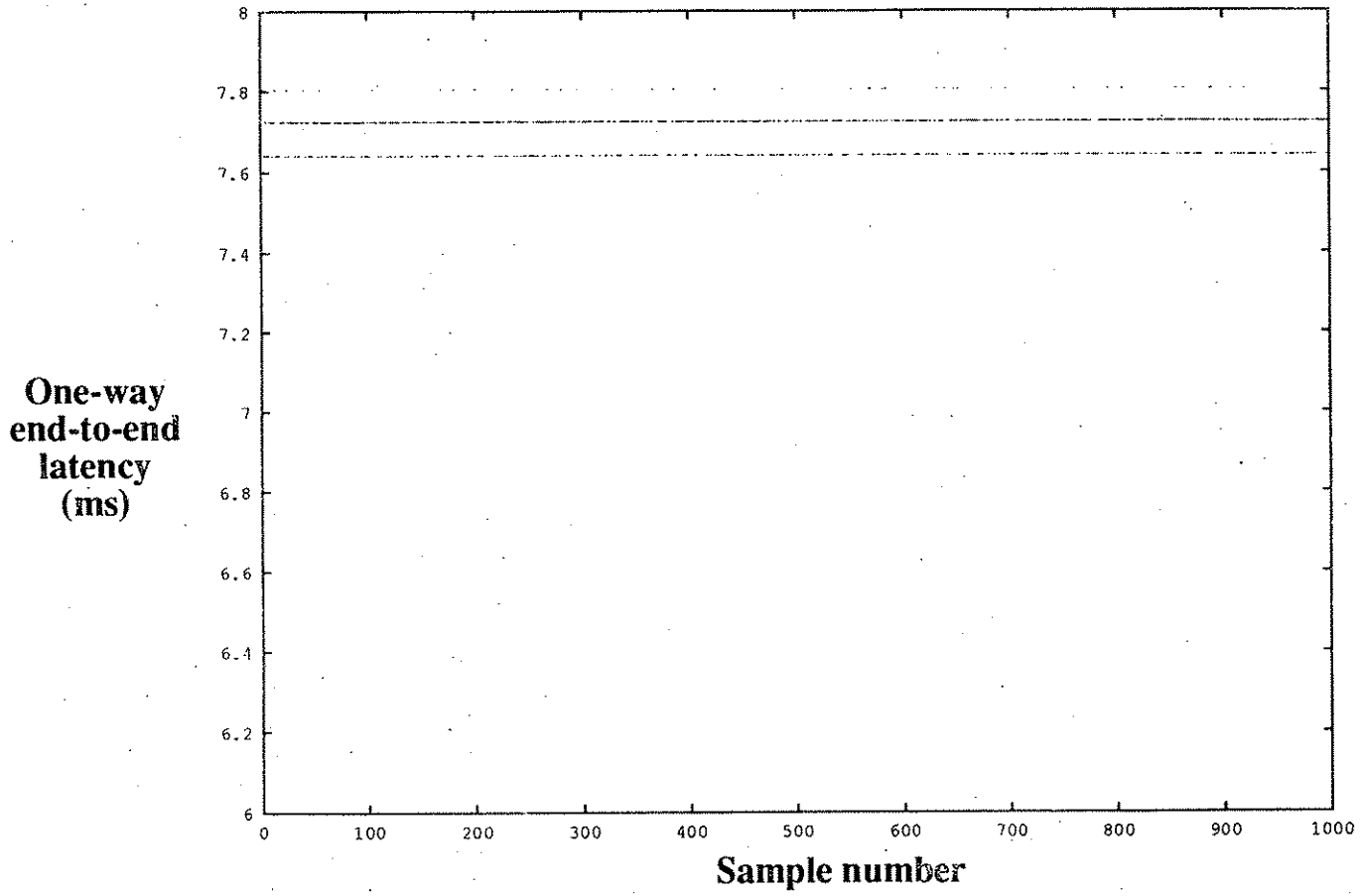
1000 samples
No asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 5.242 ms
99.9% threshold: 5.366 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



1000 samples
No asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 7.699 ms
99.9% threshold: 7.805 ms

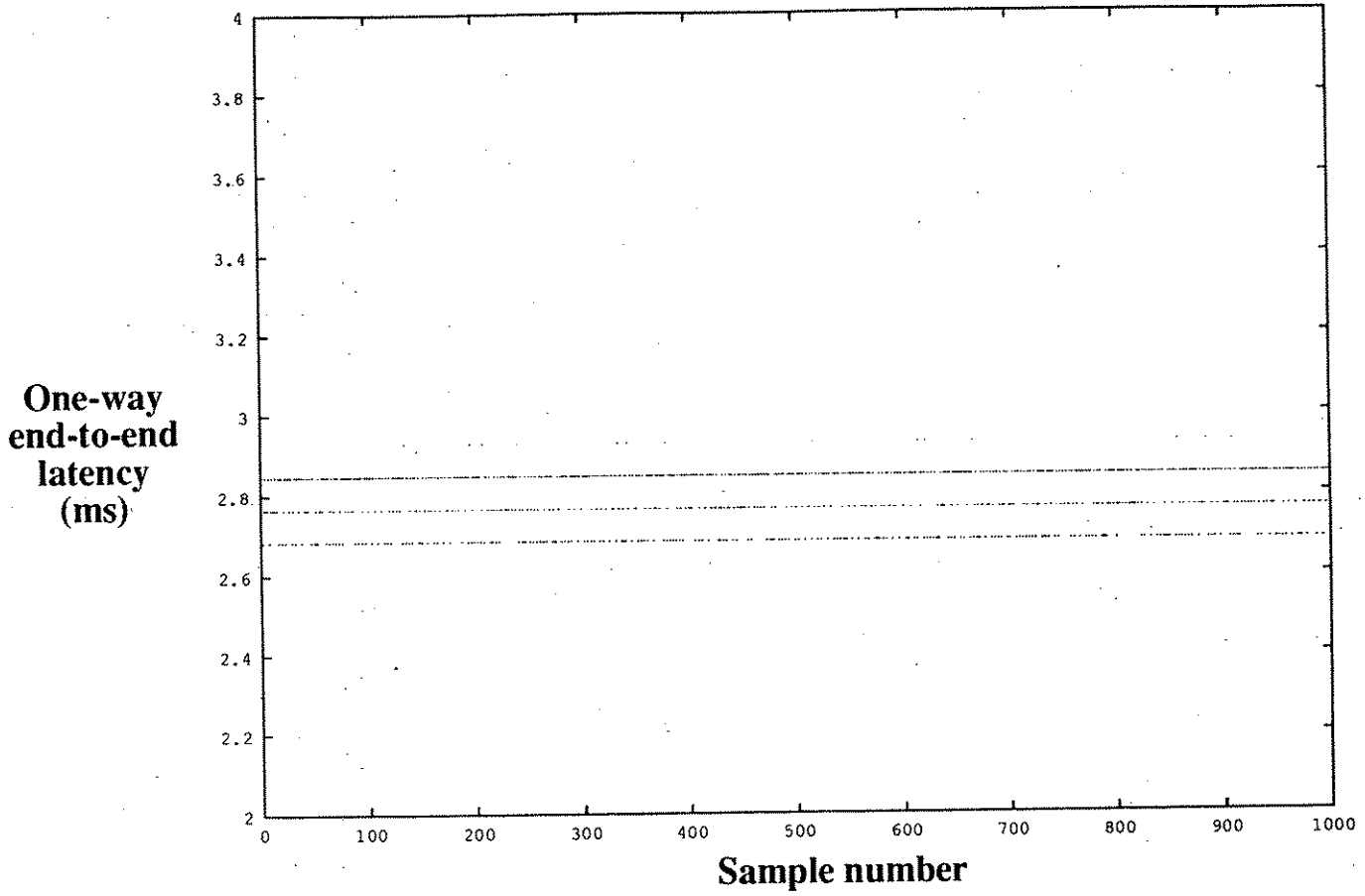
SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

Appendix B

Experiment 2: Synchronous SPPT Background FDDI Load

JITTER MEASUREMENT

Voice Data Size: 8 bytes



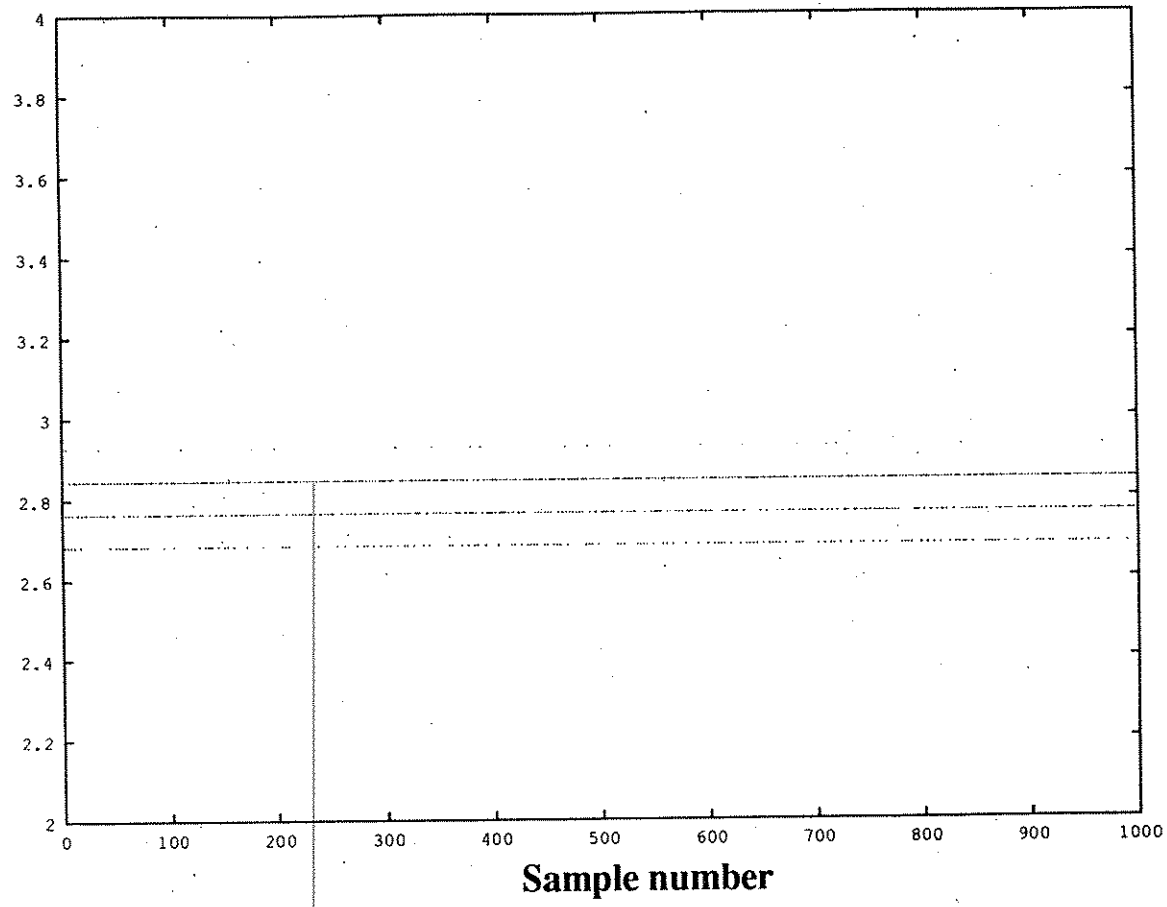
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 2.789 ms
99.9% threshold: 2.927 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 16 bytes



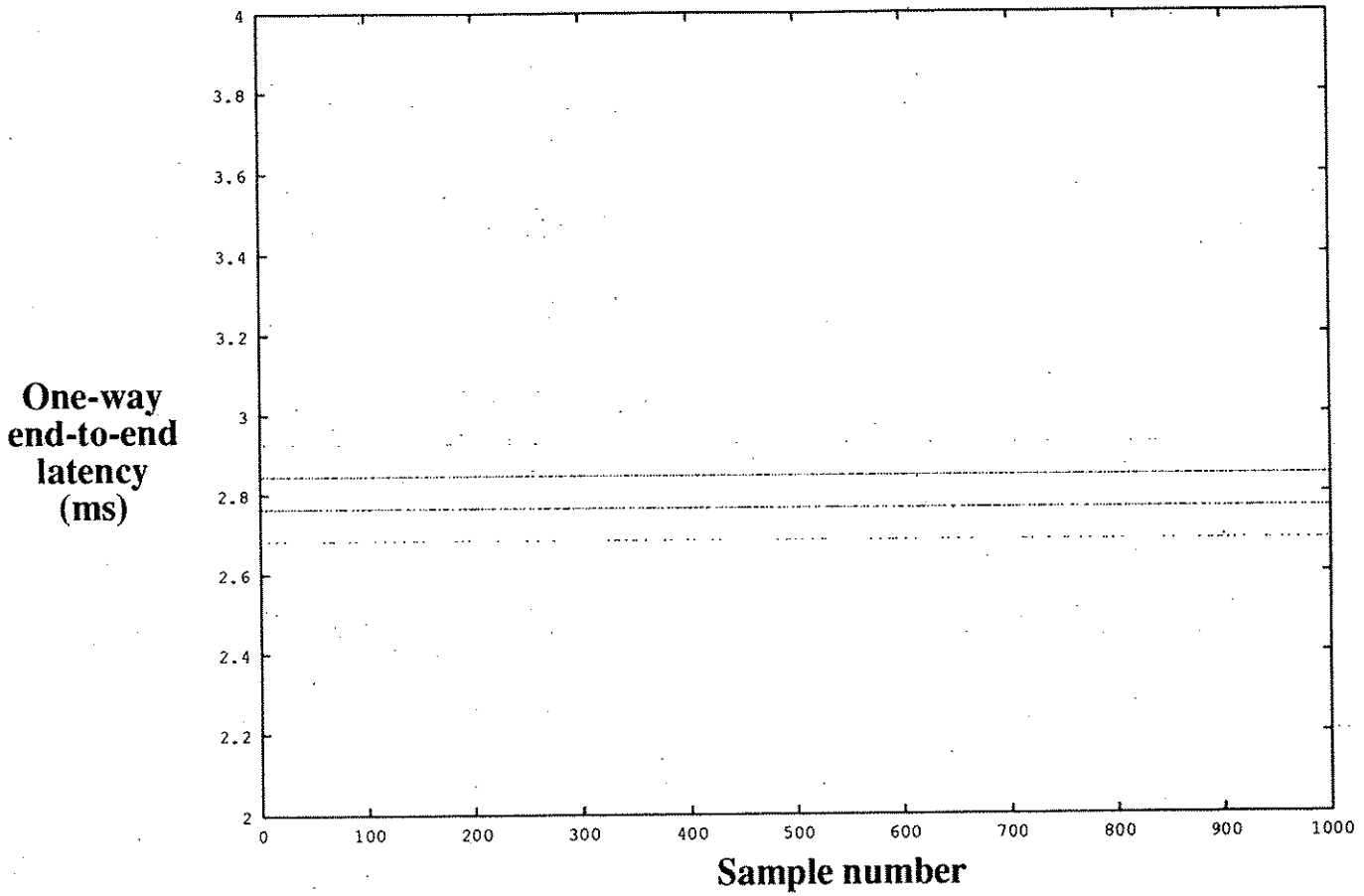
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 2.796 ms
99.9% threshold: 2.927 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 32 bytes



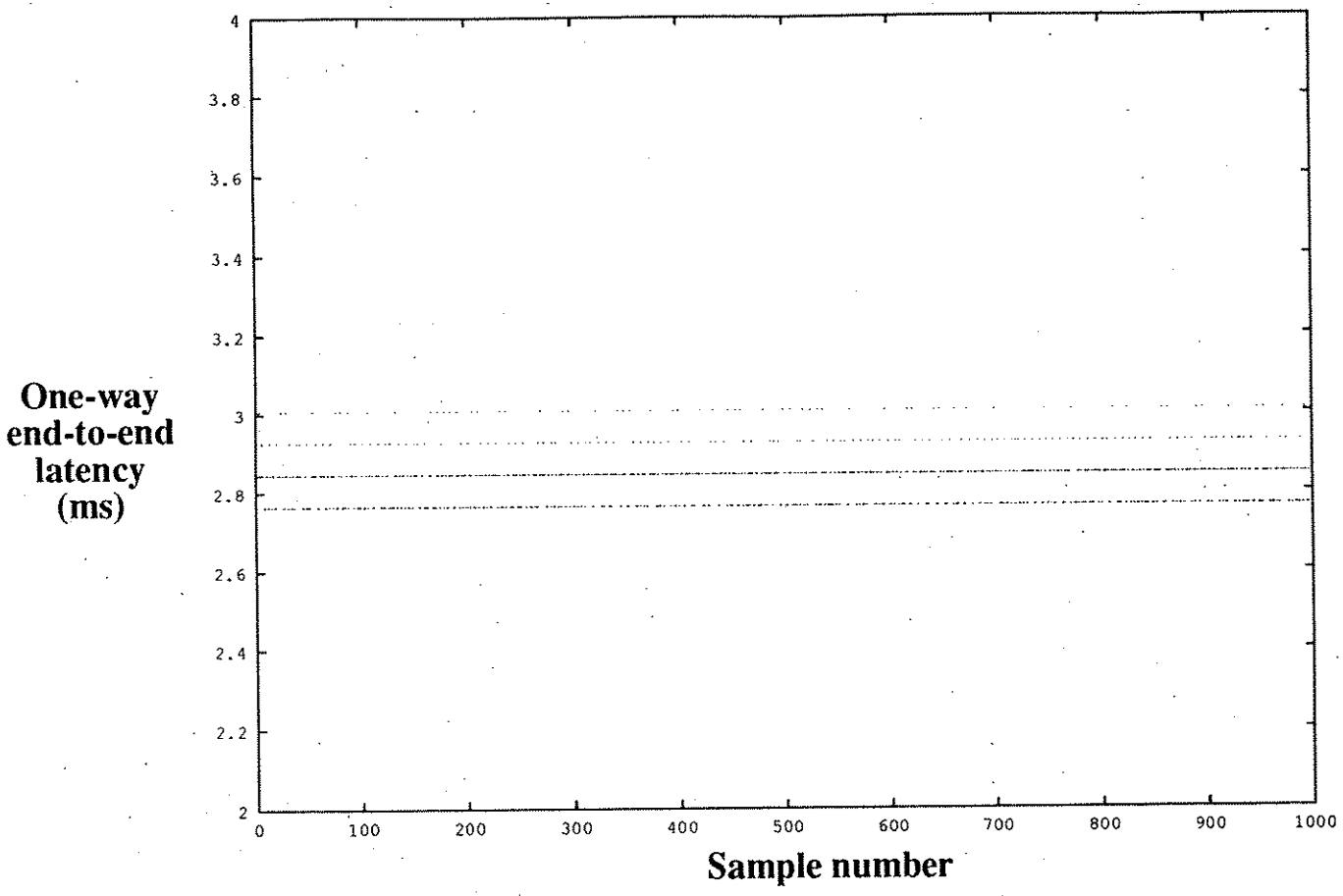
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 2.799 ms
99.9% threshold: 2.927 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 64 bytes



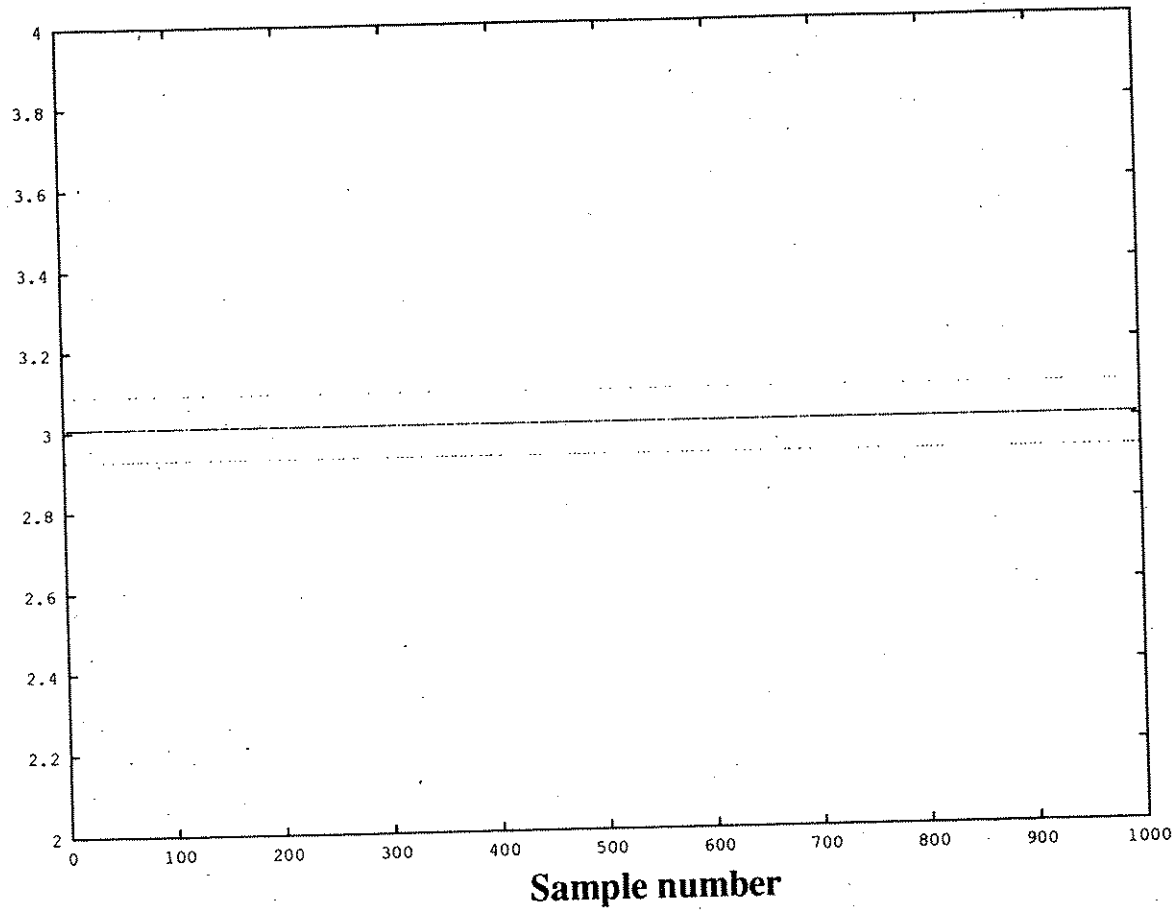
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 2.836 ms
99.9% threshold: 3.008 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 128 bytes



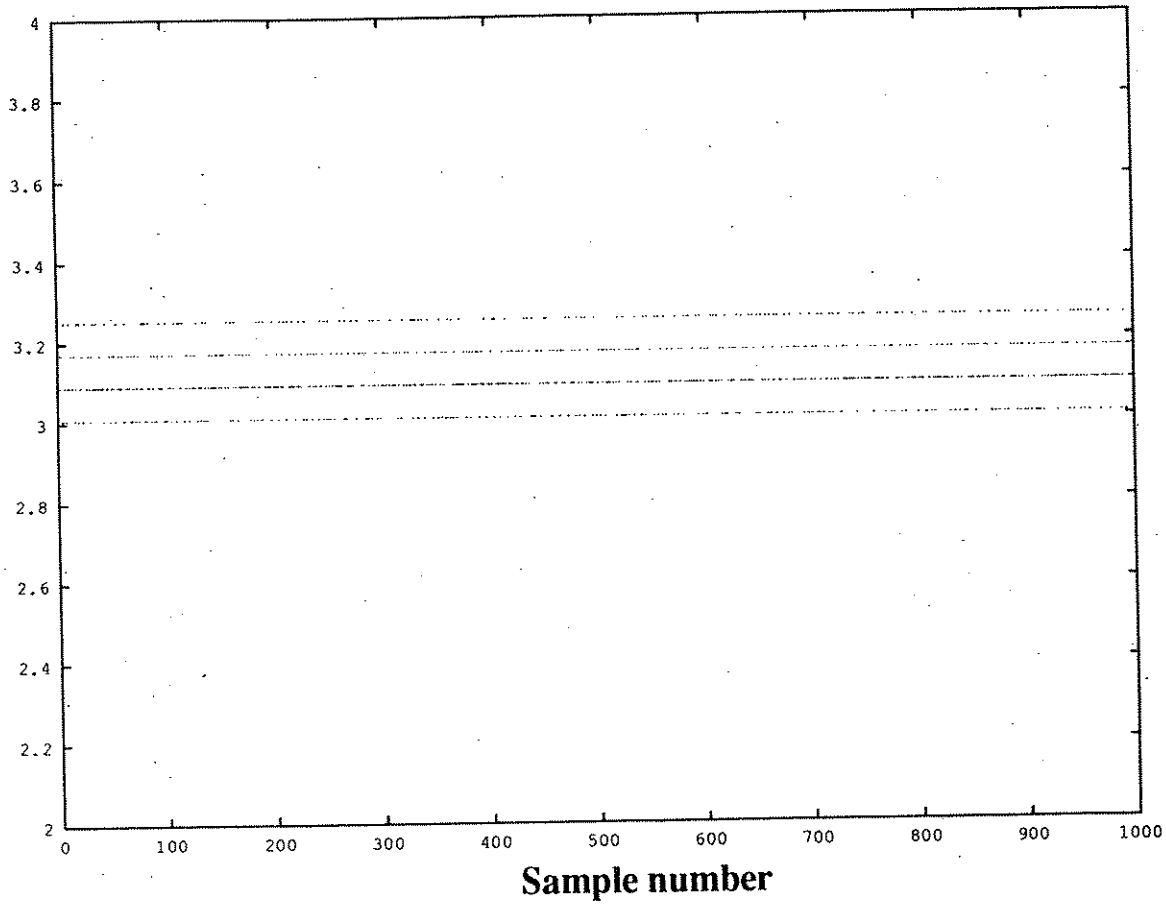
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.002 ms
99.9% threshold: 3.089 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 256 bytes



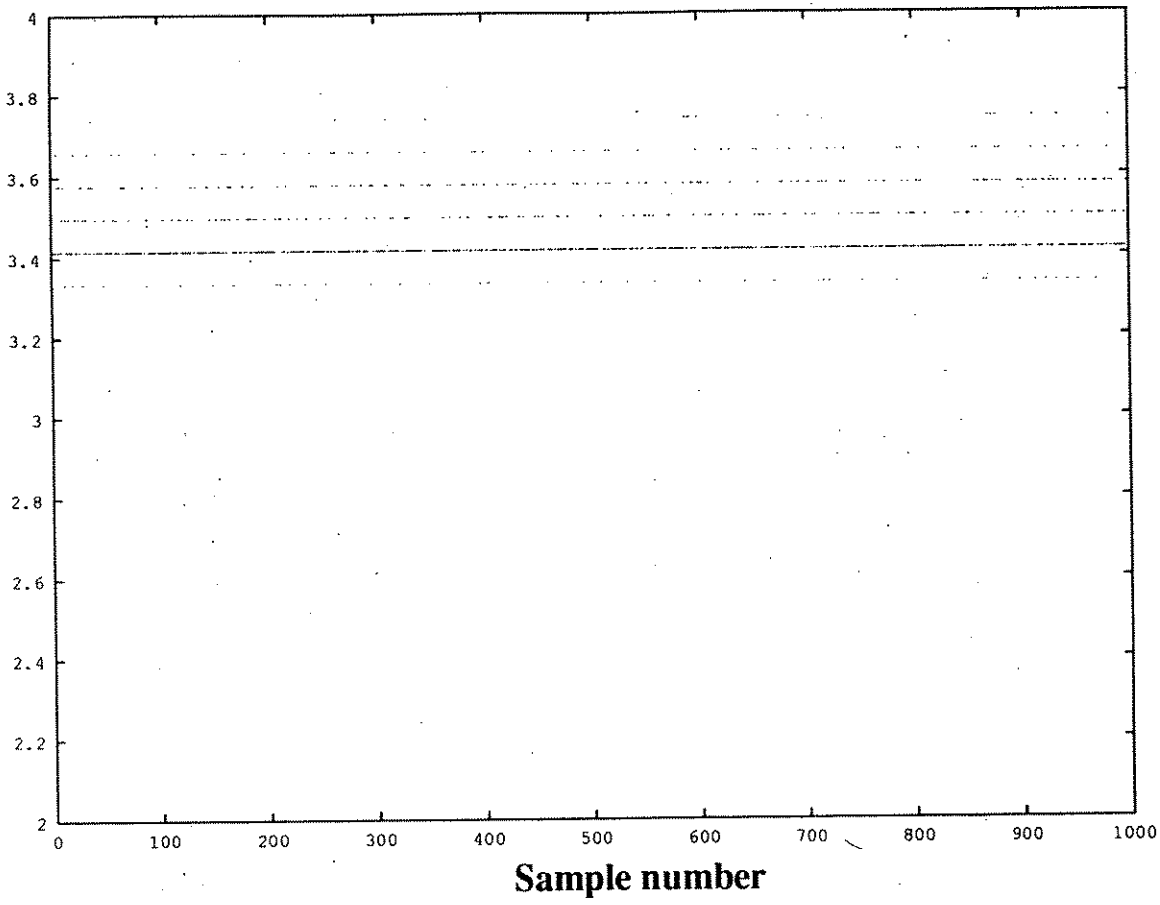
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.112 ms
99.9% threshold: 3.333 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 512 bytes



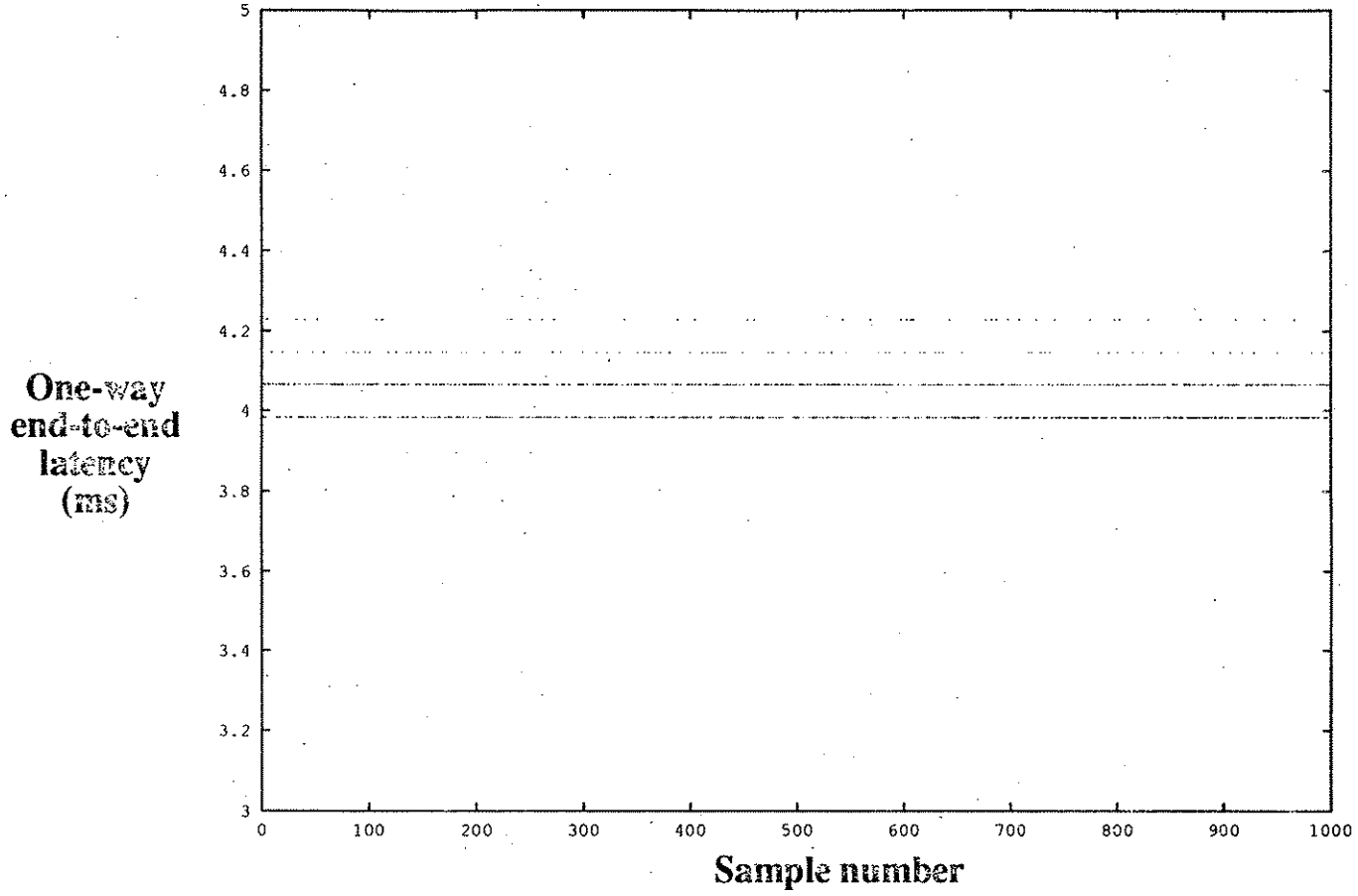
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.464 ms
99.9% threshold: 3.740 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



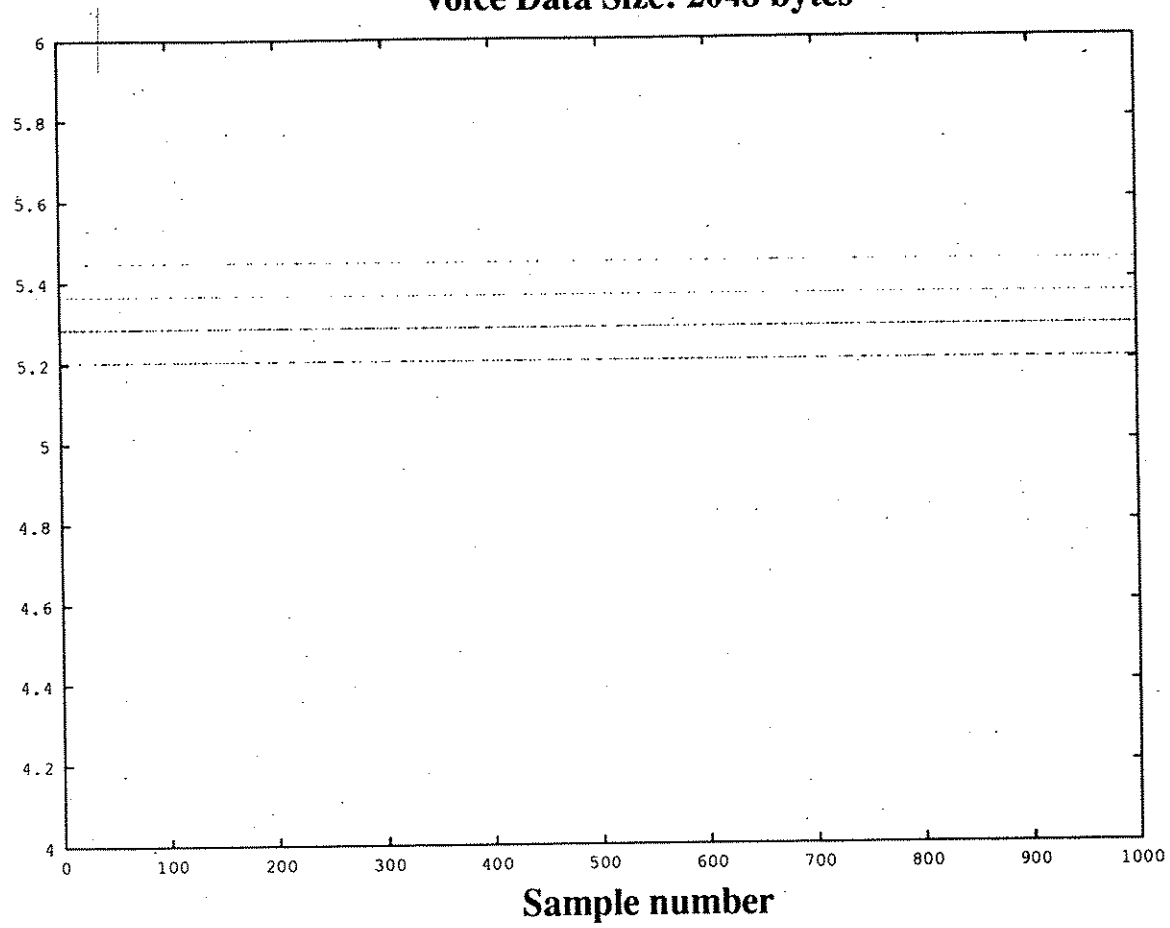
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 4.042 ms
99.9% threshold: 4.228 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



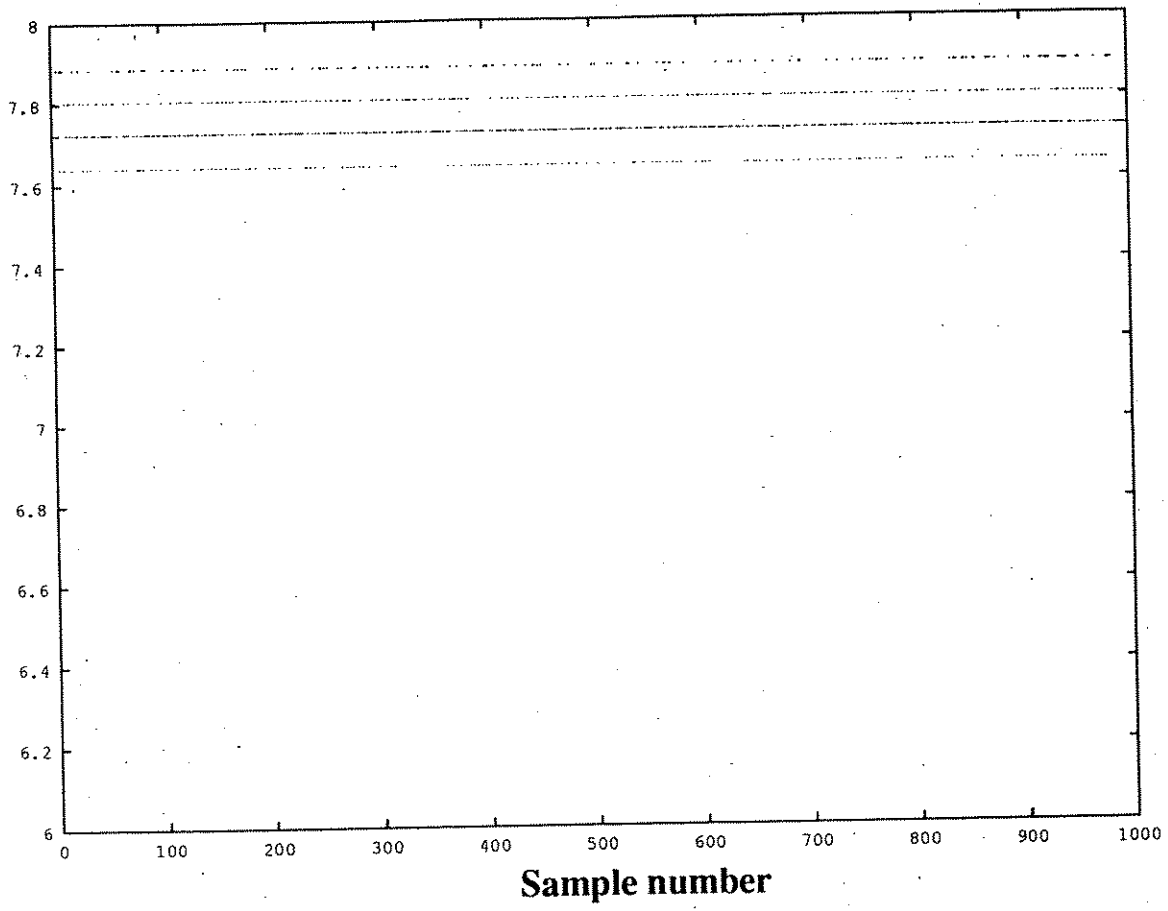
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 5.284 ms
99.9% threshold: 5.528 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



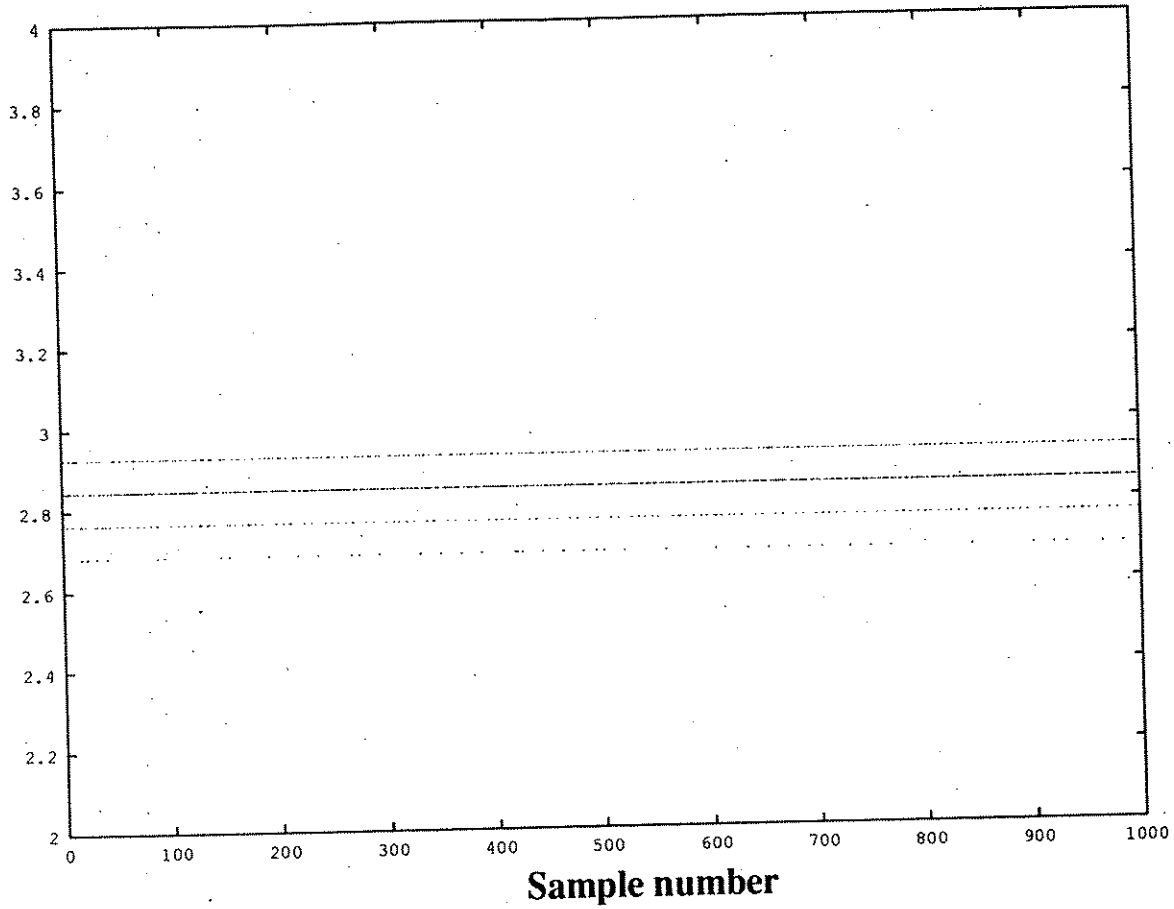
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 7.753 ms
99.9% threshold: 7.967 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 8 bytes



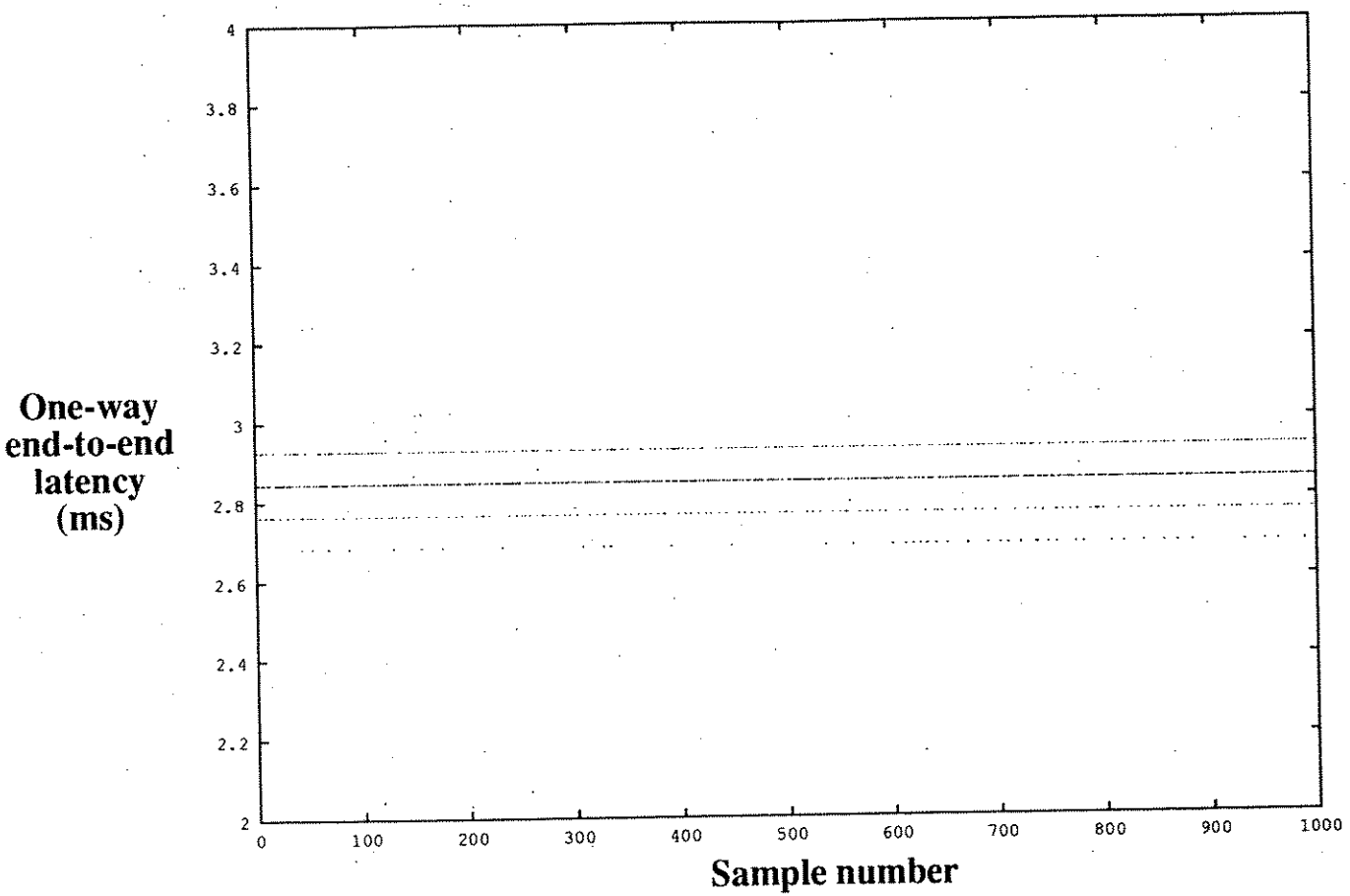
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 2.847 ms
99.9% threshold: 2.927 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 16 bytes



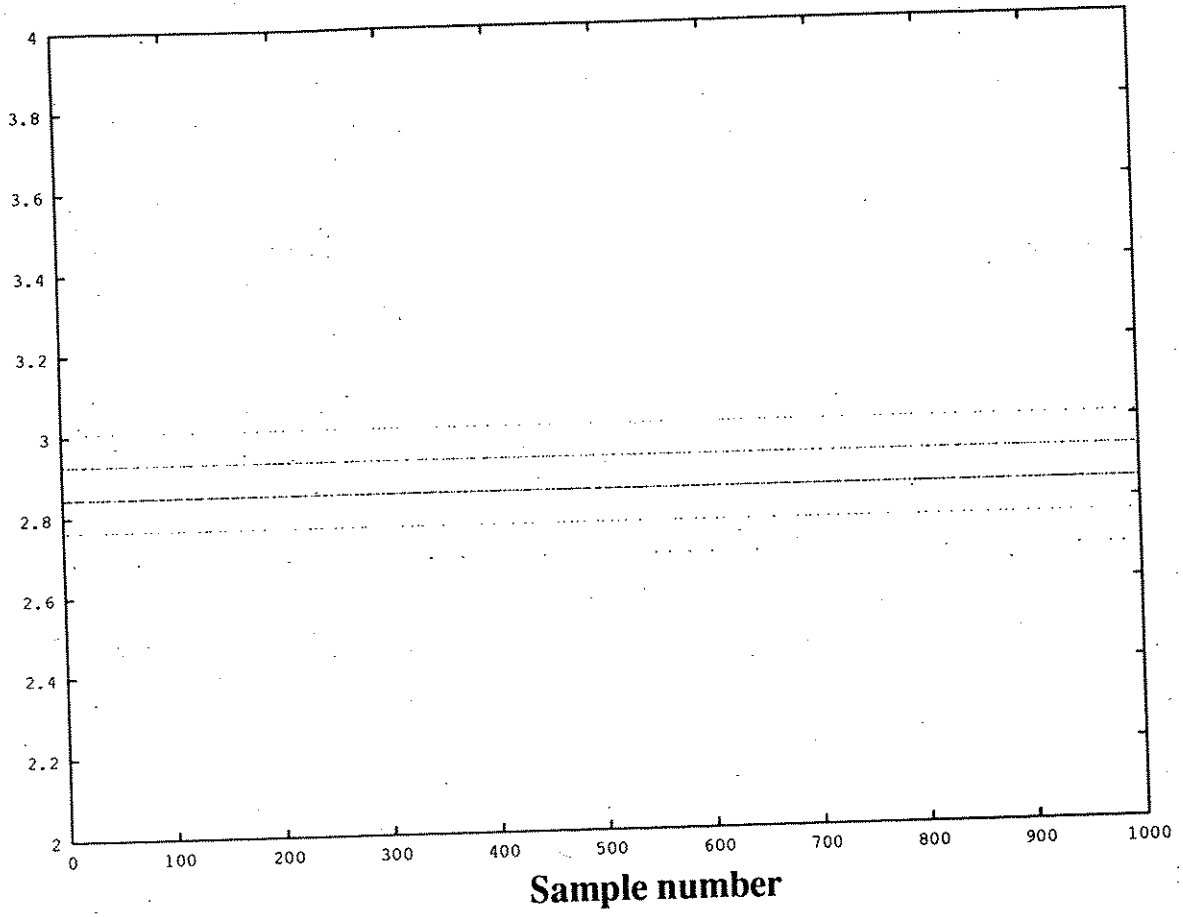
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 2.847 ms
99.9% threshold: 2.927 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 32 bytes



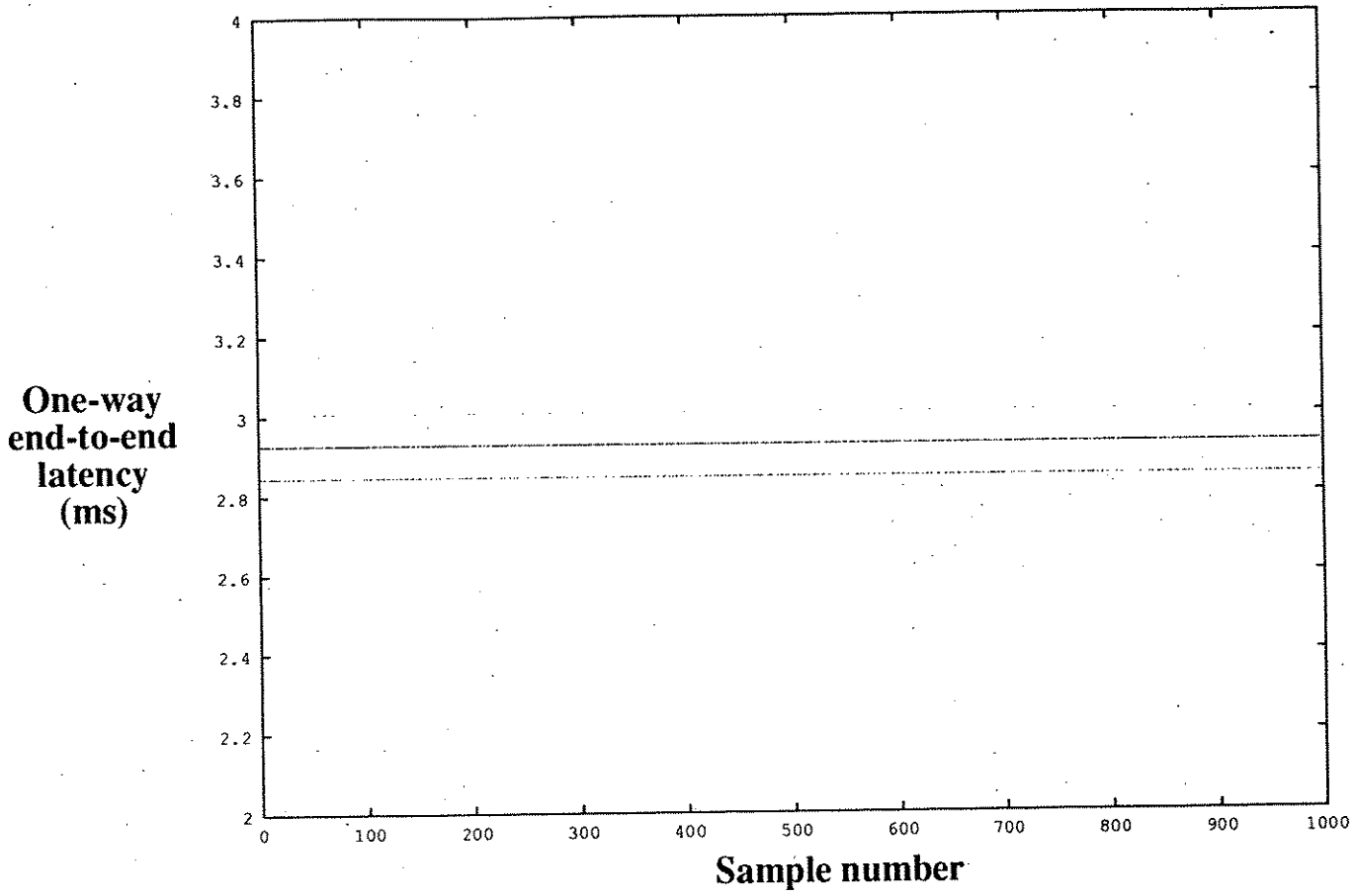
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 2.870 ms
99.9% threshold: 3.089 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 64 bytes



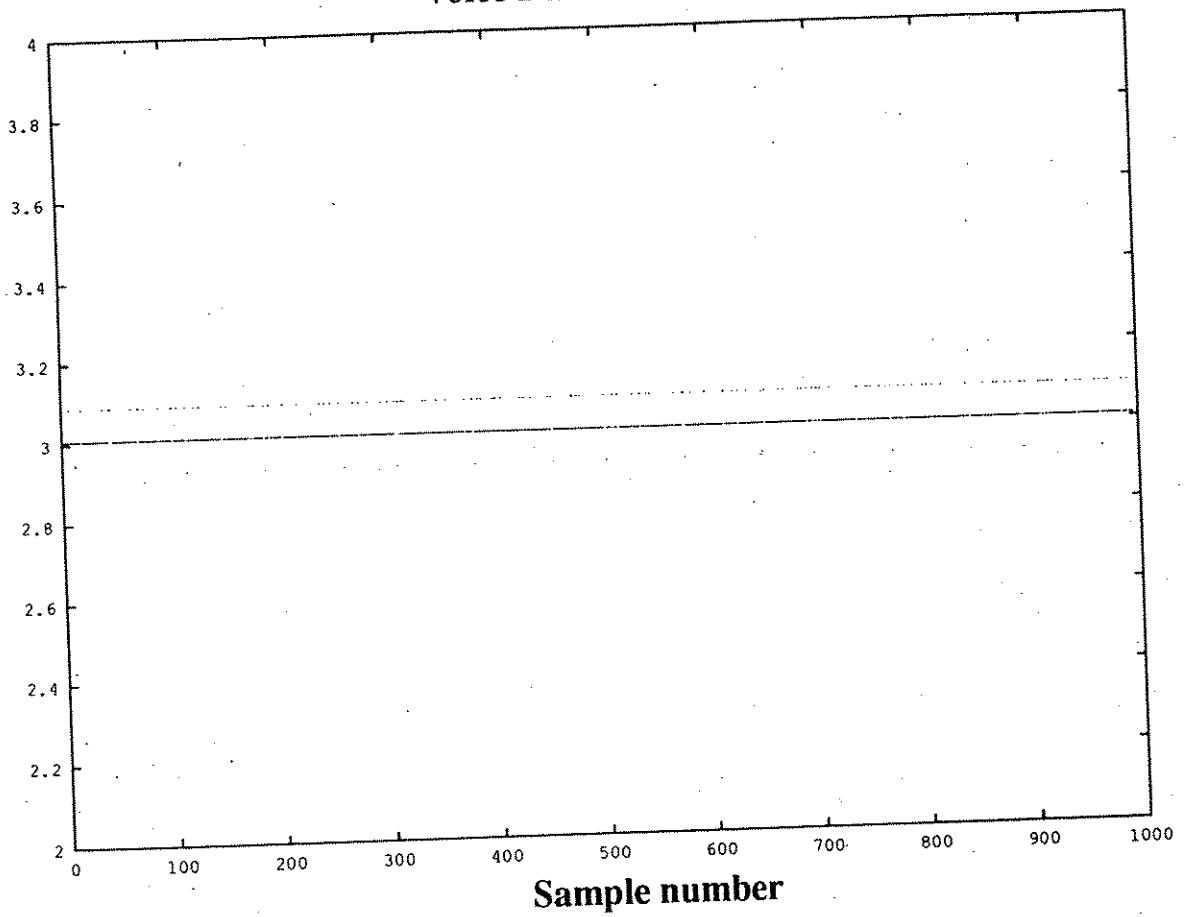
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 2.905 ms
99.9% threshold: 3.008 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 128 bytes



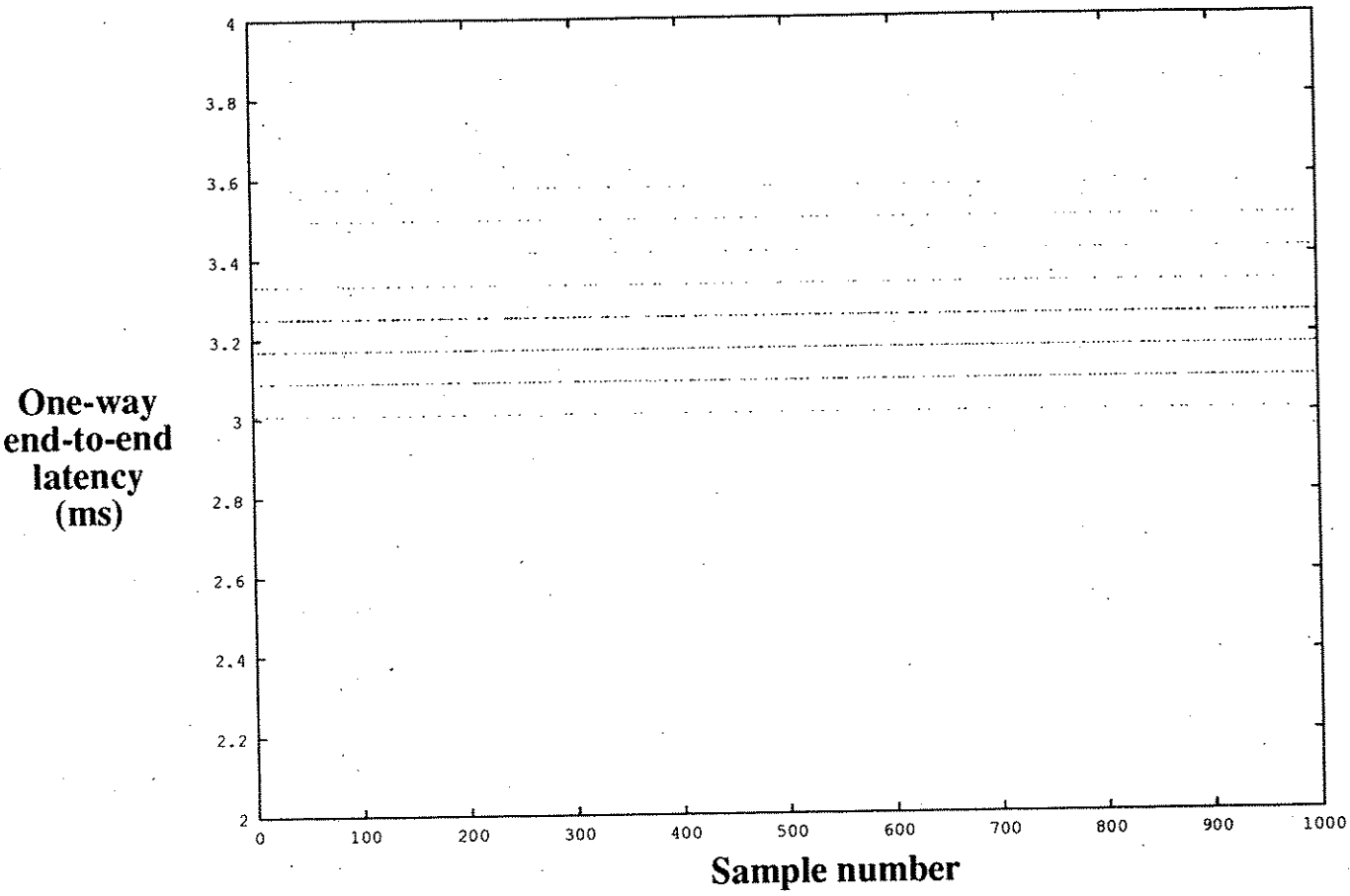
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.018 ms
99.9% threshold: 3.089 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 256 bytes



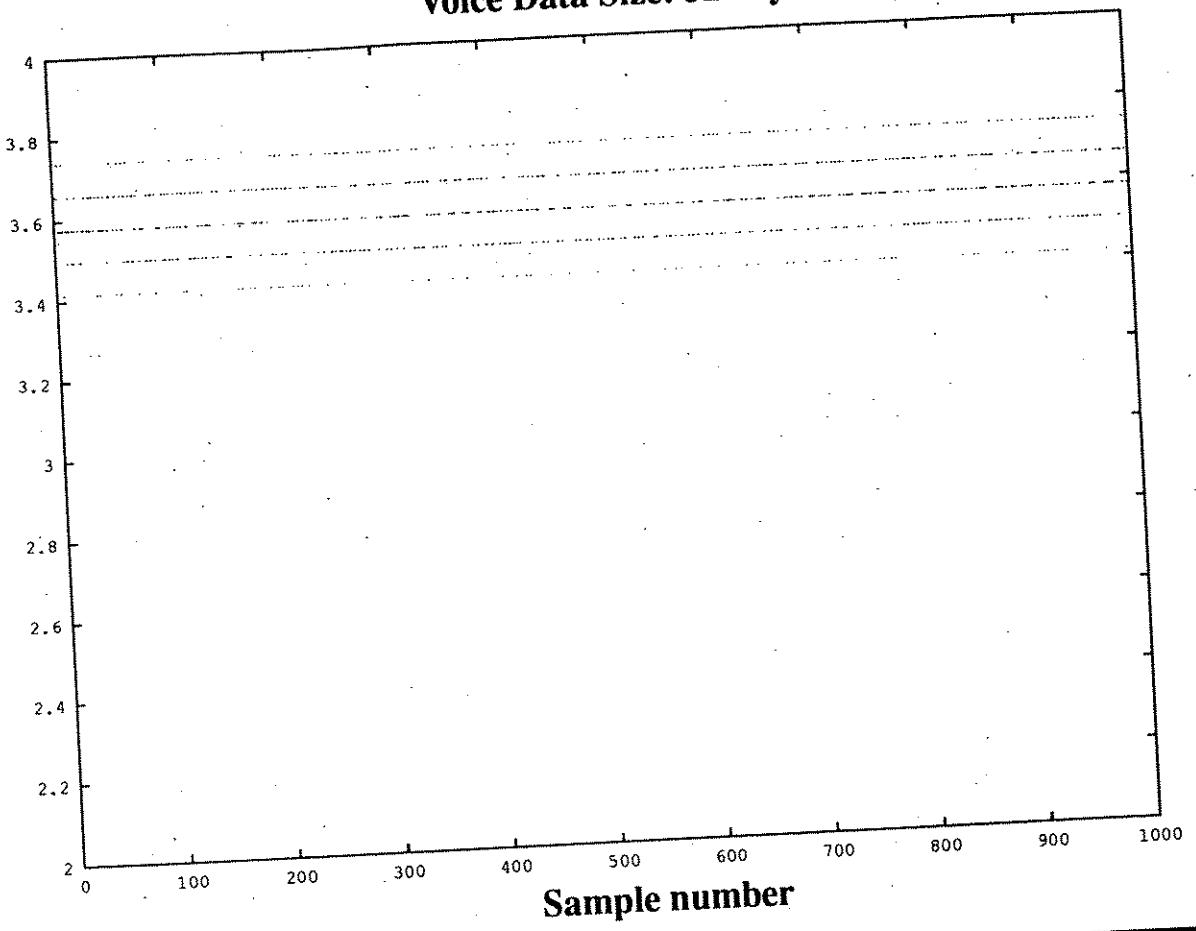
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.209 ms
99.9% threshold: 3.659 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 512 bytes



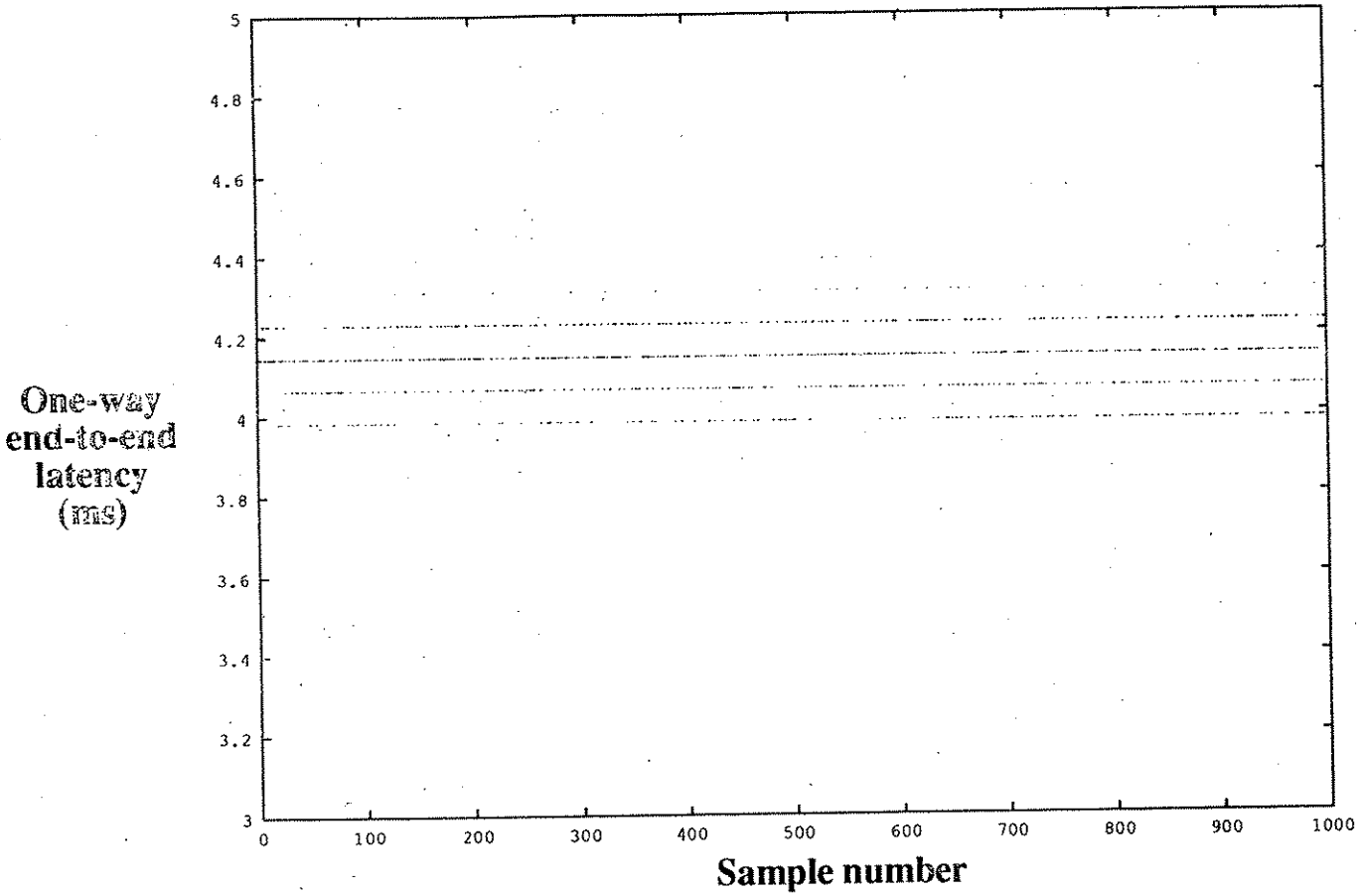
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.587 ms
99.9% threshold: 3.821 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



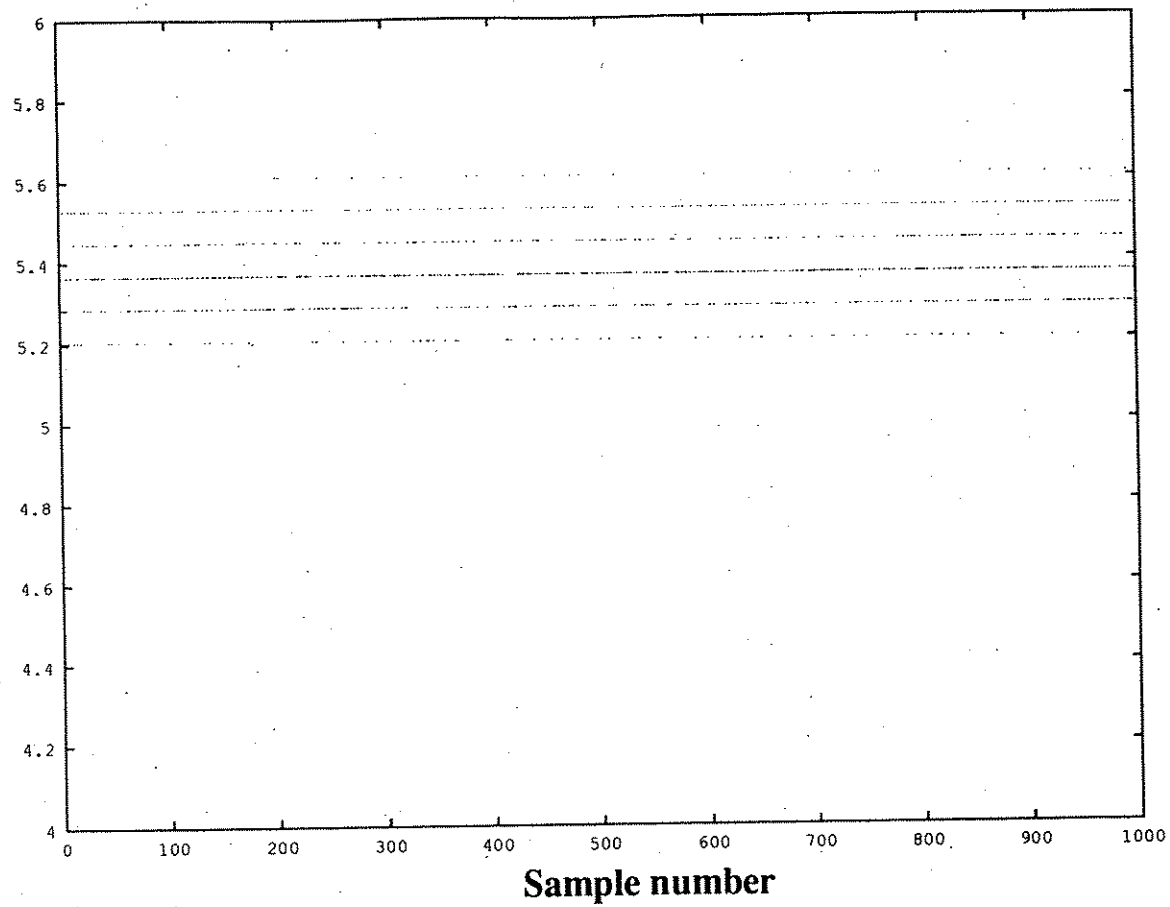
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 4.127 ms
99.9% threshold: 4.390 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



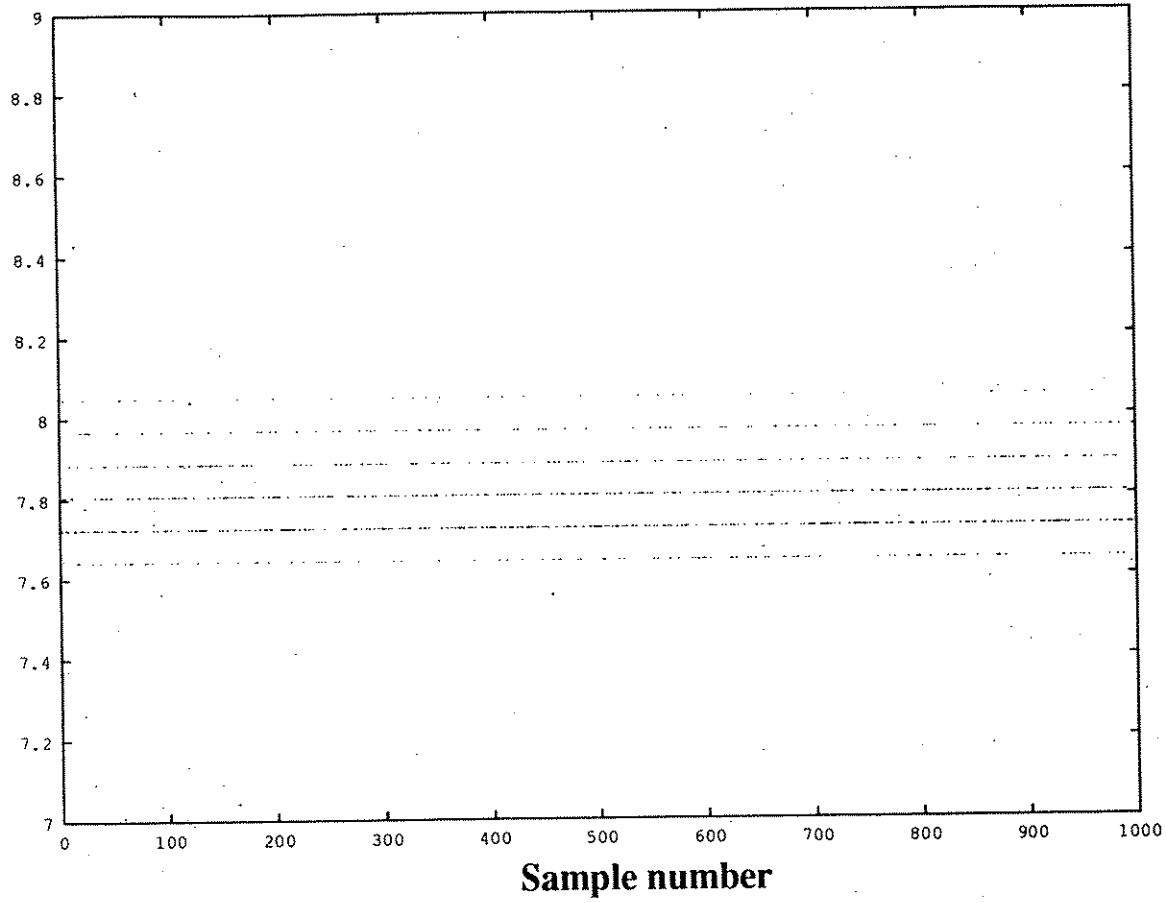
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 5.381 ms
99.9% threshold: 5.610 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



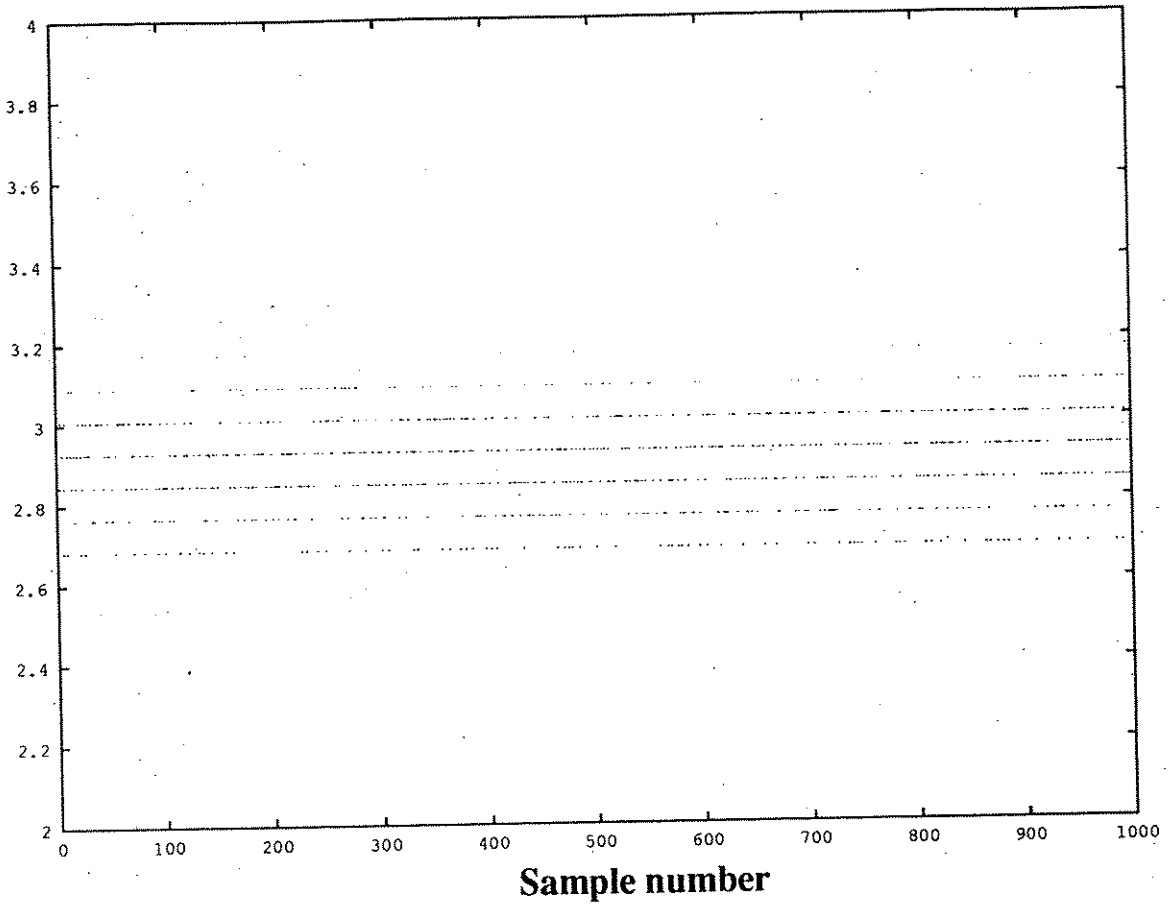
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 7.798 ms
99.9% threshold: 8.049 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 8 bytes



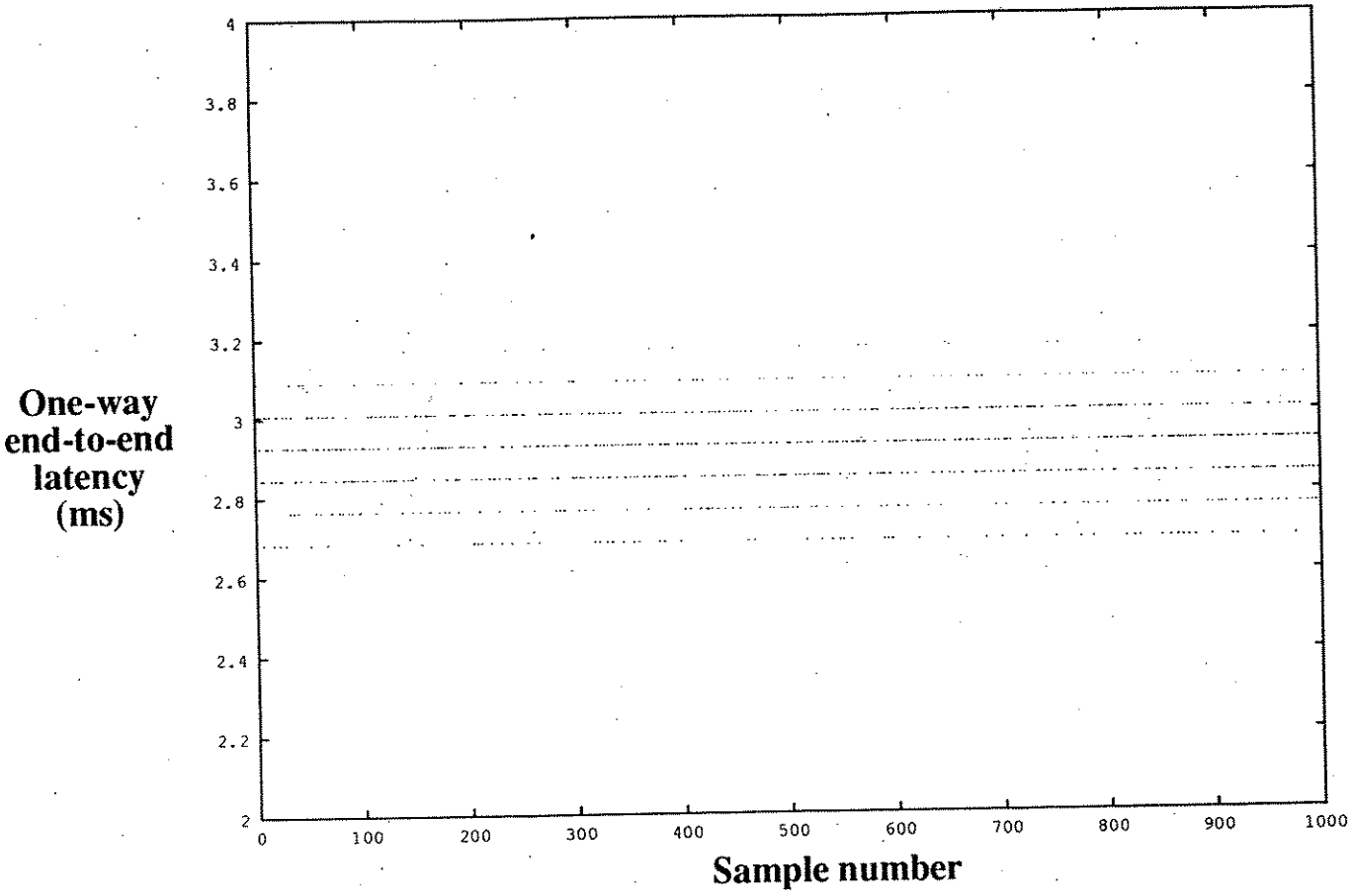
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 2.900 ms
99.9% threshold: 3.171 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 16 bytes



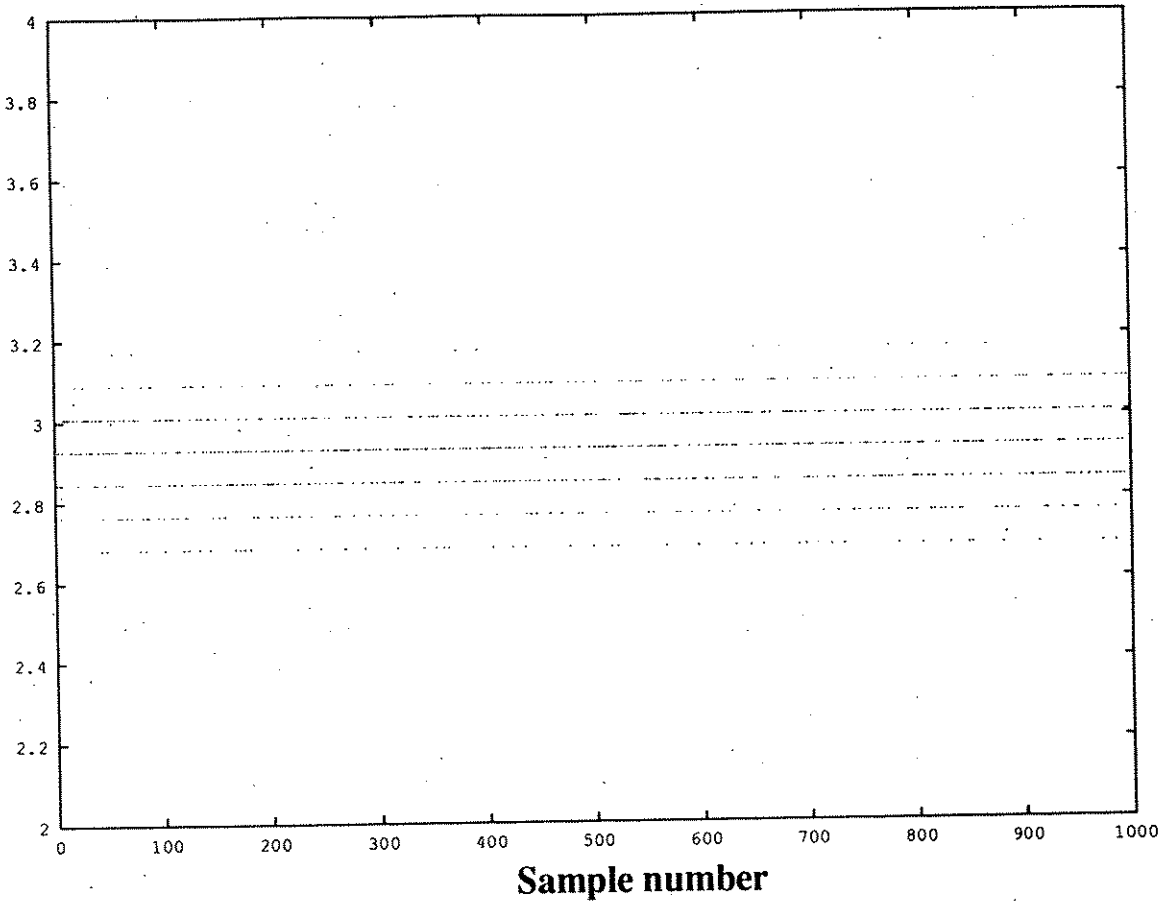
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 2.911 ms
99.9% threshold: 3.171 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 32 bytes



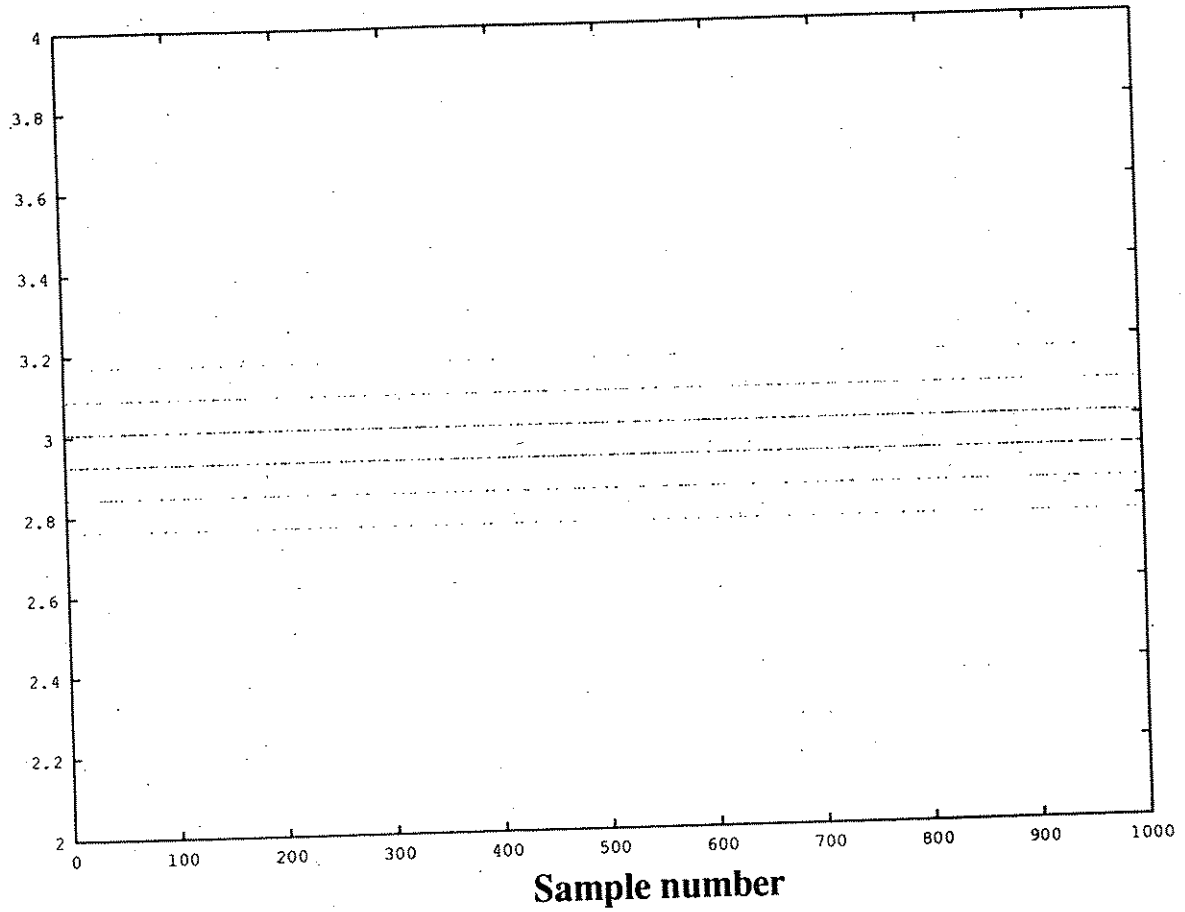
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 2.921 ms
99.9% threshold: 3.171 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 64 bytes



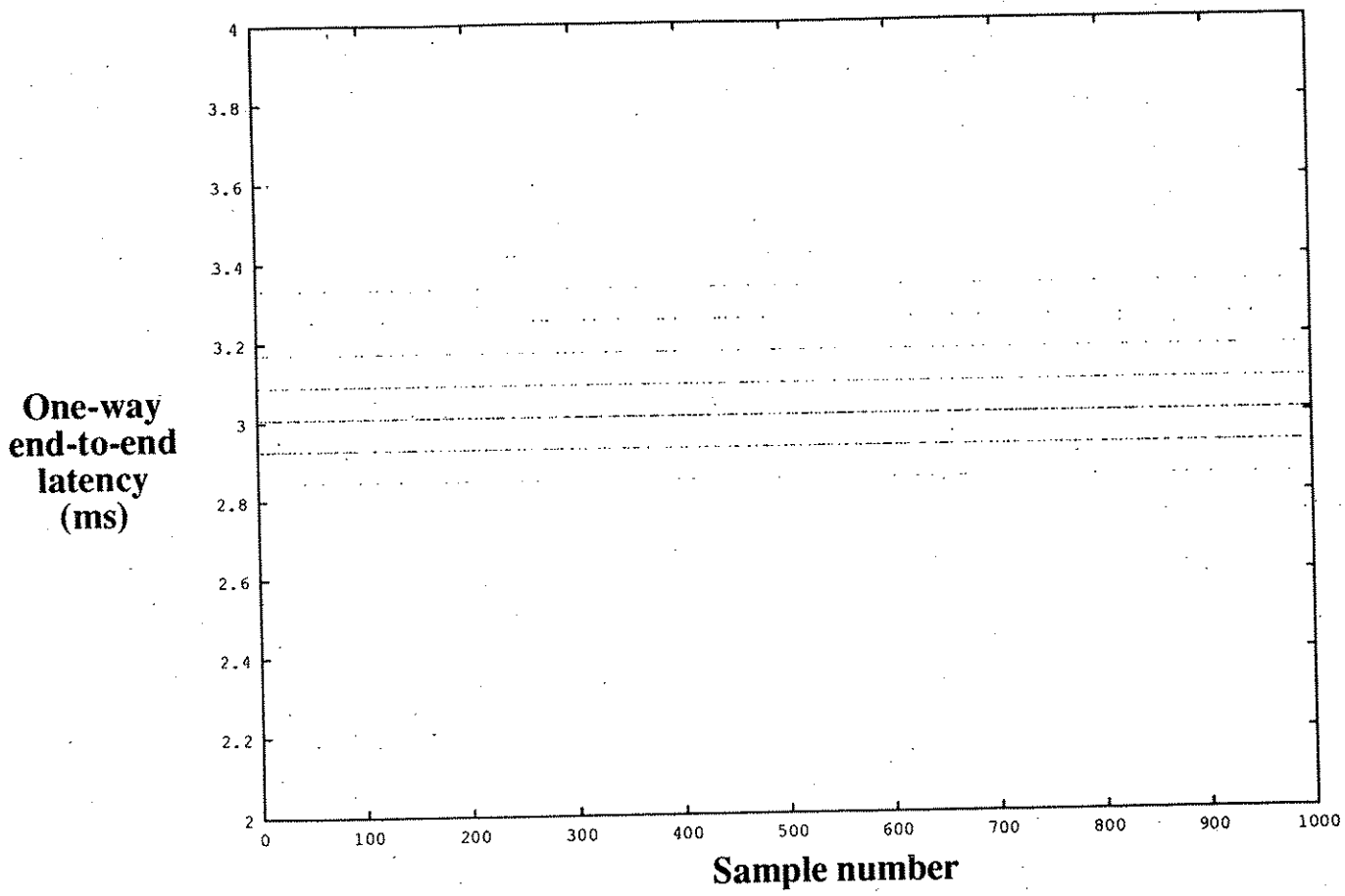
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 2.963 ms
99.9% threshold: 3.252 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 128 bytes



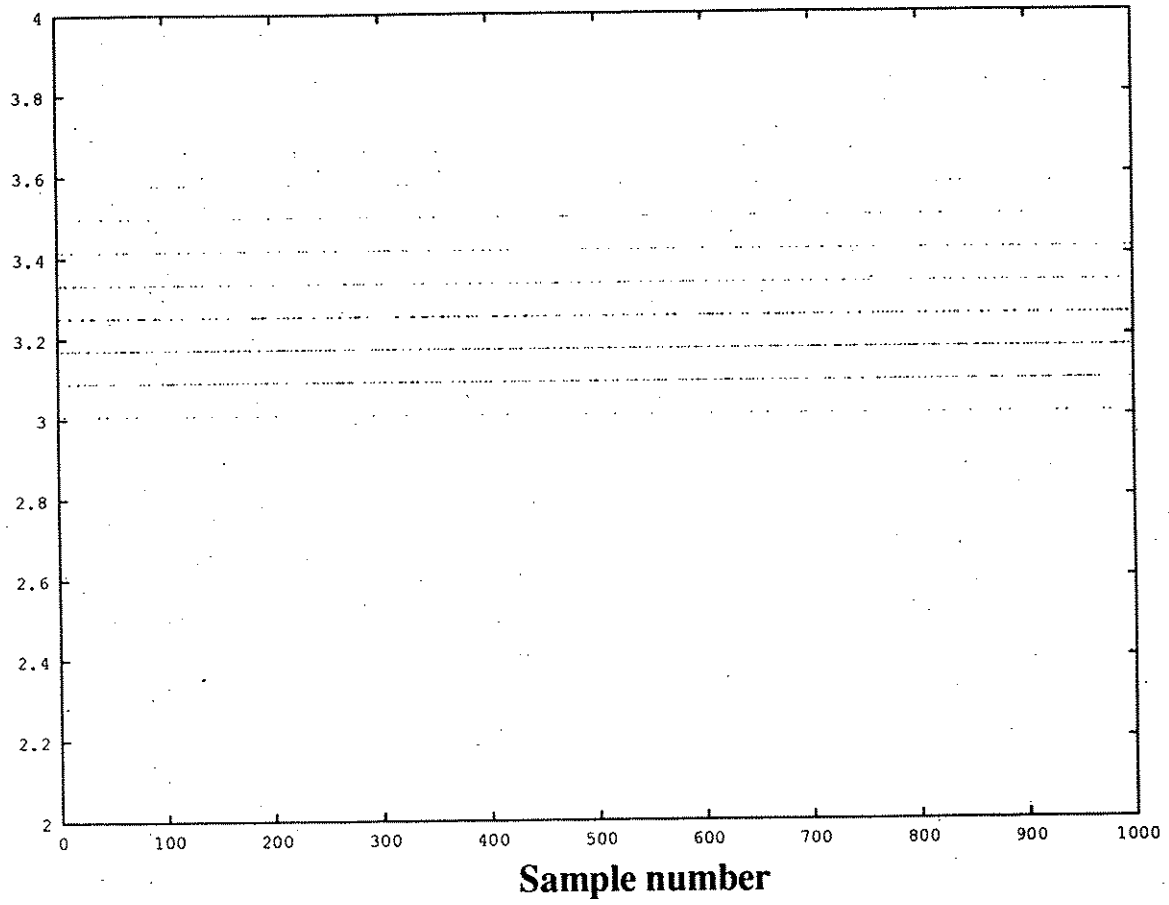
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.028 ms
99.9% threshold: 3.496 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 256 bytes



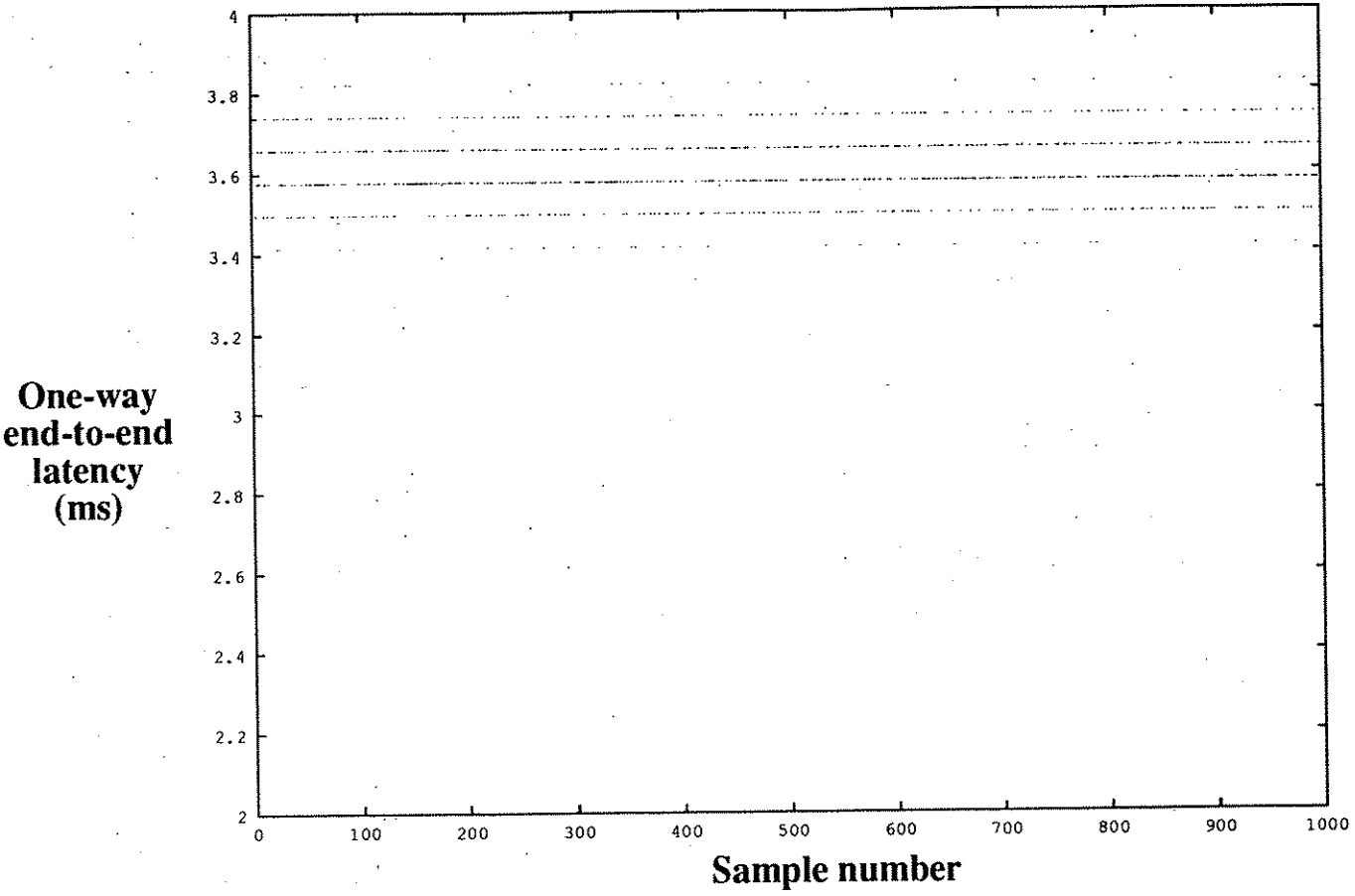
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.219 ms
99.9% threshold: 3.659 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 512 bytes



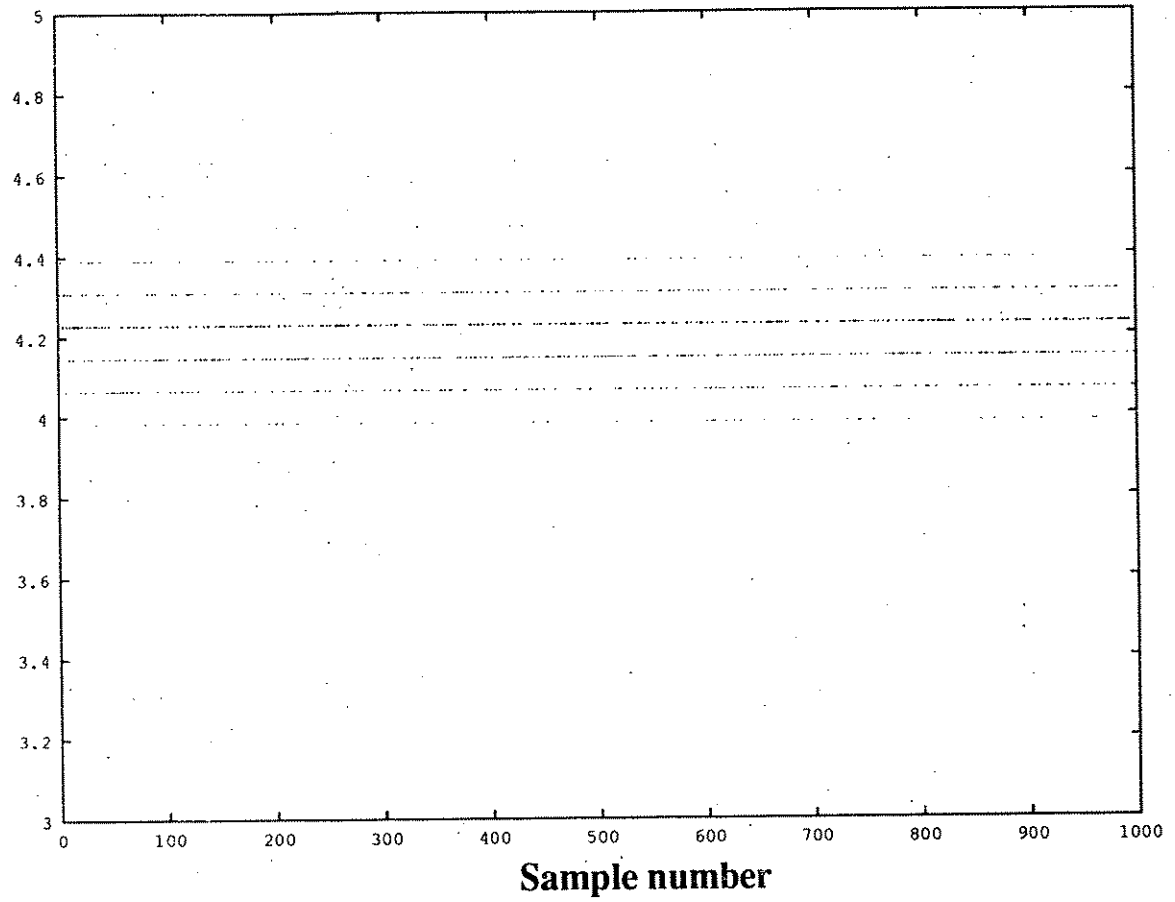
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.609 ms
99.9% threshold: 3.821 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



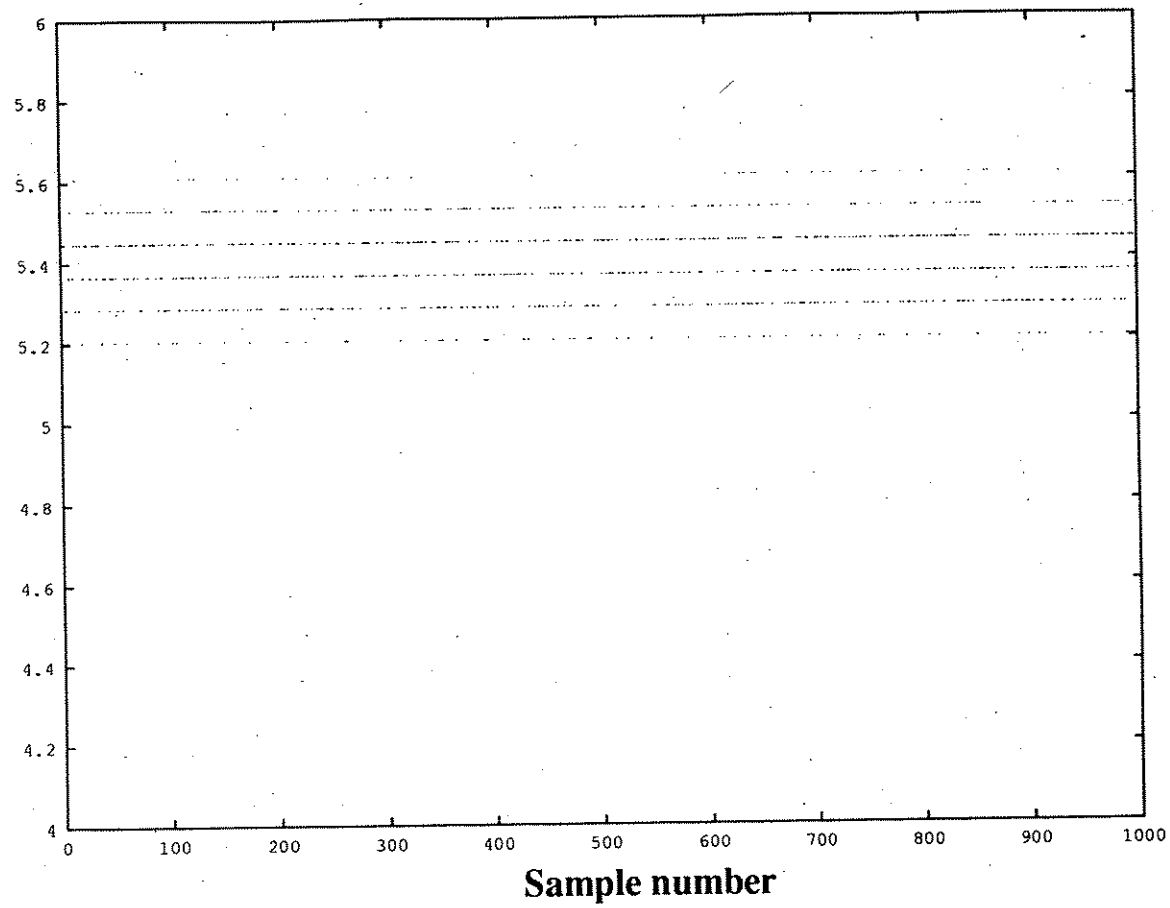
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 4.198 ms
99.9% threshold: 4.634 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



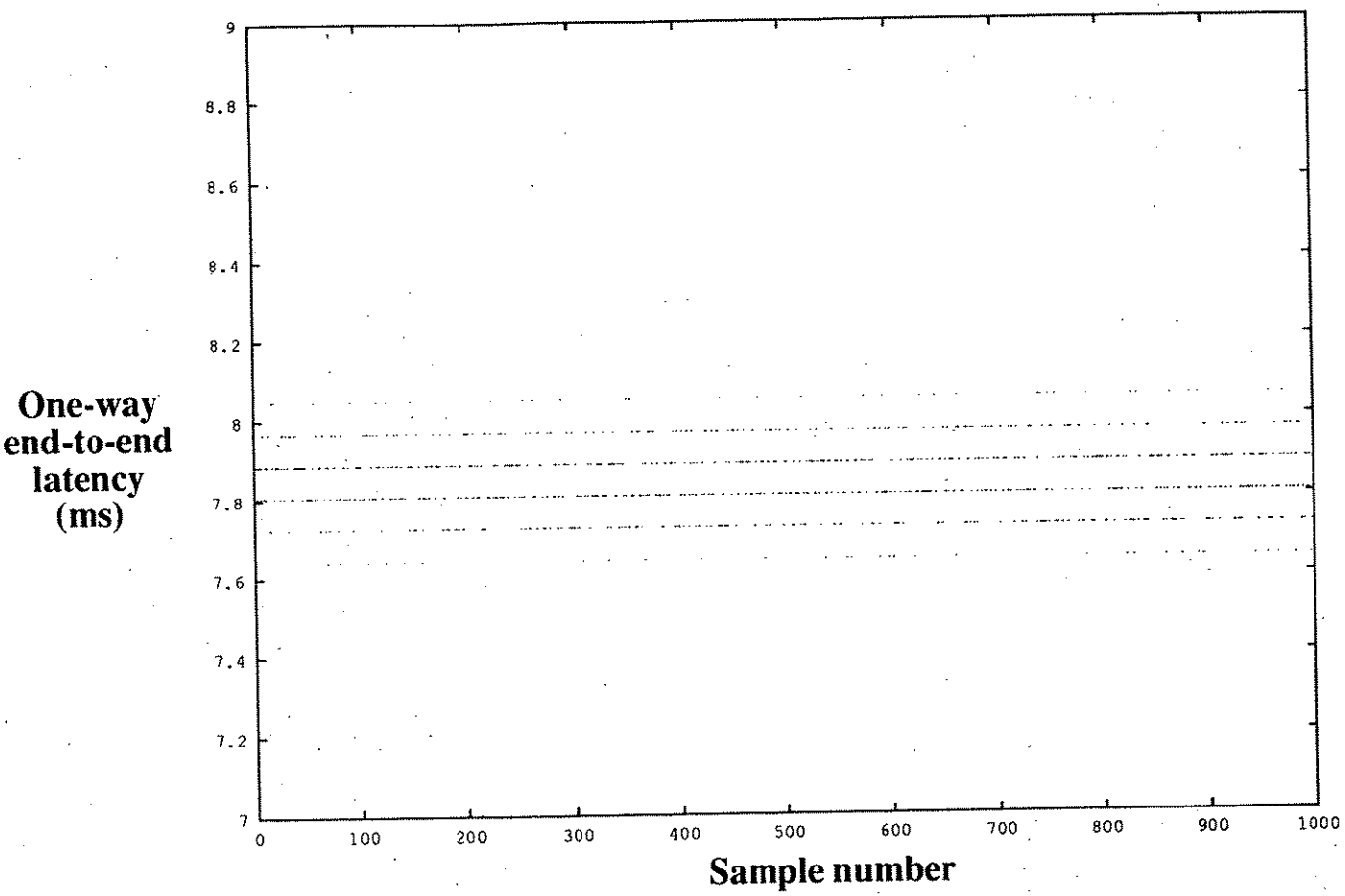
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 5.395 ms
99.9% threshold: 5.772 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



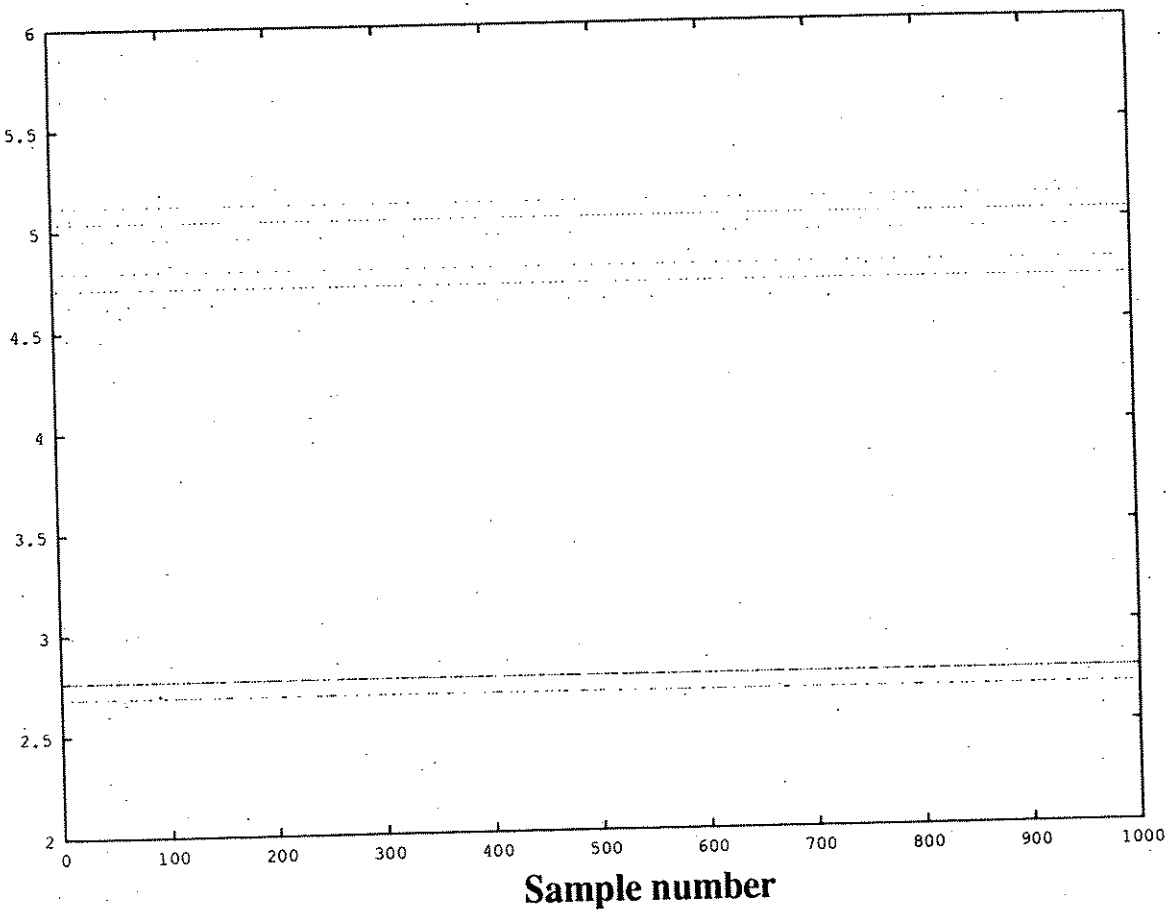
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 7.844 ms
99.9% threshold: 8.221 ms

SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 16 bytes



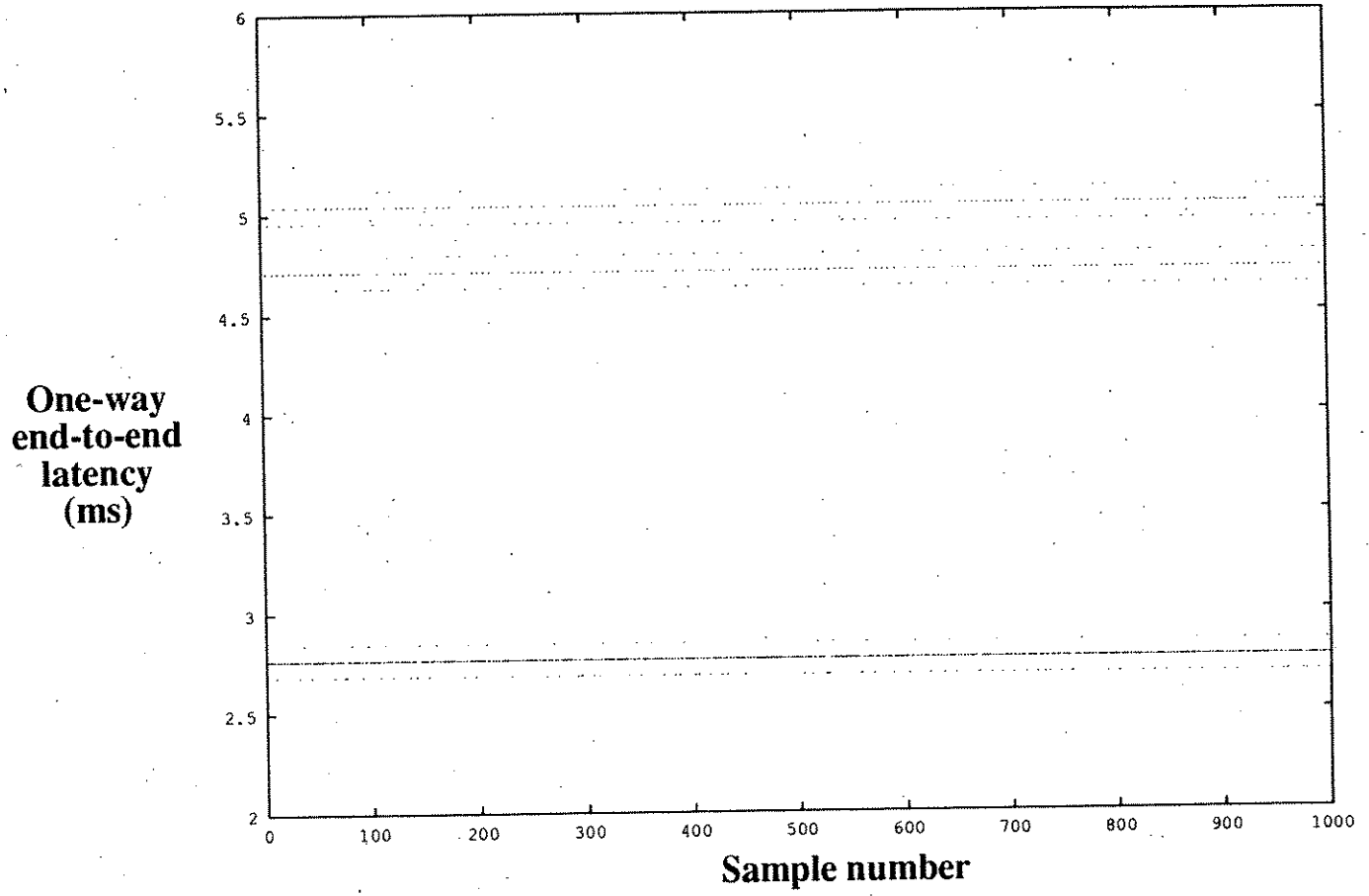
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 3.522 ms
99.9% threshold: 5.122 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 32 bytes



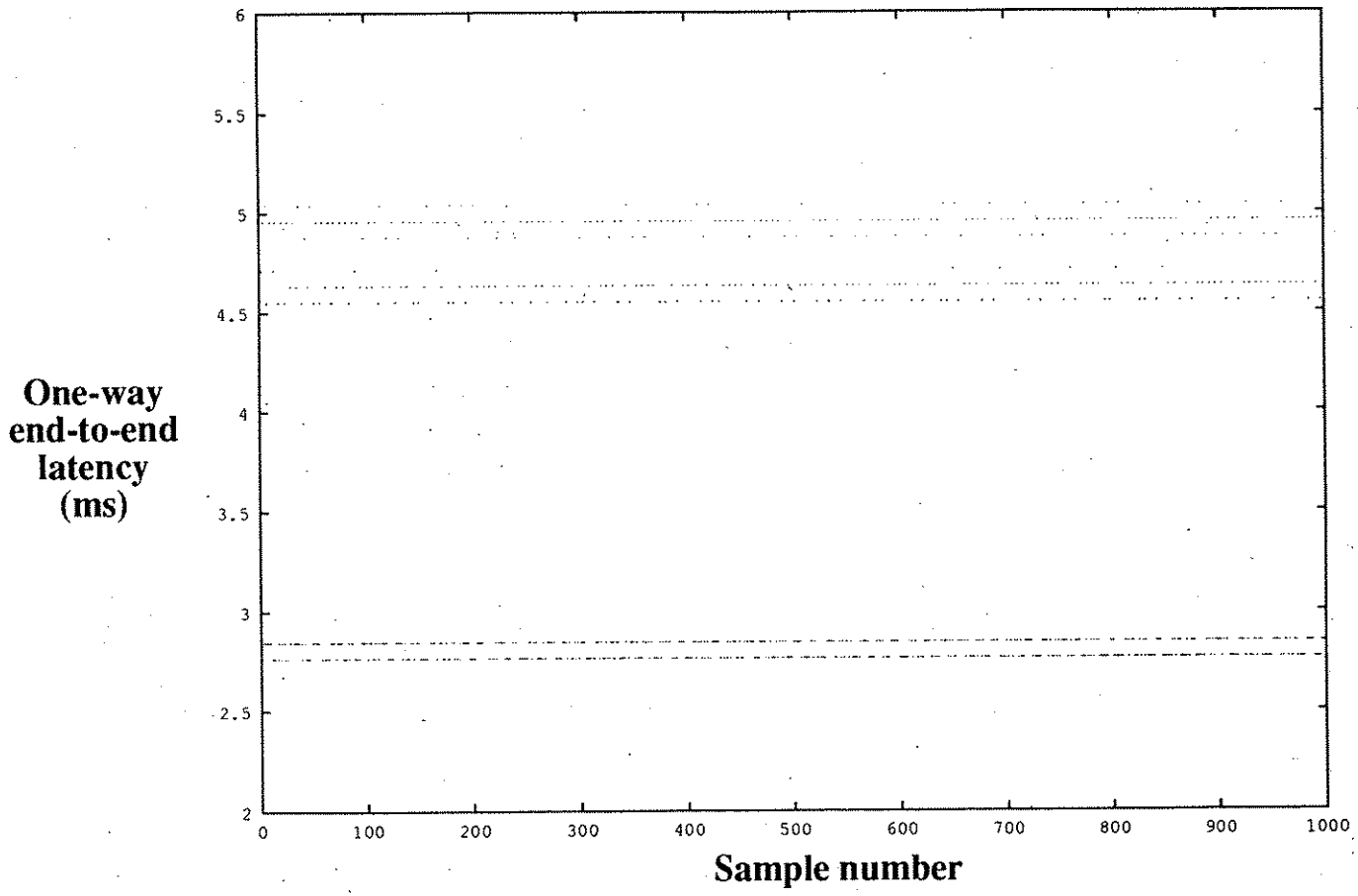
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 3.525 ms
99.9% threshold: 5.122 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 64 bytes



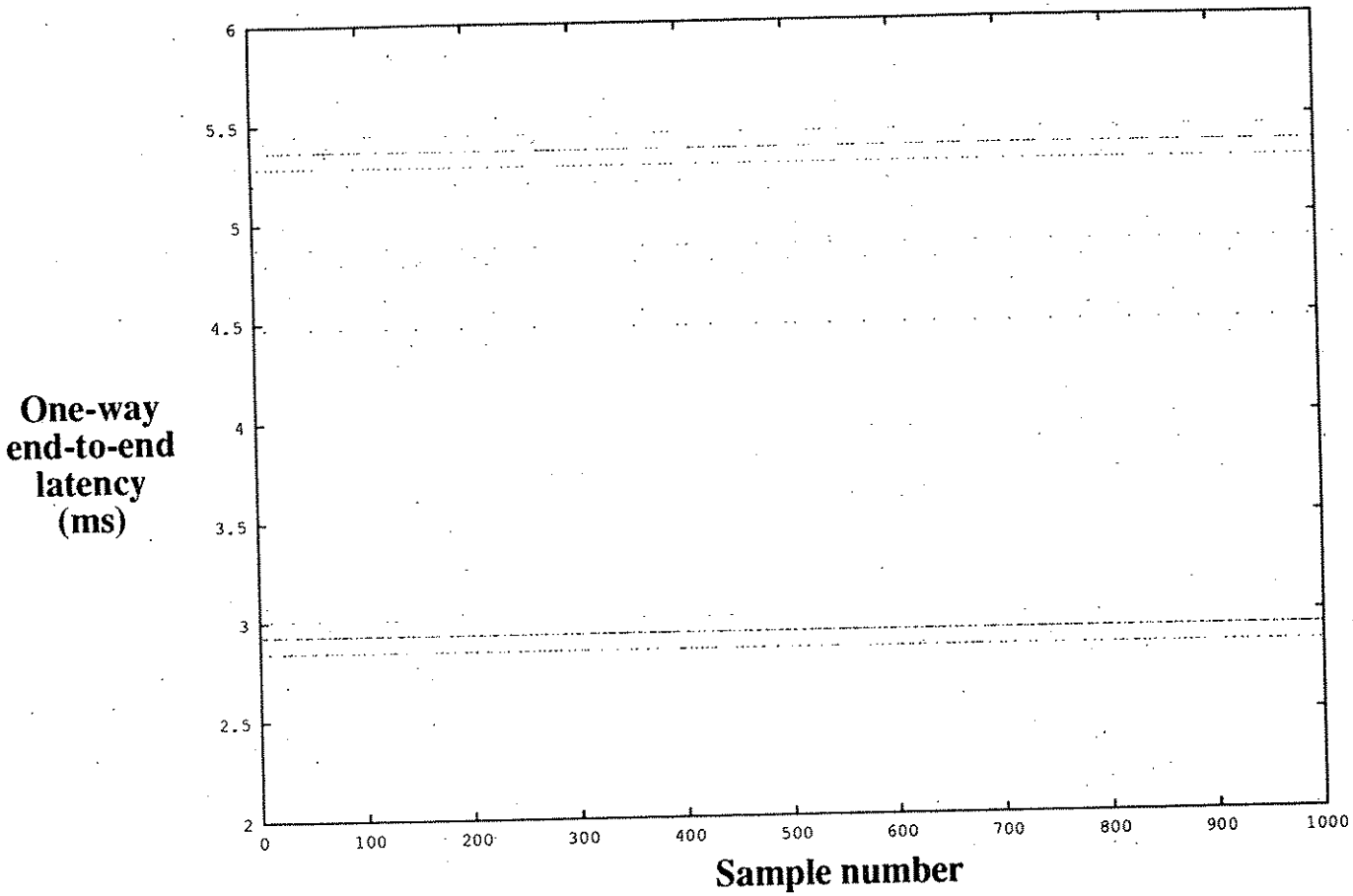
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 3.522 ms
99.9% threshold: 5.041 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 128 bytes



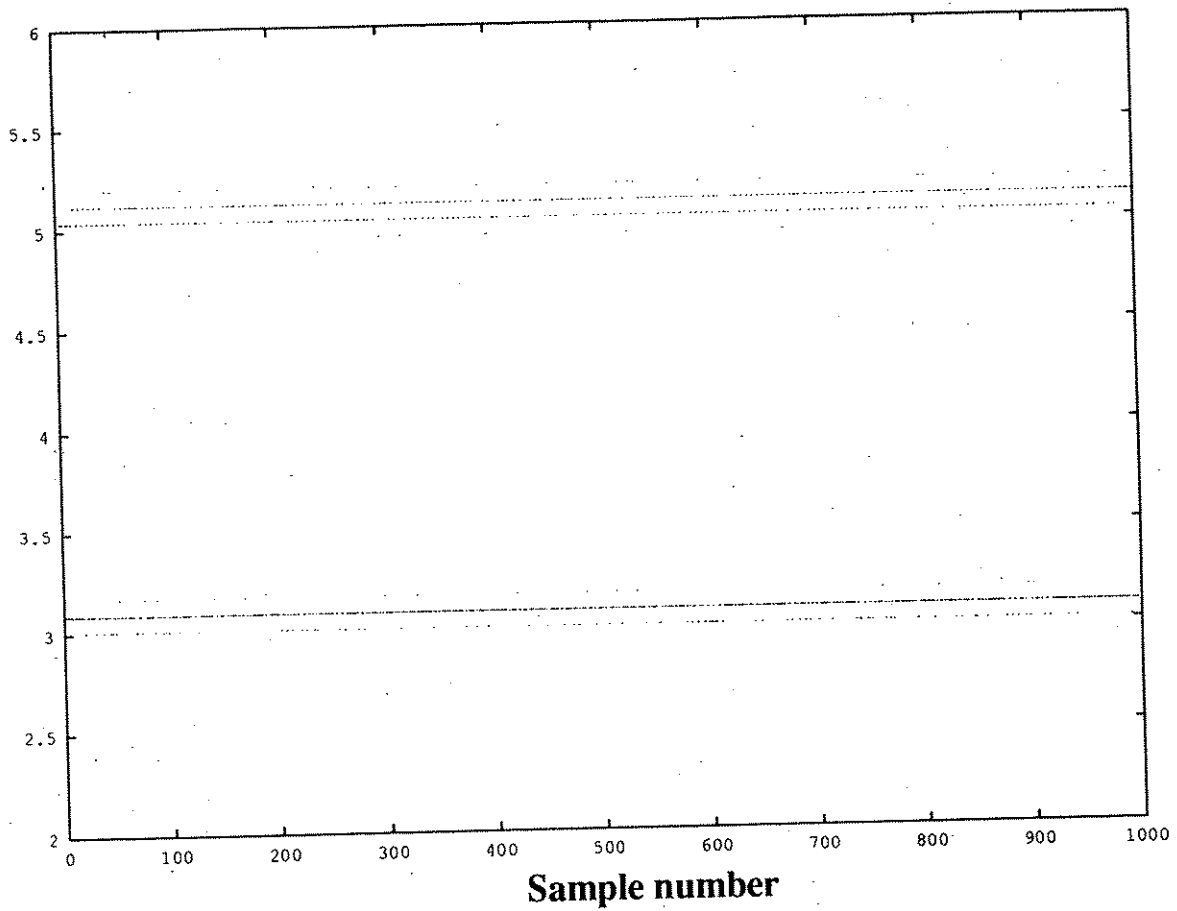
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 3.816 ms
99.9% threshold: 5.447 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 256 bytes



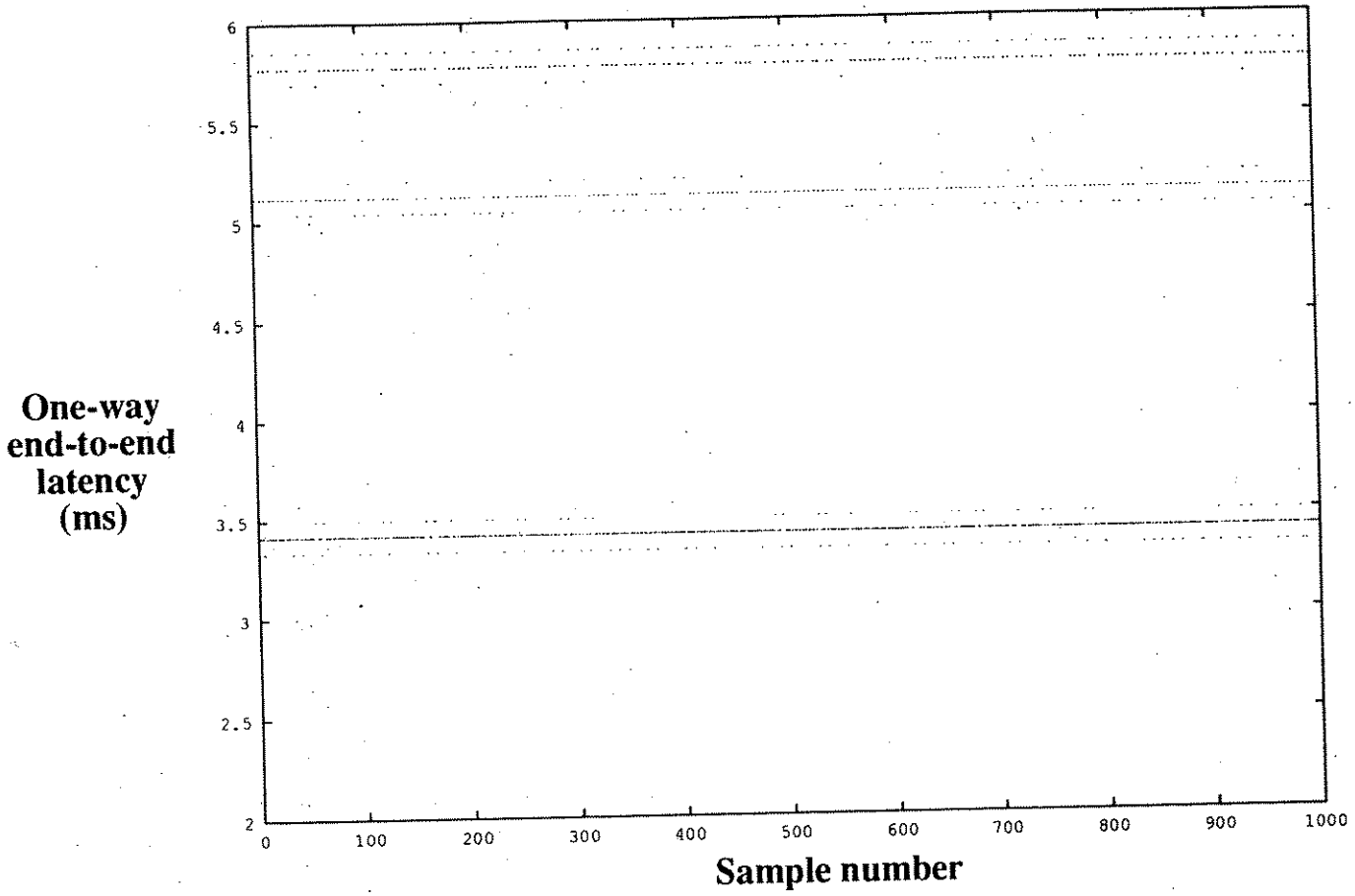
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 3.884 ms
99.9% threshold: 5.203 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 512 bytes



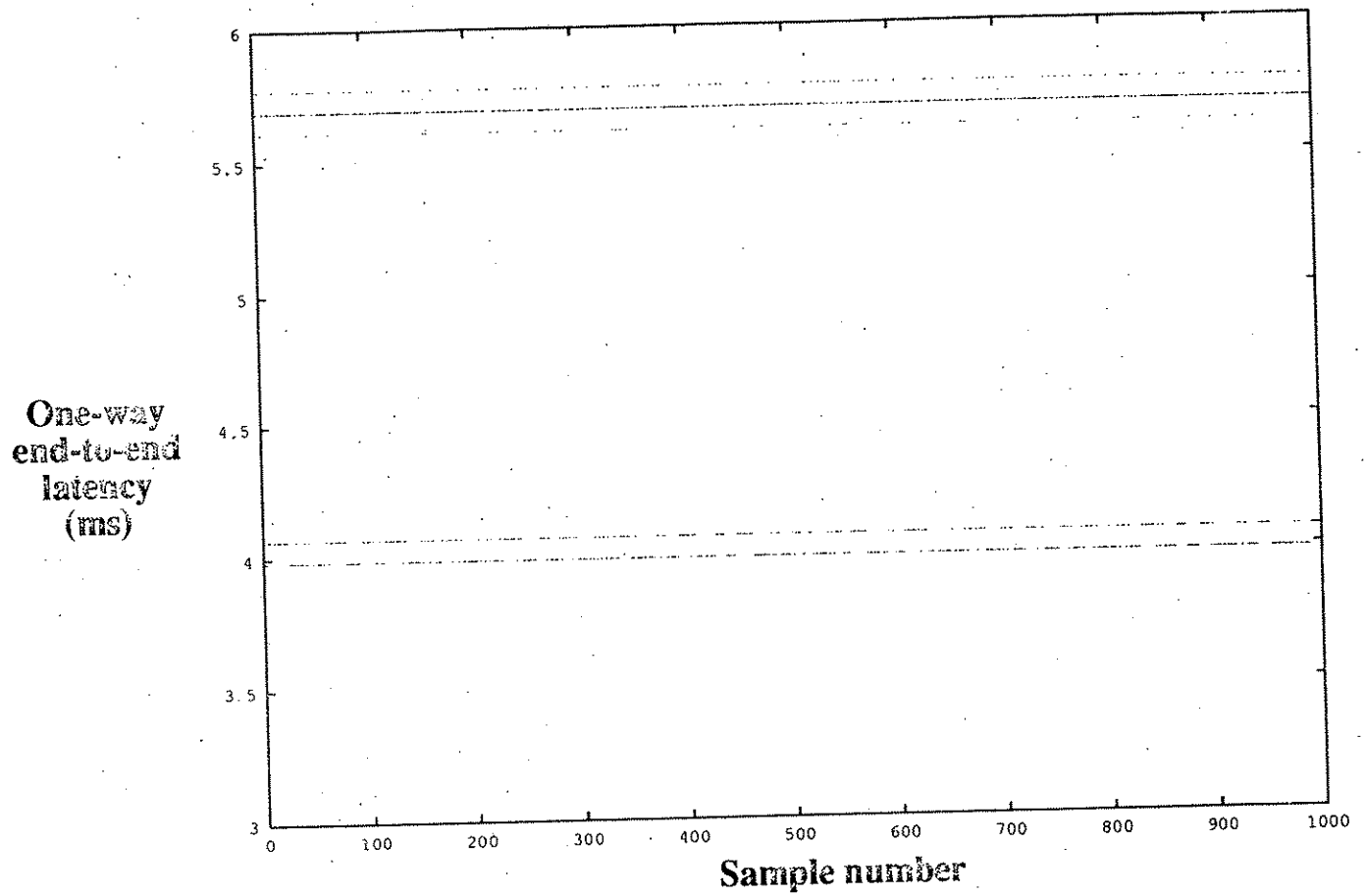
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 4.319 ms
99.9% threshold: 5.854 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



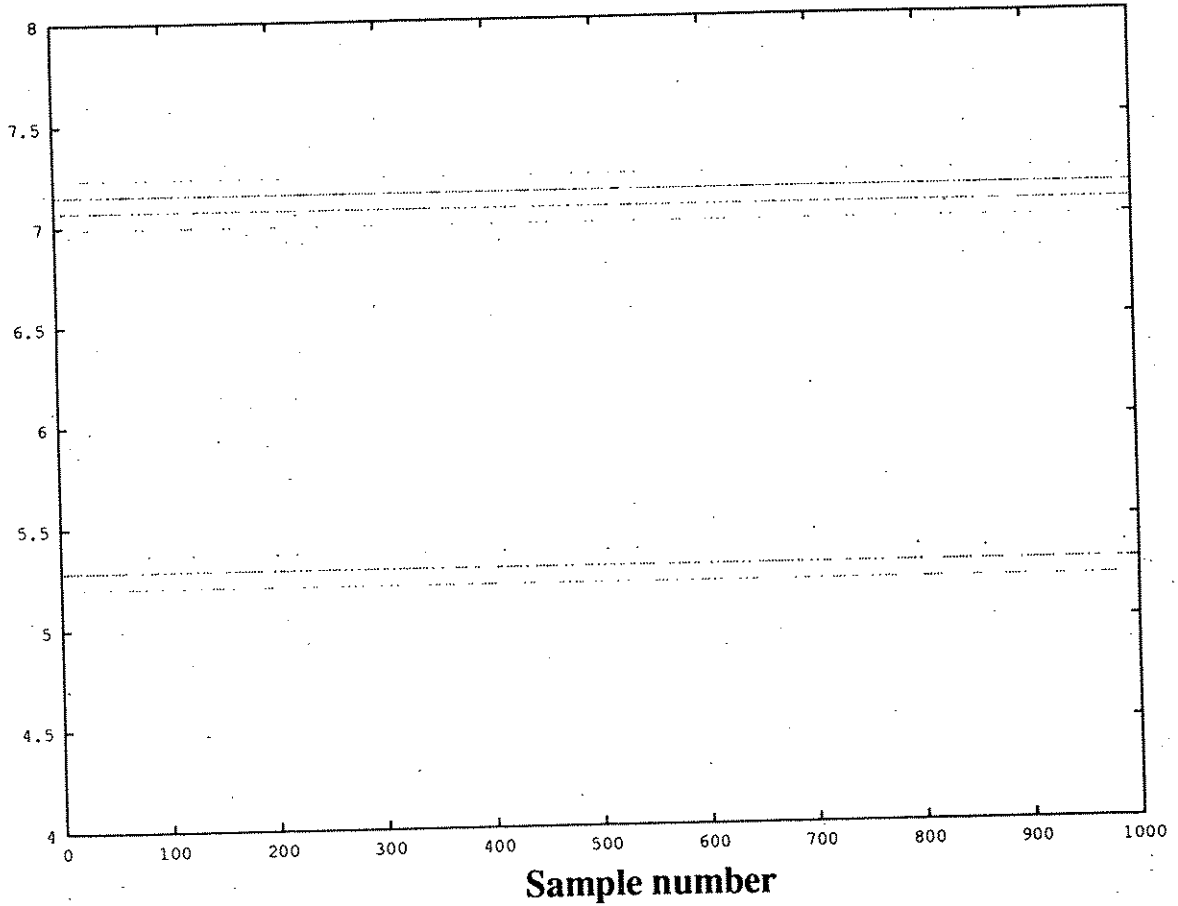
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 4.861 ms
99.9% threshold: 5.772 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



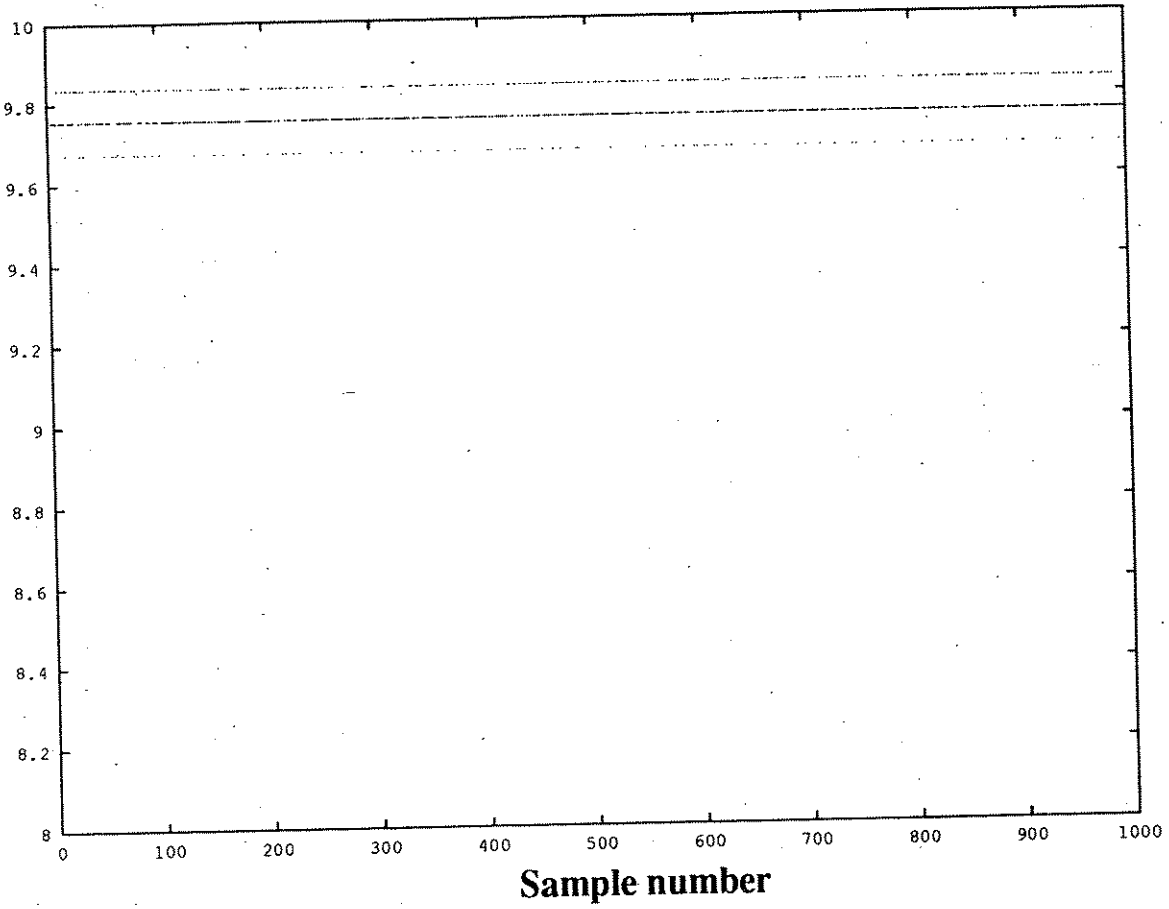
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 6.498 ms
99.9% threshold: 7.236 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



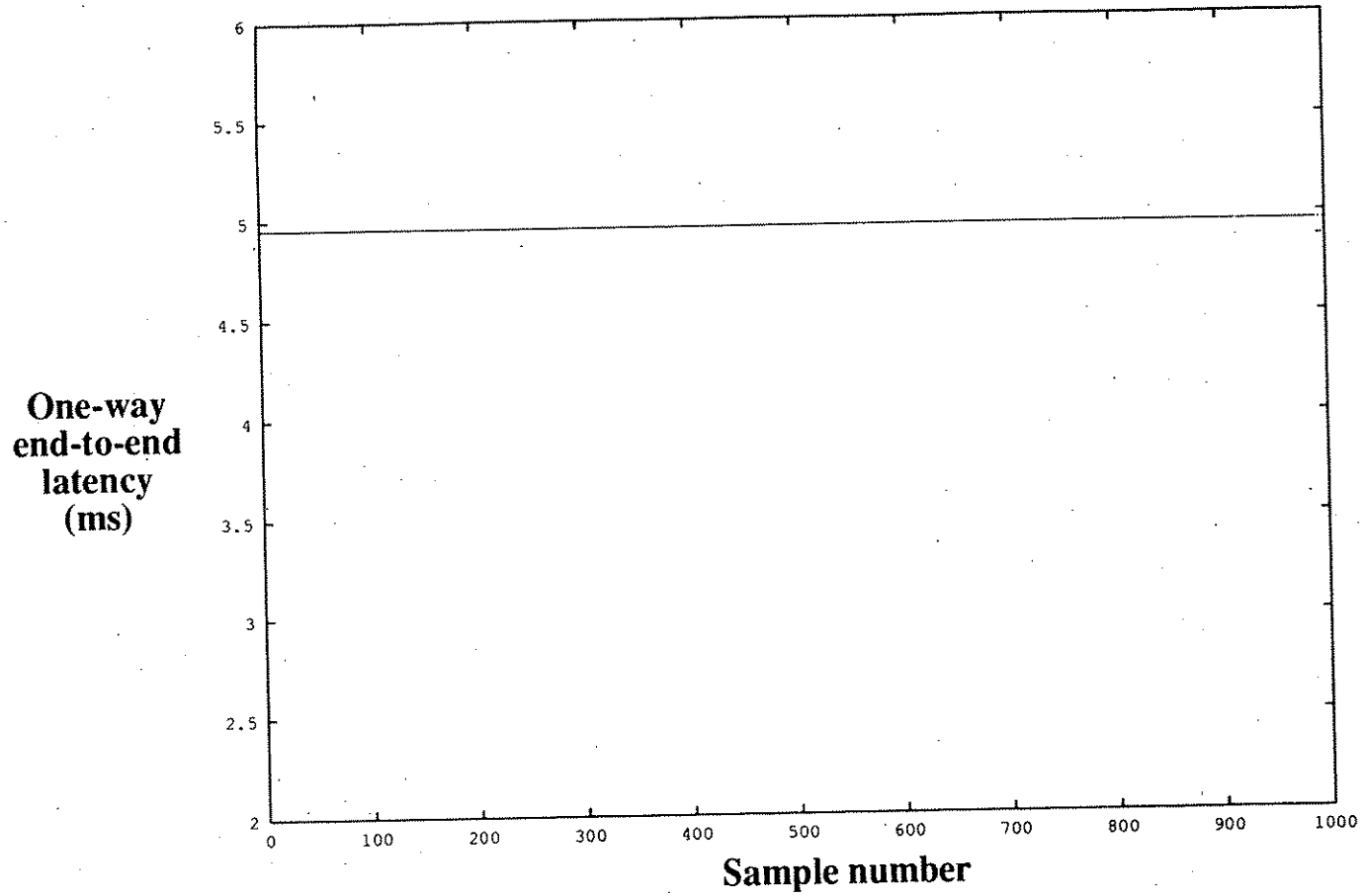
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 9.765 ms
99.9% threshold: 9.837 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 8 bytes



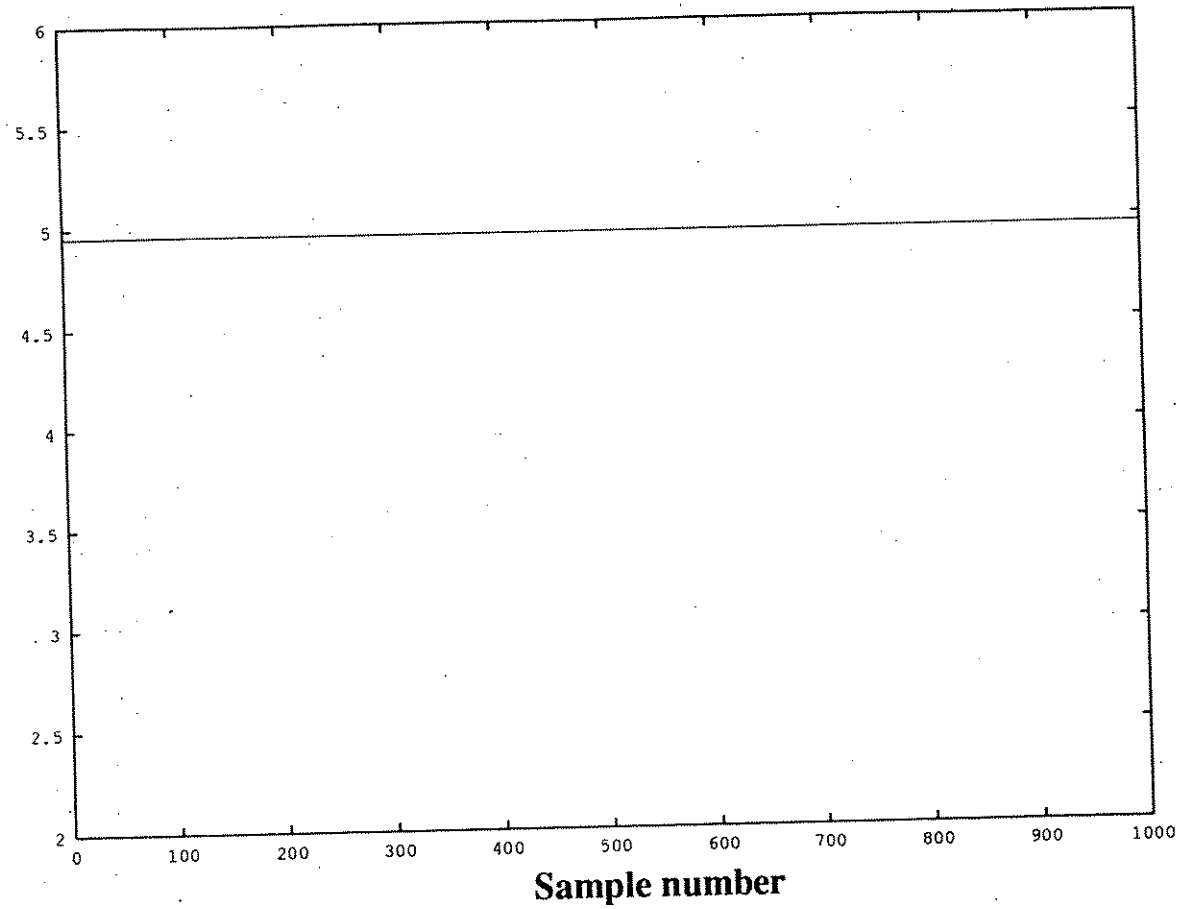
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 4.958 ms
99.9% threshold: 4.959 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 16 bytes



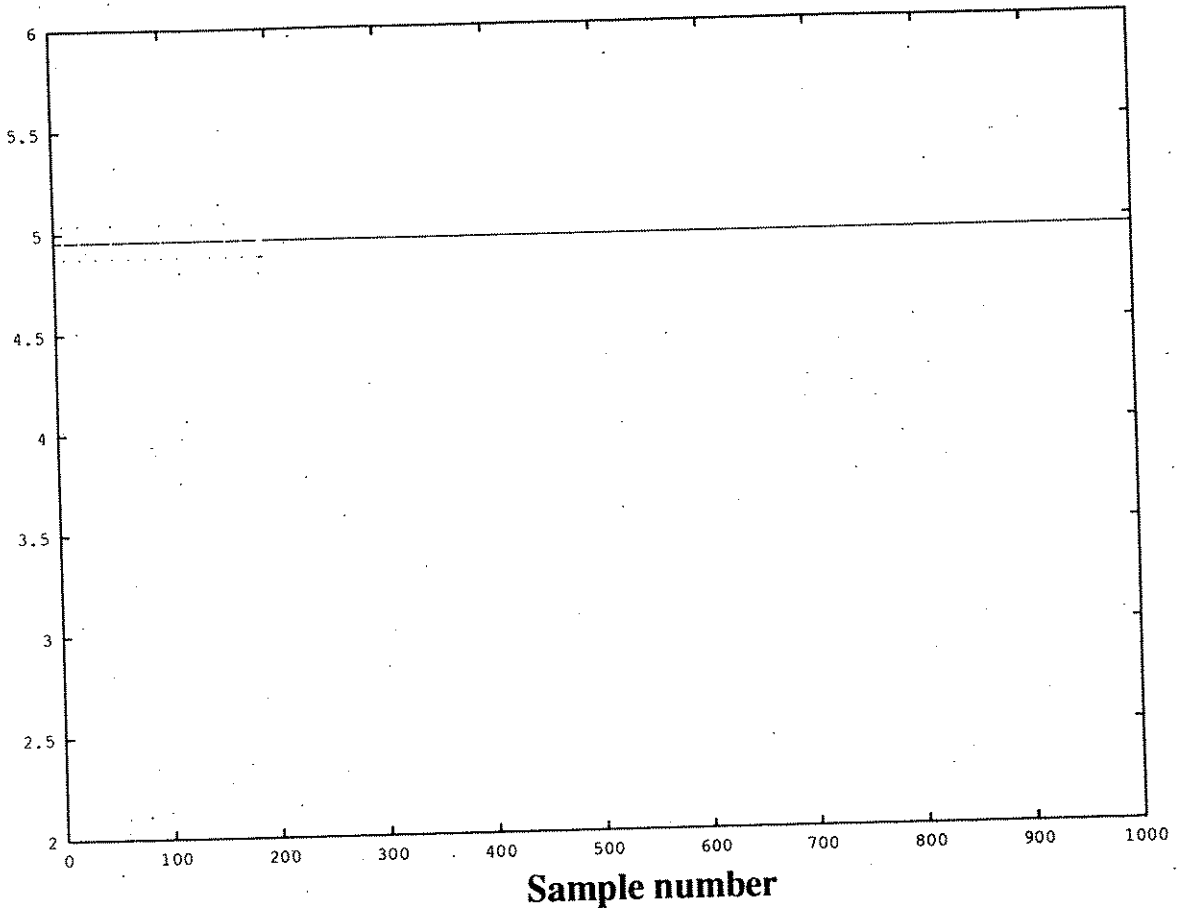
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 4.959 ms
99.9% threshold: 4.959 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 32 bytes



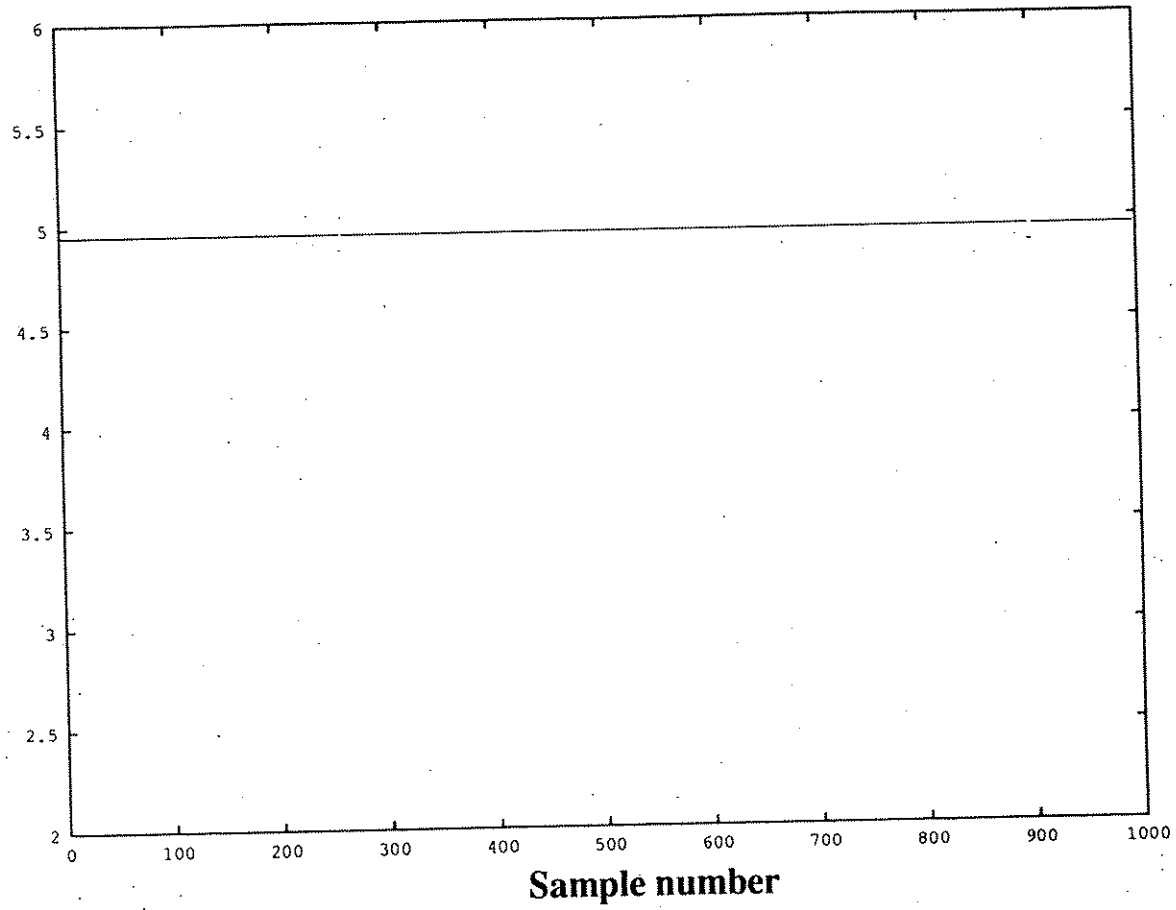
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 4.958 ms
99.9% threshold: 5.041 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 64 bytes



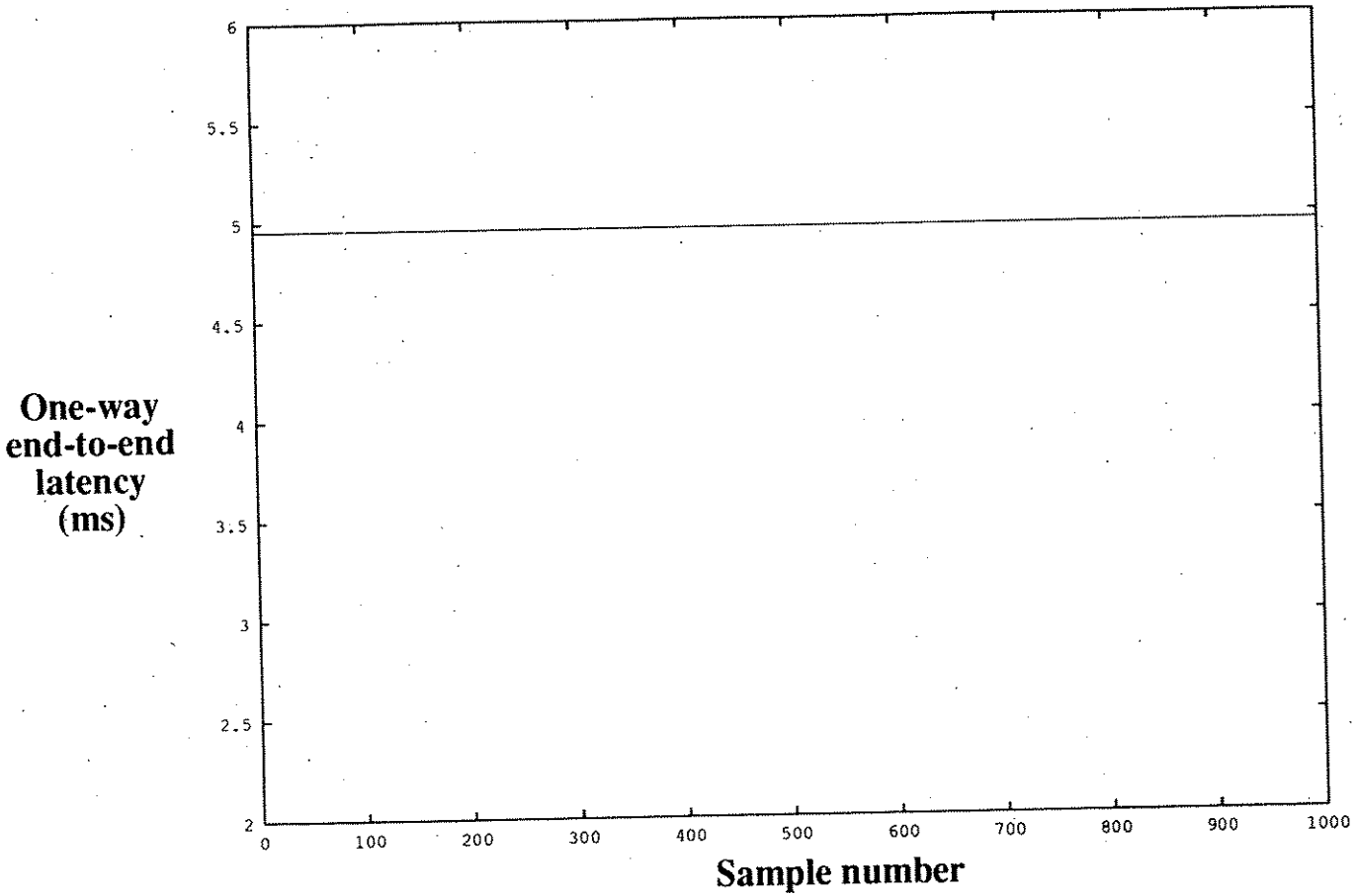
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 4.956 ms
99.9% threshold: 4.959 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 128 bytes



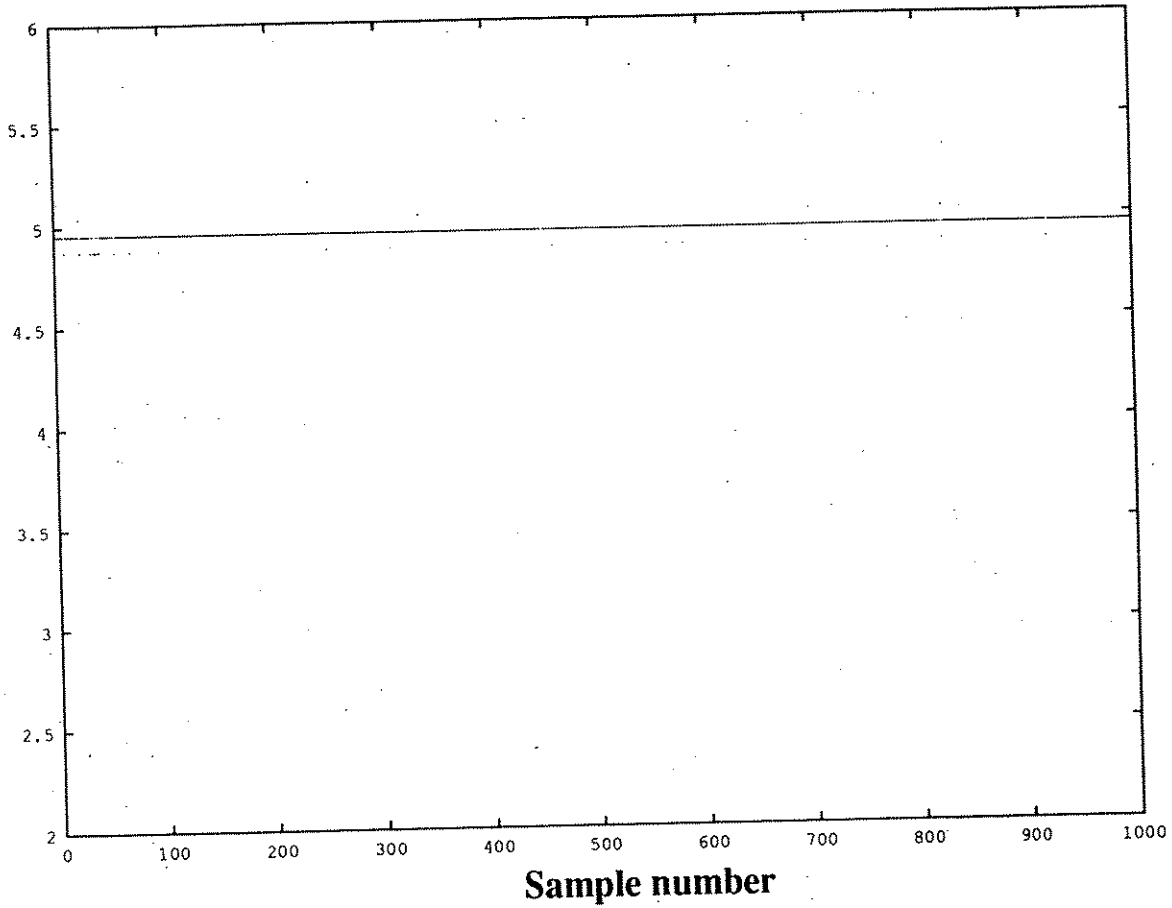
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 4.957 ms
99.9% threshold: 4.959 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 256 bytes



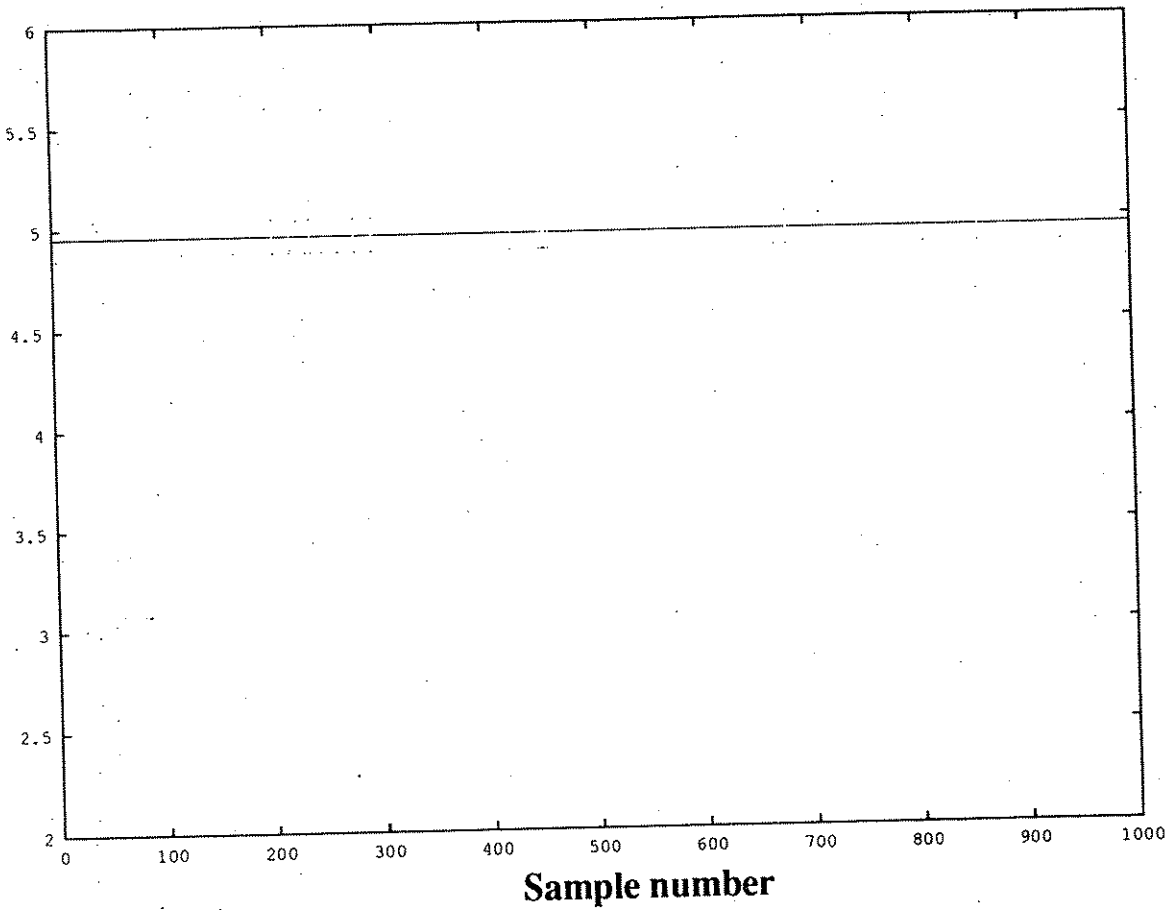
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 4.957 ms
99.9% threshold: 5.041 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 512 bytes



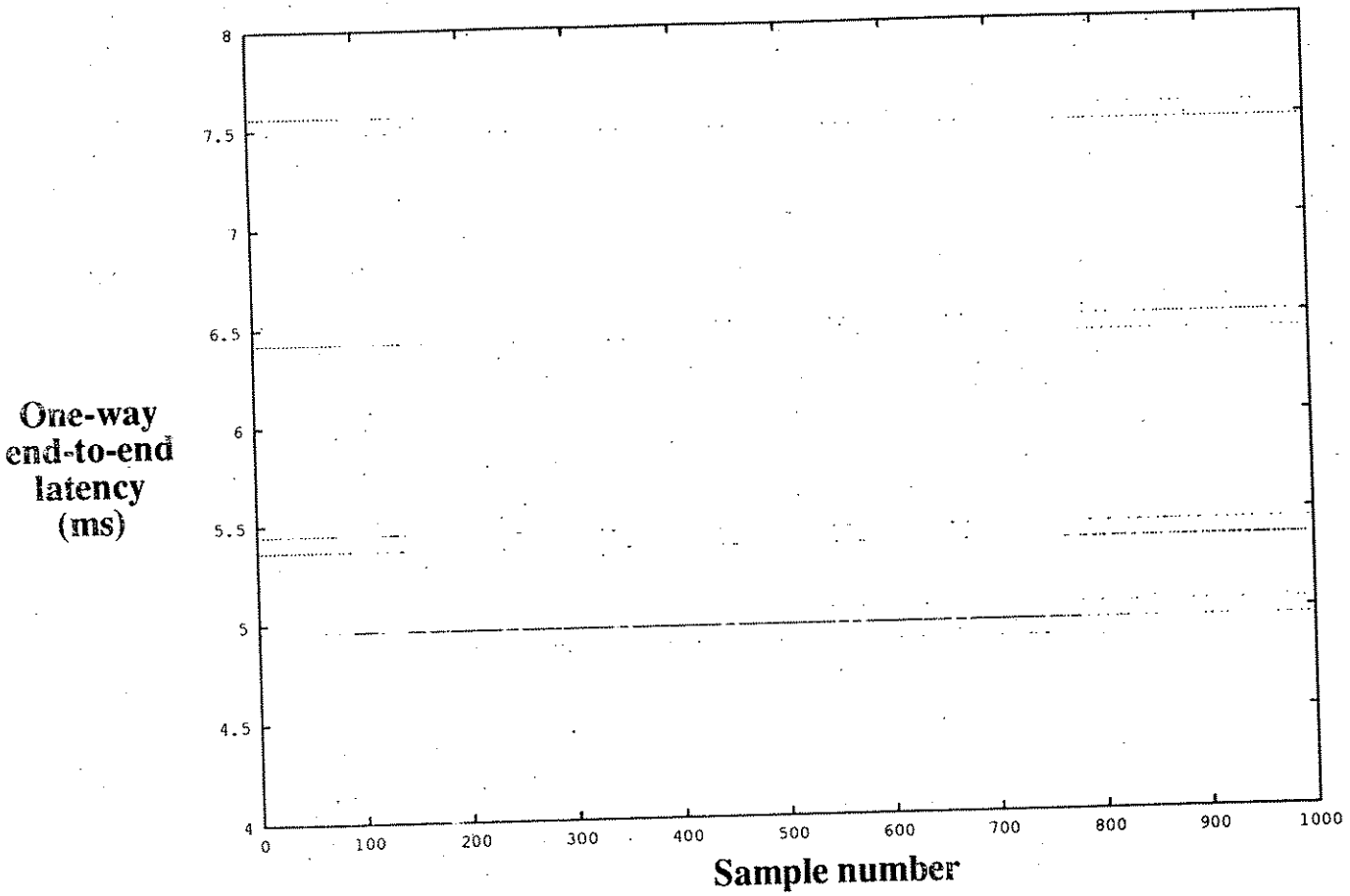
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 4.958 ms
99.9% threshold: 5.041 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



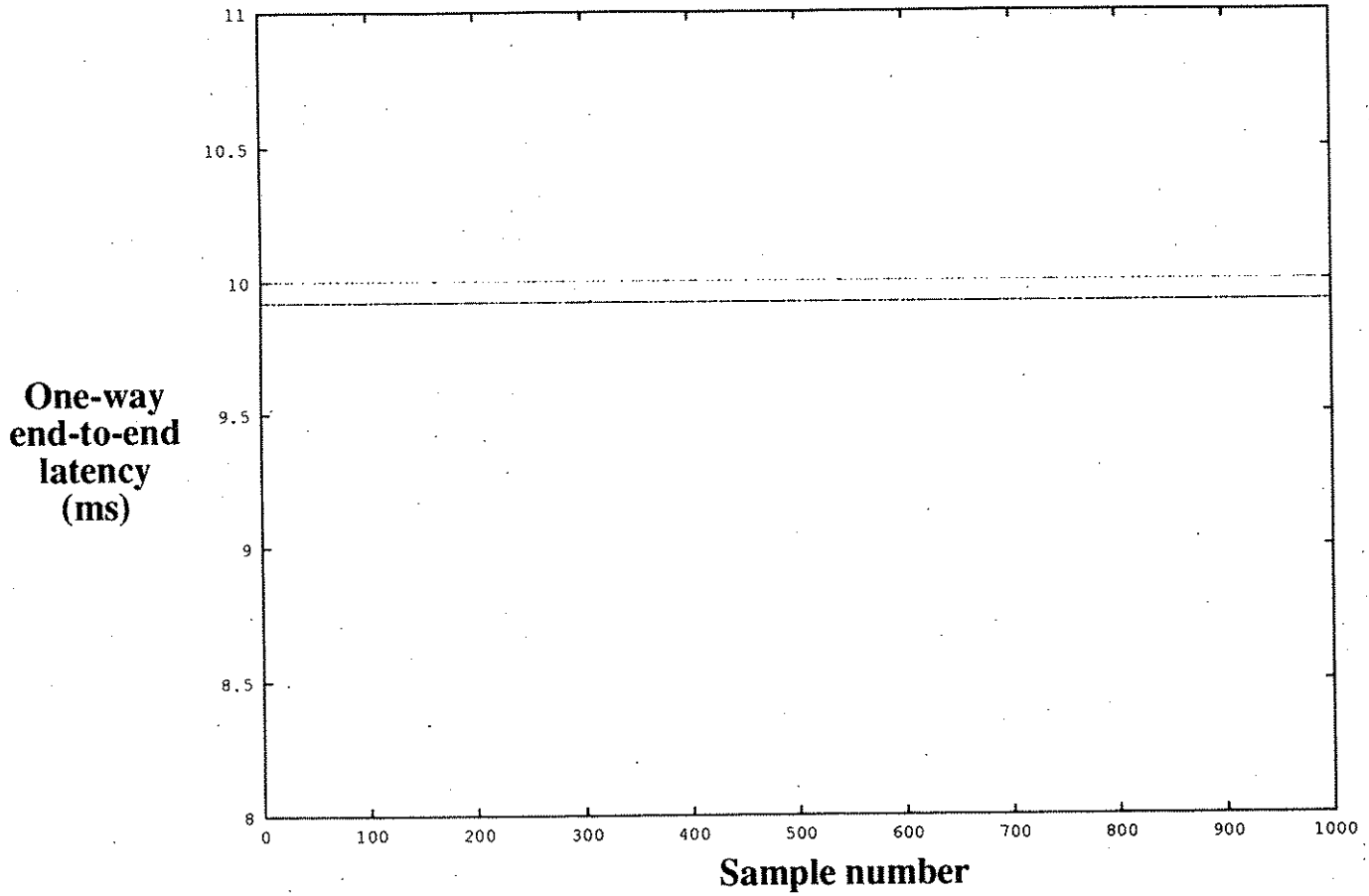
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 5.397 ms
99.9% threshold: 7.967 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



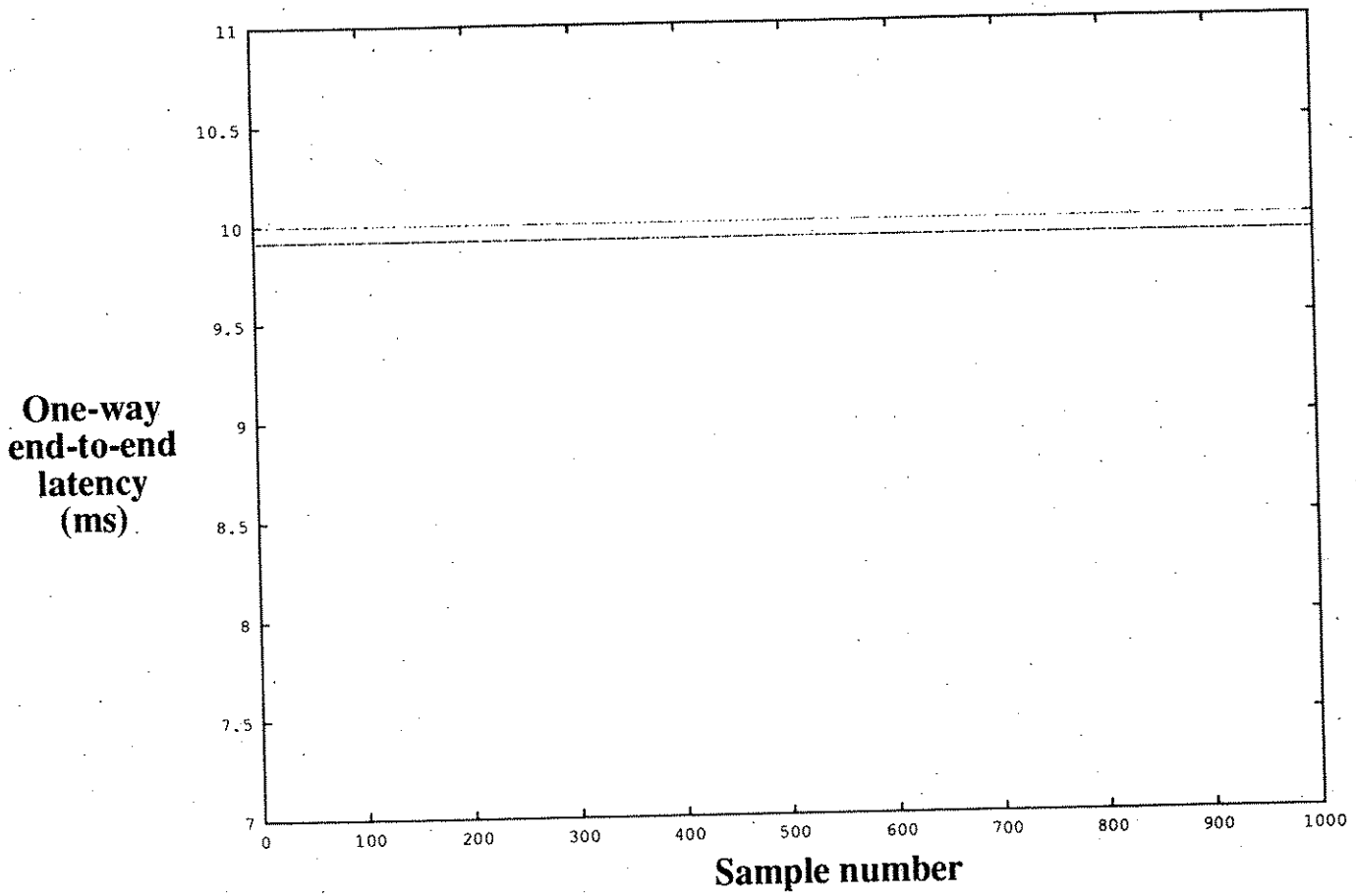
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 9.932 ms
99.9% threshold: 10.000 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



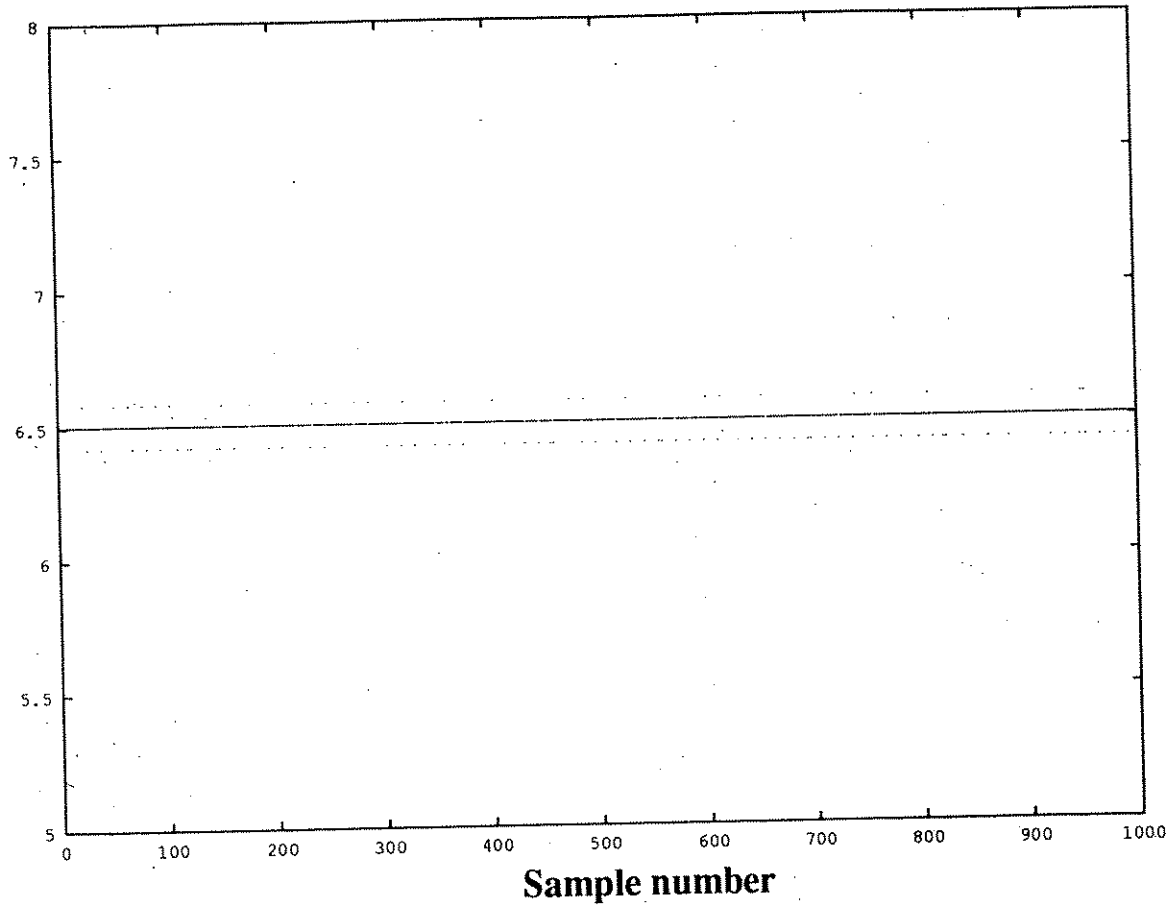
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 9.940 ms
99.9% threshold: 10.000 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 8 bytes



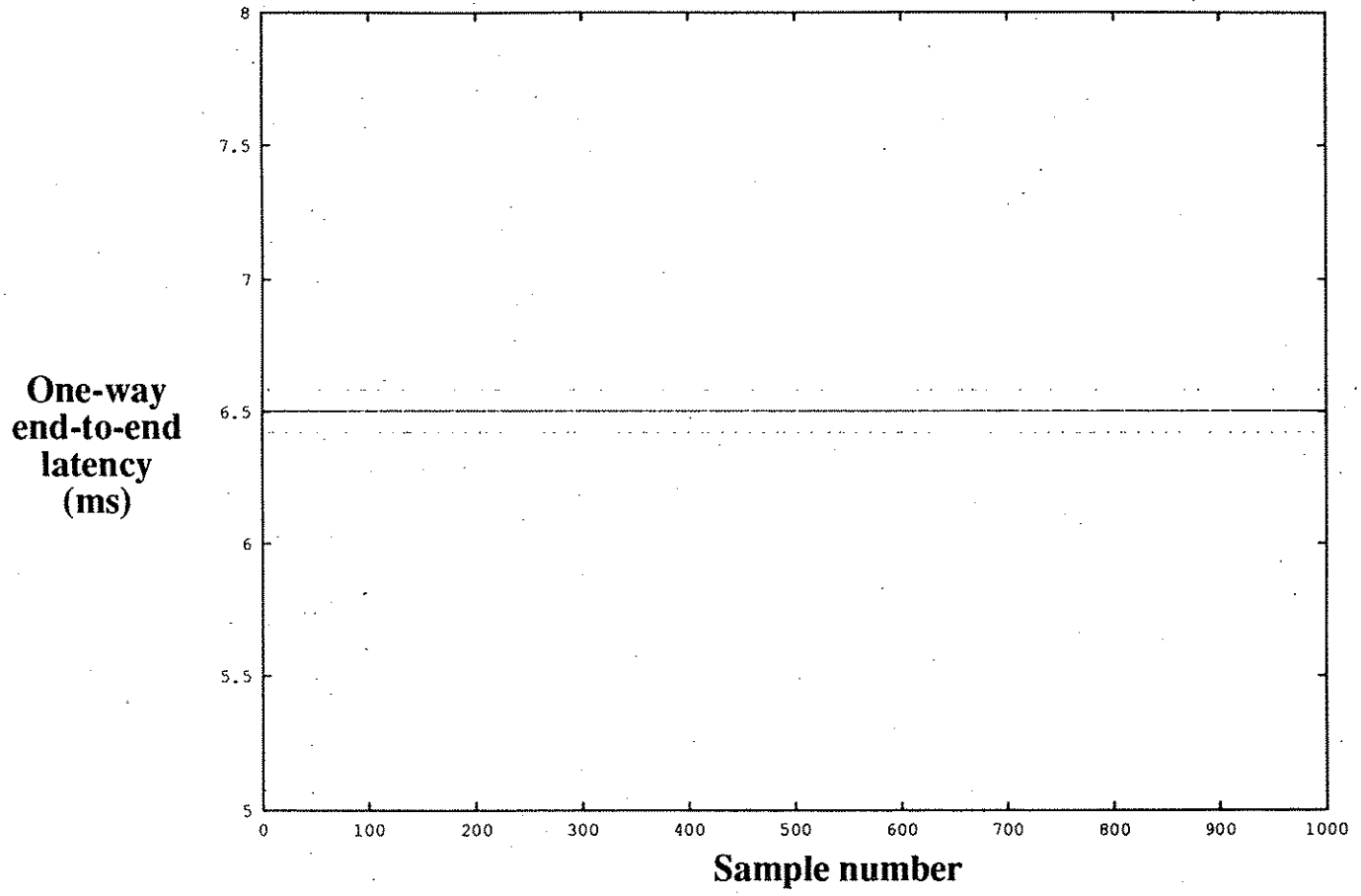
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 6.501 ms
99.9% threshold: 6.585 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 16 bytes



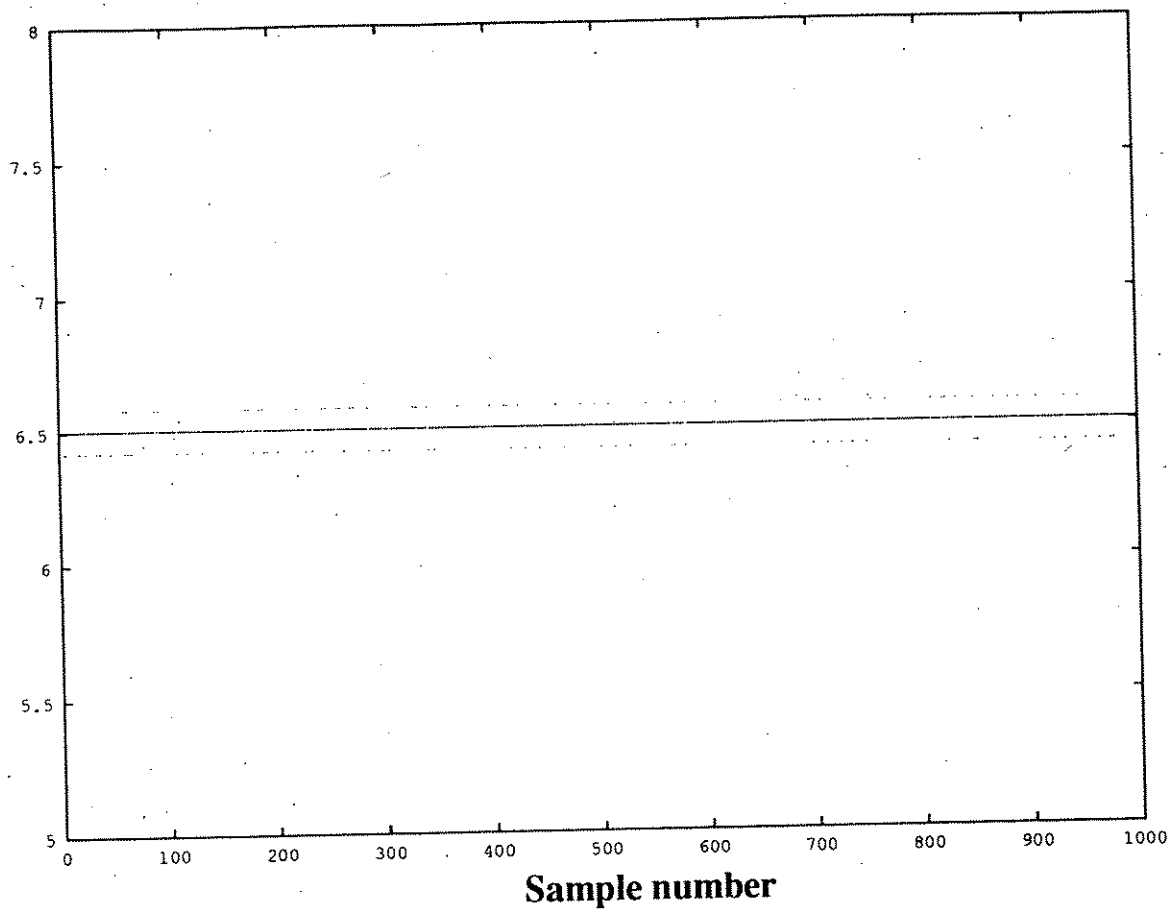
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 6.501 ms
99.9% threshold: 6.585 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 32 bytes



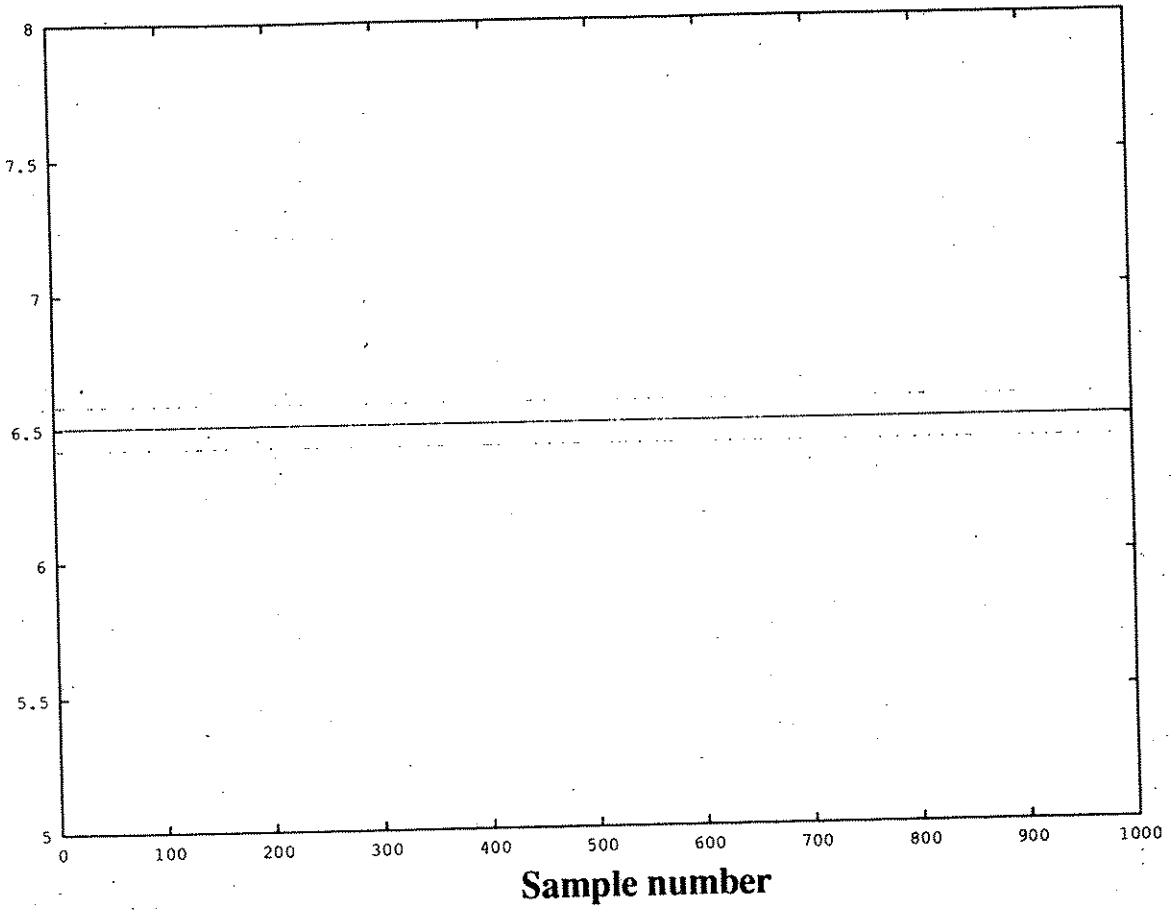
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 6.504 ms
99.9% threshold: 6.585 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 64 bytes



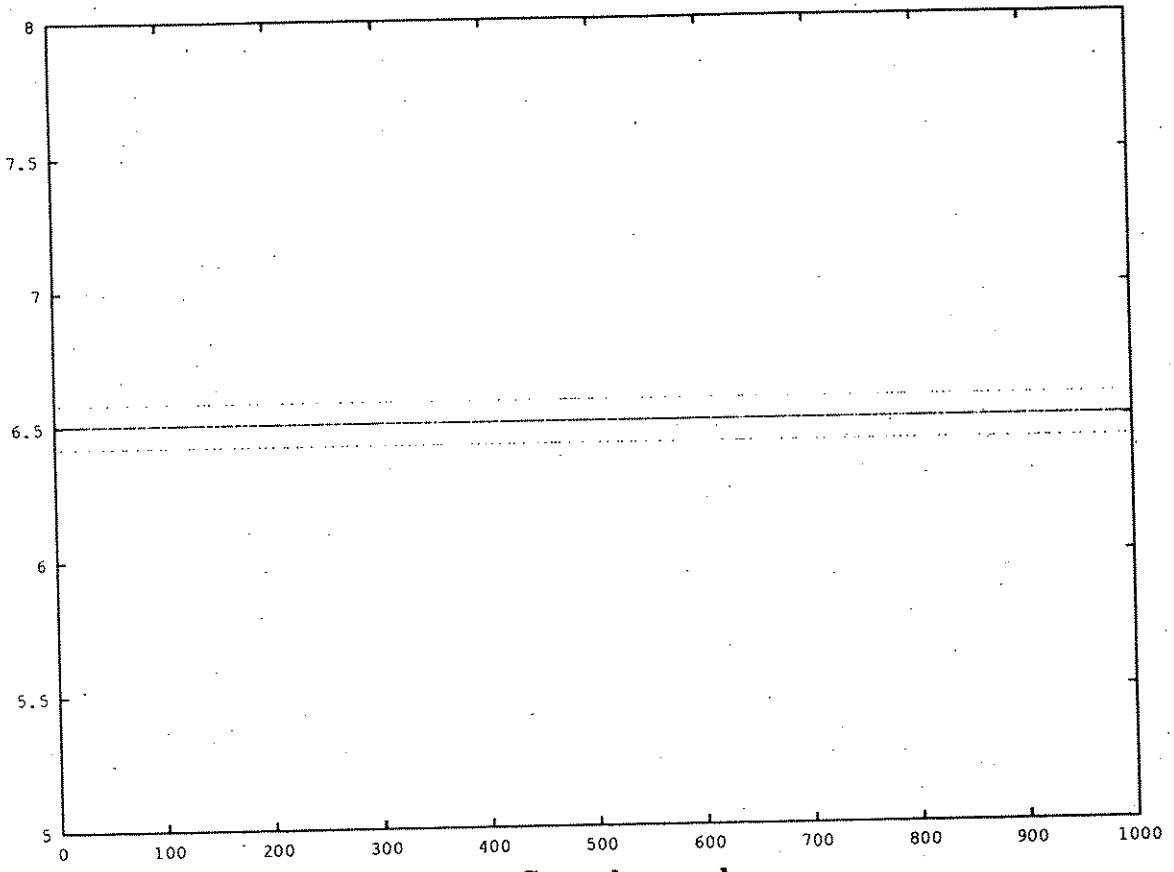
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 6.501 ms
99.9% threshold: 6.585 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 128 bytes



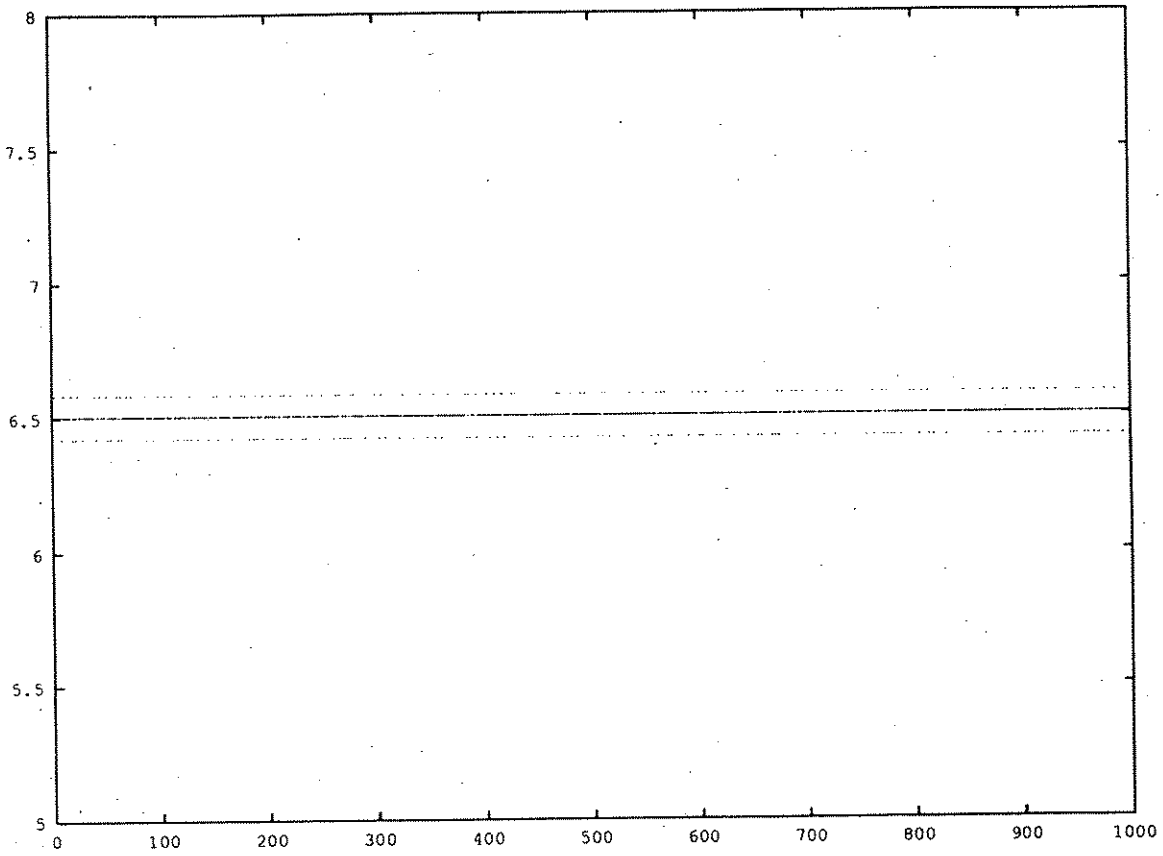
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 6.503 ms
99.9% threshold: 6.992 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 256 bytes



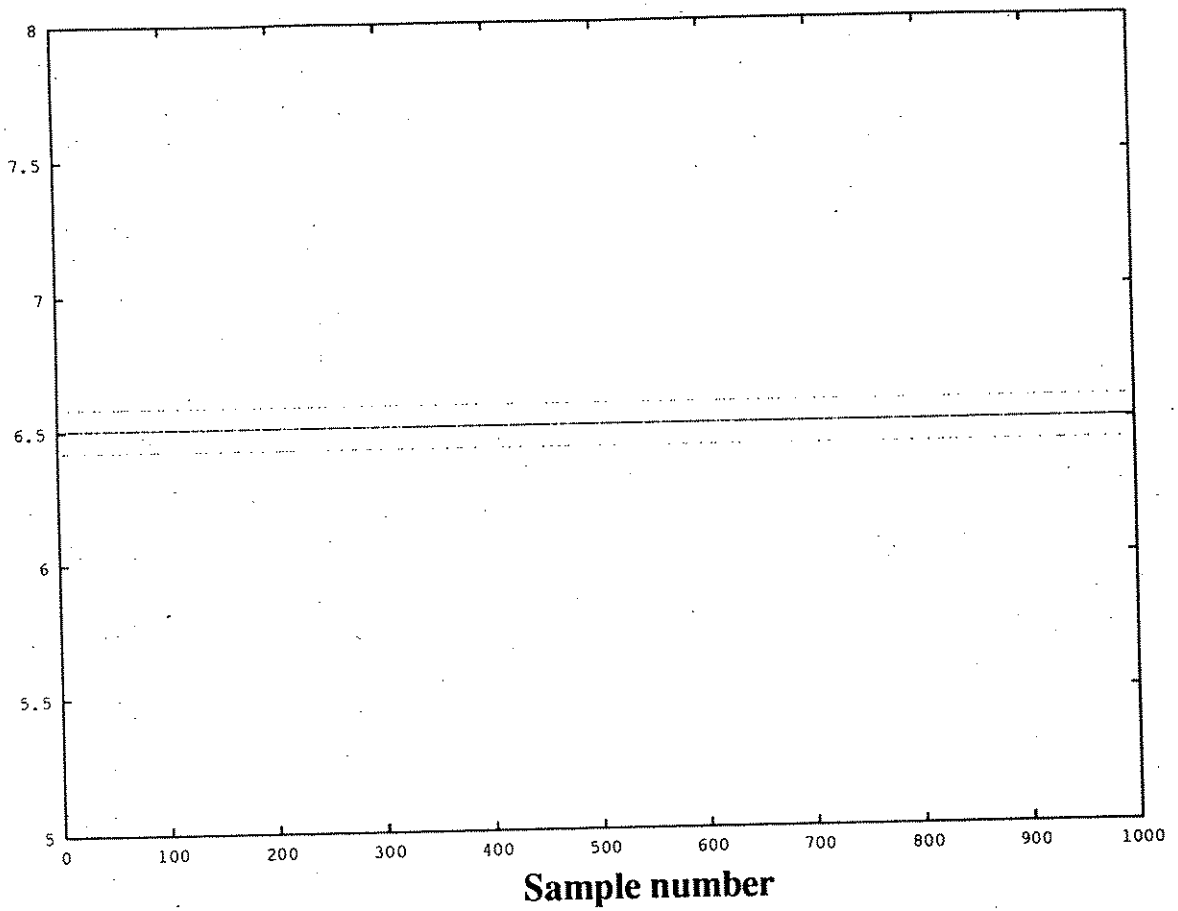
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 6.504 ms
99.9% threshold: 6.585 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 512 bytes



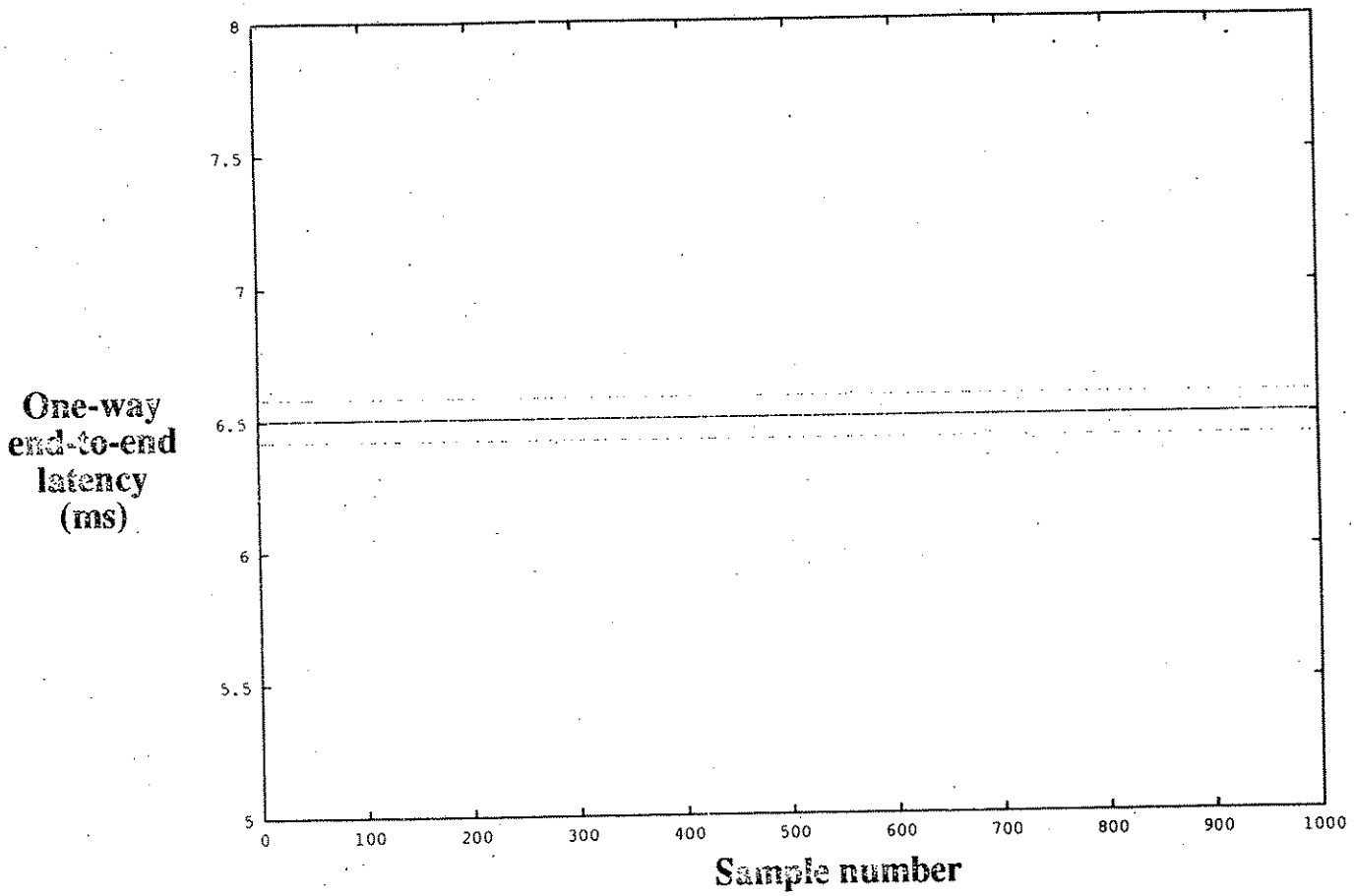
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 6.505 ms
99.9% threshold: 6.585 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



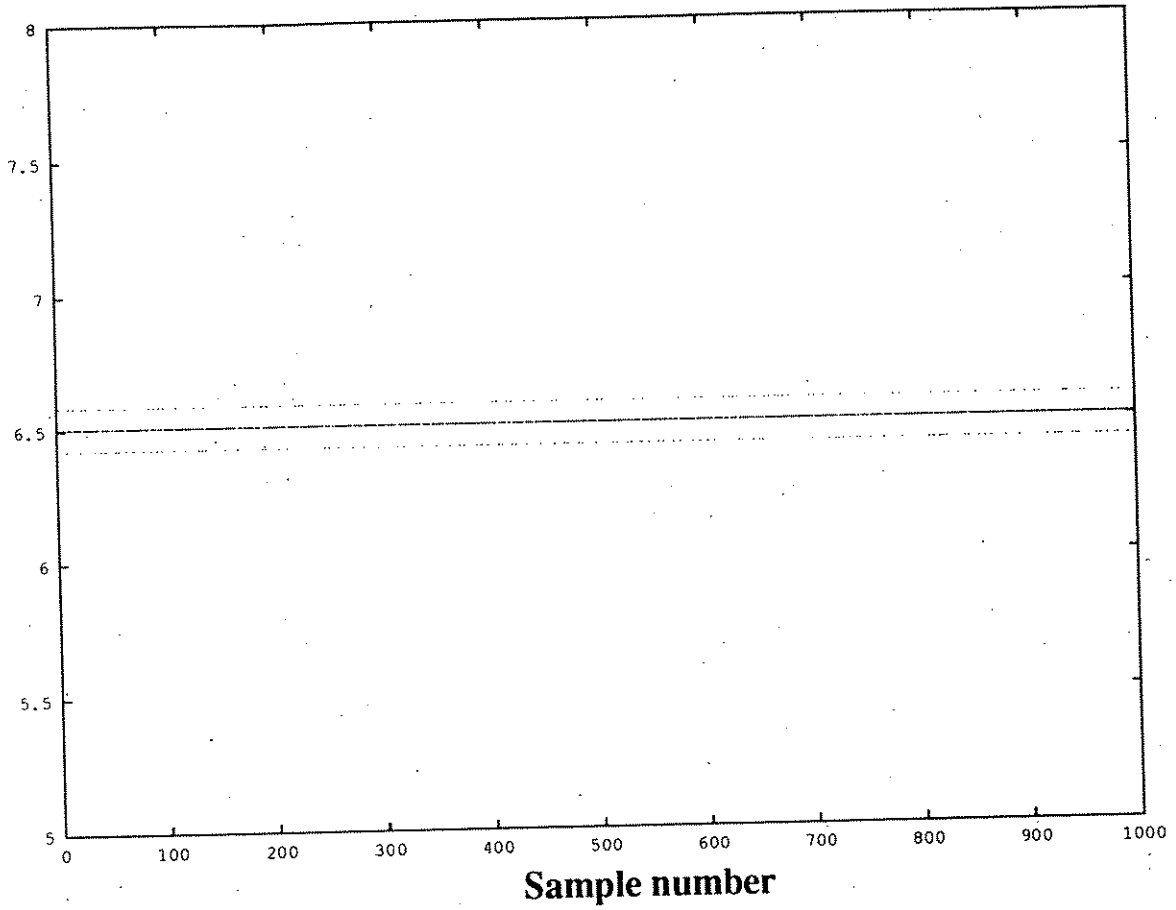
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 6.502 ms
99.9% threshold: 6.585 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



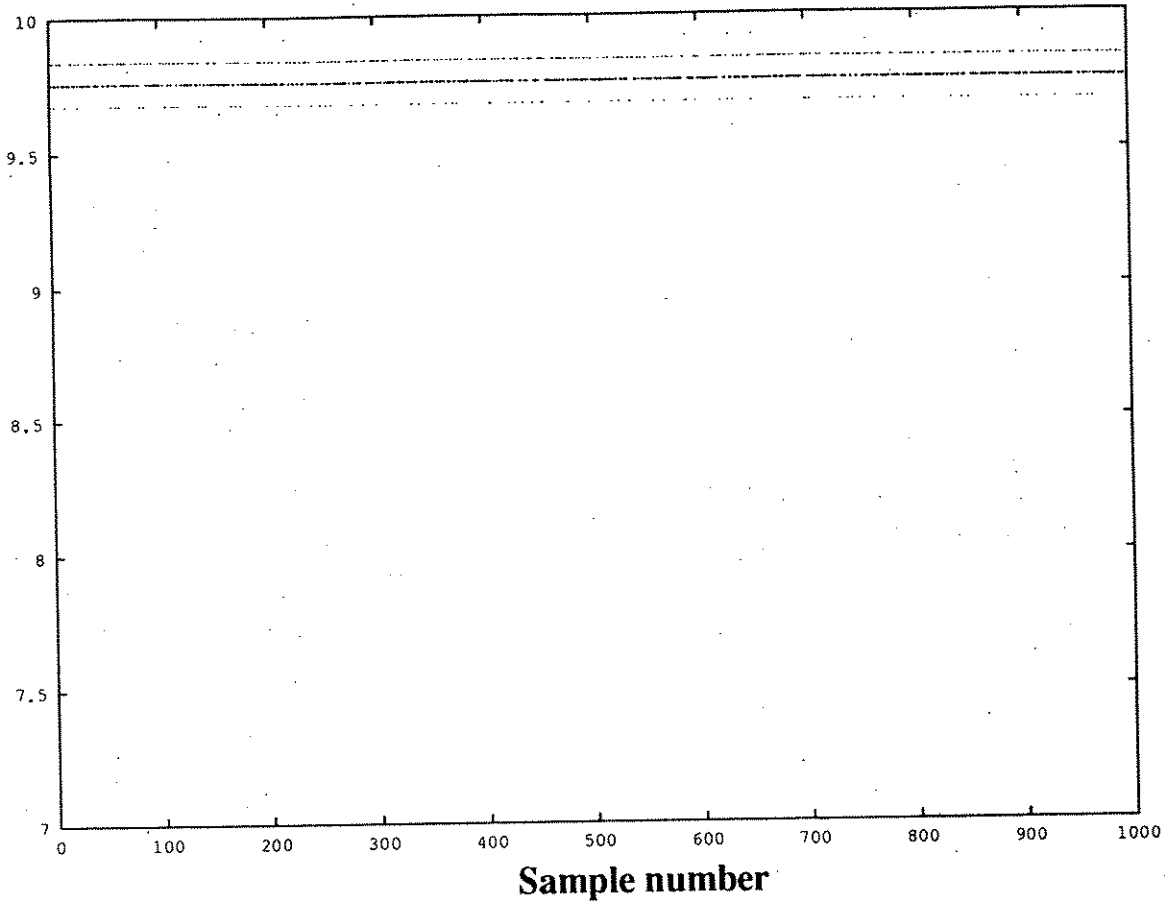
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 6.503 ms
99.9% threshold: 6.667 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 9.774 ms
99.9% threshold: 9.919 ms

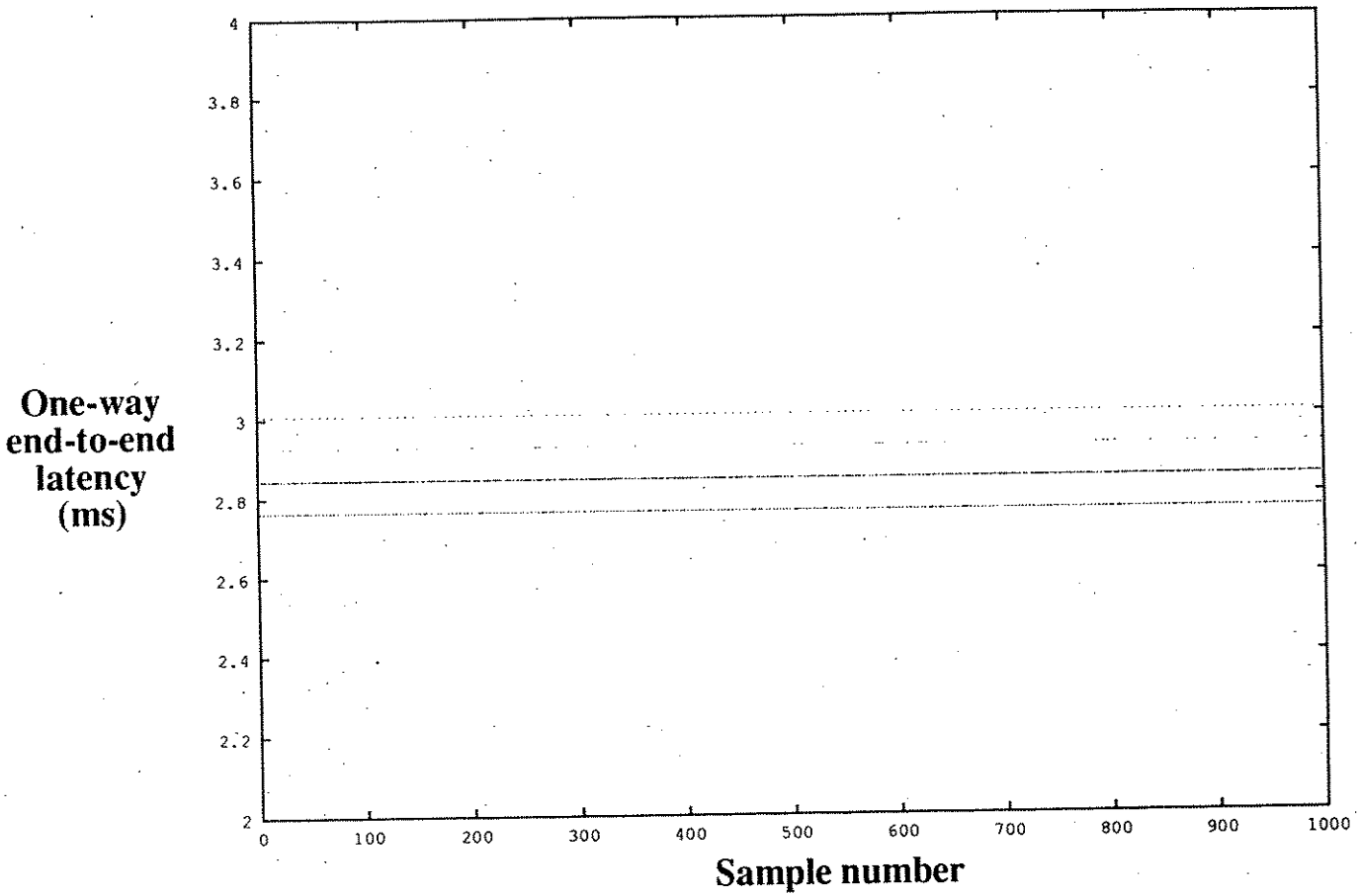
SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

Appendix D

Experiment 4: Retransmission

JITTER MEASUREMENT

Voice Data Size: 8 bytes



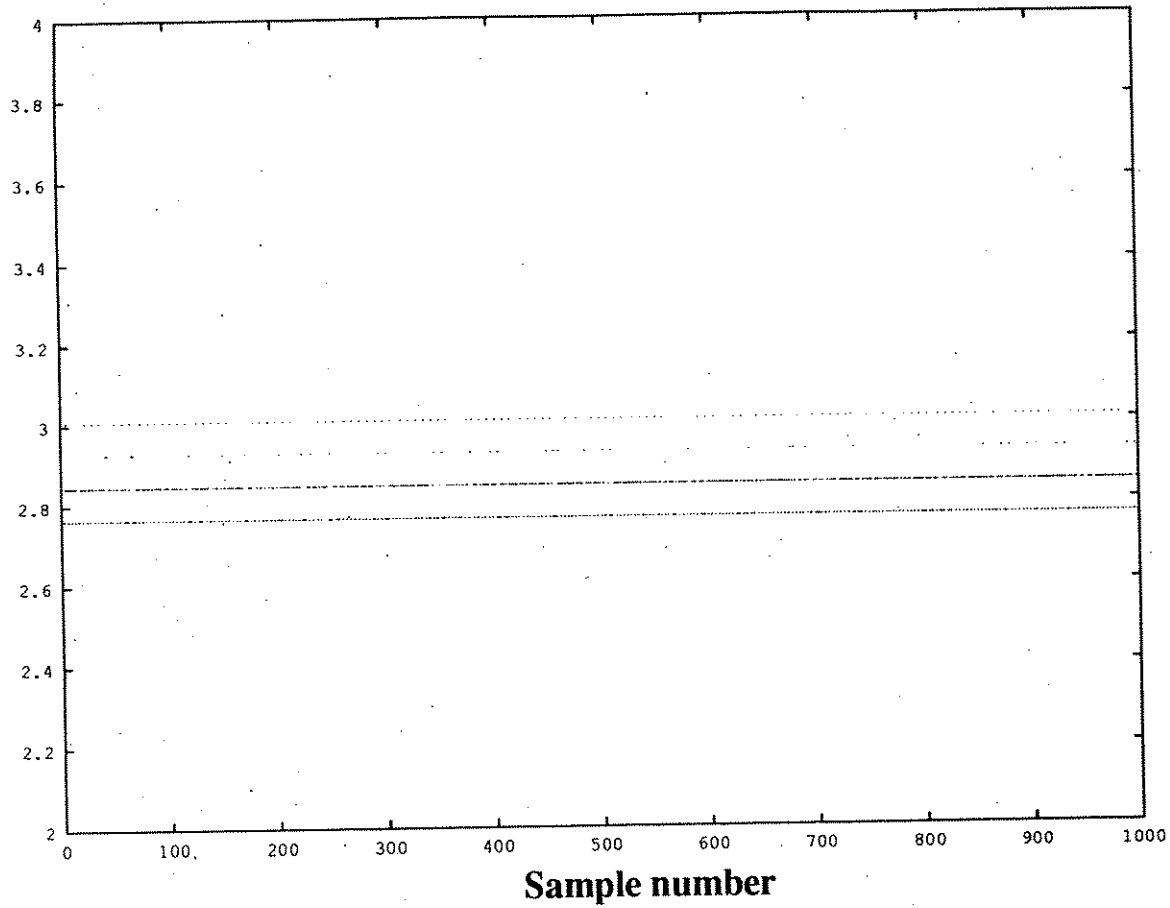
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
1% packet loss
Voice data in FDDI synchronous class

Average latency: 2.838 ms
99.9% threshold: 3.008 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 16 bytes



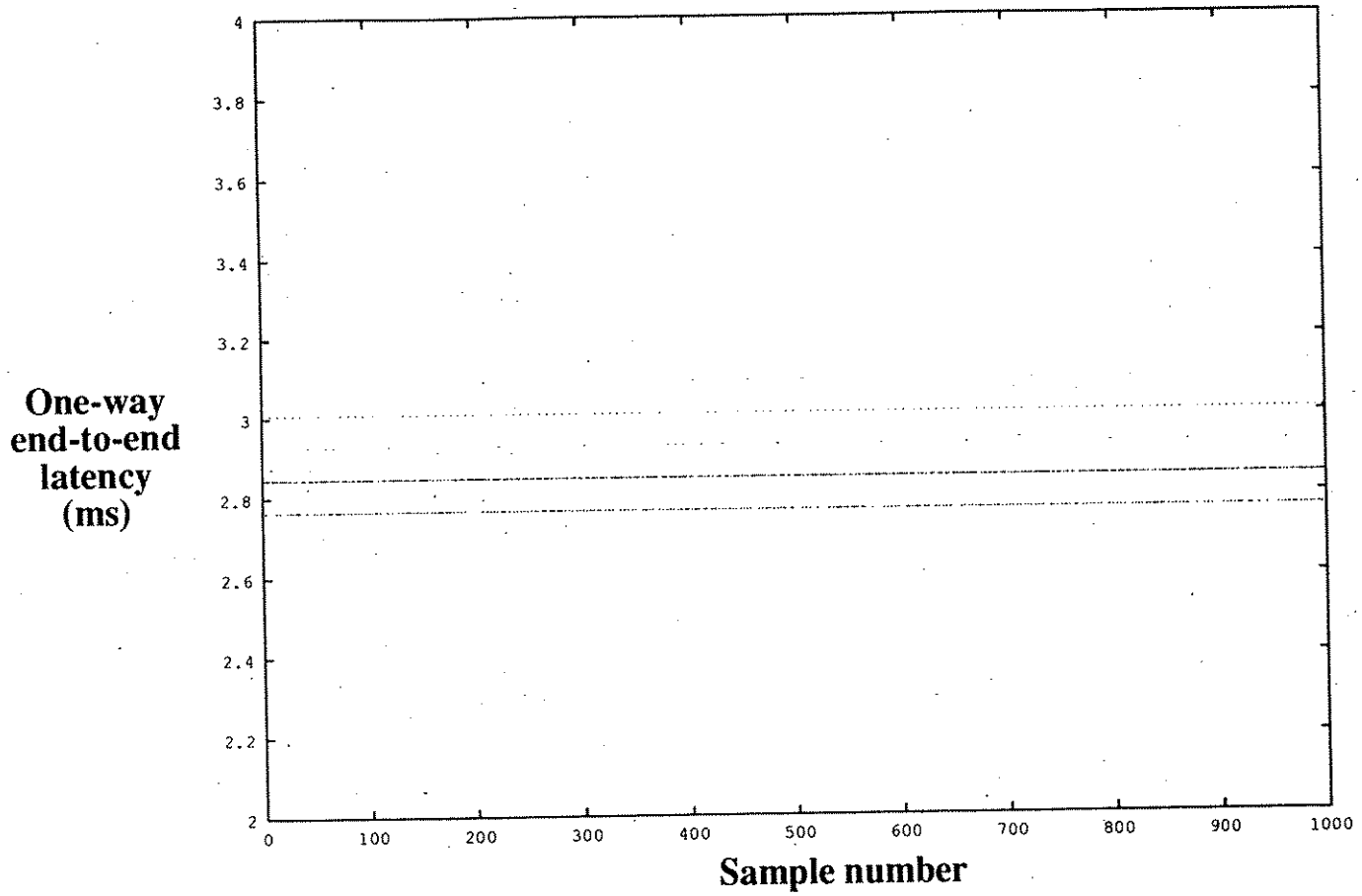
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
1% packet loss
Voice data in FDDI synchronous class

Average latency: 2.838 ms
99.9% threshold: 3.089 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 32 bytes



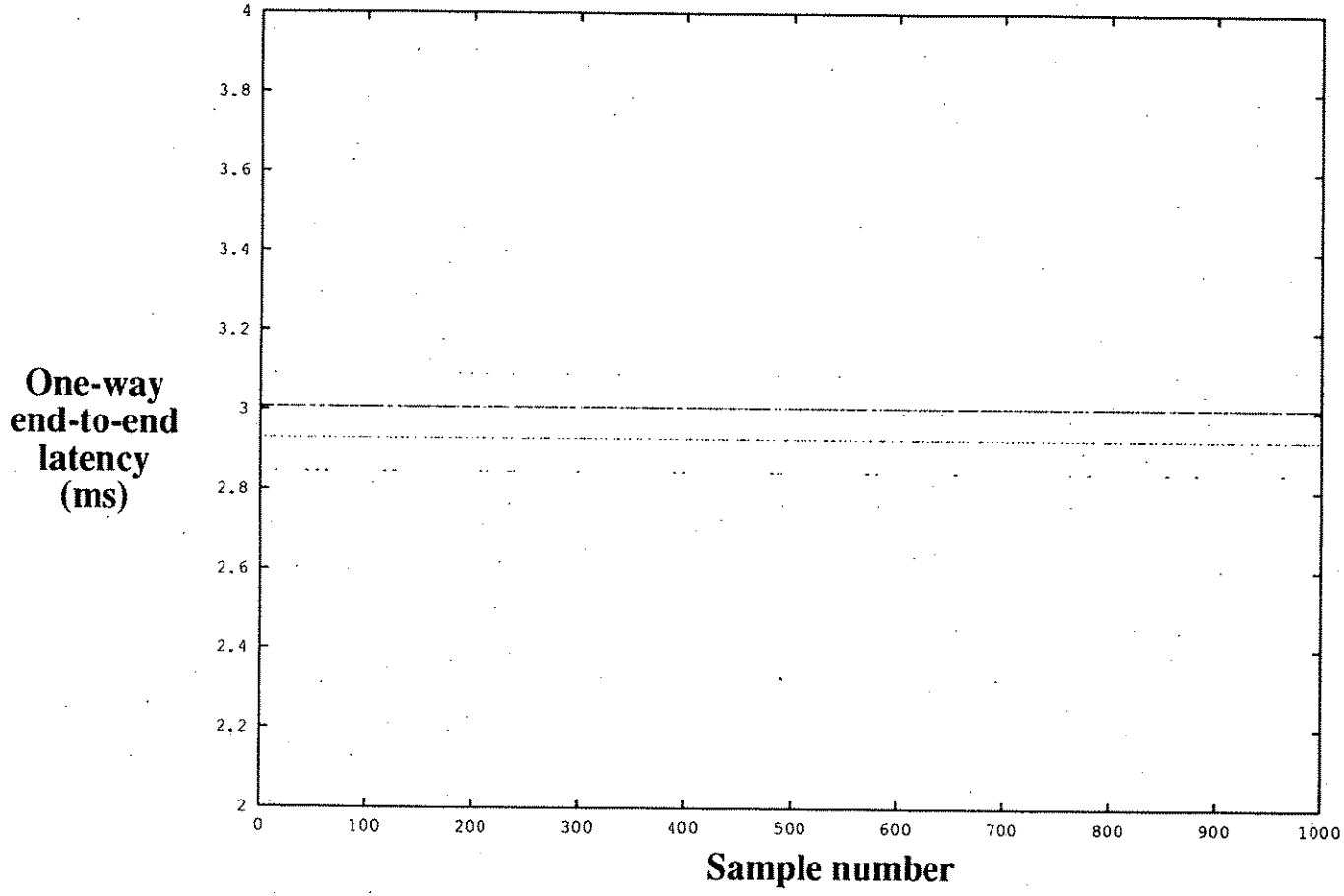
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
1% packet loss
Voice data in FDDI synchronous class

Average latency: 2.940 ms
99.9% threshold: 3.089 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 64 bytes



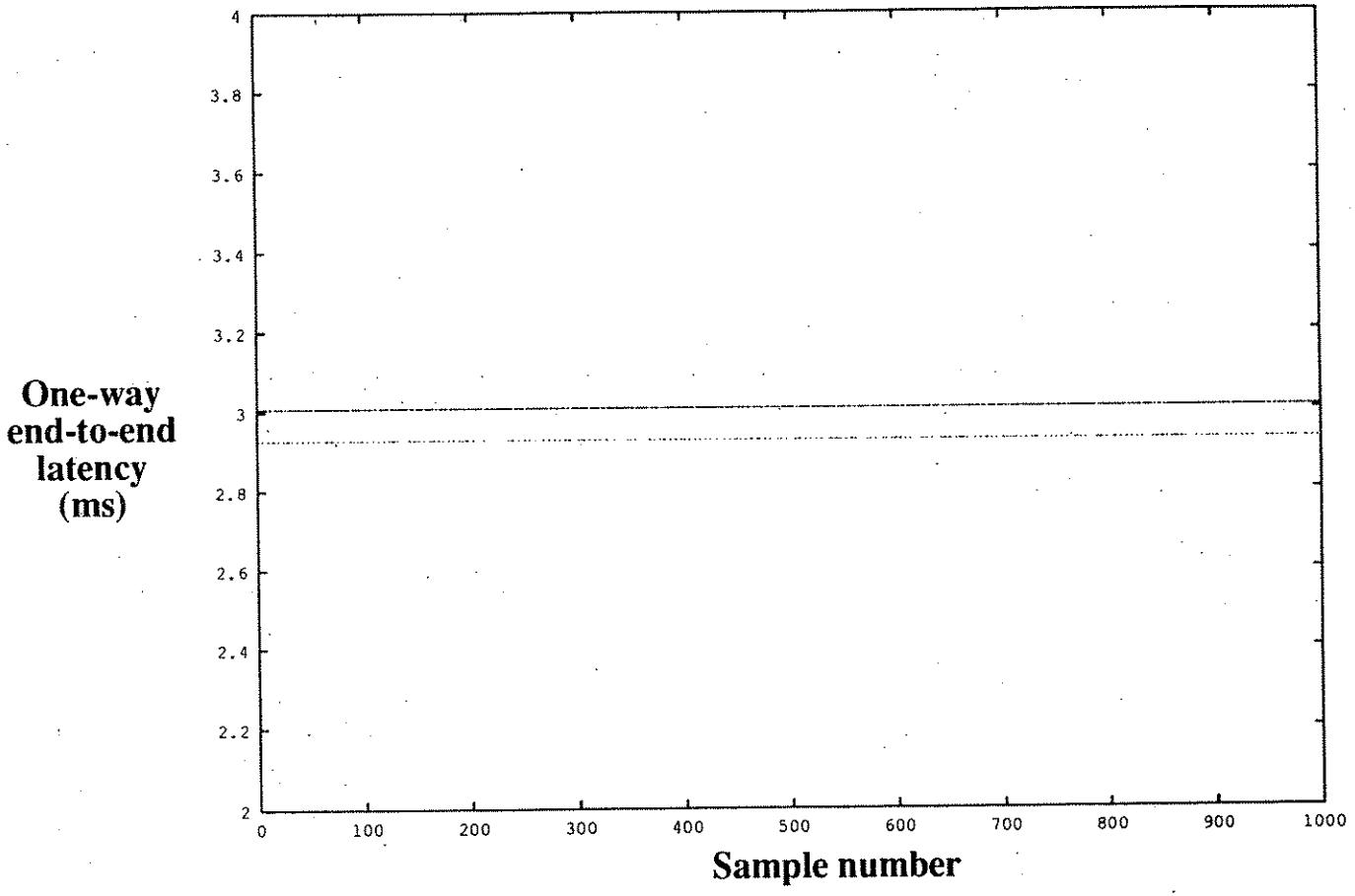
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
1% packet loss
Voice data in FDDI synchronous class

Average latency: 2.980 ms
99.9% threshold: 3.089 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 128 bytes



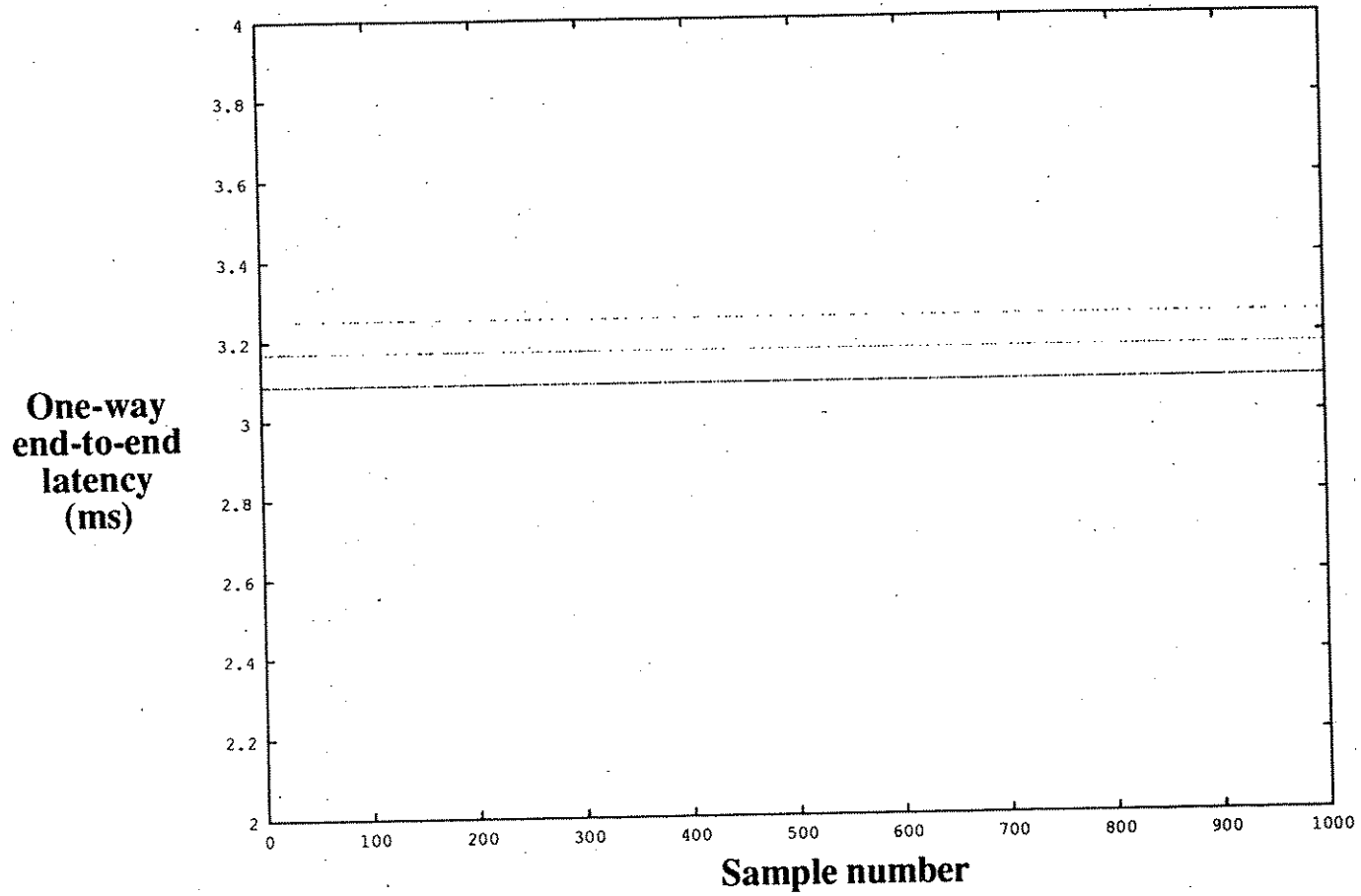
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
1% packet loss
Voice data in FDDI synchronous class

Average latency: 2.989 ms
99.9% threshold: 3.089 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 256 bytes



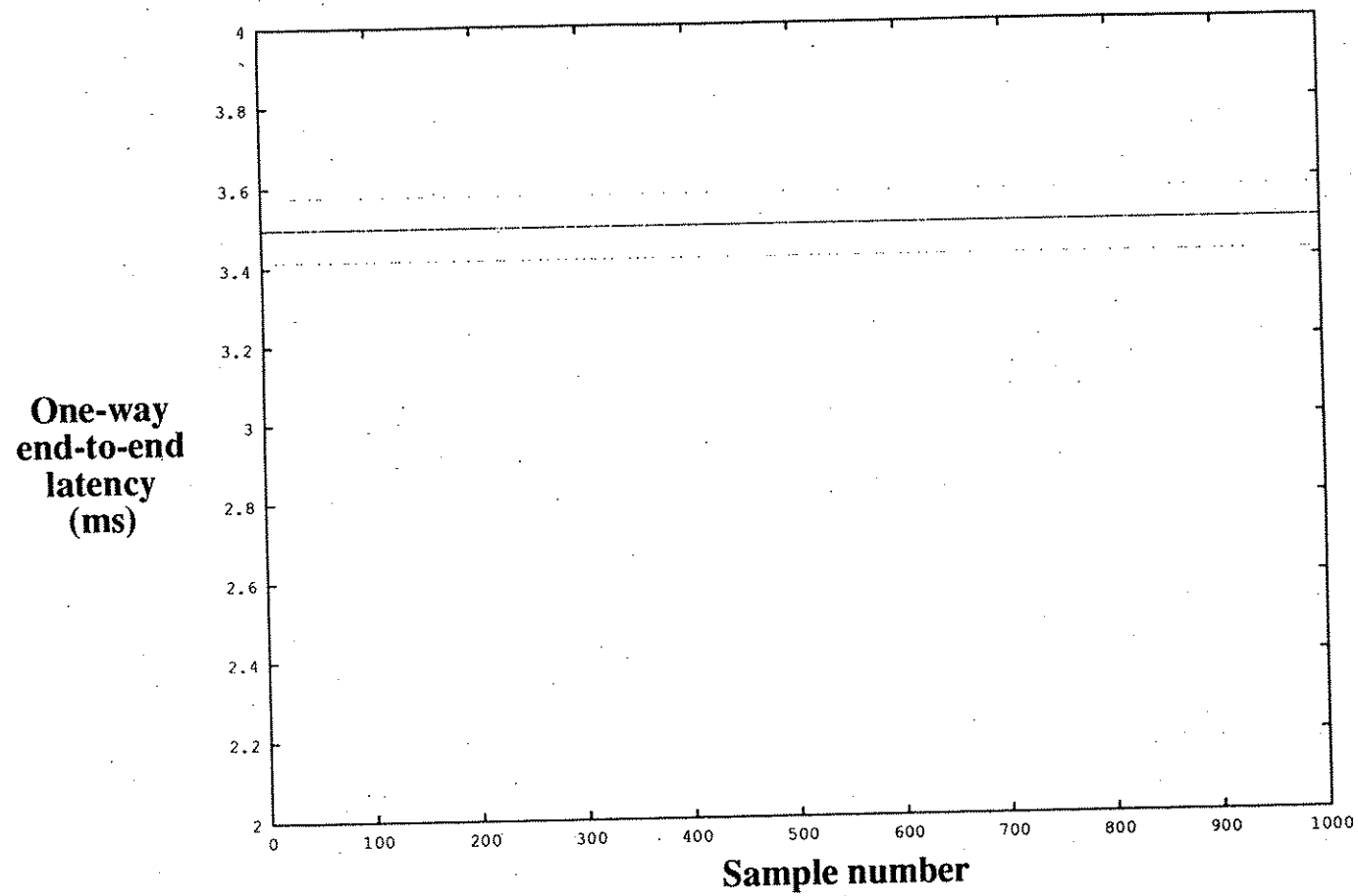
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
1% packet loss
Voice data in FDDI synchronous class

Average latency: 3.122 ms
99.9% threshold: 3.333 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 512 bytes



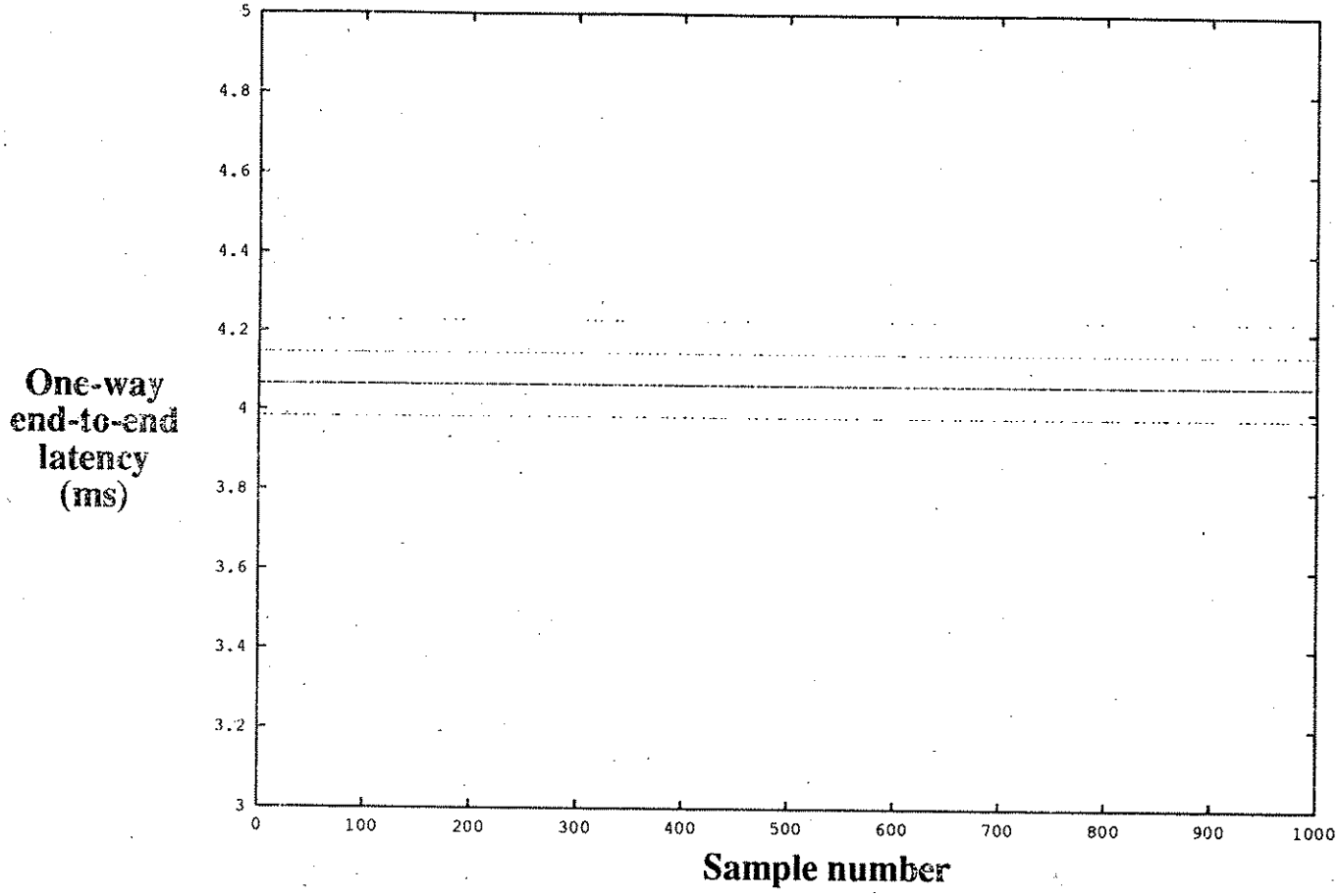
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
1% packet loss
Voice data in FDDI synchronous class

Average latency: 3.492 ms
99.9% threshold: 3.577 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



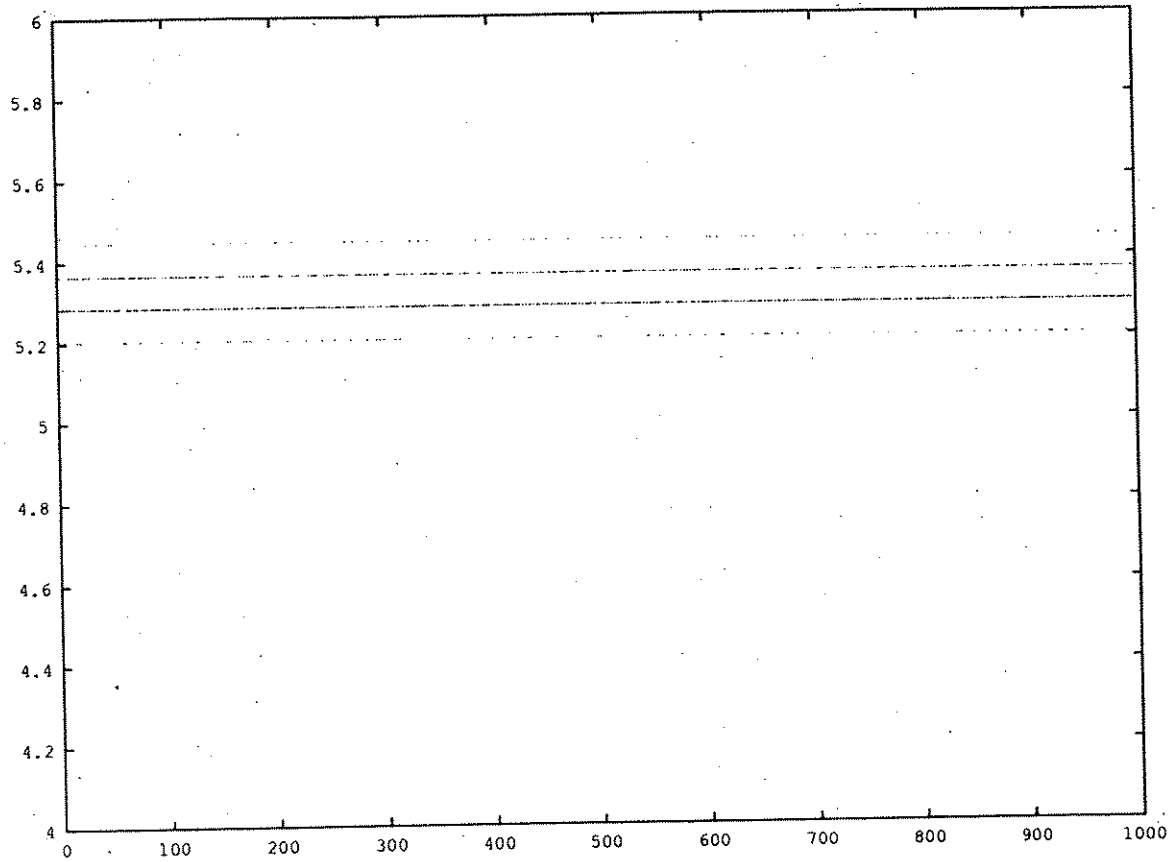
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
1% packet loss
Voice data in FDDI synchronous class

Average latency: 4.068 ms
99.9% threshold: 4.228 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



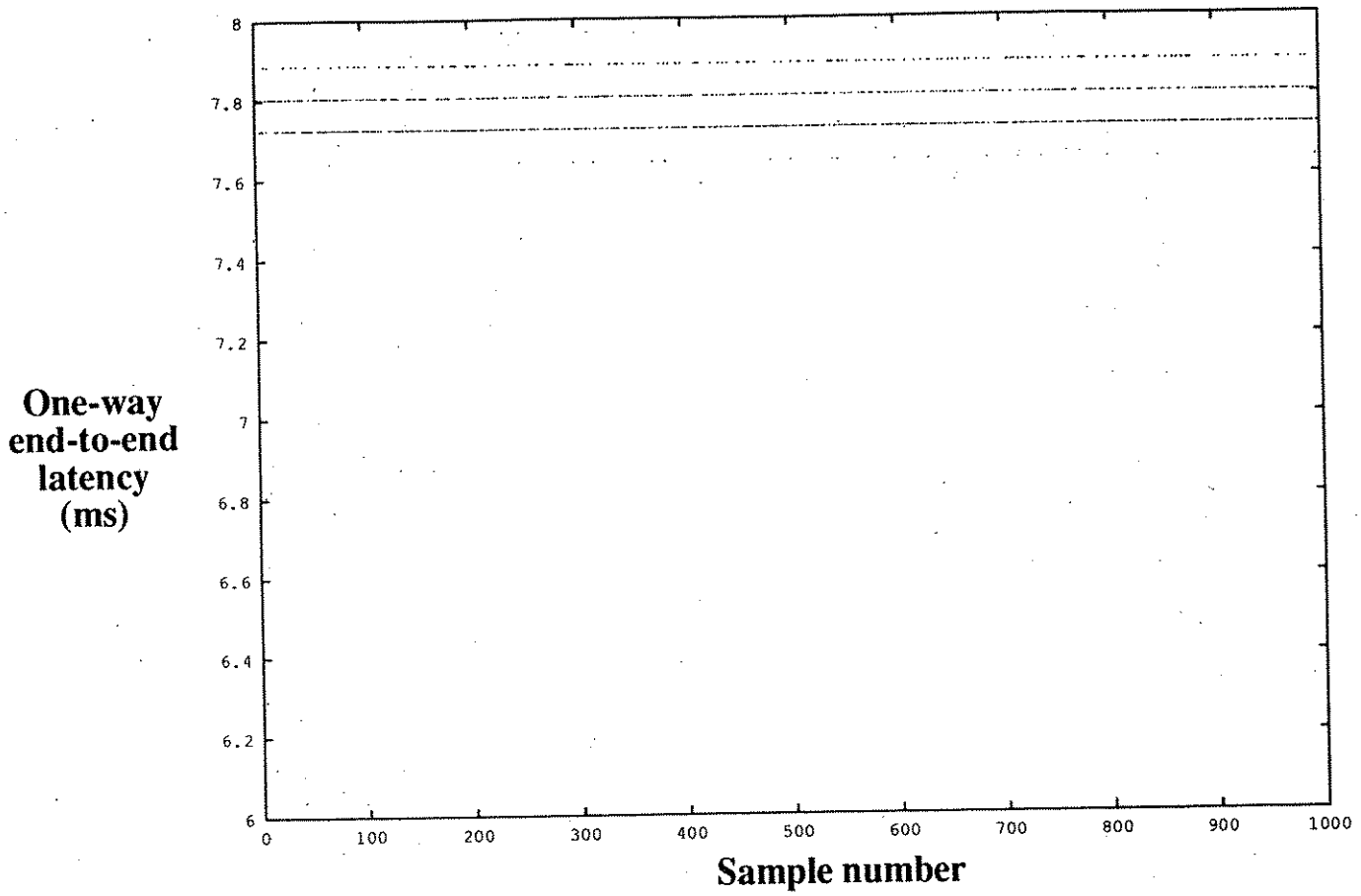
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
1% packet loss
Voice data in FDDI synchronous class

Average latency: 5.316 ms
99.9% threshold: 5.447 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



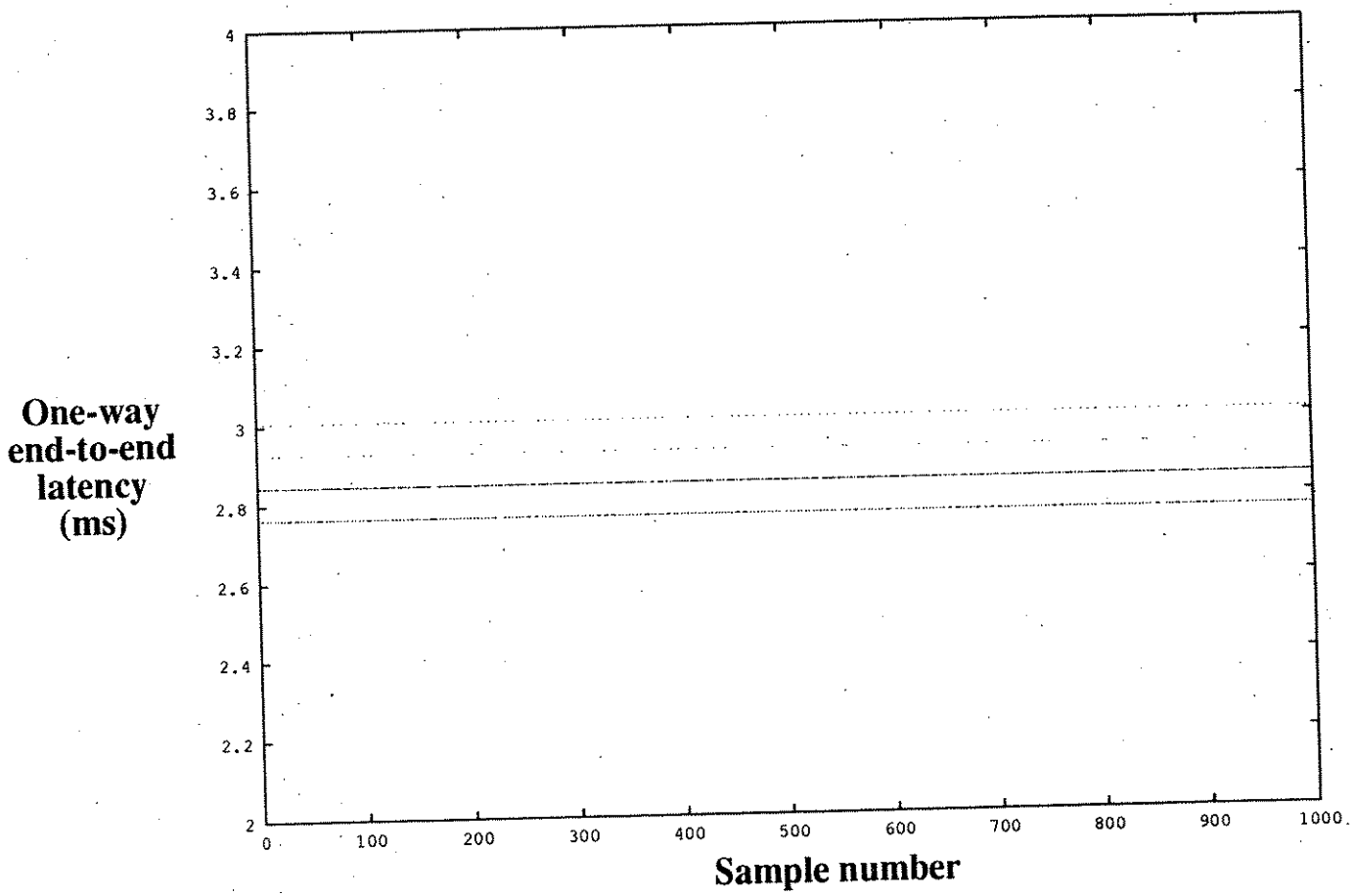
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
1% packet loss
Voice data in FDDI synchronous class

Average latency: 7.779 ms
99.9% threshold: 7.967 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 8 bytes



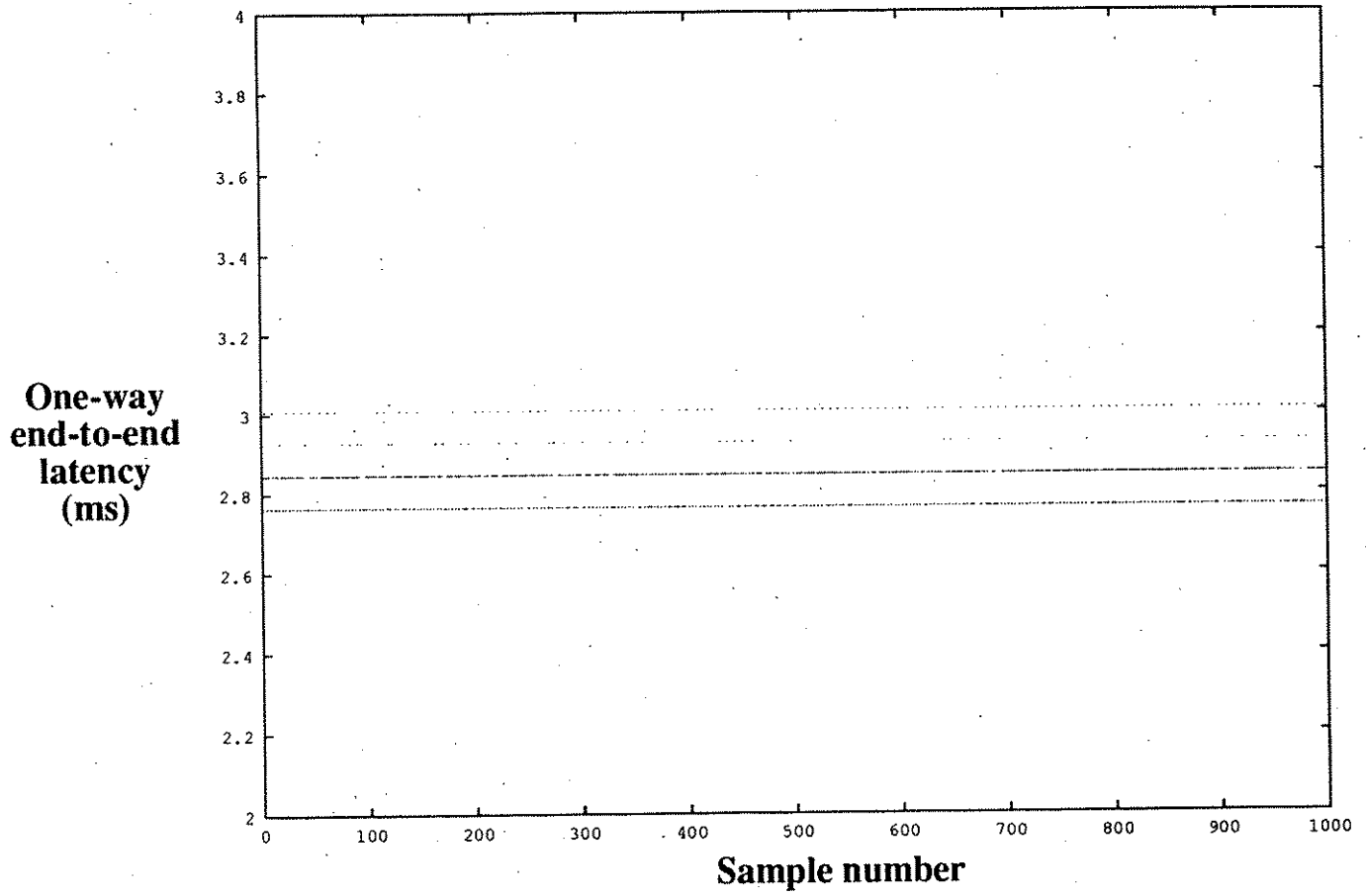
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
5% packet loss
Voice data in FDDI synchronous class

Average latency: 2.837 ms
99.9% threshold: 3.008 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 16 bytes



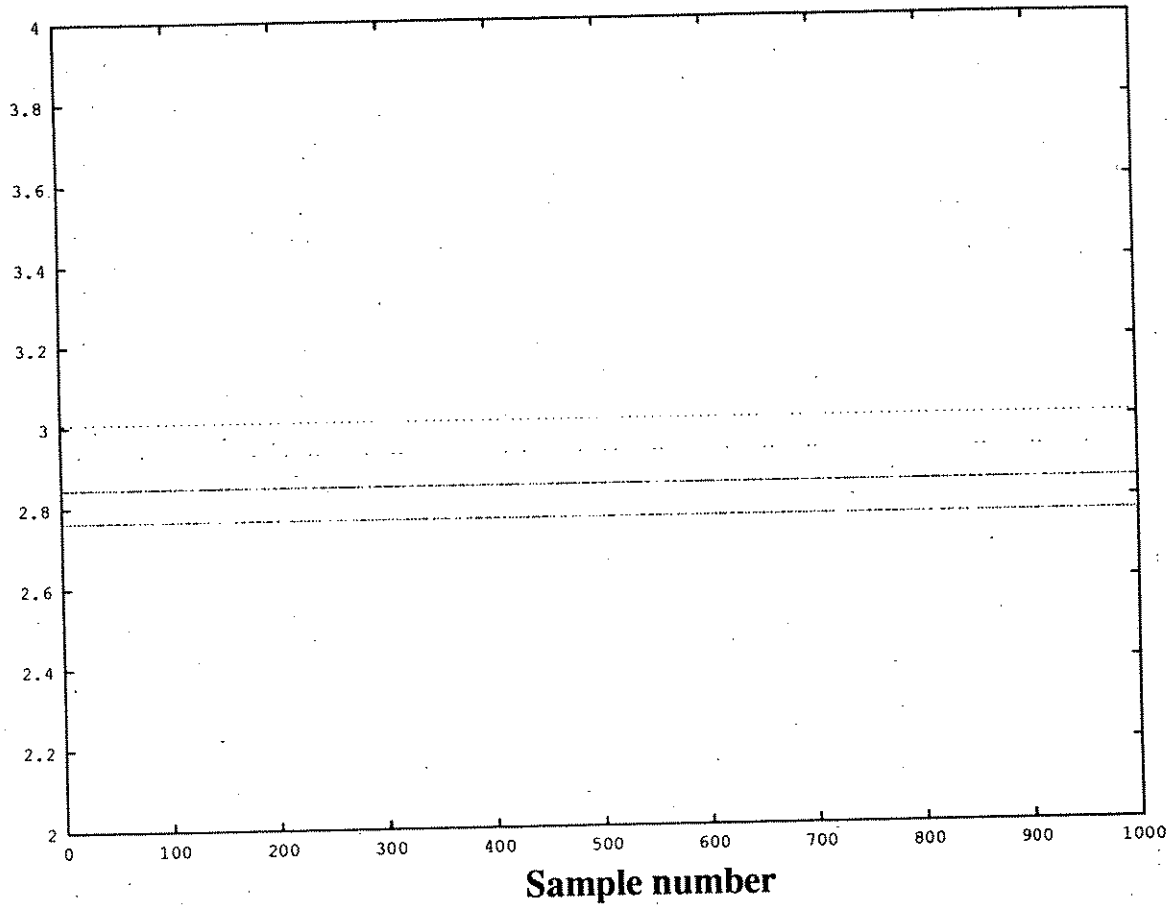
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
5% packet loss
Voice data in FDDI synchronous class

Average latency: 2.837 ms
99.9% threshold: 3.008 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 32 bytes



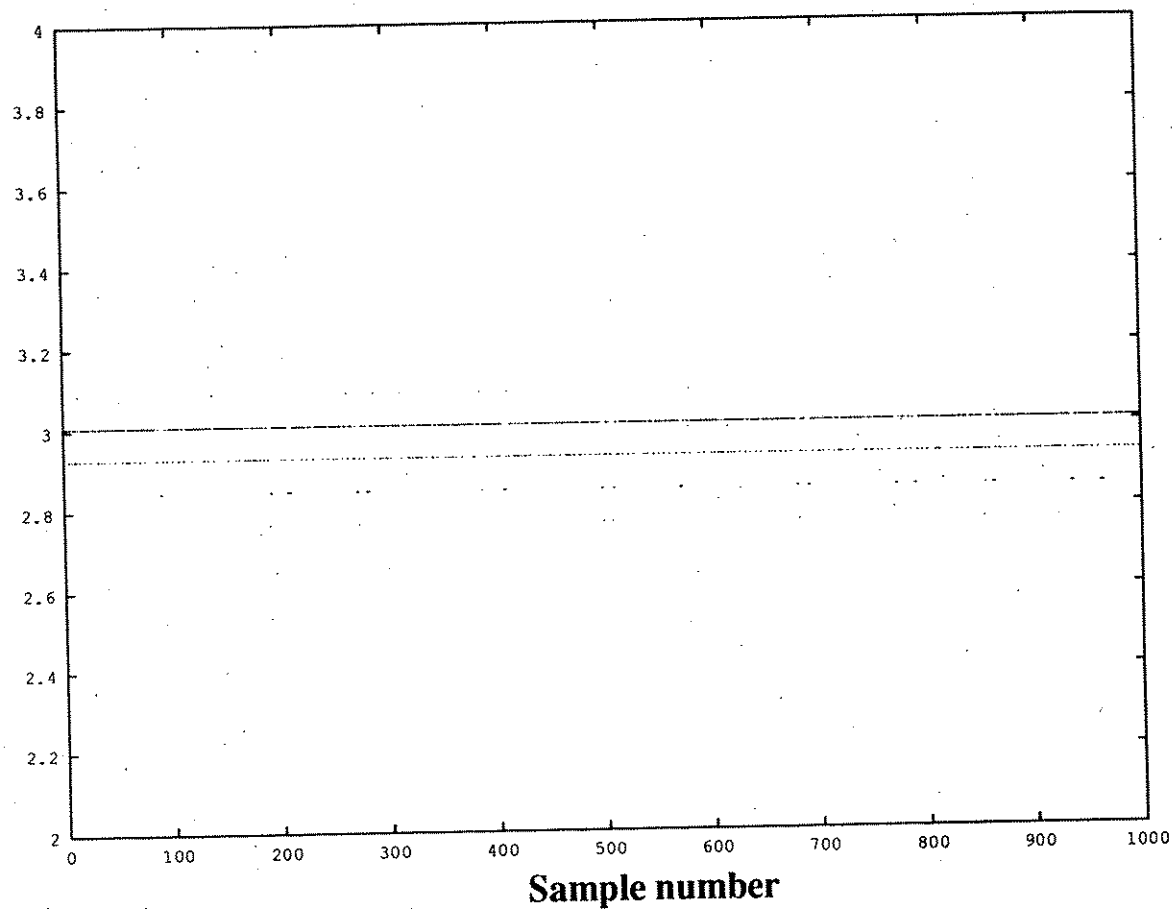
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
5% packet loss
Voice data in FDDI synchronous class

Average latency: 2.838 ms
99.9% threshold: 3.008 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 64 bytes



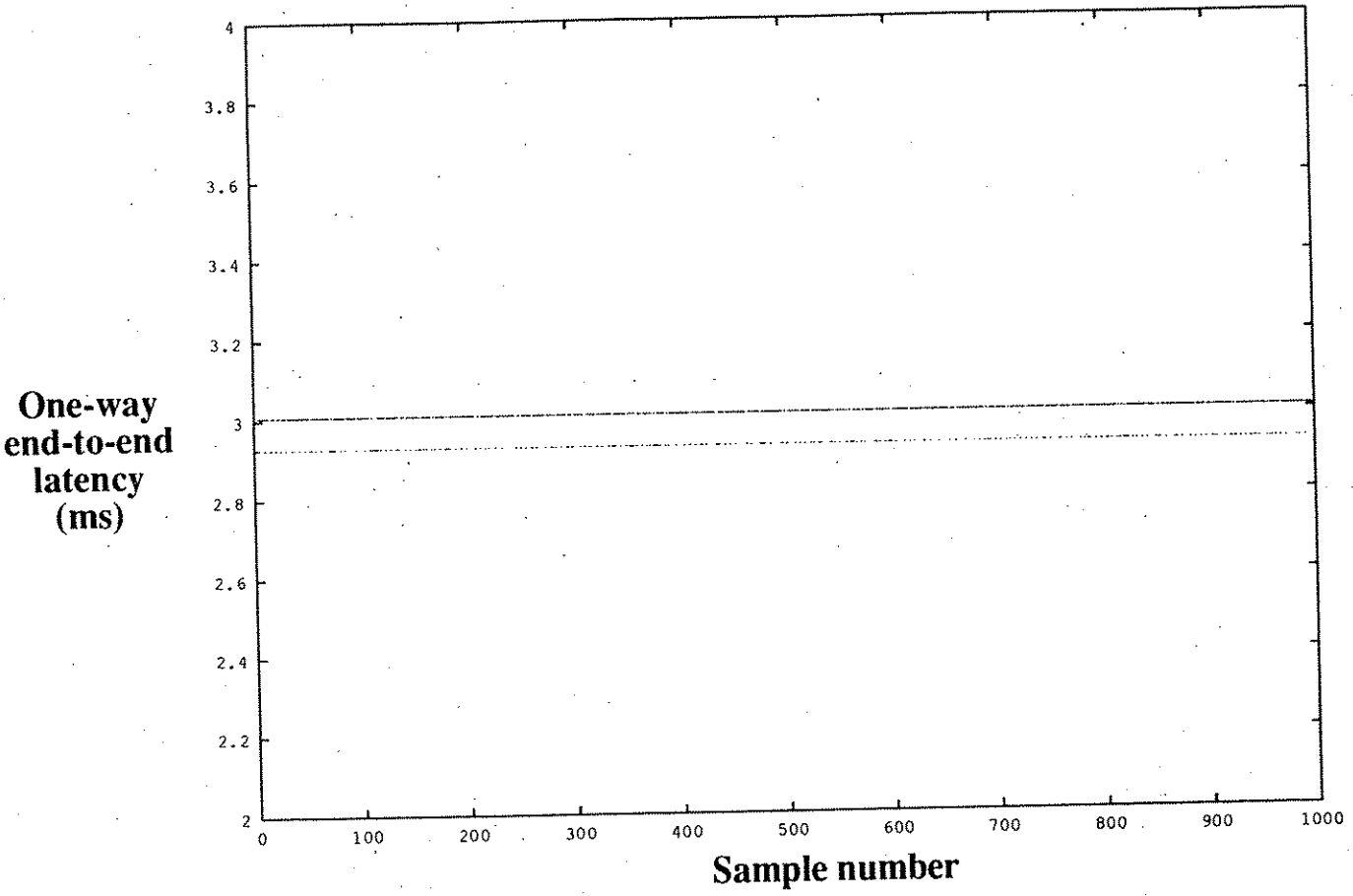
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
5% packet loss
Voice data in FDDI synchronous class

Average latency: 2.979 ms
99.9% threshold: 3.089 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 128 bytes



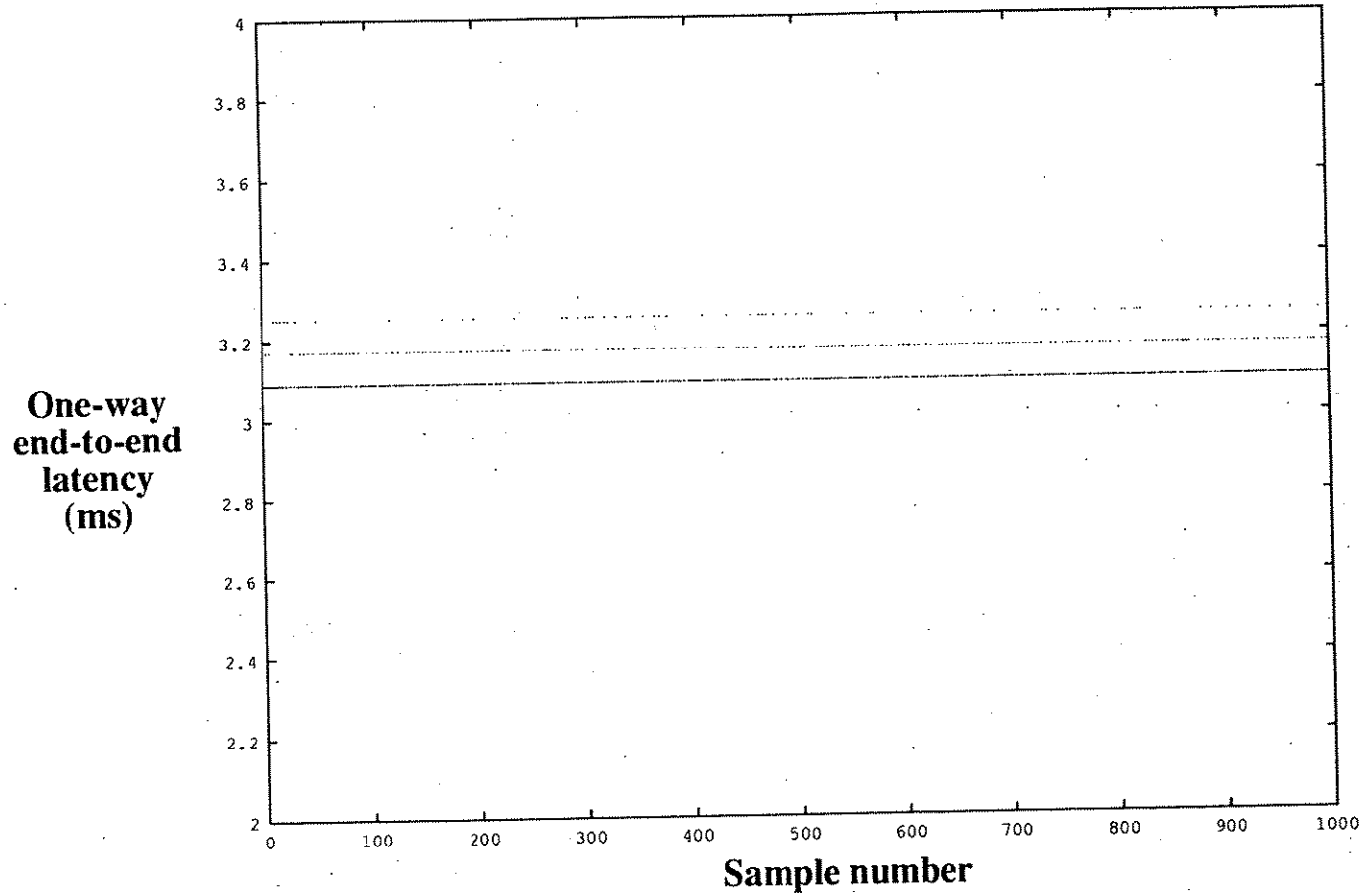
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
5% packet loss
Voice data in FDDI synchronous class

Average latency: 2.988 ms
99.9% threshold: 3.089 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 256 bytes



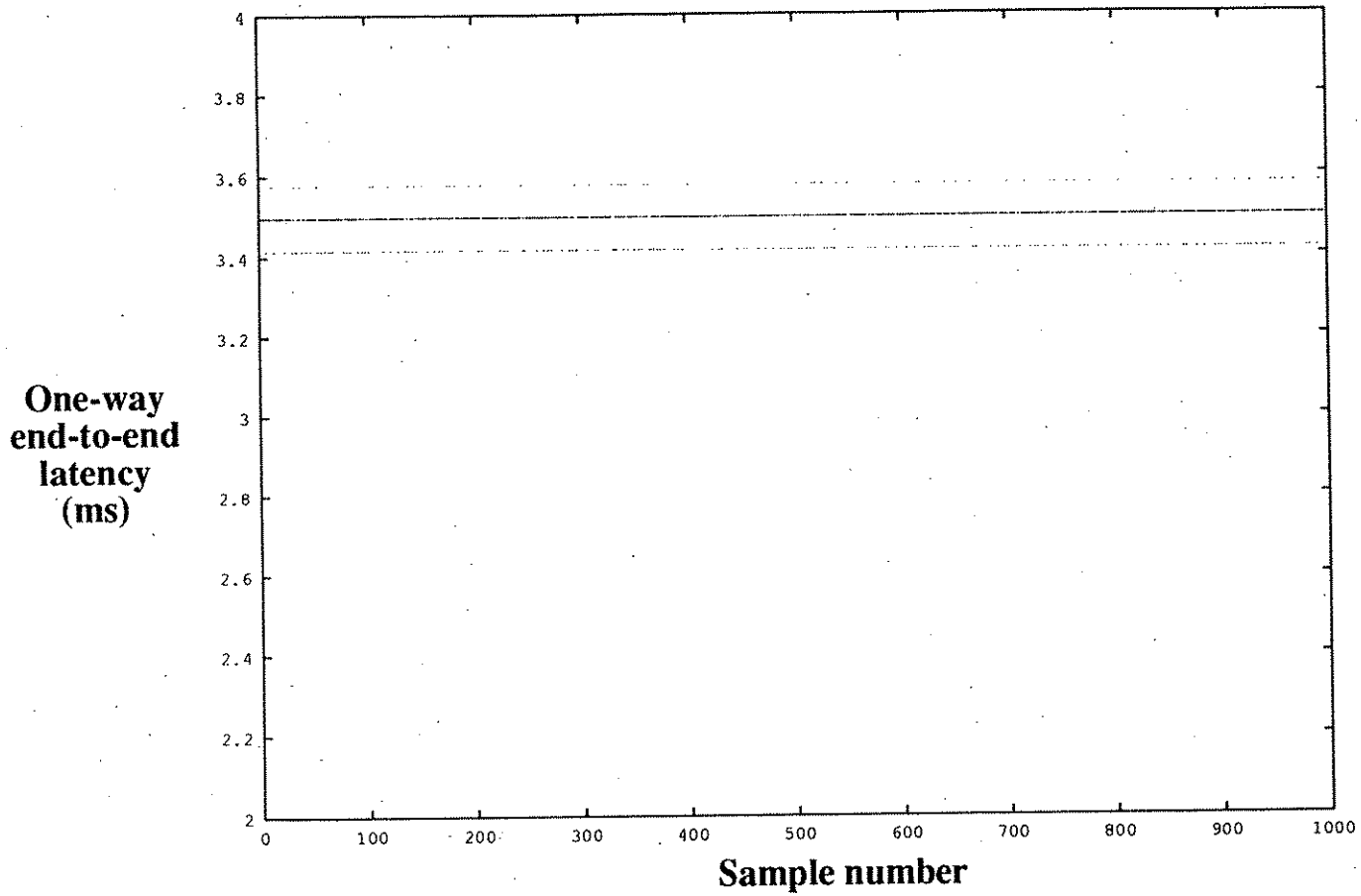
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
5% packet loss
Voice data in FDDI synchronous class

Average latency: 3.117 ms
99.9% threshold: 3.252 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 512 bytes



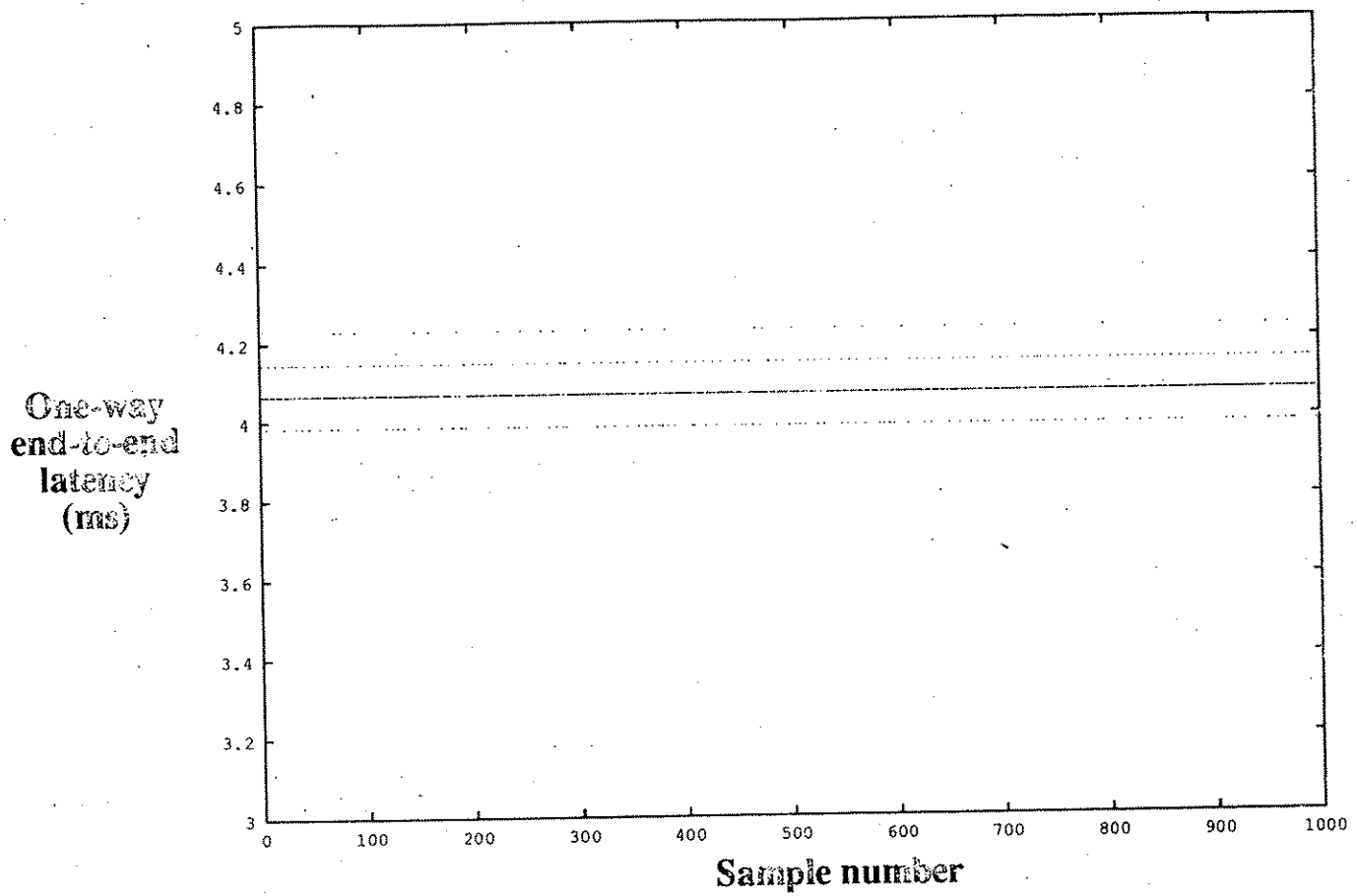
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
5% packet loss
Voice data in FDDI synchronous class

Average latency: 3.487 ms
99.9% threshold: 3.577 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



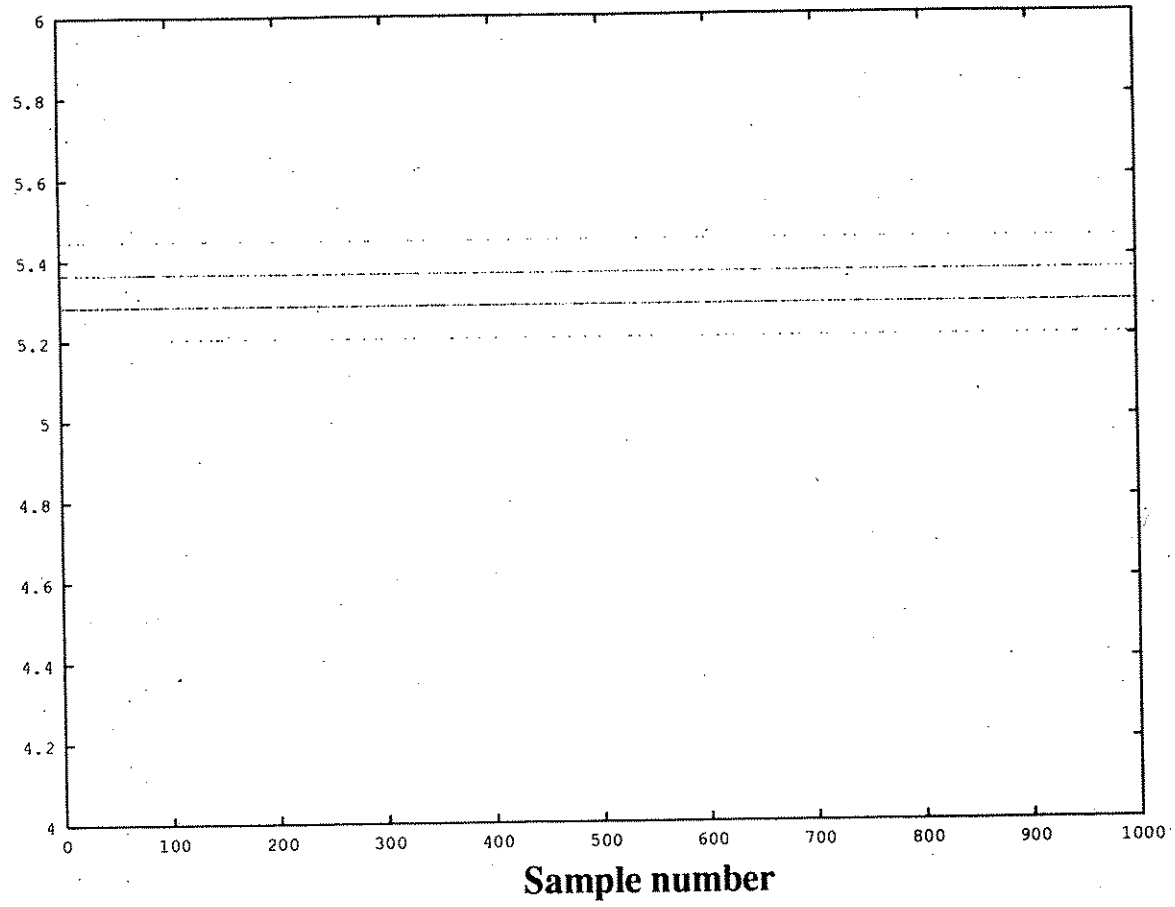
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
5% packet loss
Voice data in FDDI synchronous class

Average latency: 4.070 ms
99.9% threshold: 4.228 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



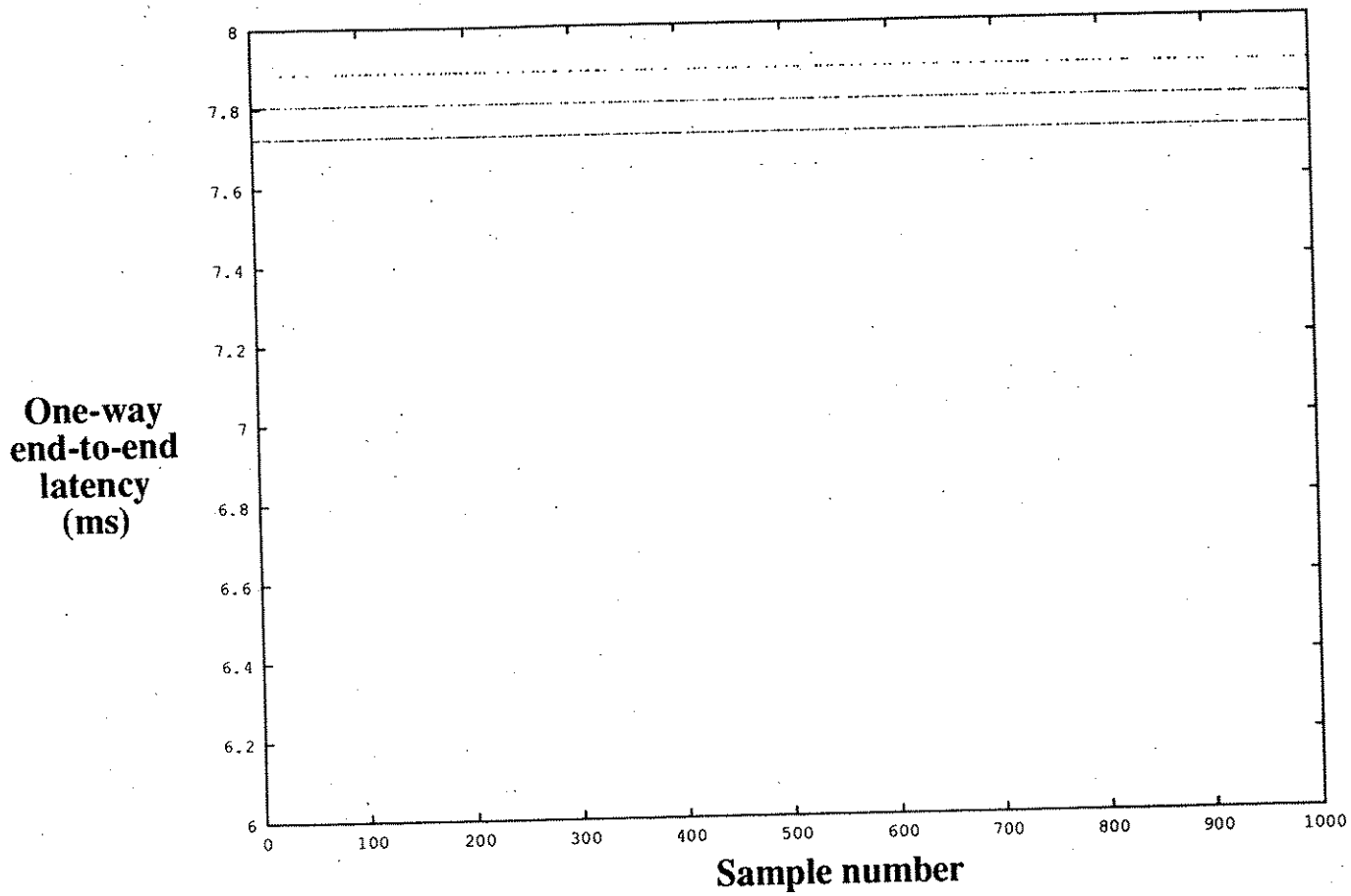
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
5% packet loss
Voice data in FDDI synchronous class

Average latency: 5.317 ms
99.9% threshold: 5.528 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
5% packet loss
Voice data in FDDI synchronous class

Average latency: 7.776 msecs
99.9% threshold: 7.886 msecs

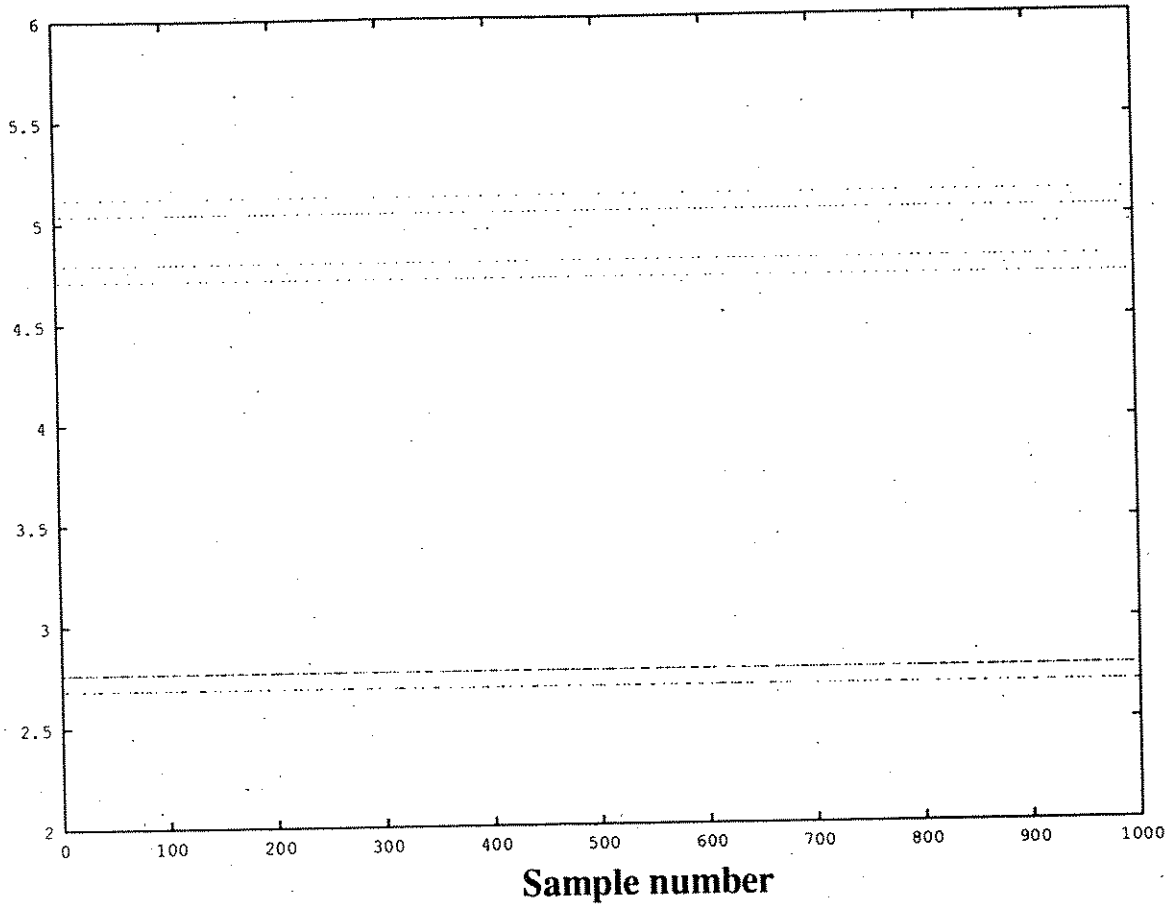
RETRANSMISSIONS

Appendix C

Experiment 3: Synchronous MPPT Background FDDI Load

JITTER MEASUREMENT

Voice Data Size: 8 bytes



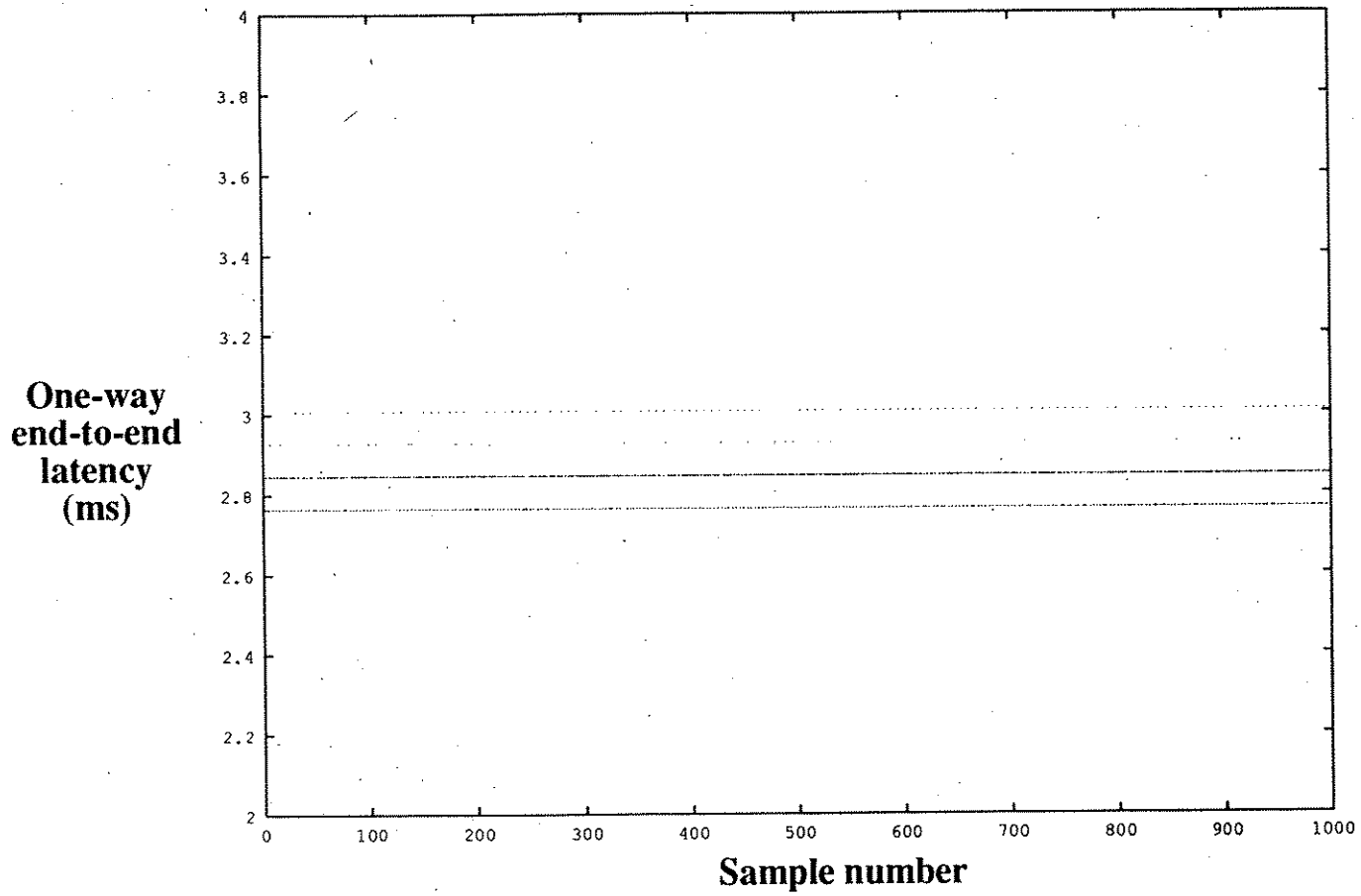
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Voice data in FDDI synchronous class

Average latency: 3.526 ms
99.9% threshold: 5.122 ms

SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

JITTER MEASUREMENT

Voice Data Size: 8 bytes



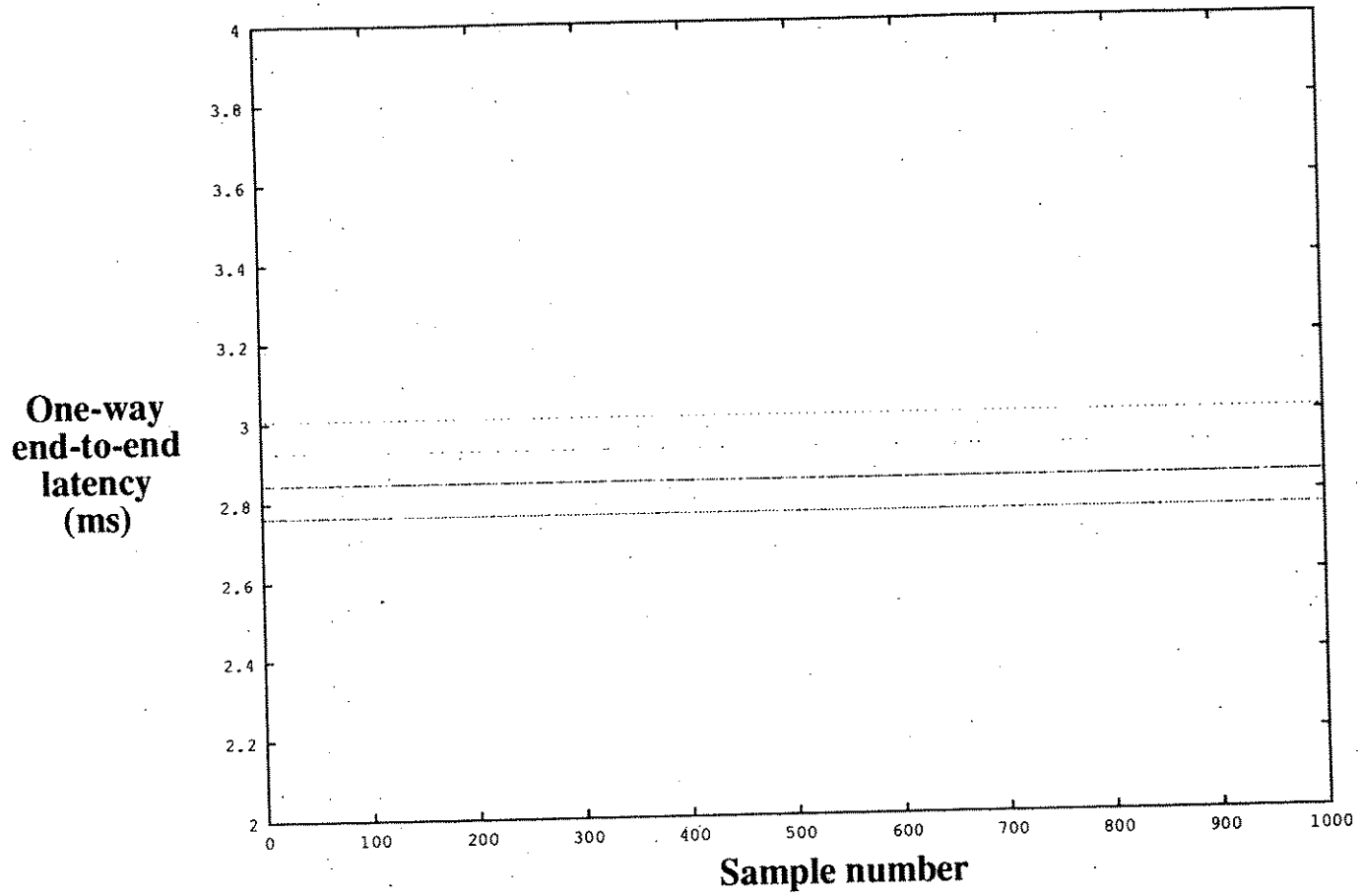
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
10% packet loss
Voice data in FDDI synchronous class

Average latency: 2.838 ms
99.9% threshold: 3.008 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 16 bytes



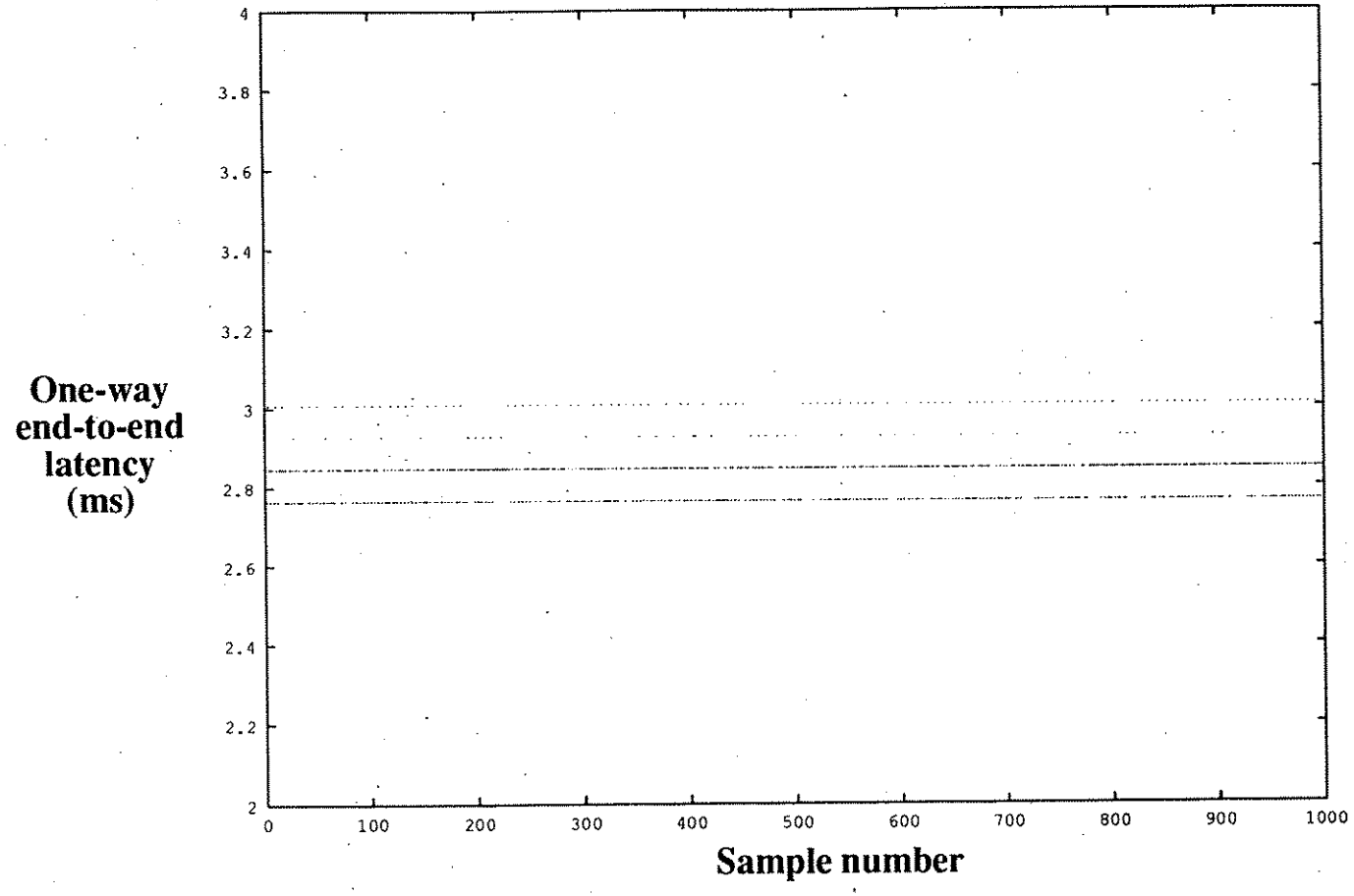
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
10% packet loss
Voice data in FDDI synchronous class

Average latency: 2.838 ms
99.9% threshold: 3.008 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 32 bytes



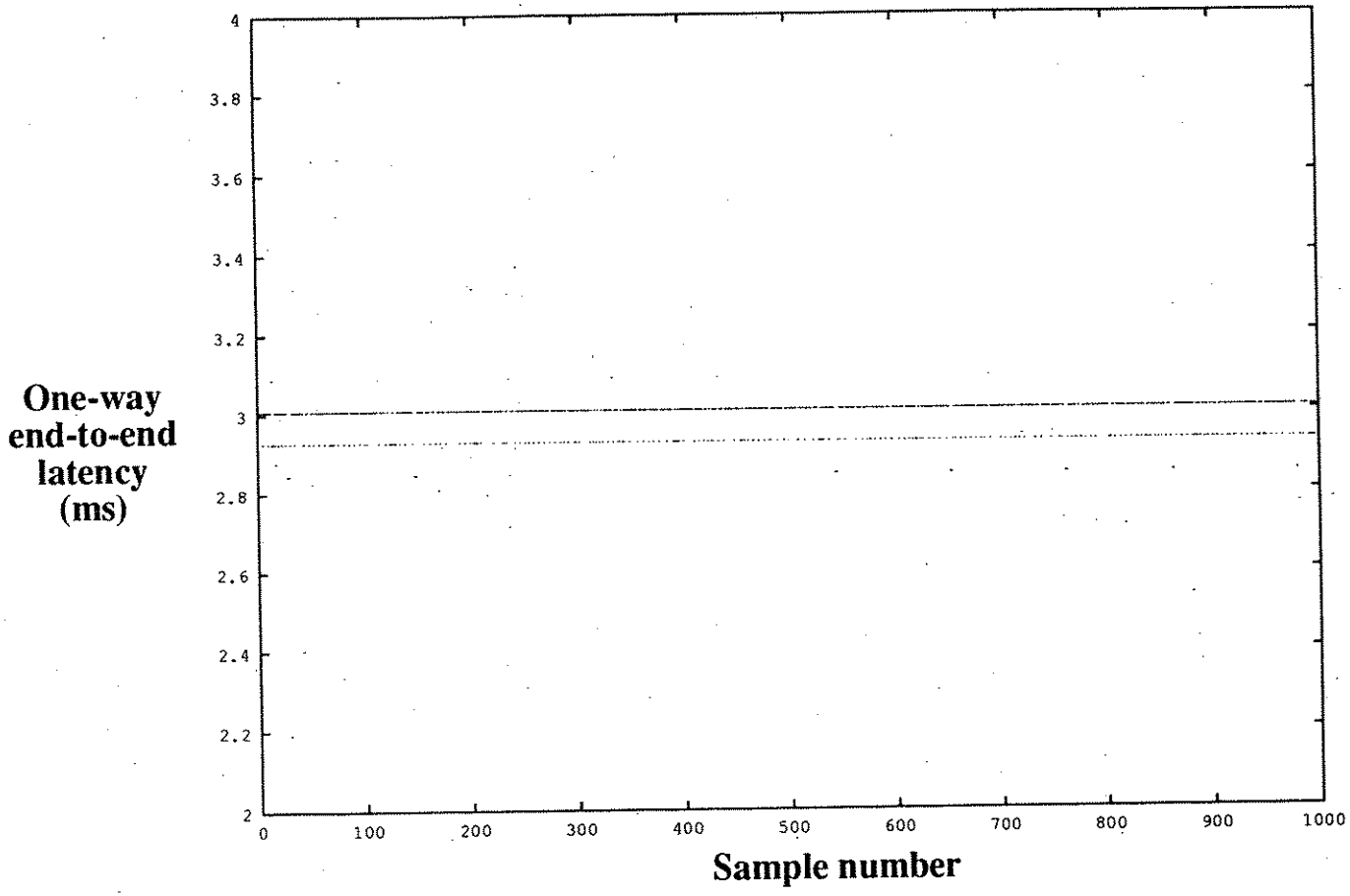
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
10% packet loss
Voice data in FDDI synchronous class

Average latency: 2.836 ms
99.9% threshold: 3.008 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 64 bytes



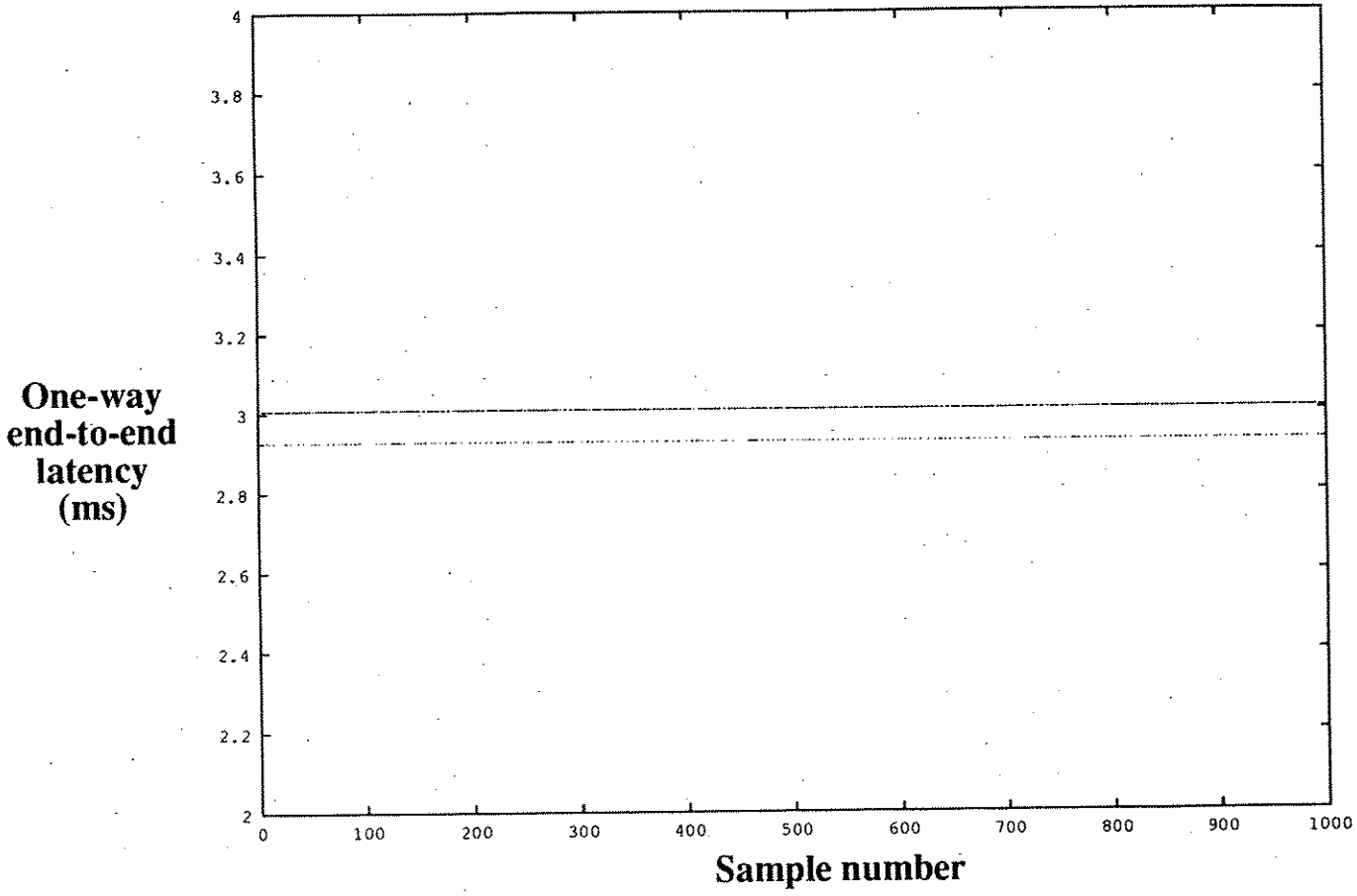
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
10% packet loss
Voice data in FDDI synchronous class

Average latency: 2.985 ms
99.9% threshold: 3.089 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 128 bytes



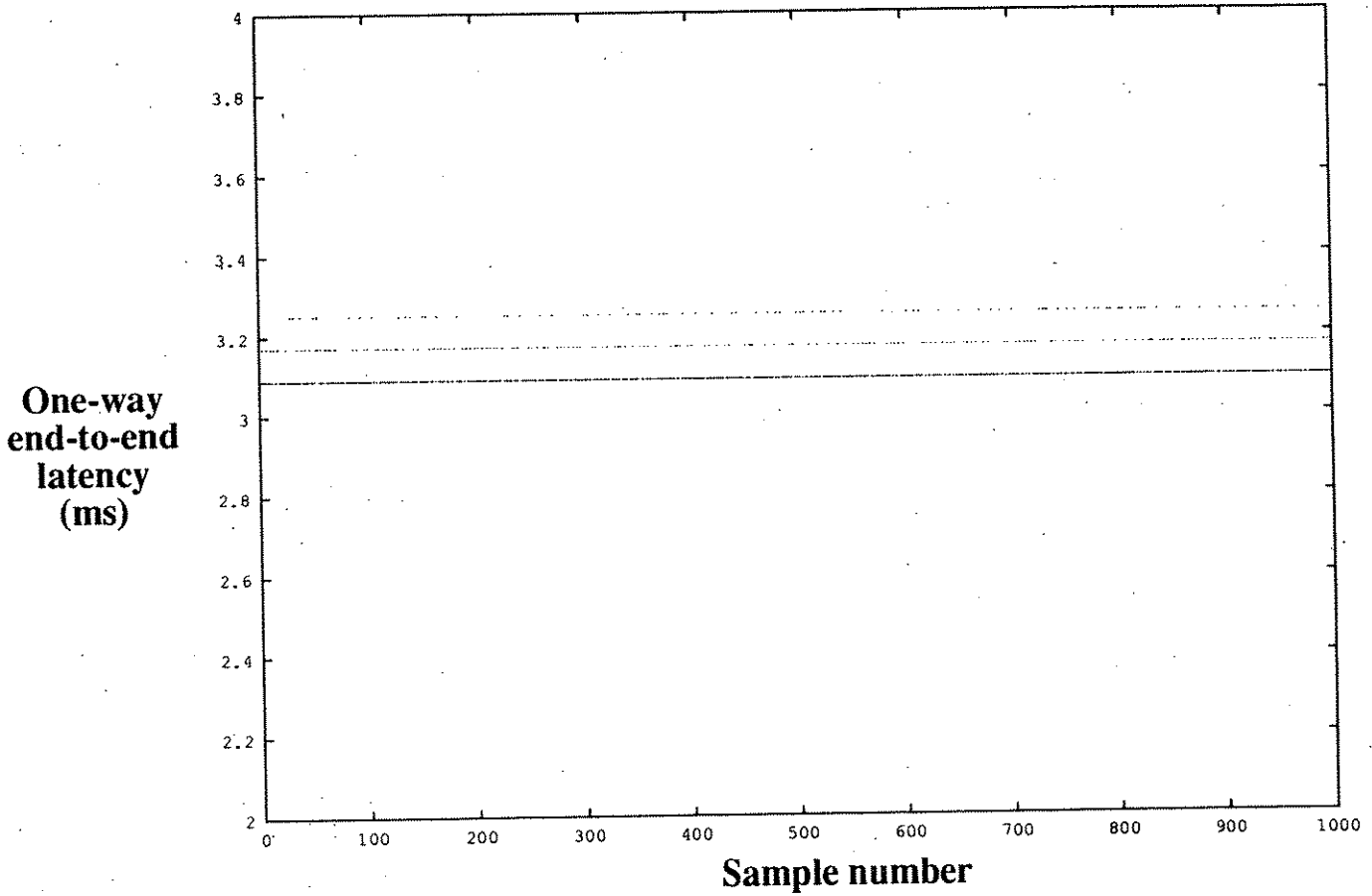
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
10% packet loss
Voice data in FDDI synchronous class

Average latency: 2.988 ms
99.9% threshold: 3.089 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 256 bytes



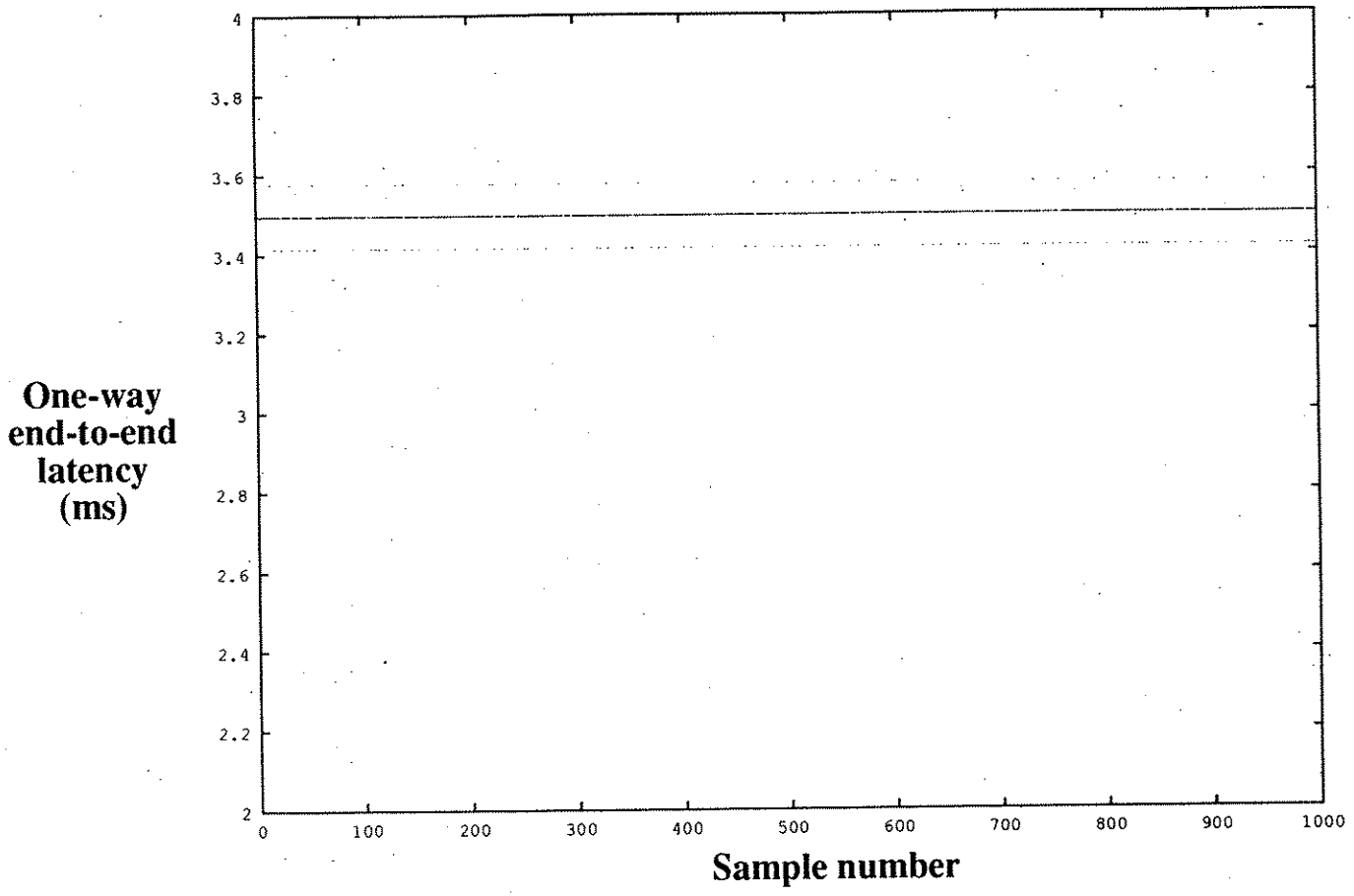
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
10% packet loss
Voice data in FDDI synchronous class

Average latency: 3.120 ms
99.9% threshold: 3.252 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 512 bytes



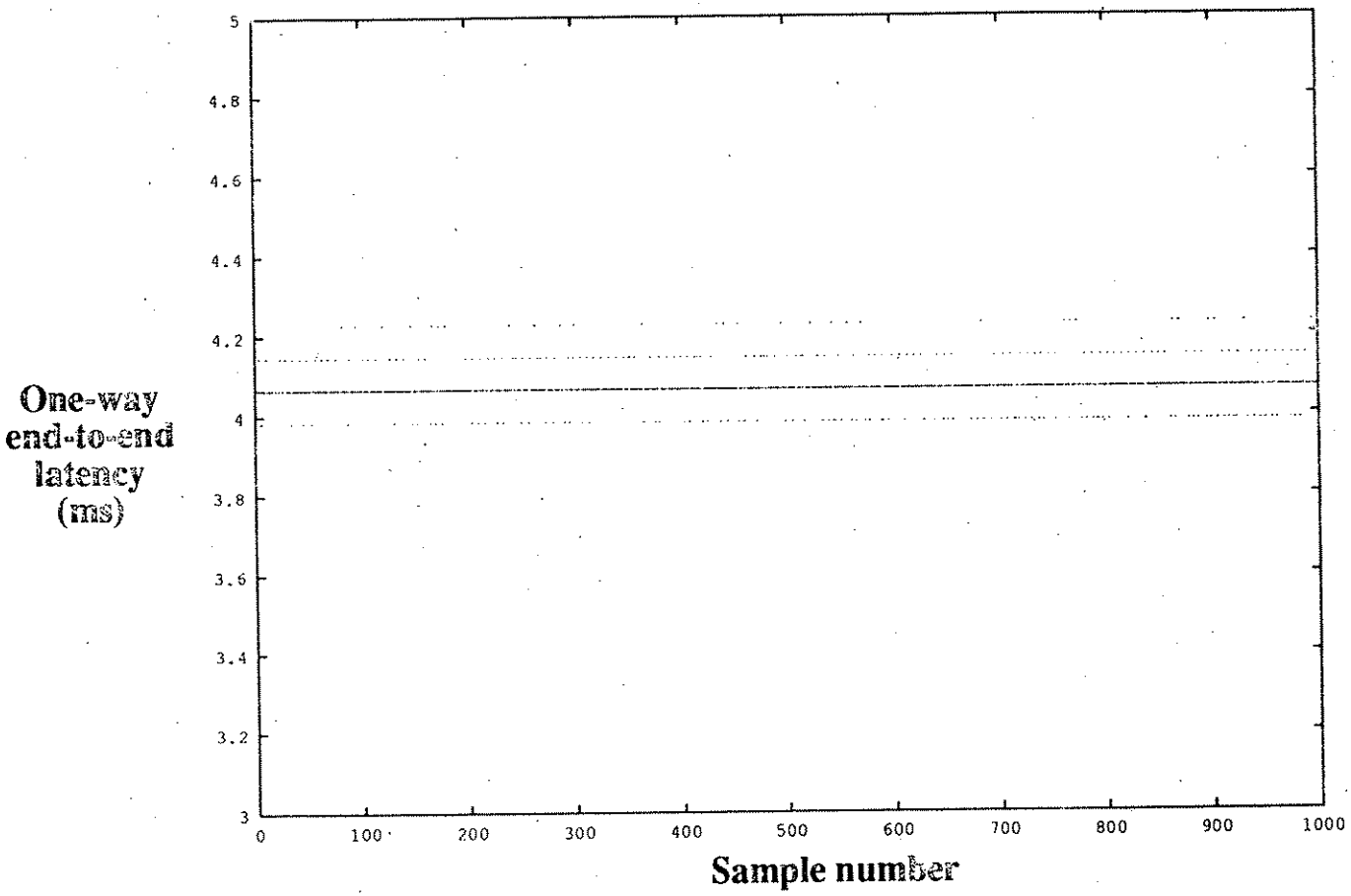
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
10% packet loss
Voice data in FDDI synchronous class

Average latency: 3.491 ms
99.9% threshold: 3.577 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



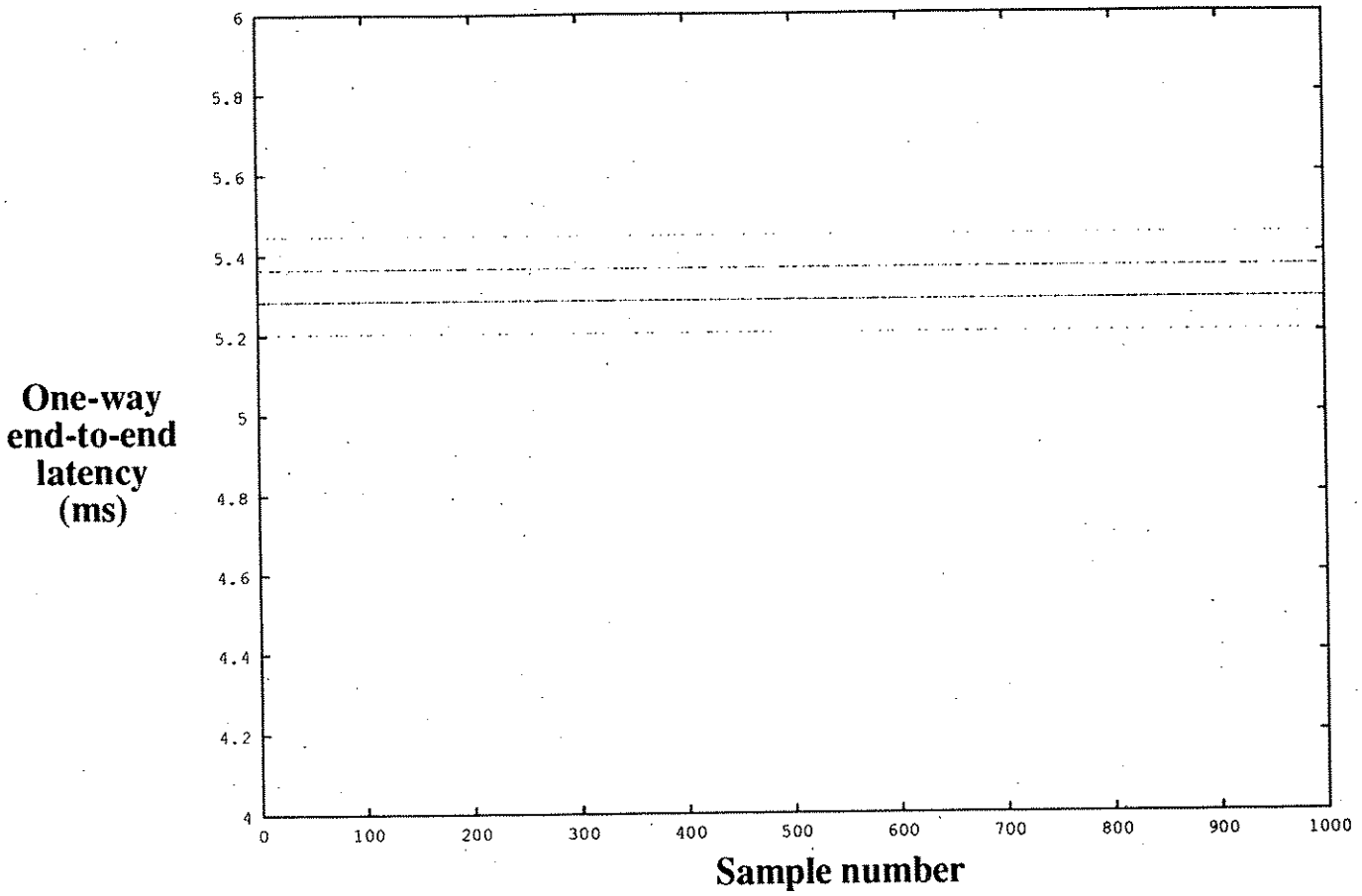
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
10% packet loss
Voice data in FDDI synchronous class

Average latency: 4.071 ms
99.9% threshold: 4.228 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



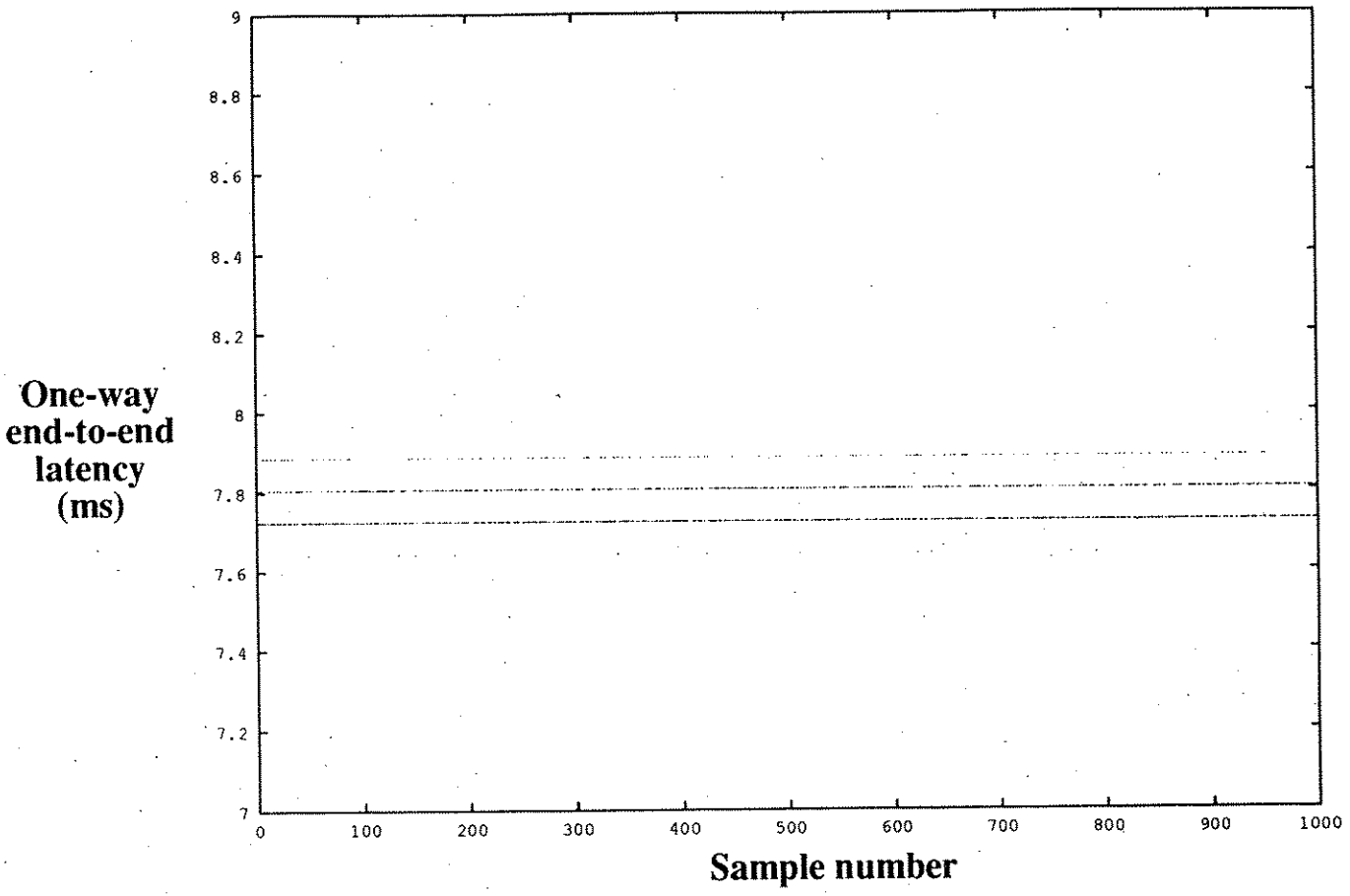
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
10% packet loss
Voice data in FDDI synchronous class

Average latency: 5.313 ms
99.9% threshold: 5.447 ms

RETRANSMISSIONS

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
10% packet loss
Voice data in FDDI synchronous class

Average latency: 7.776 ms
99.9% threshold: 8.049 ms

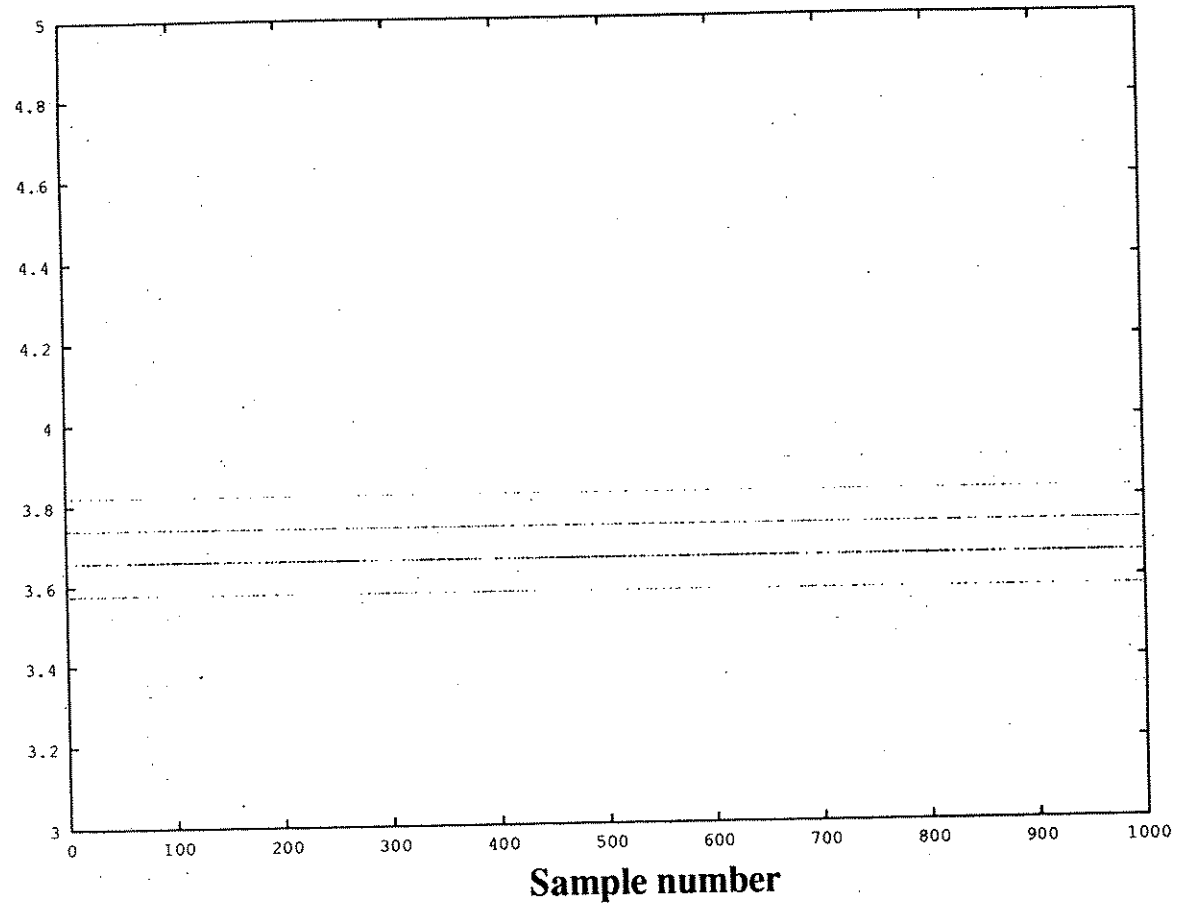
RETRANSMISSIONS

Appendix E

Experiment 5: Background Asynchronous Processor Load

JITTER MEASUREMENT

Voice Data Size: 8 bytes



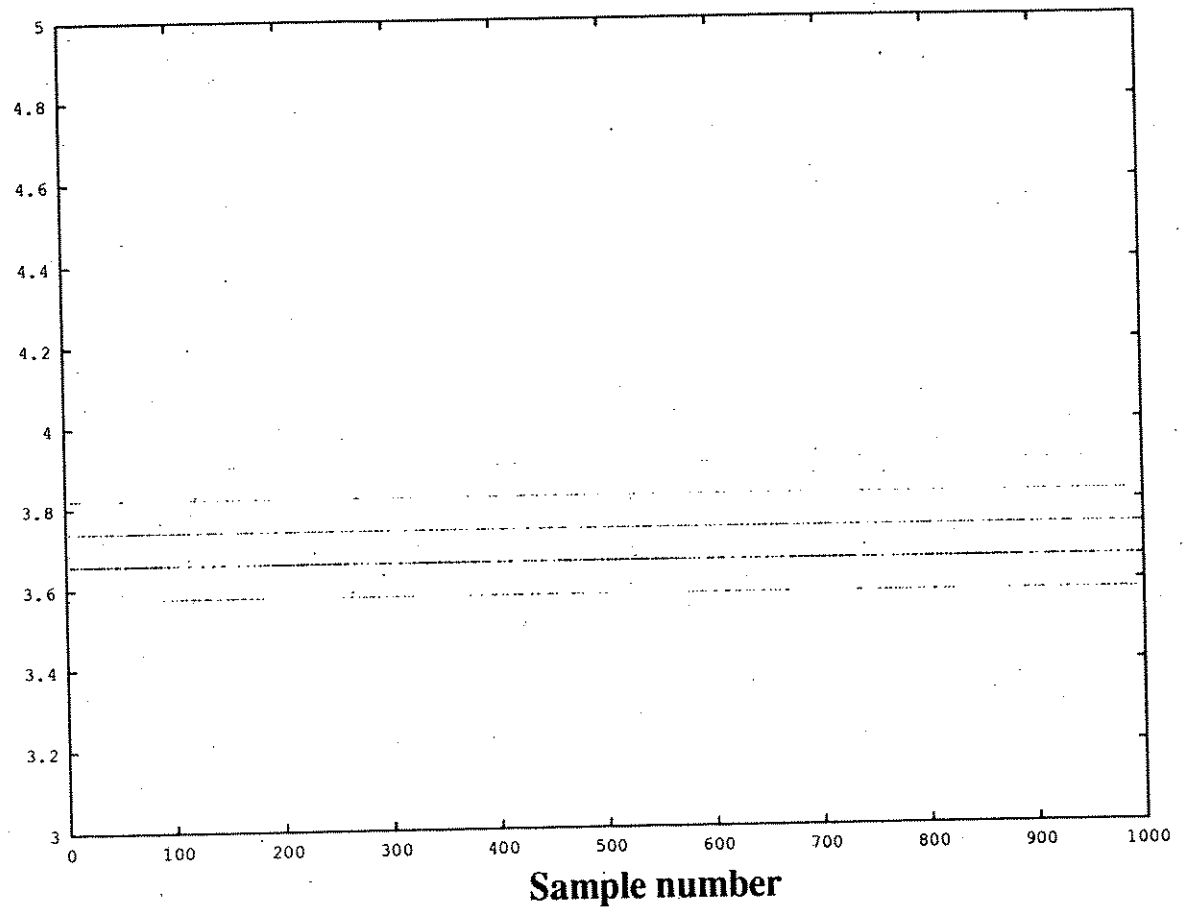
1000 samples
120 msg/sec asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 3.689 ms
99.9% threshold: 3.984 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 16 bytes



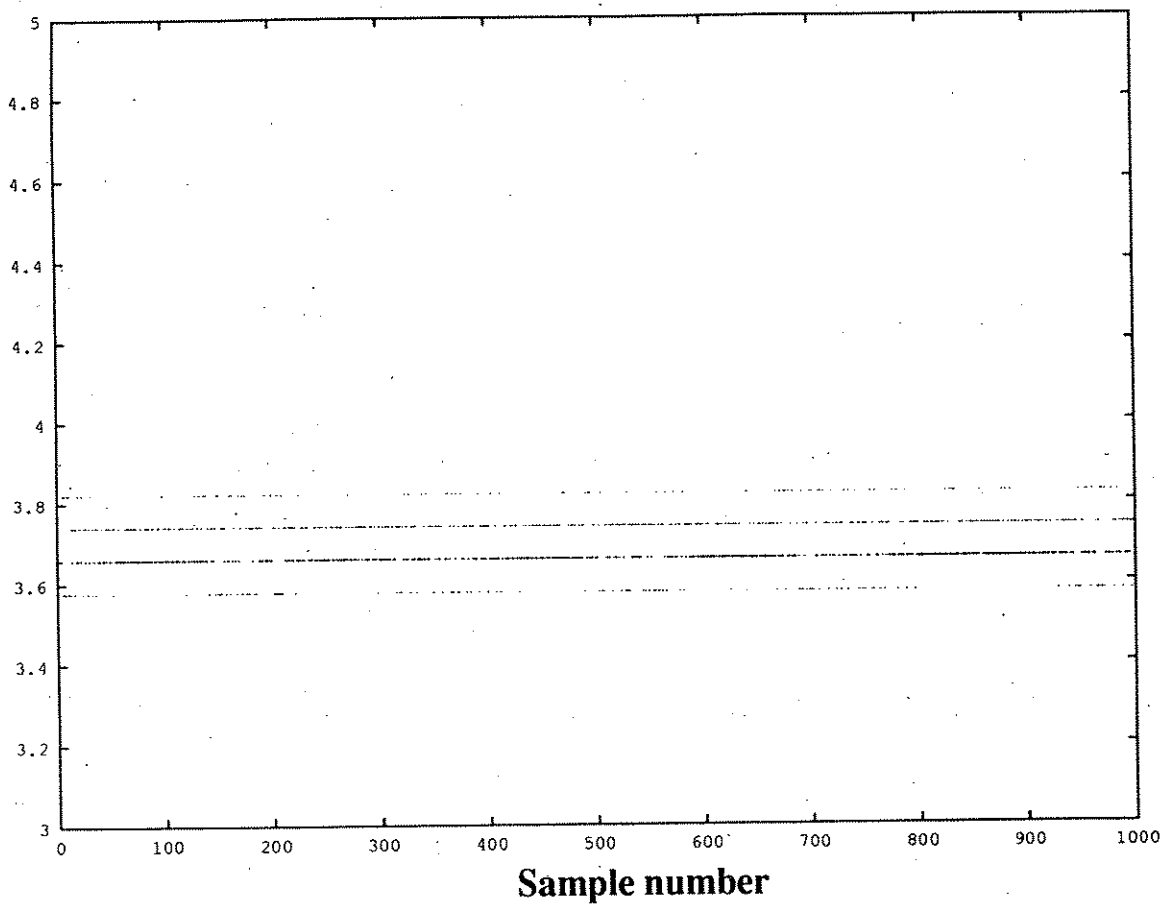
1000 samples
120 msg/sec asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 3.688 ms
99.9% threshold: 3.902 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 32 bytes



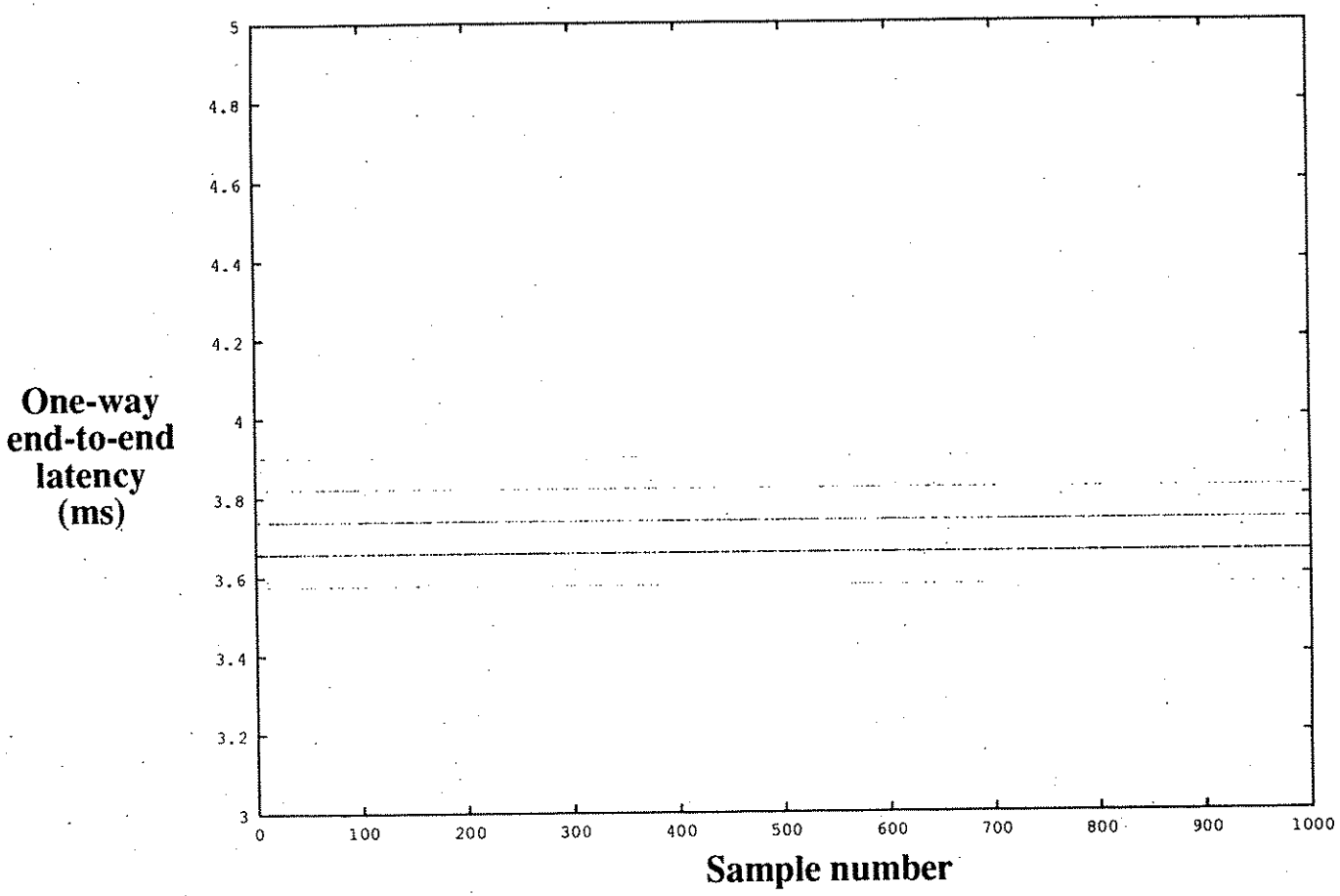
1000 samples
120 msg/sec asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: **3.693 ms**
99.9% threshold: **3.902 ms**

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 64 bytes



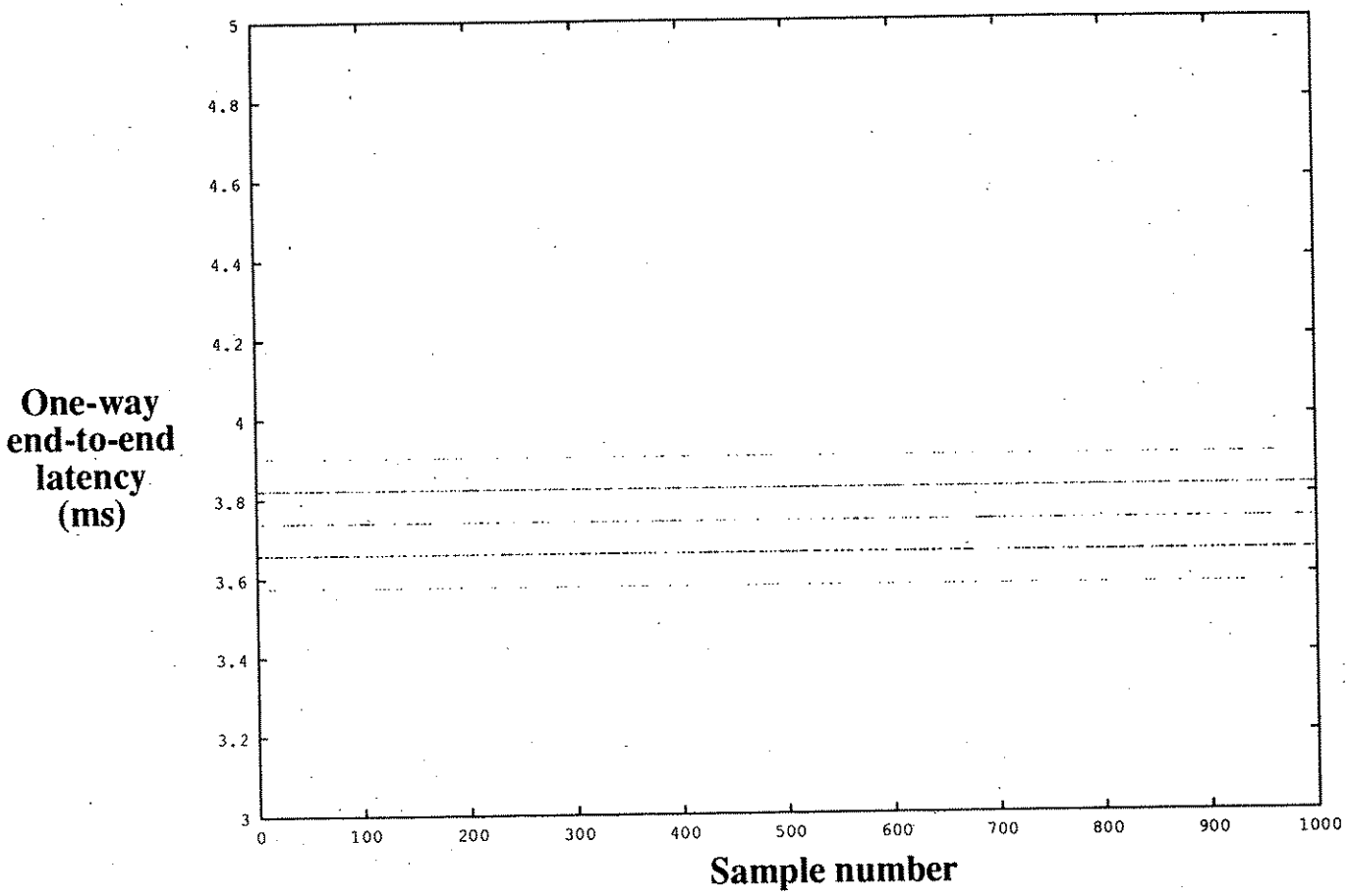
1000 samples
120 msg/sec asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 3.709 ms
99.9% threshold: 3.902 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 128 bytes



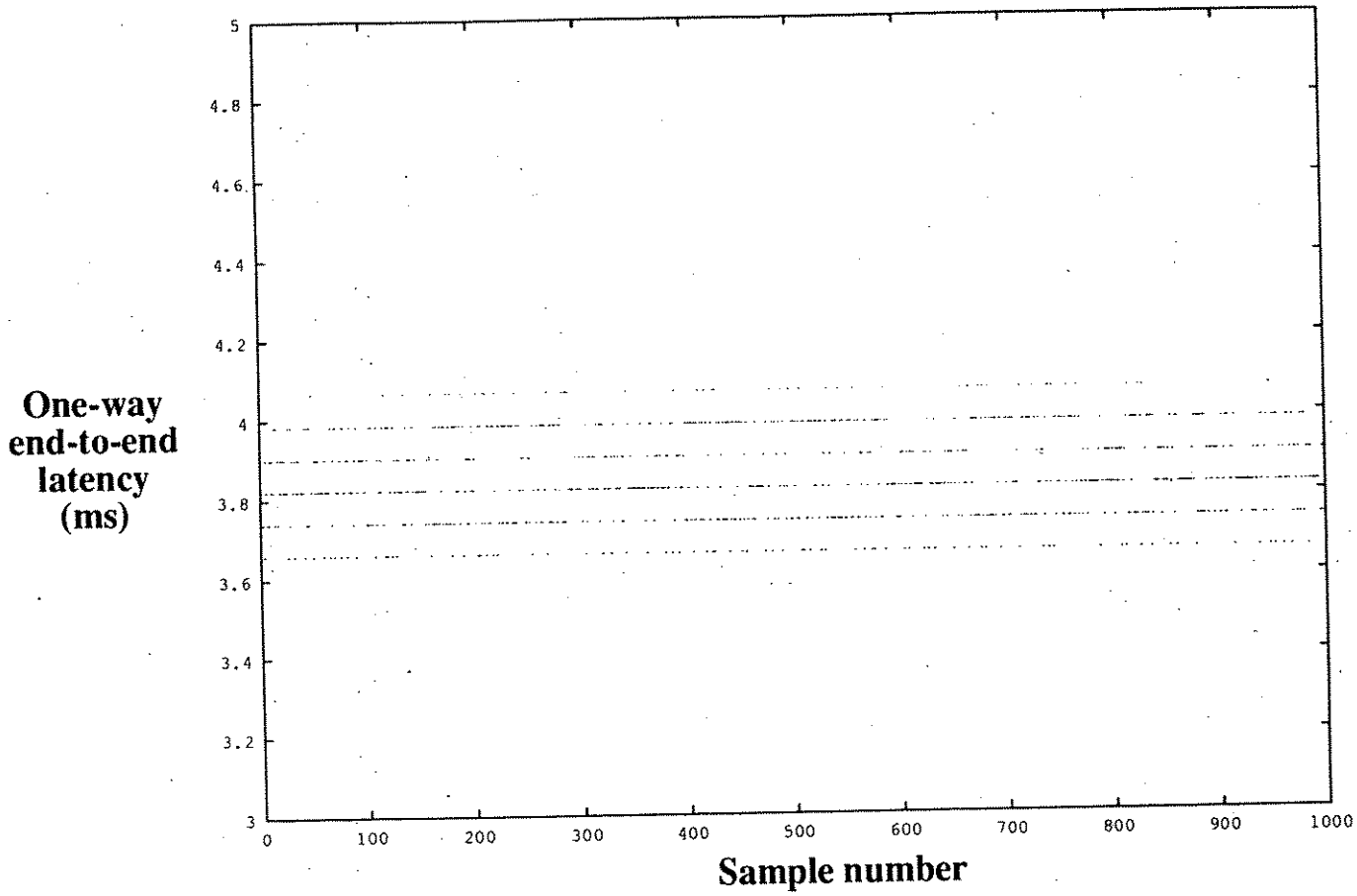
1000 samples
120 msg/sec asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 3.745 ms
99.9% threshold: 3.902 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 256 bytes



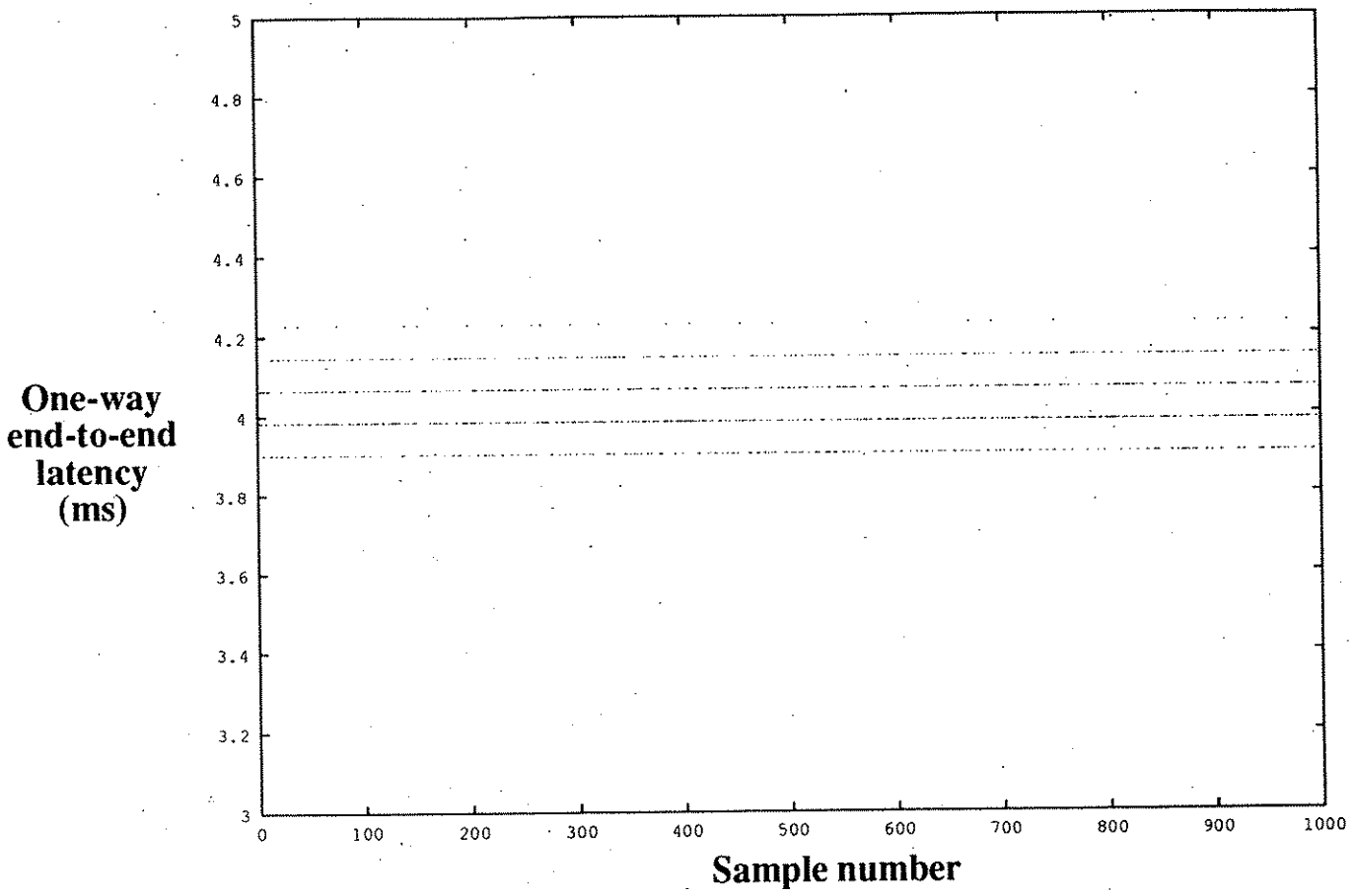
1000 samples
120 msg/sec asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 3.850 ms
99.9% threshold: 4.065 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 512 bytes



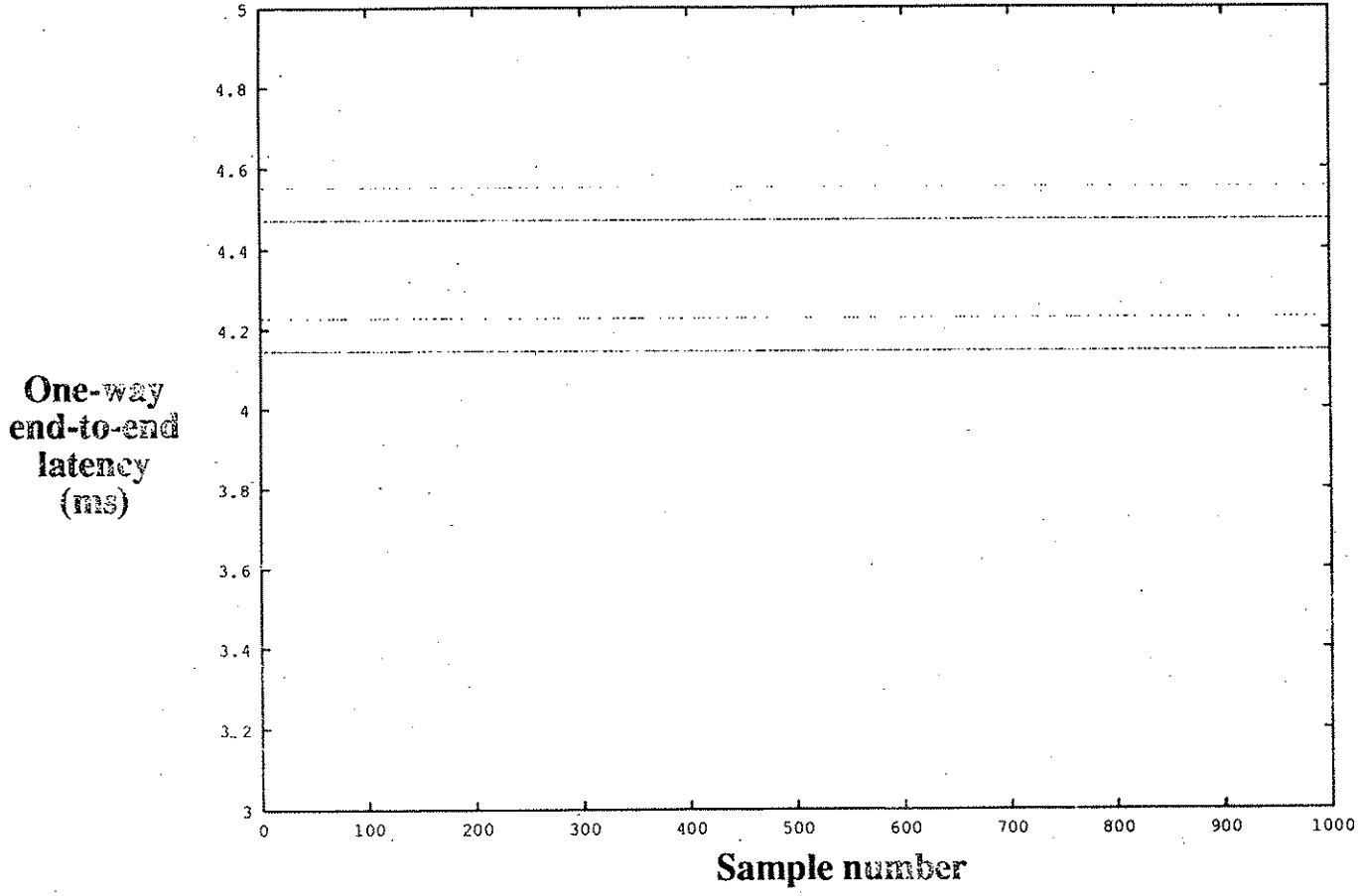
1000 samples
120 msg/sec asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 4.024 ms
99.9% threshold: 4.228 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



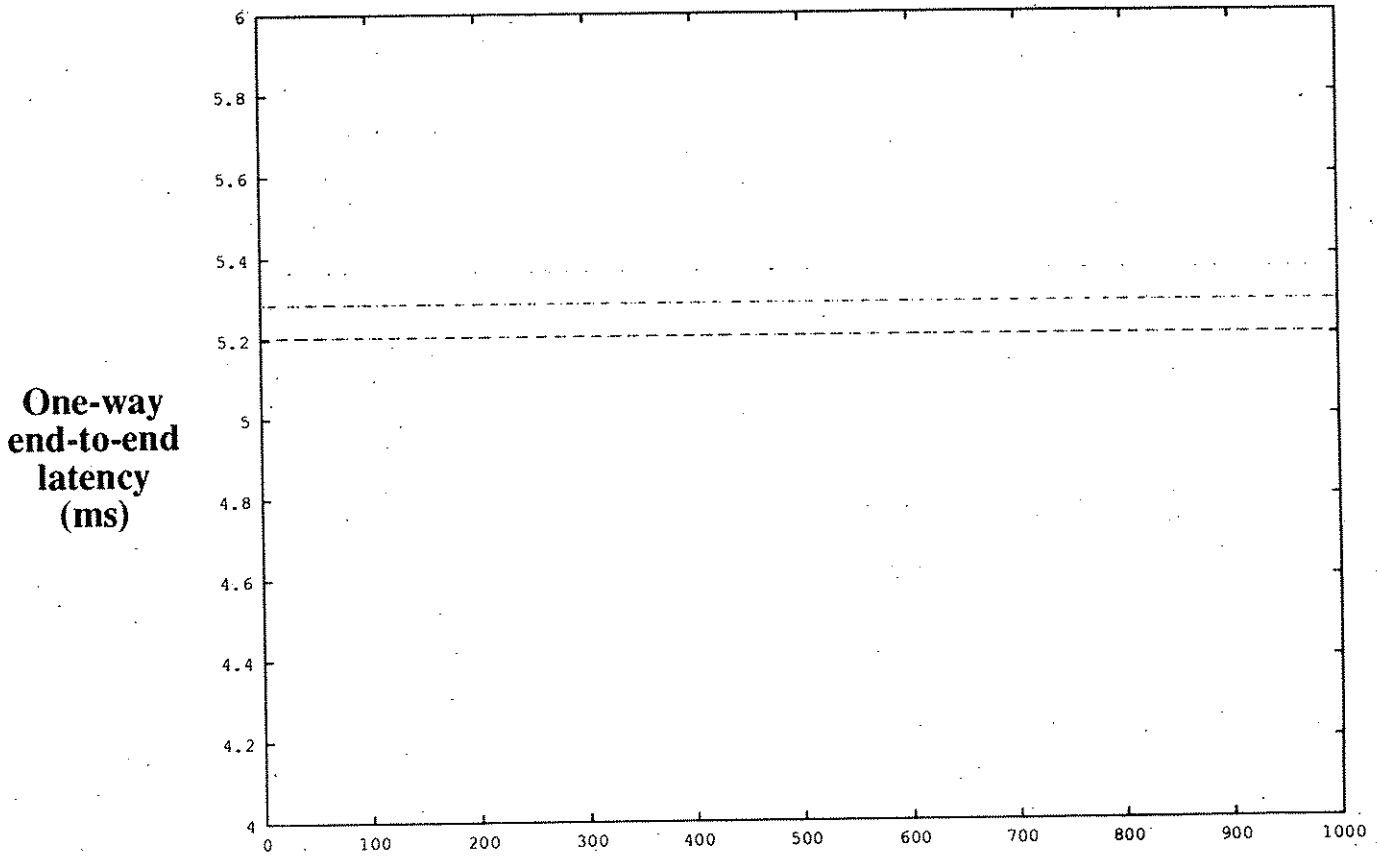
1000 samples
120 msg/sec asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 4.324 ms
99.9% threshold: 4.553 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



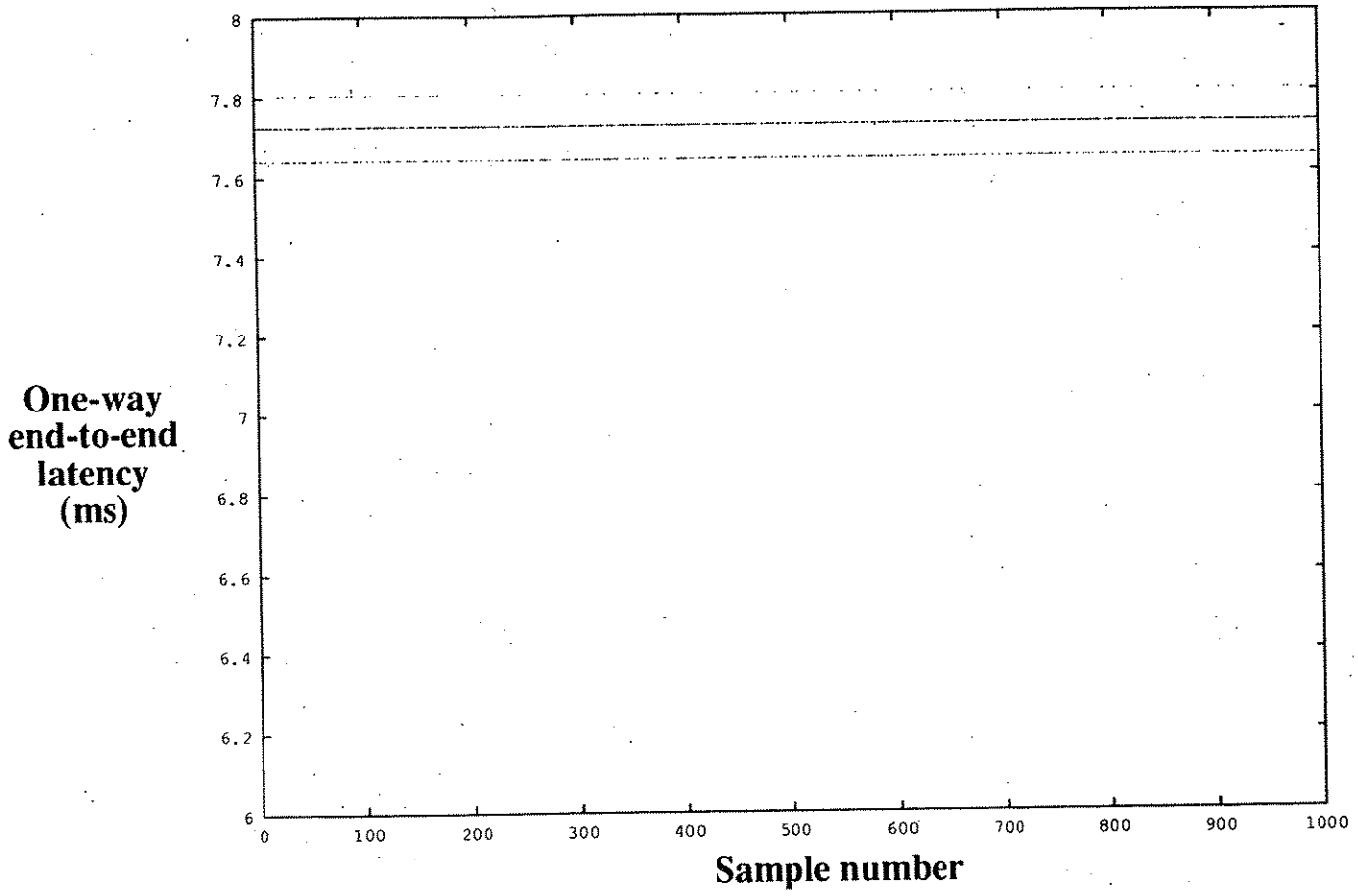
1000 samples
120 msg/sec asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 5.241 ms
99.9% threshold: 5.366 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



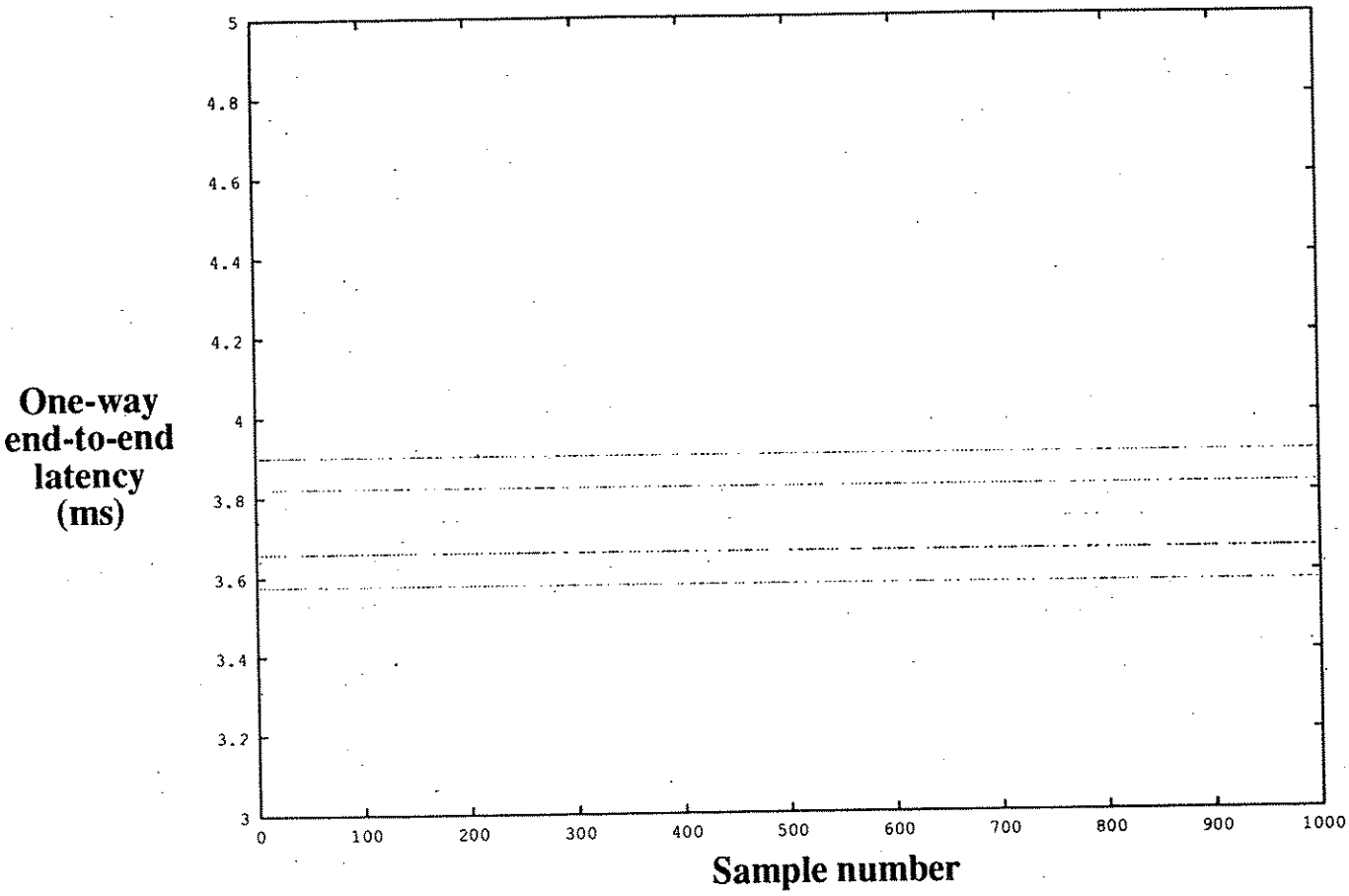
1000 samples
120 msg/sec asynchronous processor load
No background FDDI load
Voice data in FDDI synchronous class

Average latency: 7.704 ms
99.9% threshold: 7.805 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 8 bytes



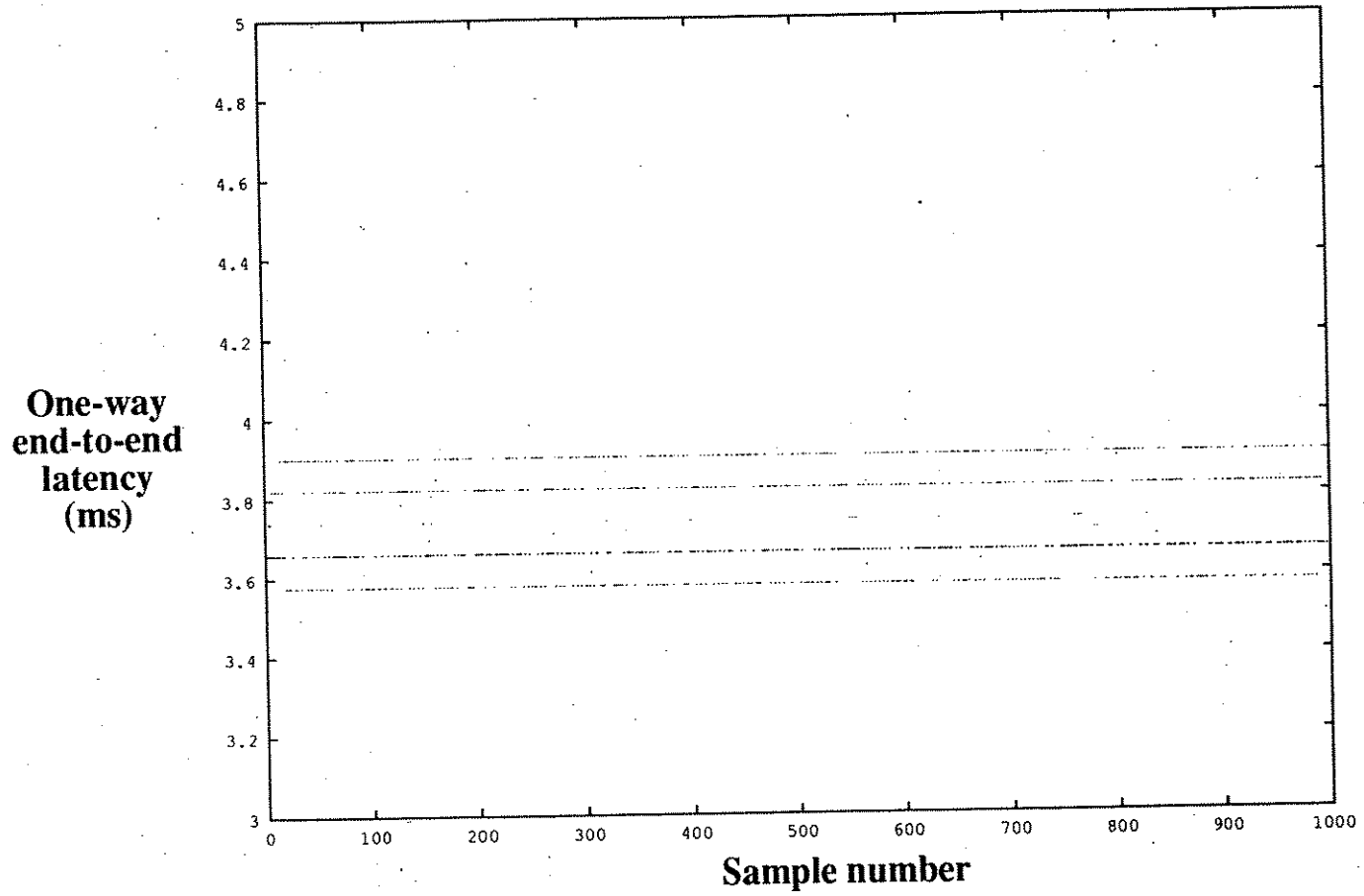
1000 samples
120 msg/sec asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.740 ms
99.9% threshold: 3.984 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 16 bytes



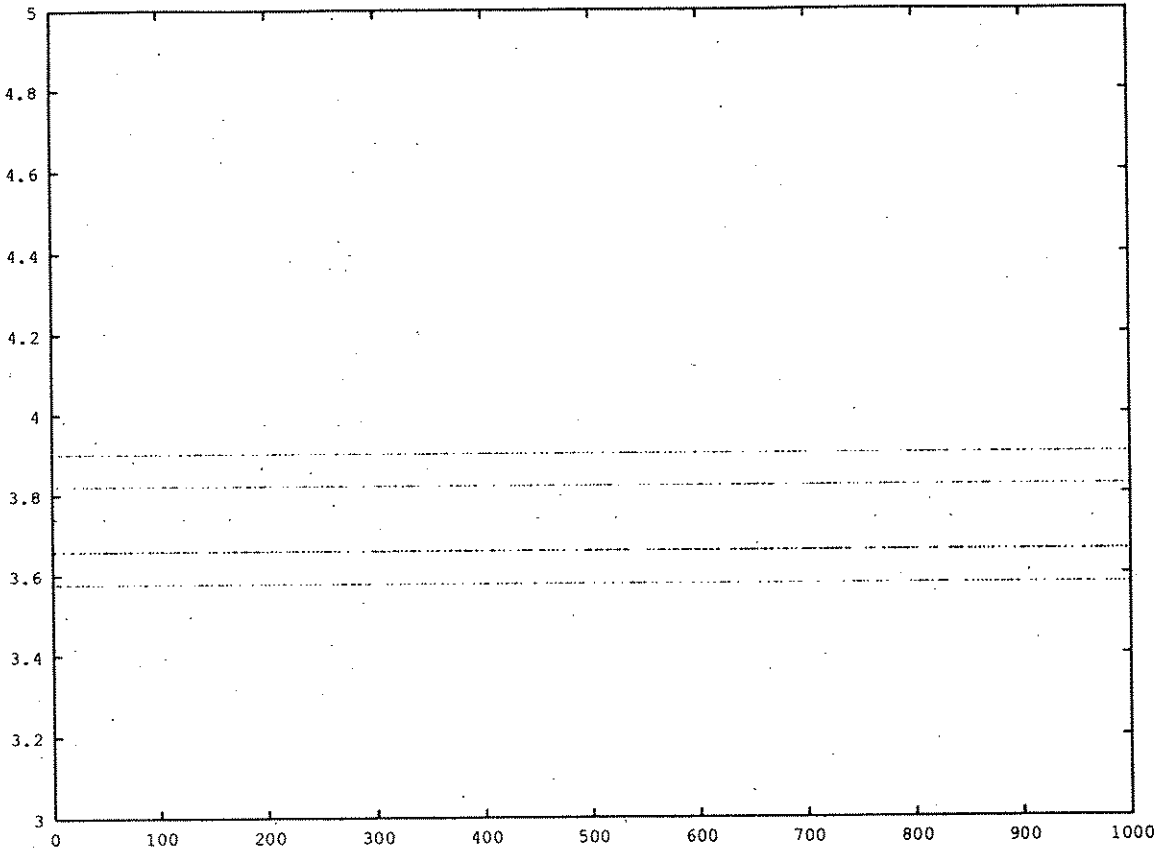
1000 samples
120 msg/sec asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: **3.740 ms**
99.9% threshold: **3.984 ms**

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 32 bytes



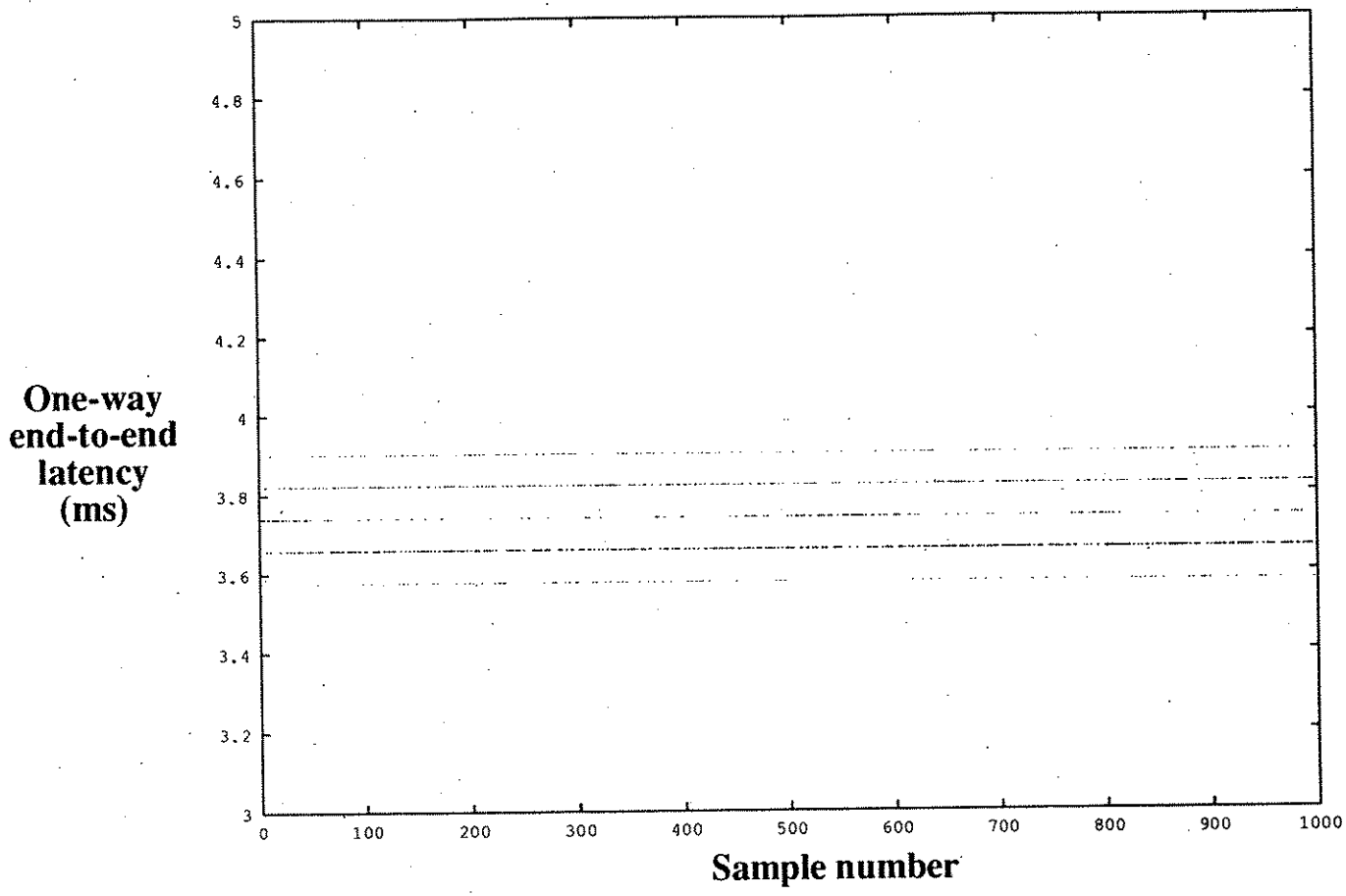
1000 samples
120 msg/sec asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.743 ms
99.9% threshold: 3.984 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 64 bytes



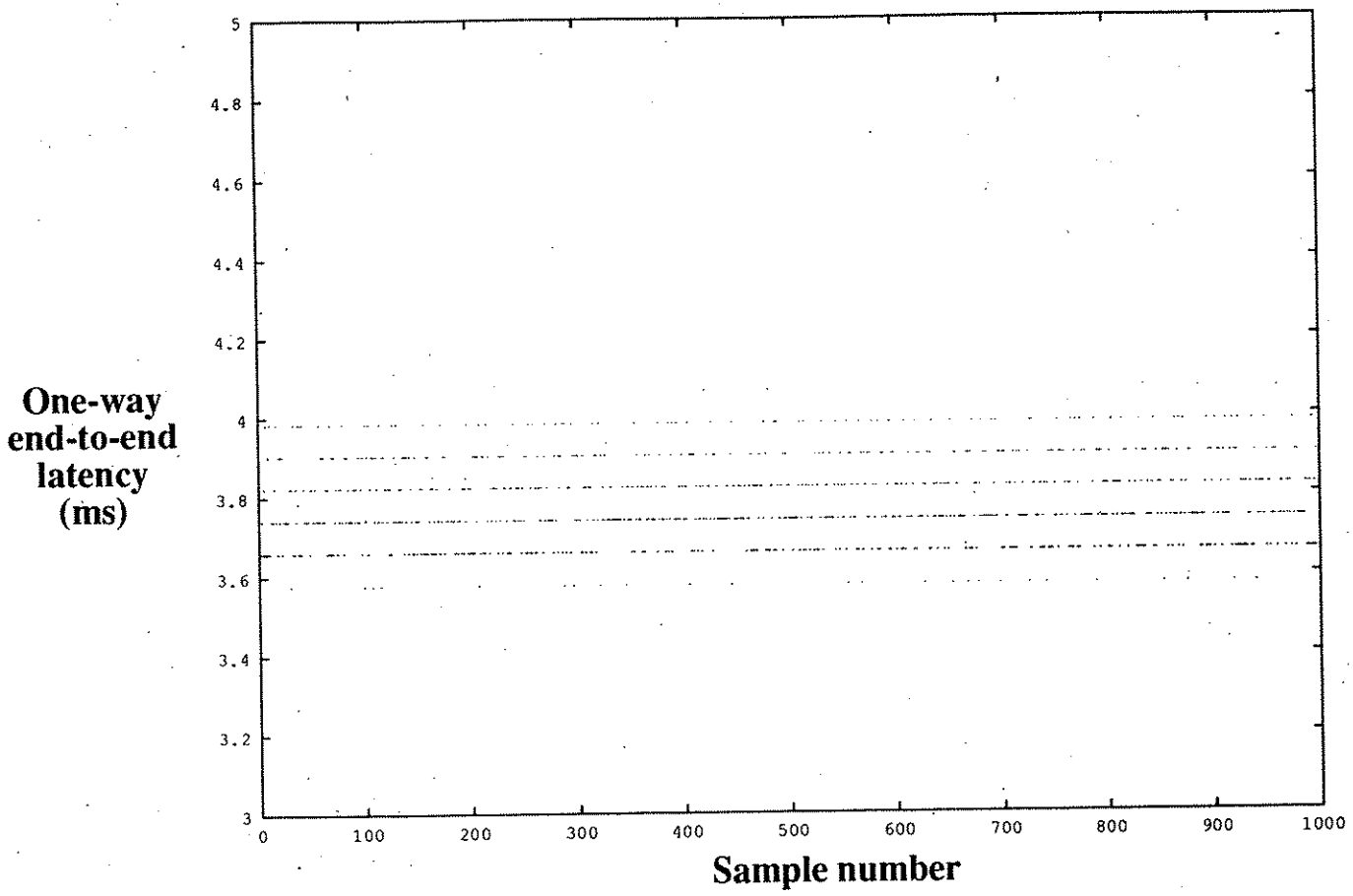
1000 samples
120 msg/sec asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.743 ms
99.9% threshold: 3.984 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 128 bytes



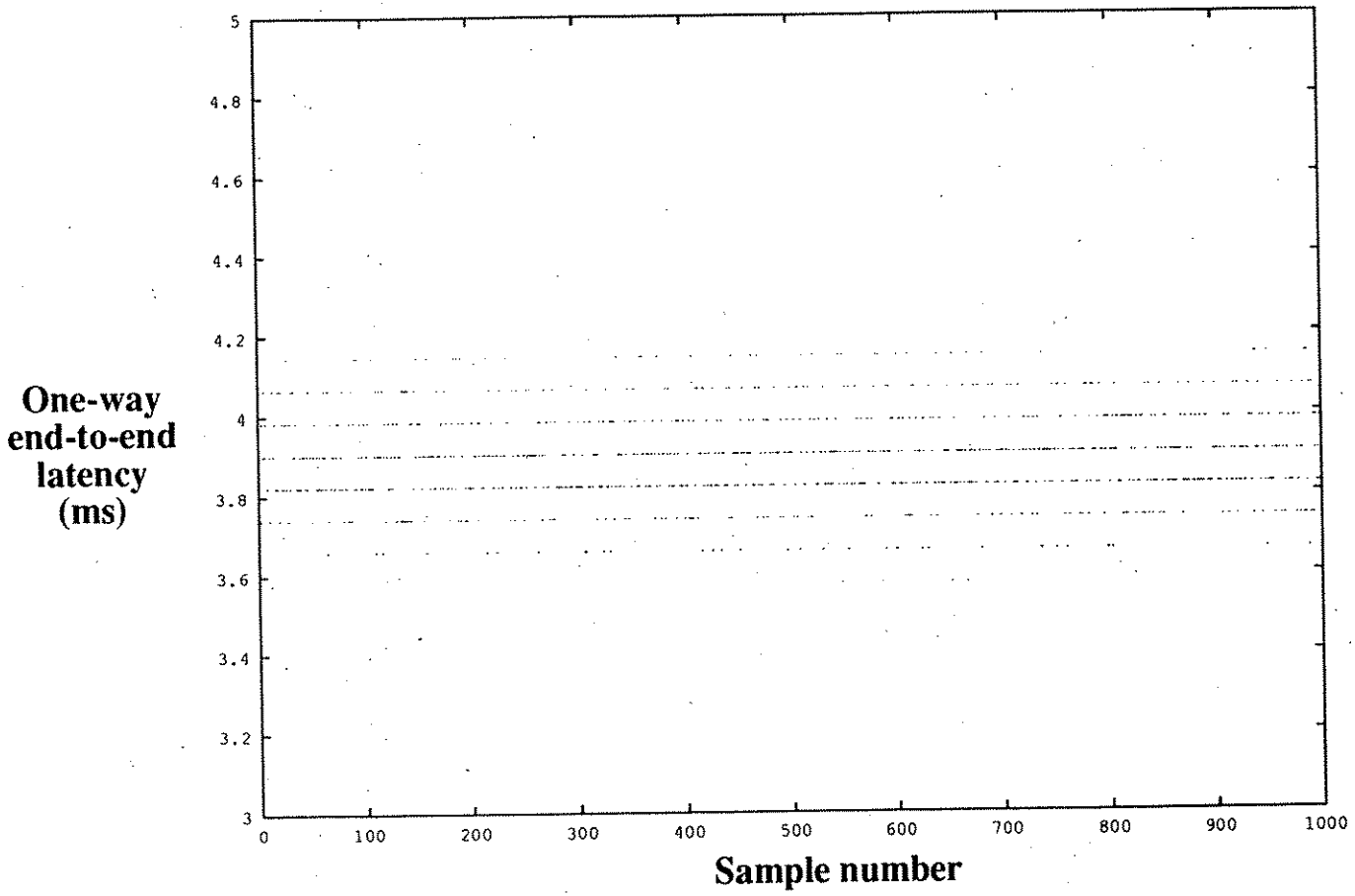
1000 samples
120 msg/sec asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.781 ms
99.9% threshold: 4.065 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 256 bytes



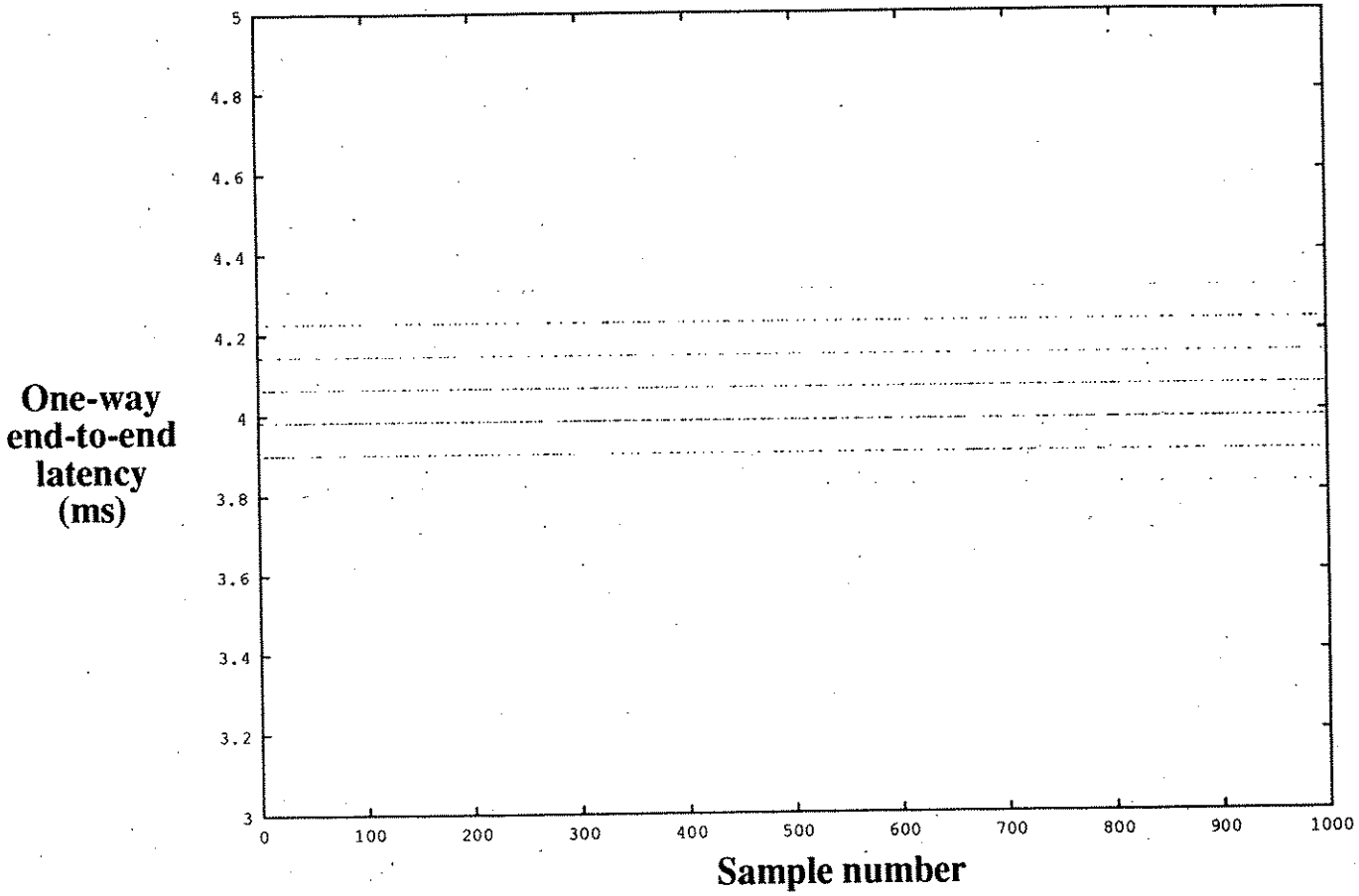
1000 samples
120 msg/sec asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.890 ms
99.9% threshold: 4.146 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 512 bytes



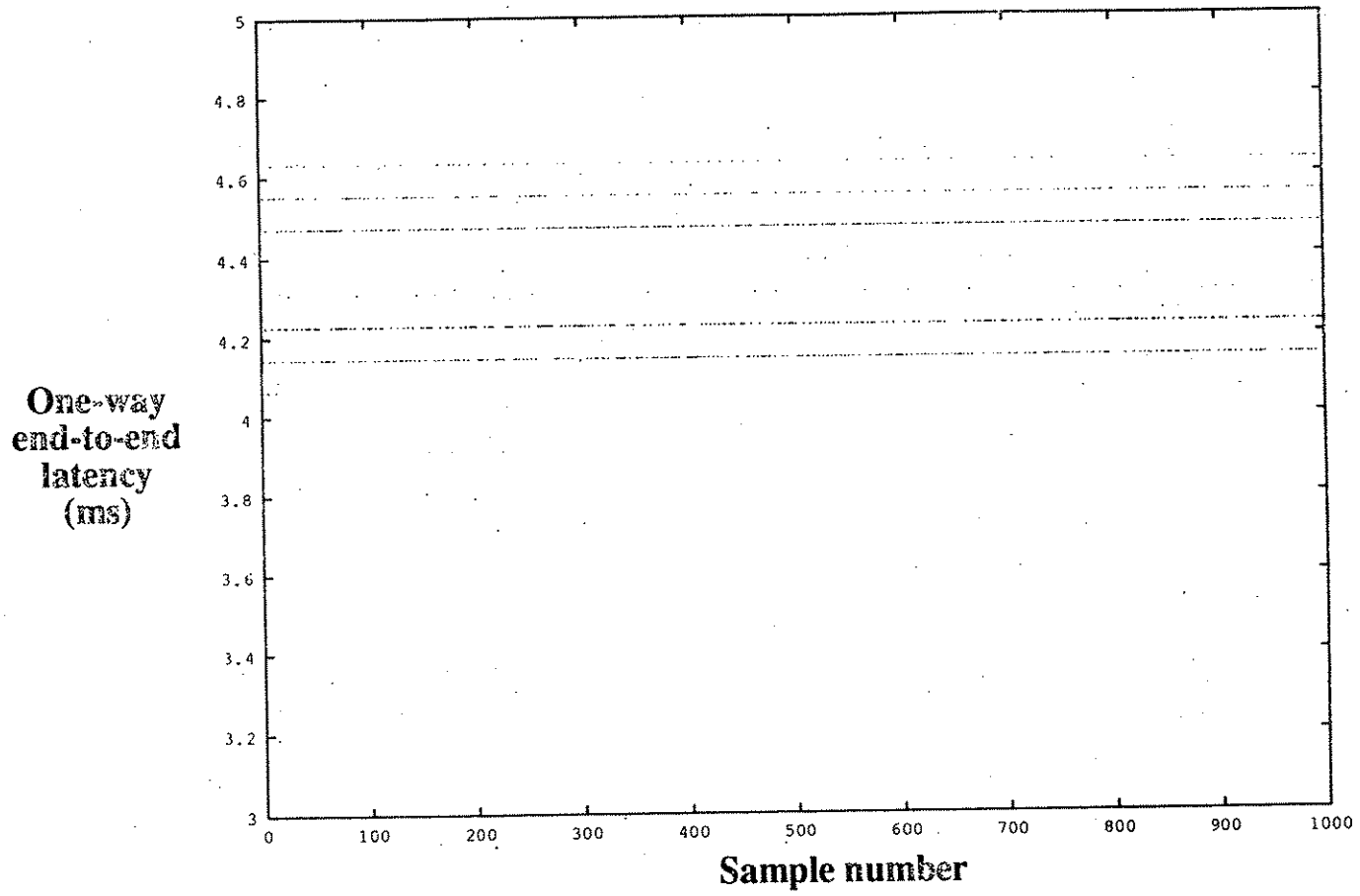
1000 samples
120 msg/sec asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 4.045 ms
99.9% threshold: 4.472 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



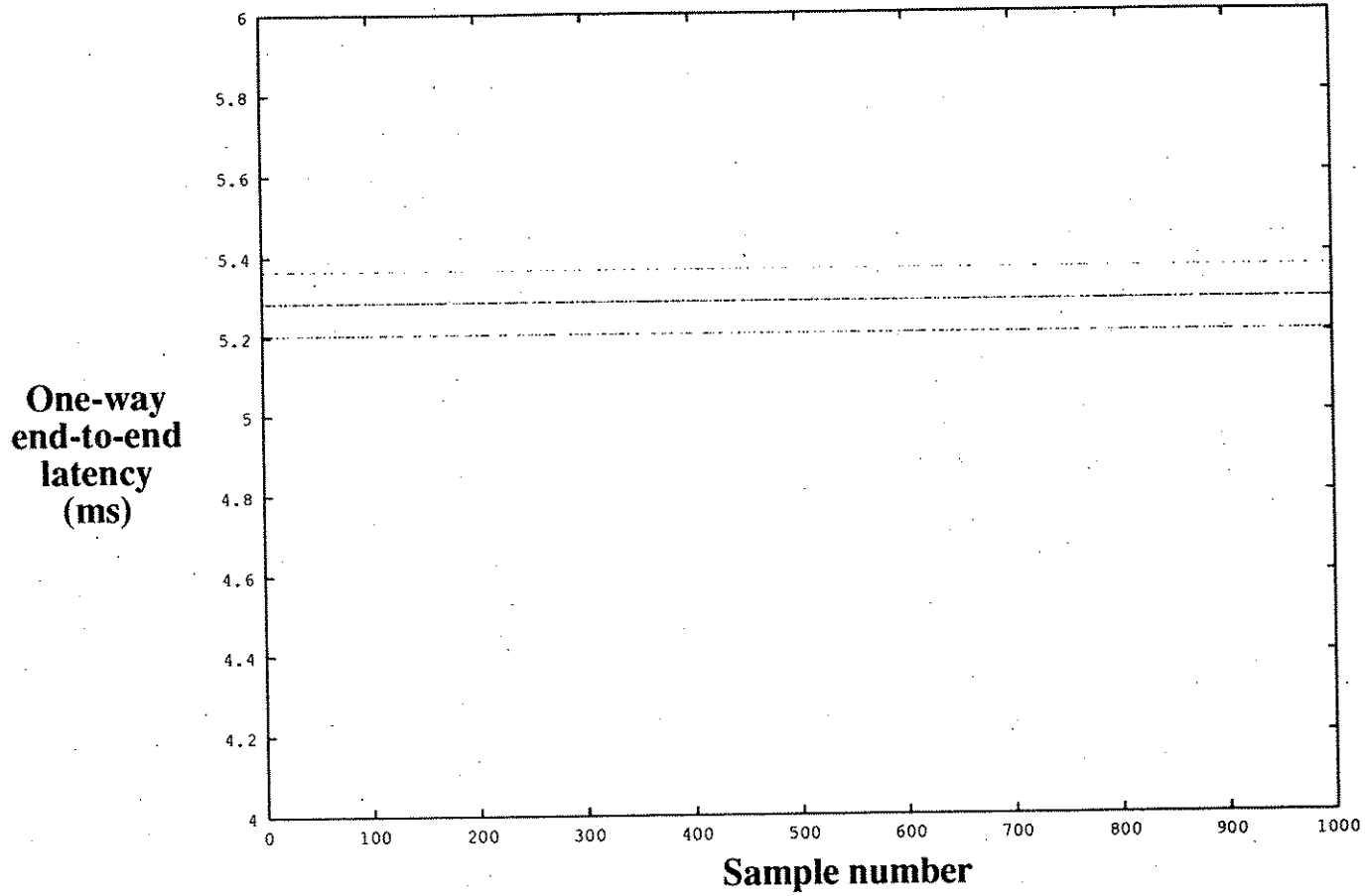
1000 samples
120 msg/sec asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 4.350 ms
99.9% threshold: 4.634 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



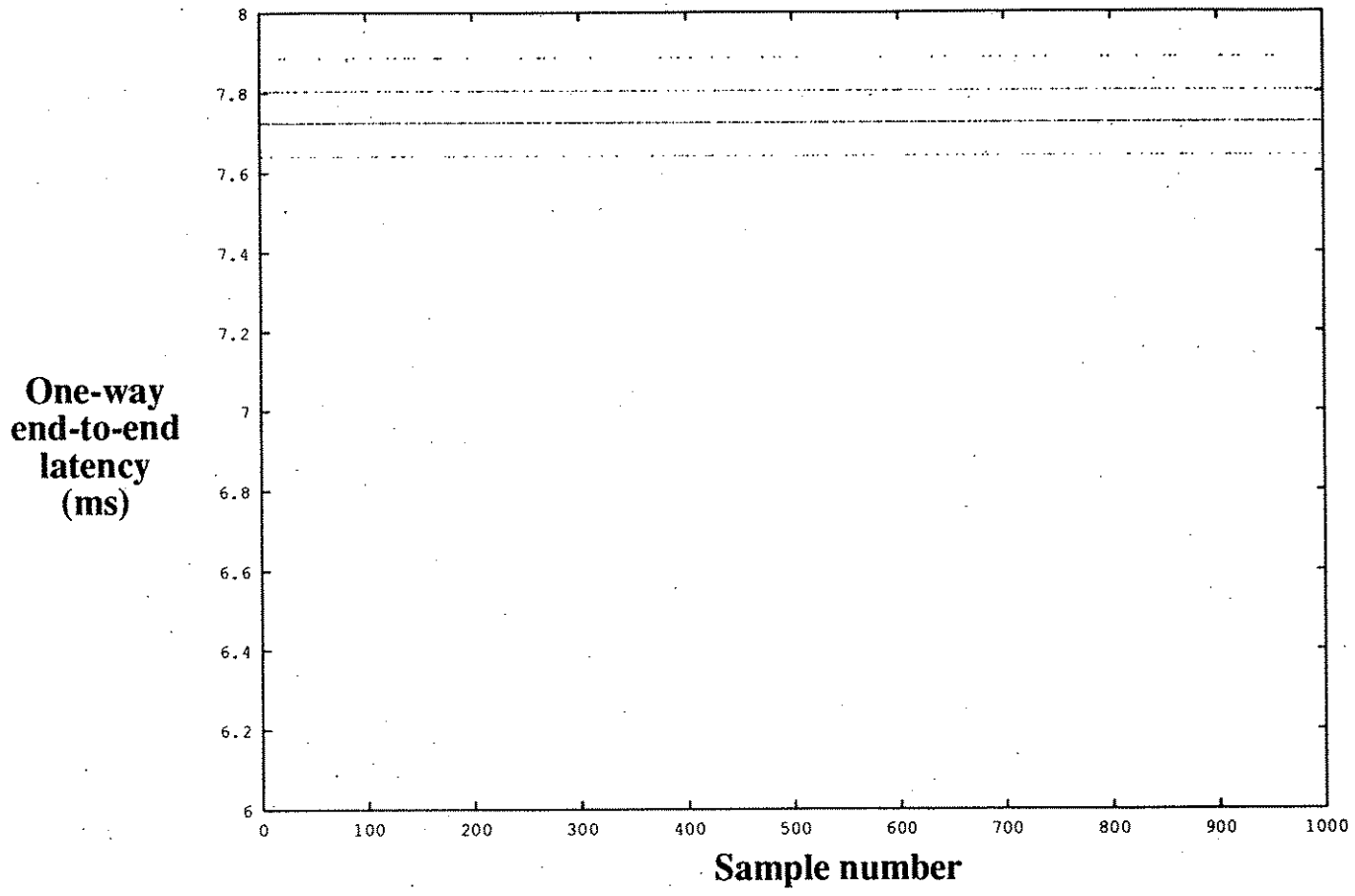
1000 samples
120 msg/sec asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 5.272 ms
99.9% threshold: 5.447 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



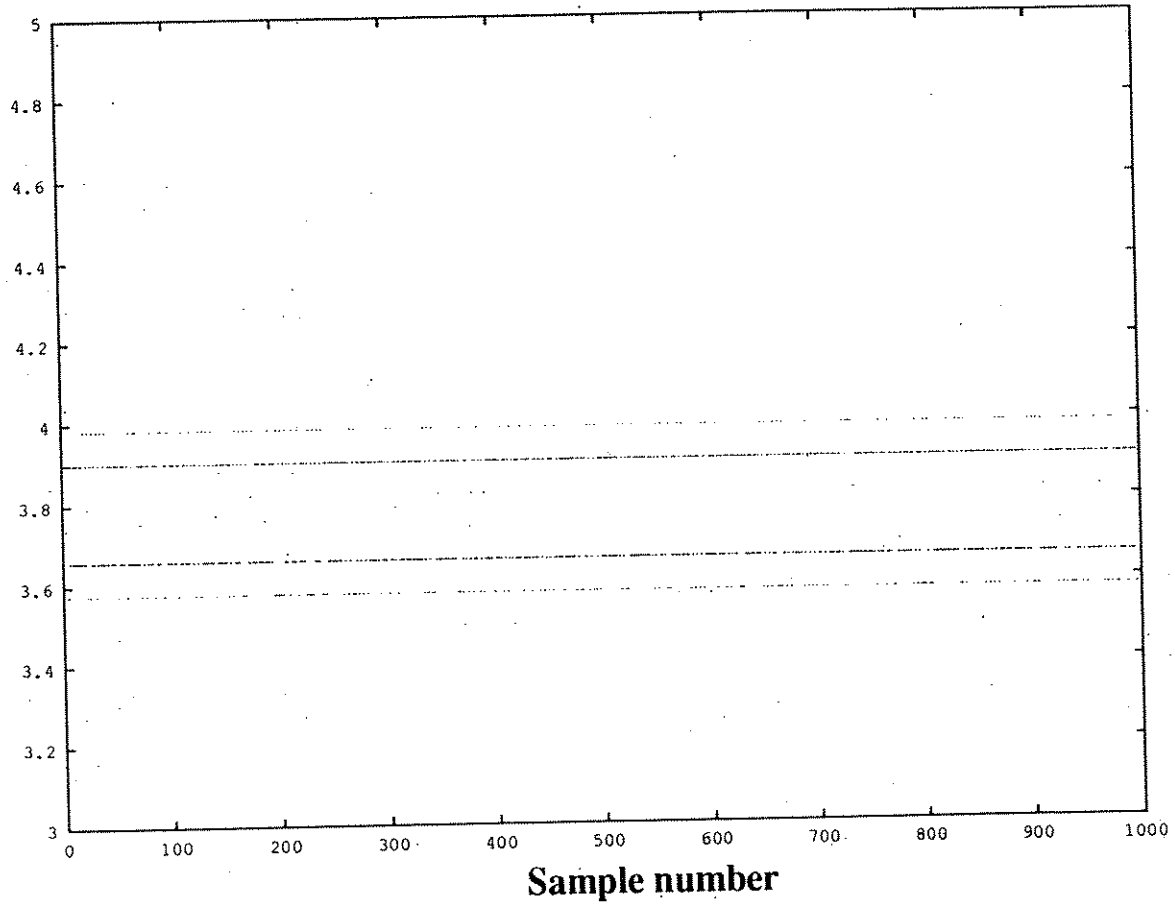
1000 samples
120 msg/sec asynchronous processor load
25 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 7.746 ms
99.9% threshold: 7.886 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 8 bytes



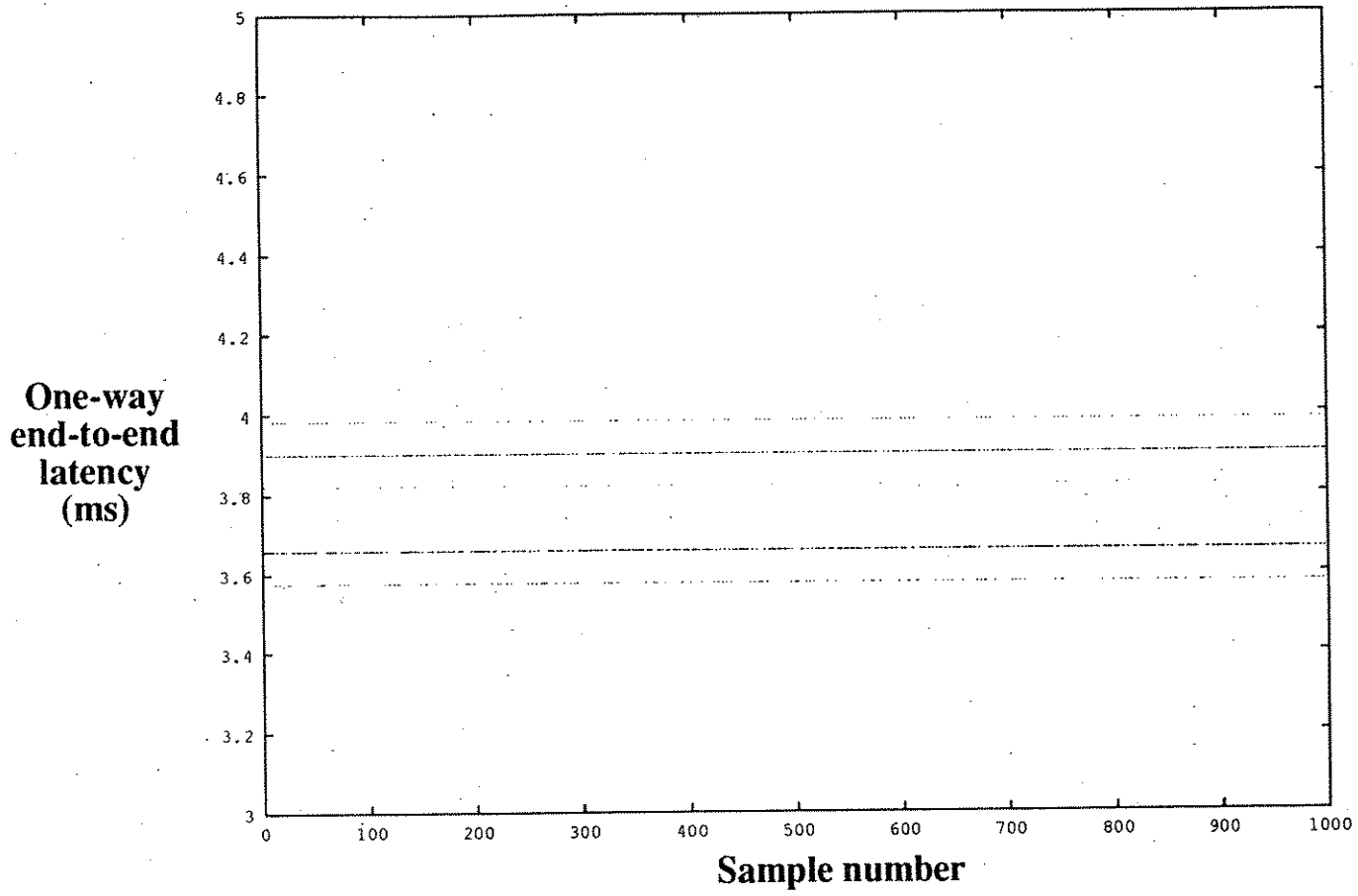
1000 samples
120 msg/sec asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.775 ms
99.9% threshold: 3.984 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 16 bytes



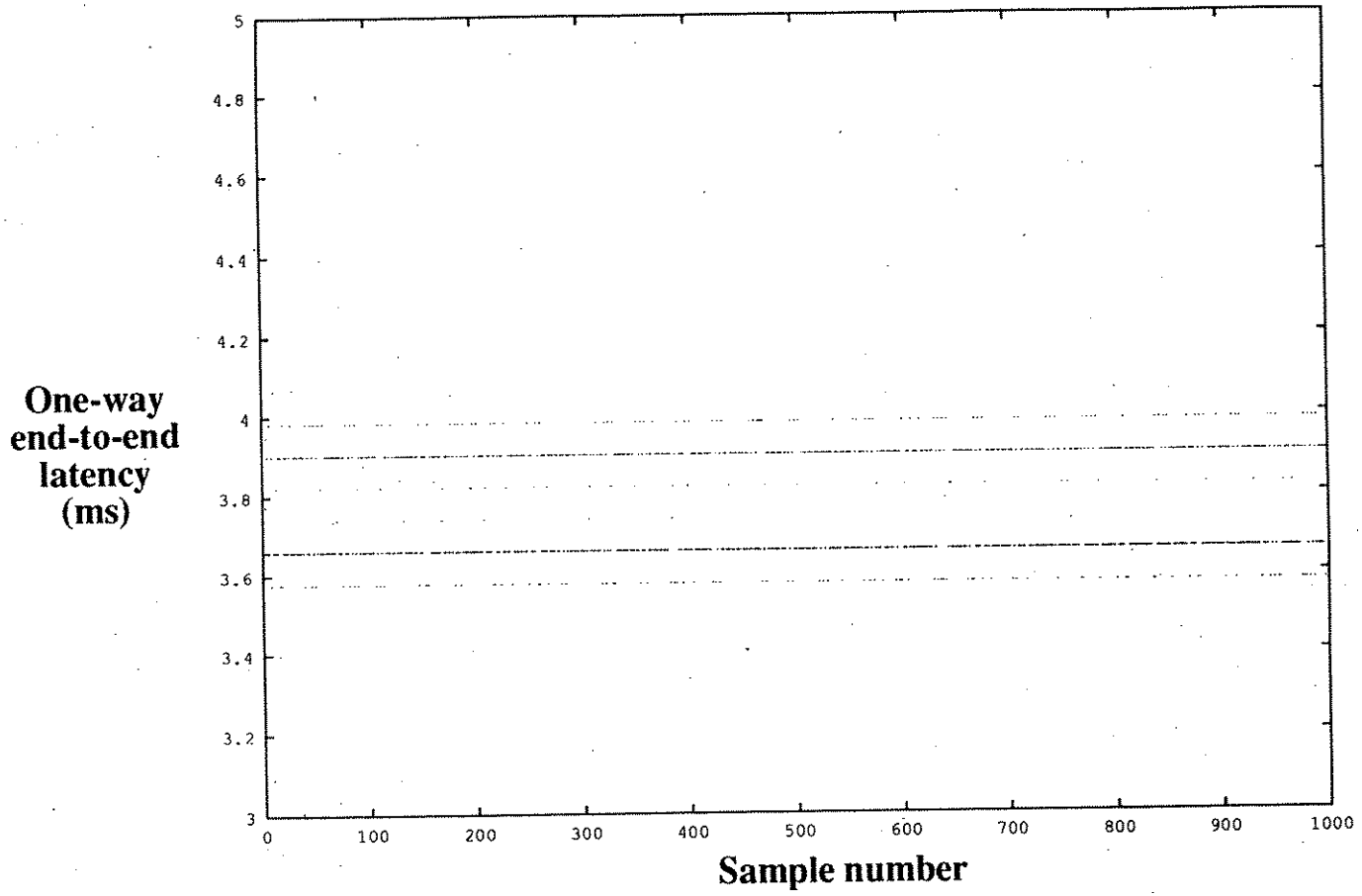
1000 samples
120 msg/sec asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.776 ms
99.9% threshold: 4.065 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 32 bytes



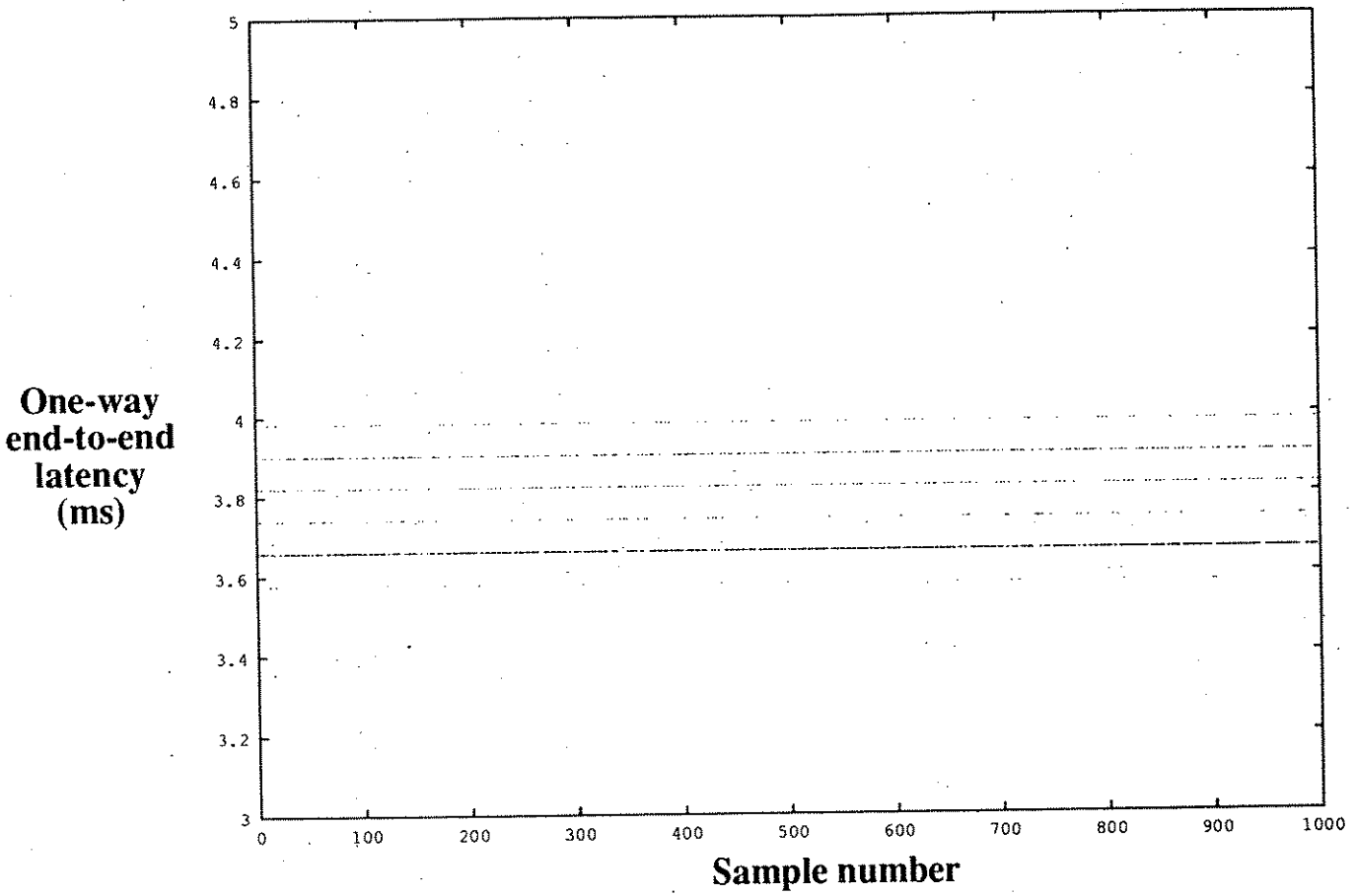
1000 samples
120 msg/sec asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.778 ms
99.9% threshold: 3.984 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 64 bytes



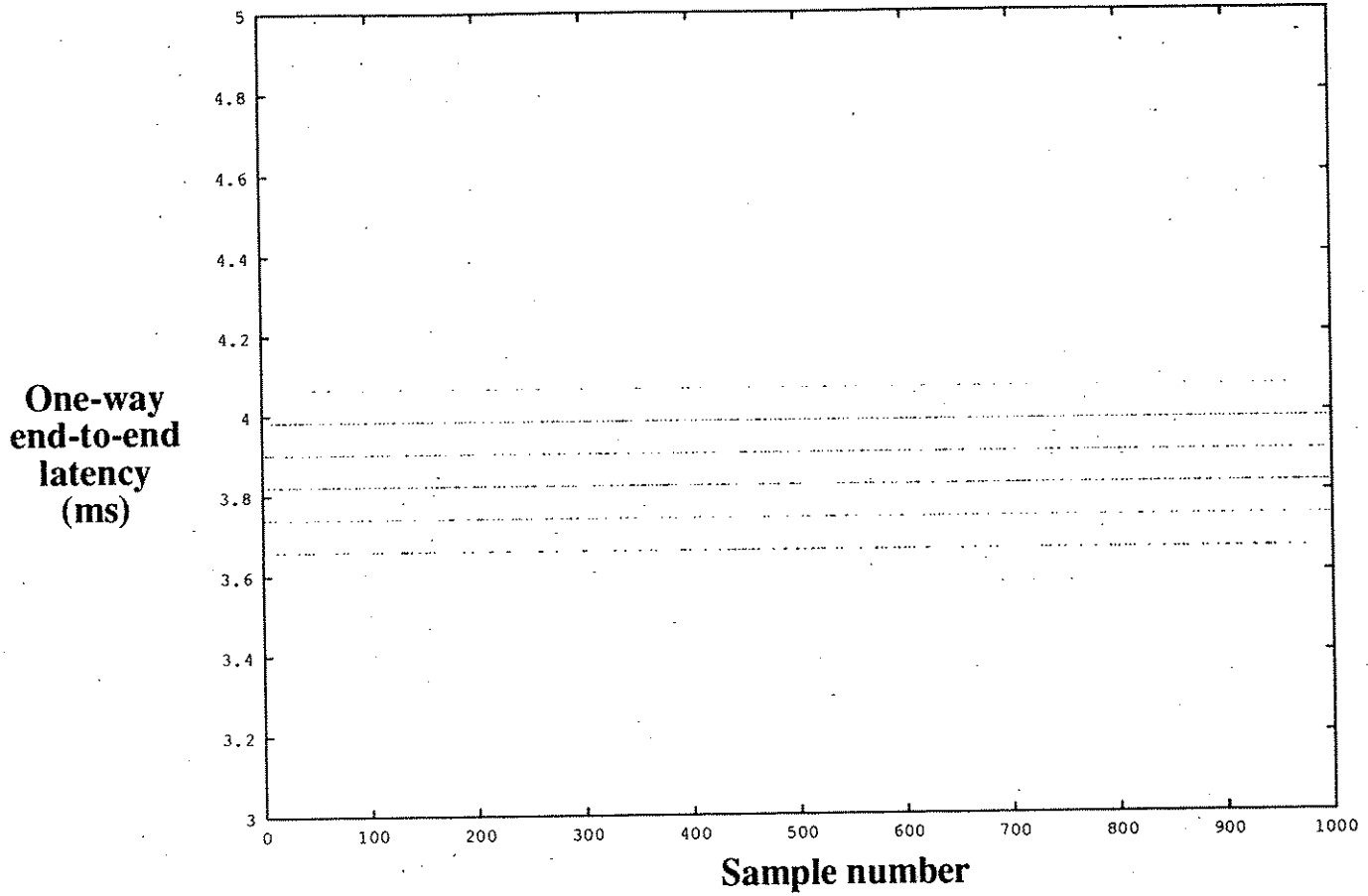
1000 samples
120 msg/sec asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.776 ms
99.9% threshold: 4.065 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 128 bytes



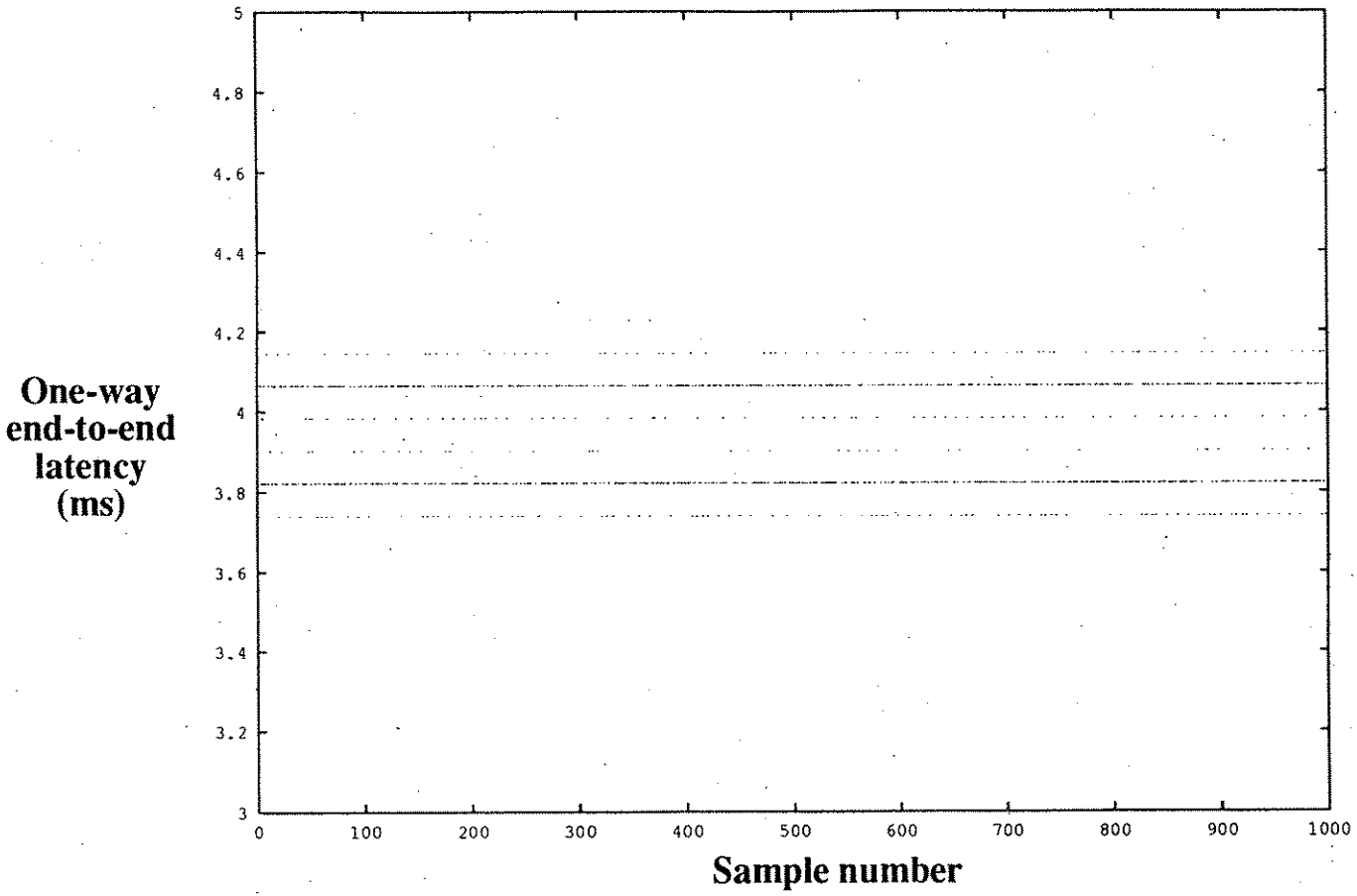
1000 samples
120 msg/sec asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.859 ms
99.9% threshold: 4.146 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 256 bytes



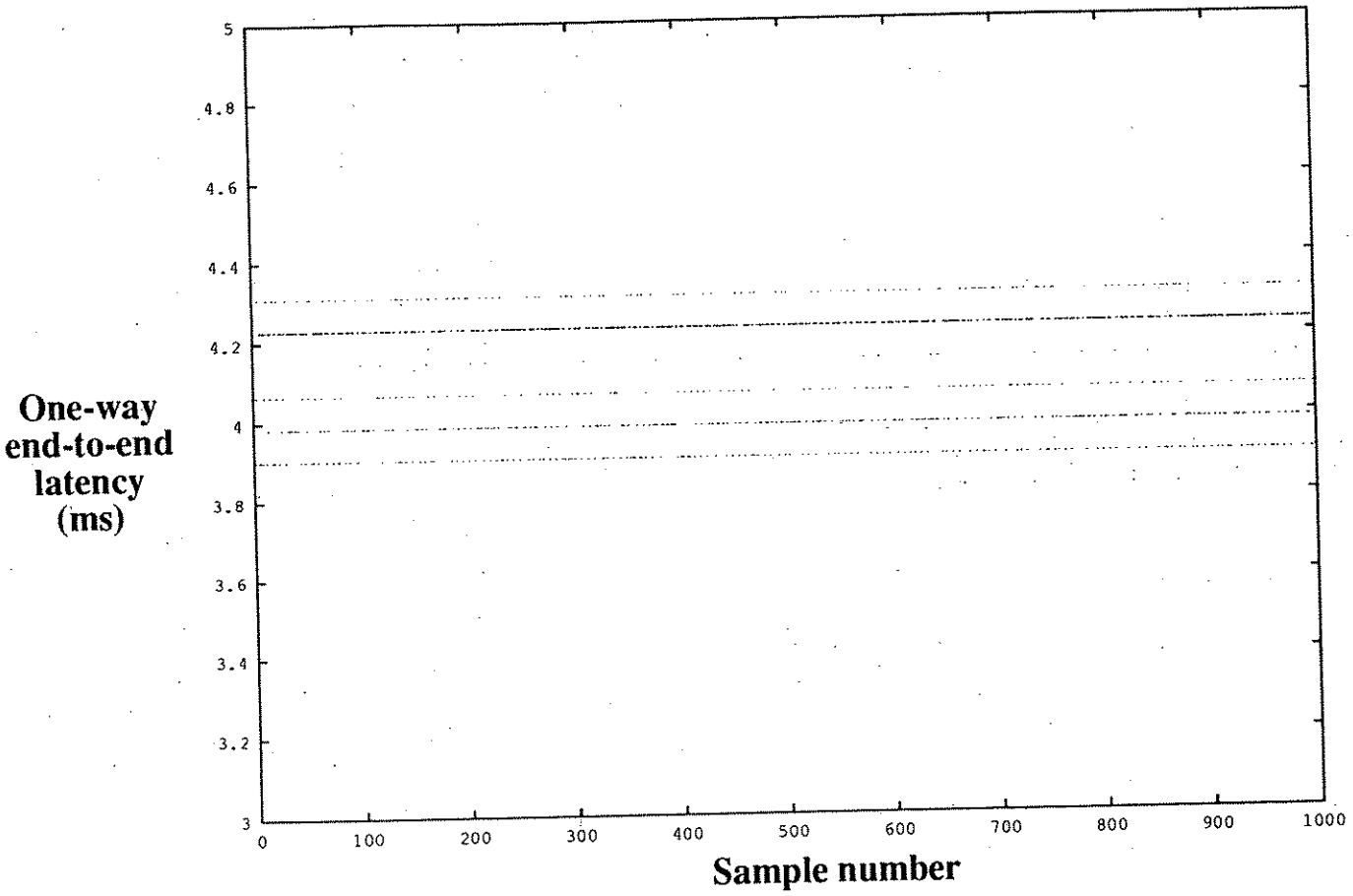
1000 samples
120 msg/sec asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 3.939 ms
99.9% threshold: 4.228 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 512 bytes



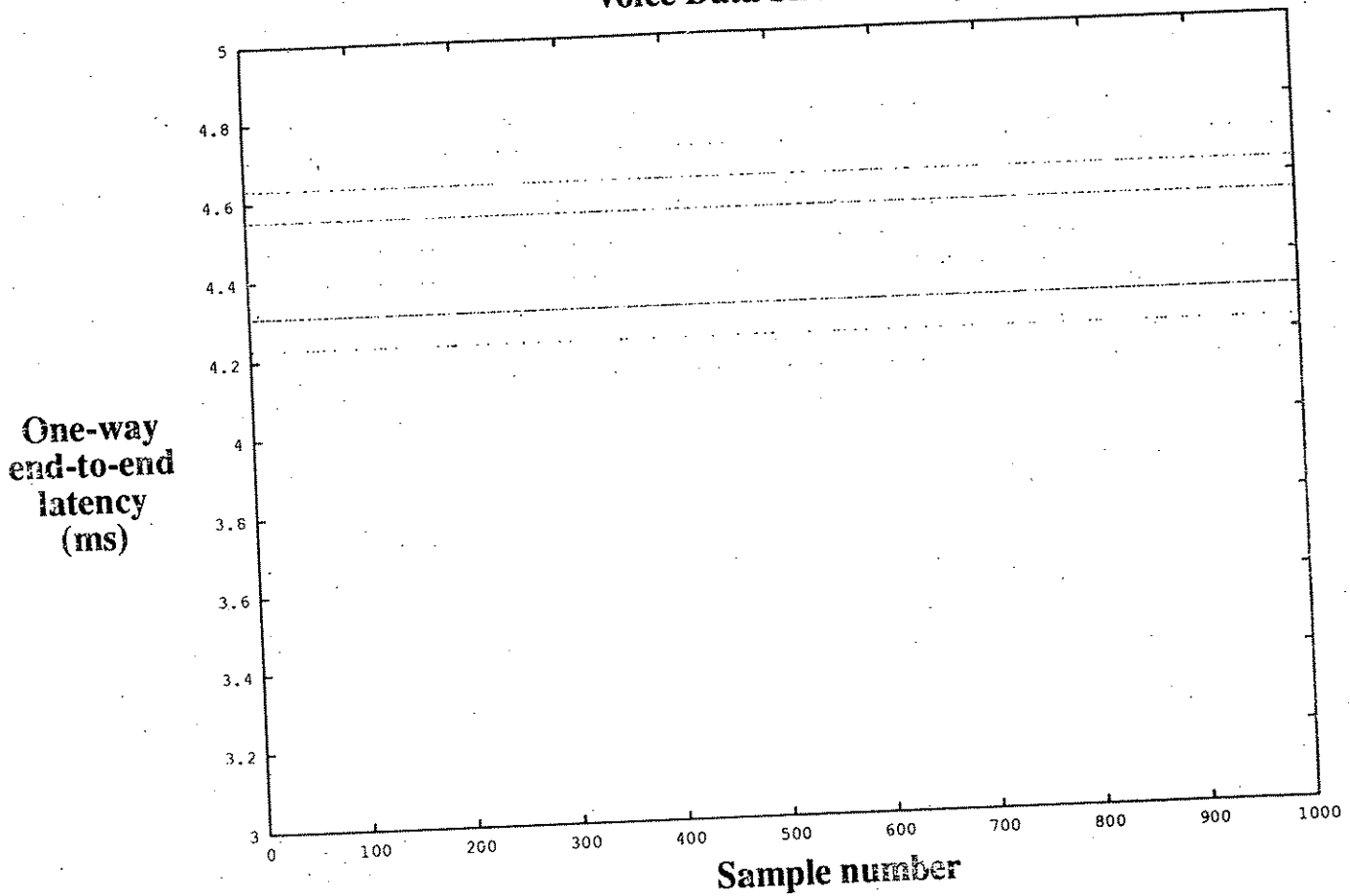
1000 samples
120 msg/sec asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 4.108 ms
99.9% threshold: 4.309 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



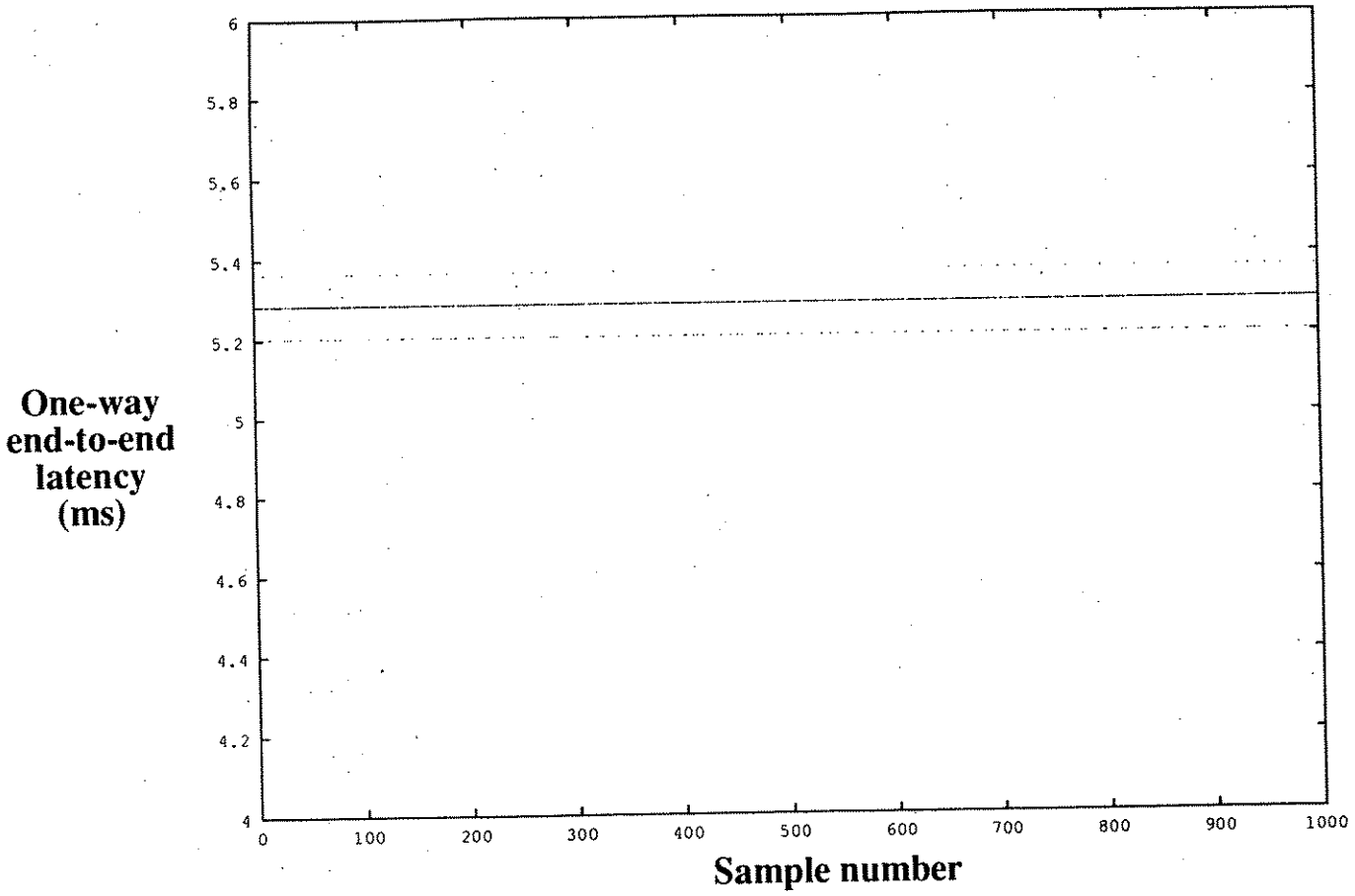
Average latency: 4.443 ms
99.9% threshold: 4.797 ms

1000 samples
120 msg/sec asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



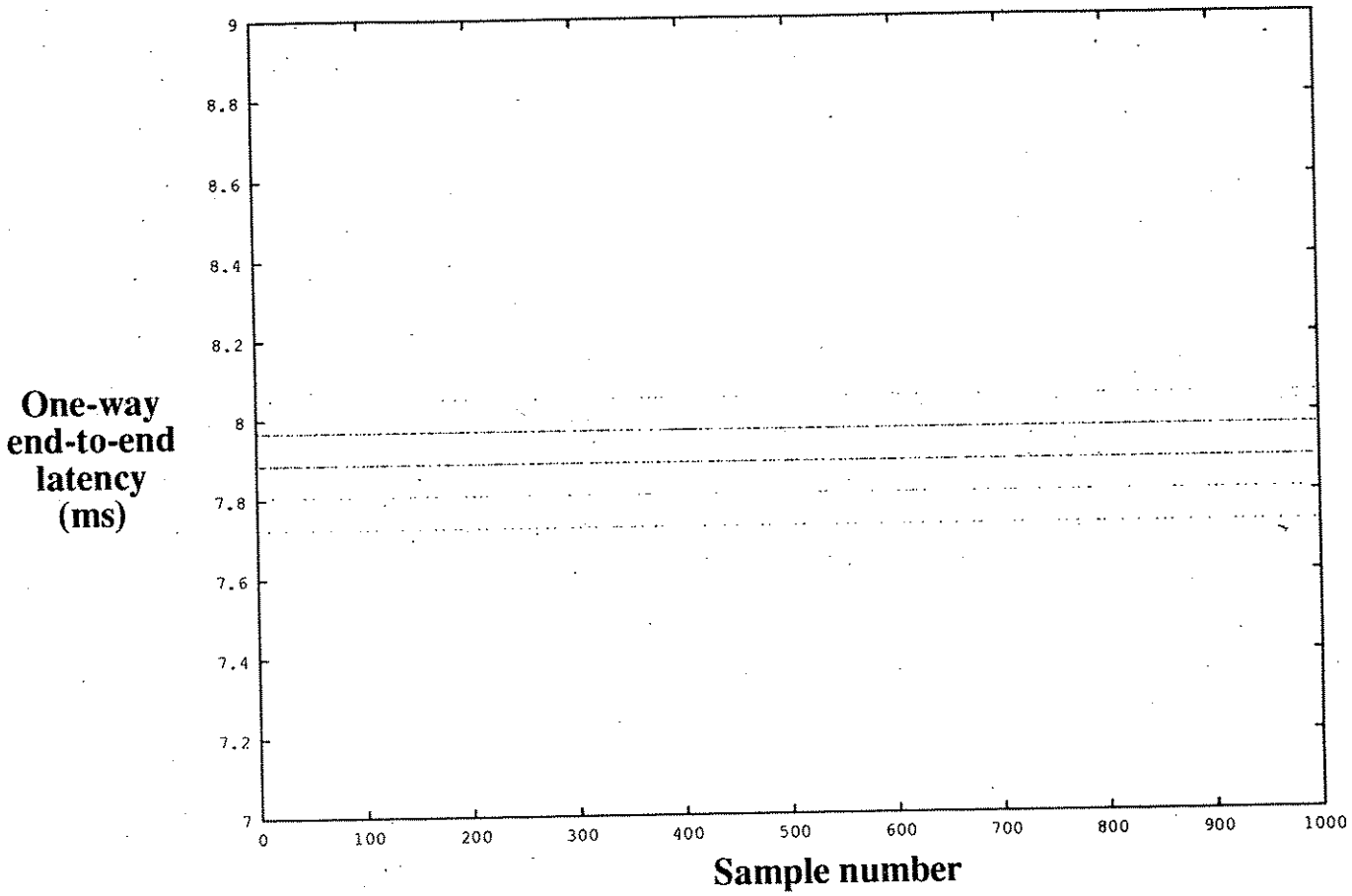
1000 samples
120 msg/sec asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 5.278 ms
99.9% threshold: 5.447 ms

ASYNCHRONOUS PROCESSOR LOAD

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



1000 samples
120 msg/sec asynchronous processor load
50 Mbits/sec background synchronous FDDI load (single packets/token)
Voice data in FDDI synchronous class

Average latency: 7.907 ms
99.9% threshold: 8.049 ms

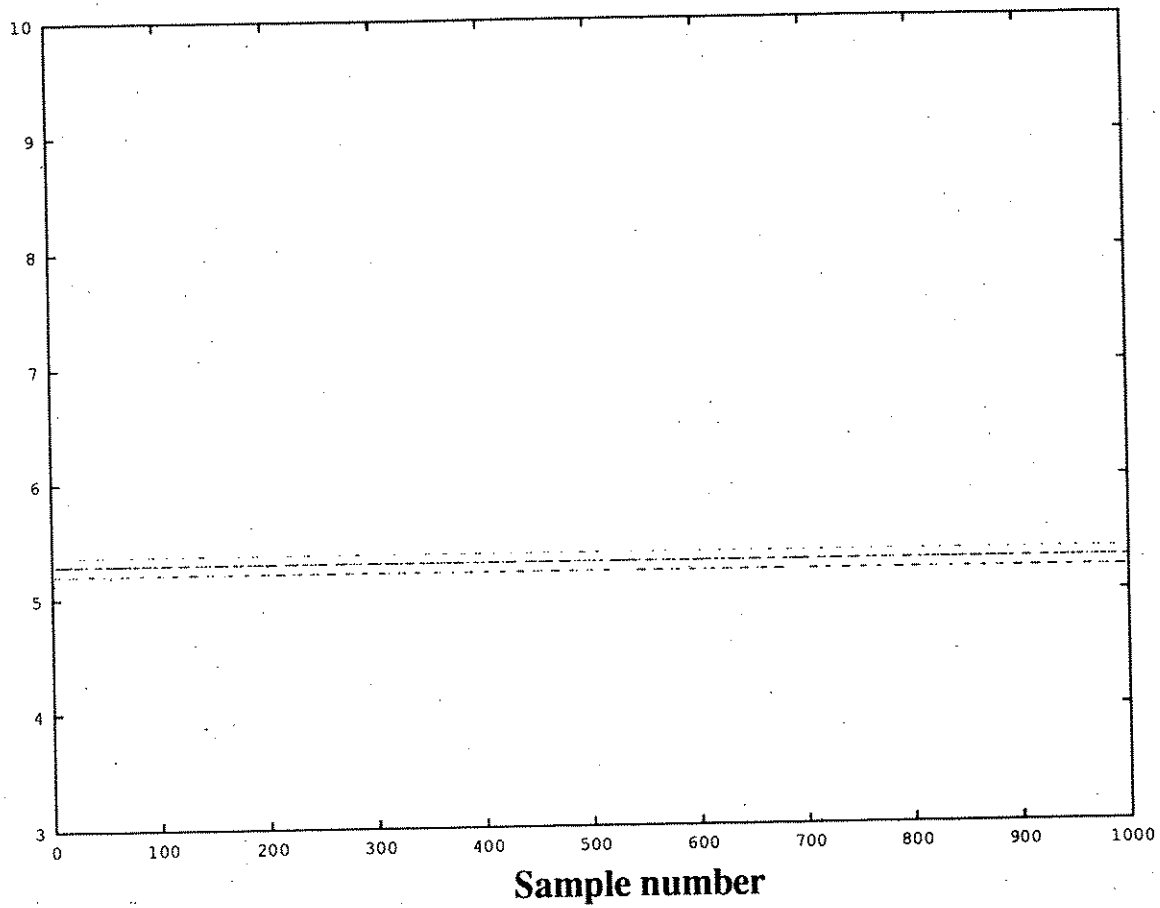
ASYNCHRONOUS PROCESSOR LOAD

Appendix F

Experiment 6: Multicast

JITTER MEASUREMENT

Voice Data Size: 8 bytes



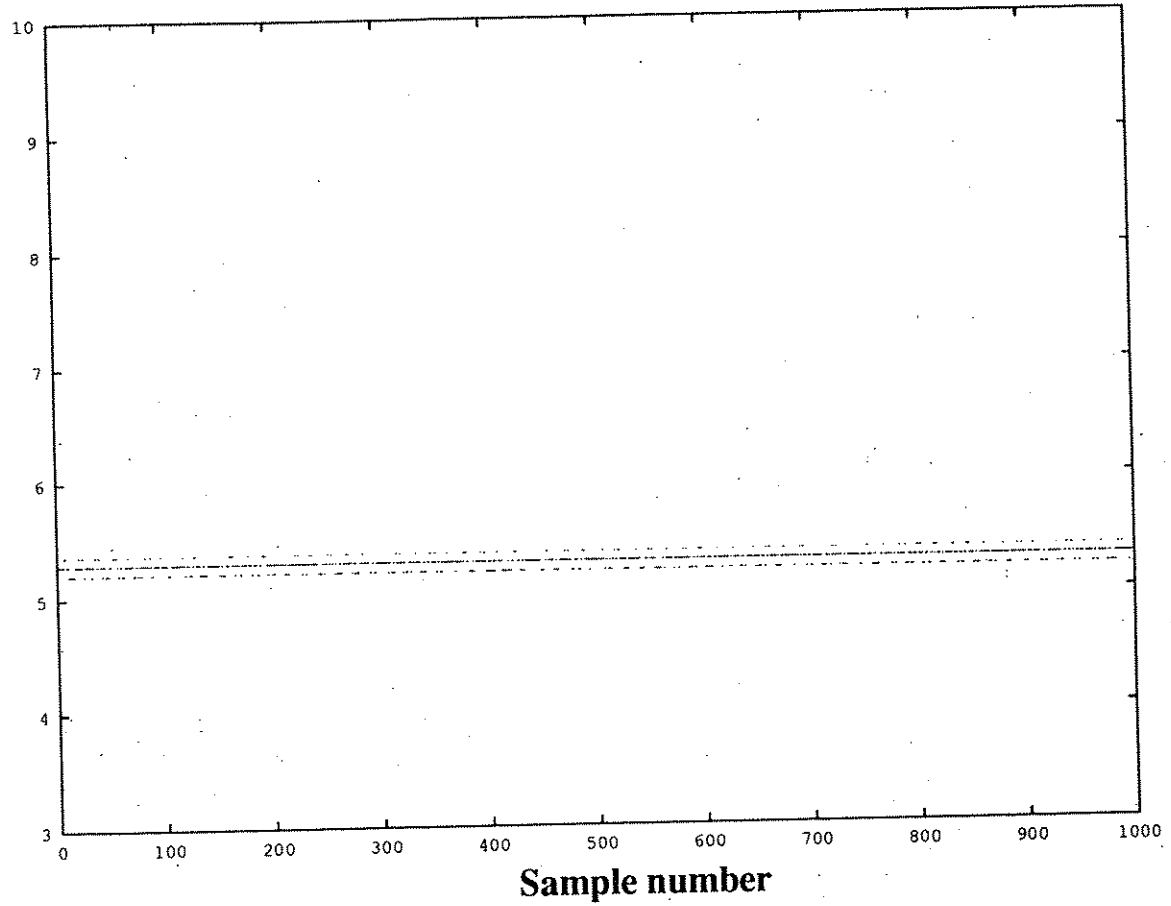
1000 samples
No asynchronous processor load
No background FDDI load
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 5.266 ms
99.9% threshold: 5.366 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 16 bytes



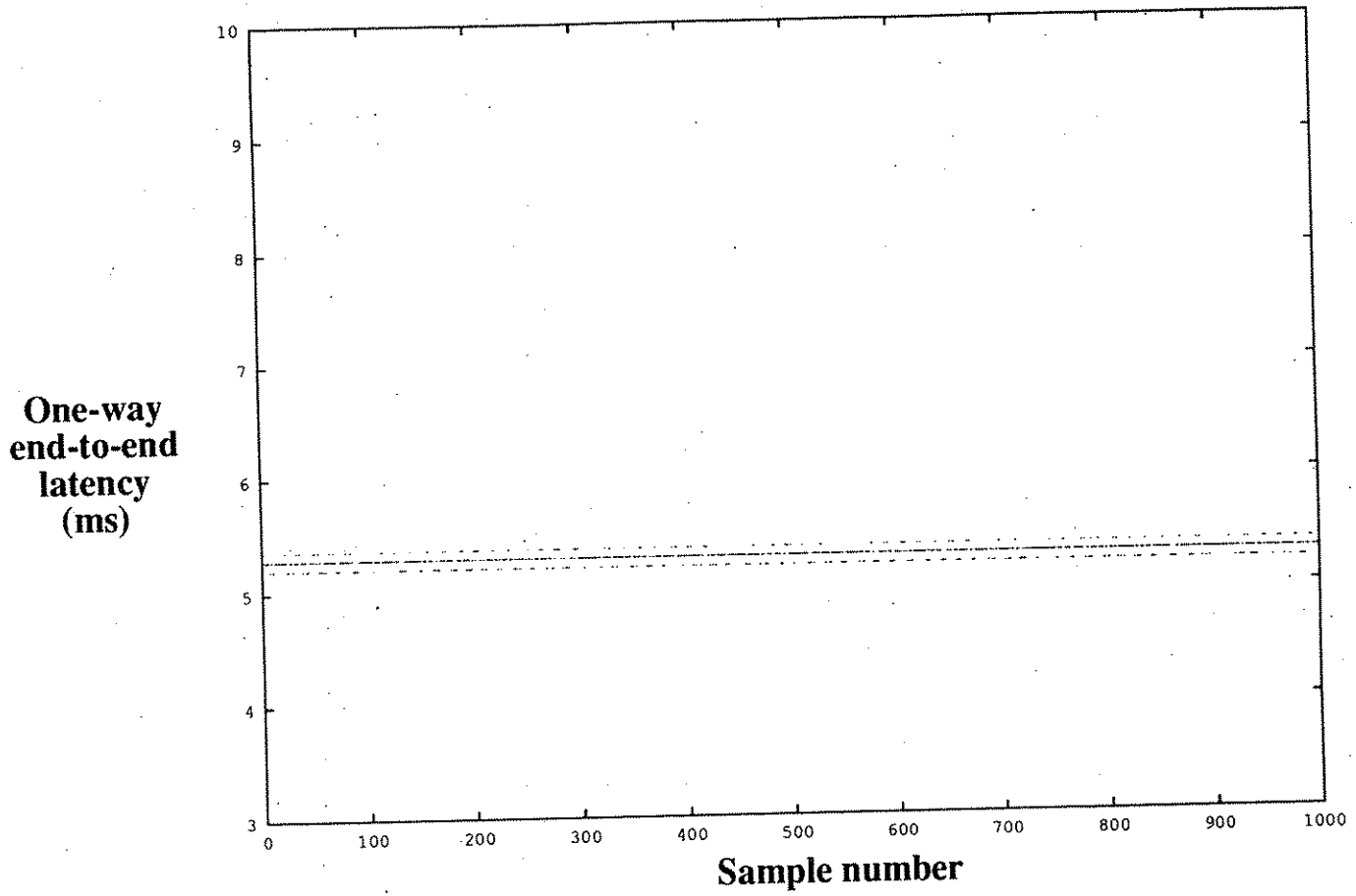
1000 samples
No asynchronous processor load
No background FDDI load
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 5.273 ms
99.9% threshold: 5.447 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 32 bytes



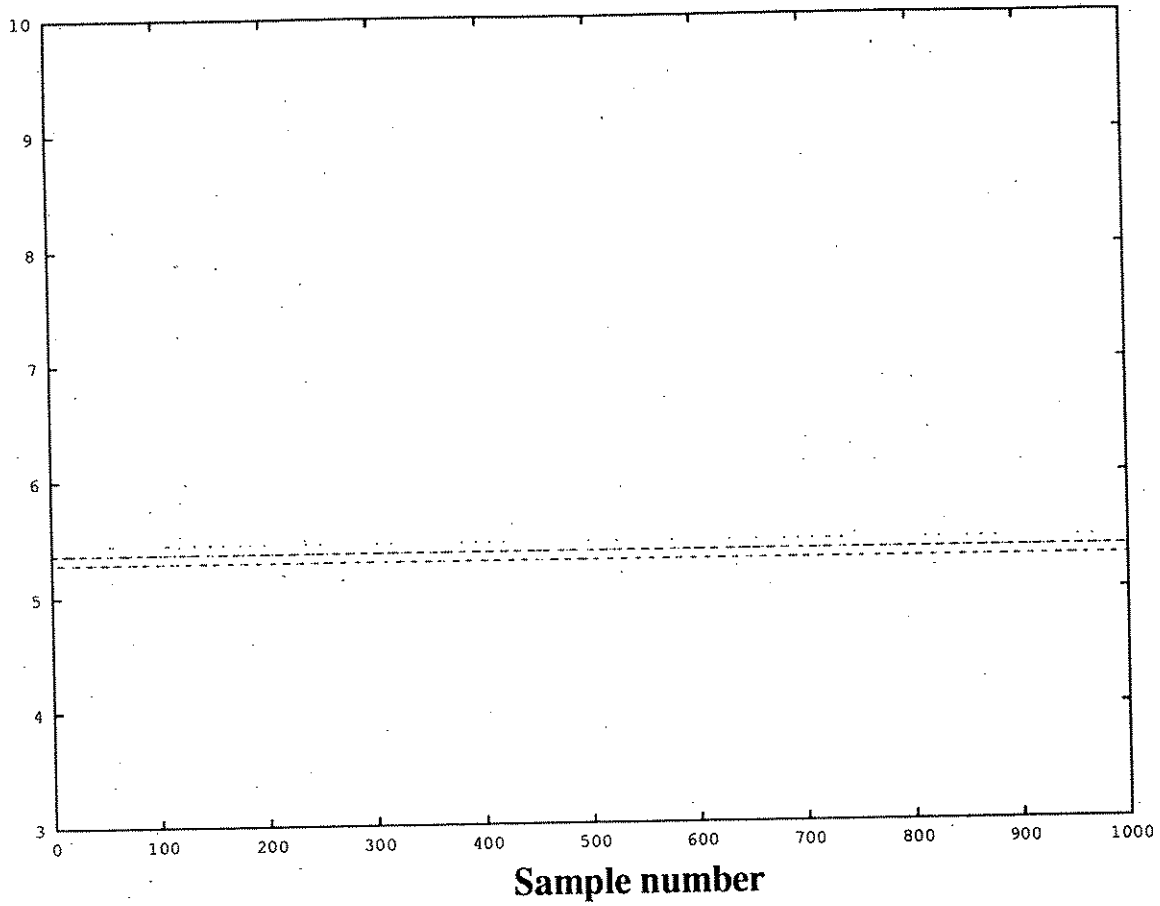
1000 samples
No asynchronous processor load
No background FDDI load
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 5.277 ms
99.9% threshold: 5.336 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 64 bytes



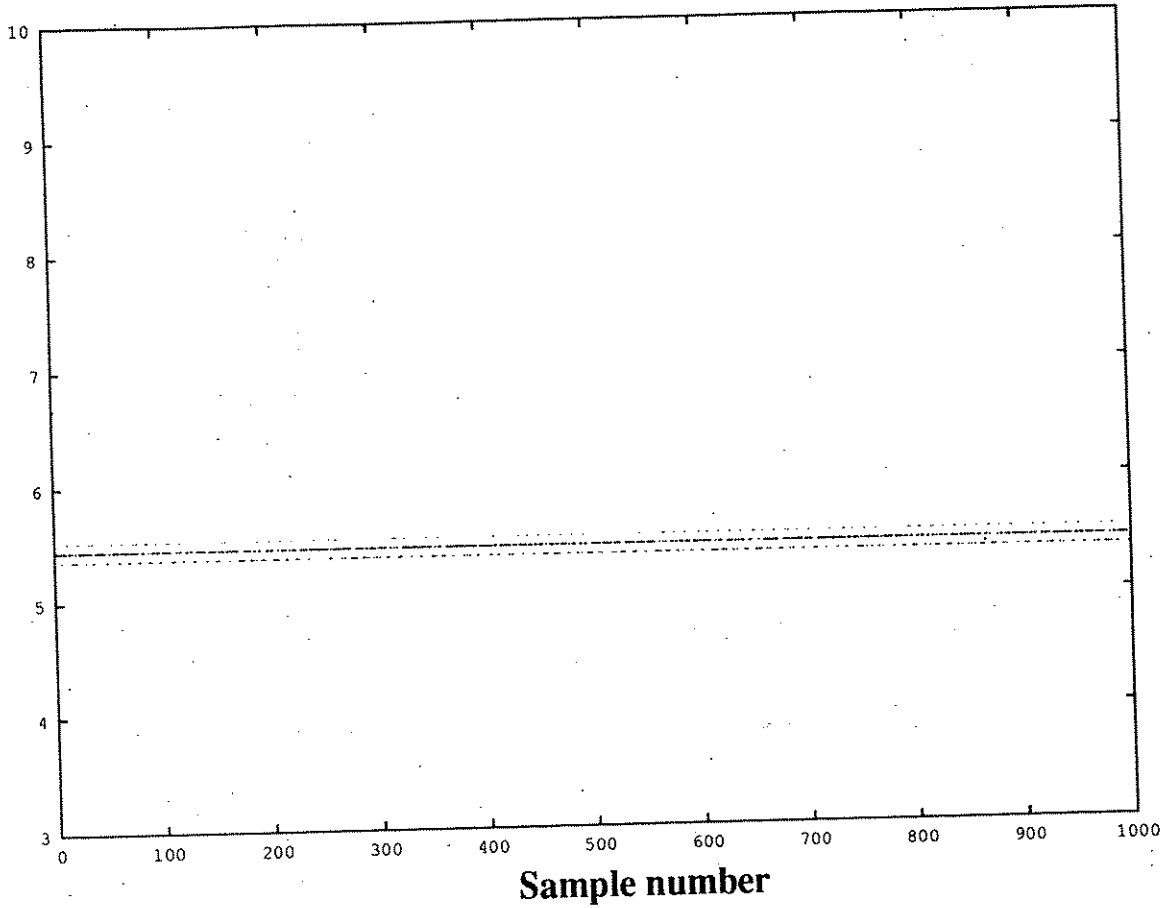
1000 samples
No asynchronous processor load
No background FDDI load
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 5.336 ms
99.9% threshold: 5.447 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 128 bytes



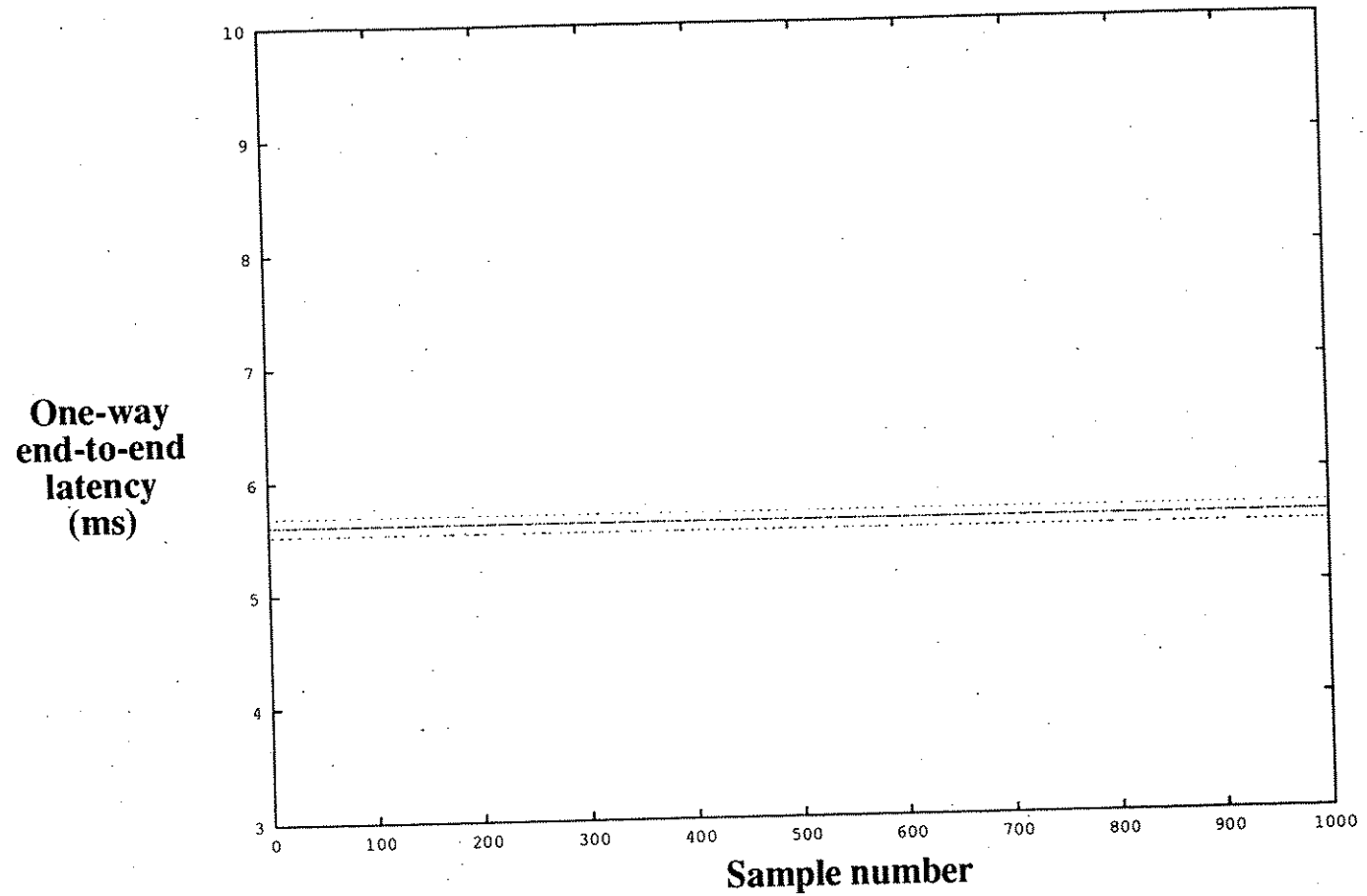
1000 samples
No asynchronous processor load
No background FDDI load
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 5.431 ms
99.9% threshold: 5.528 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 256 bytes



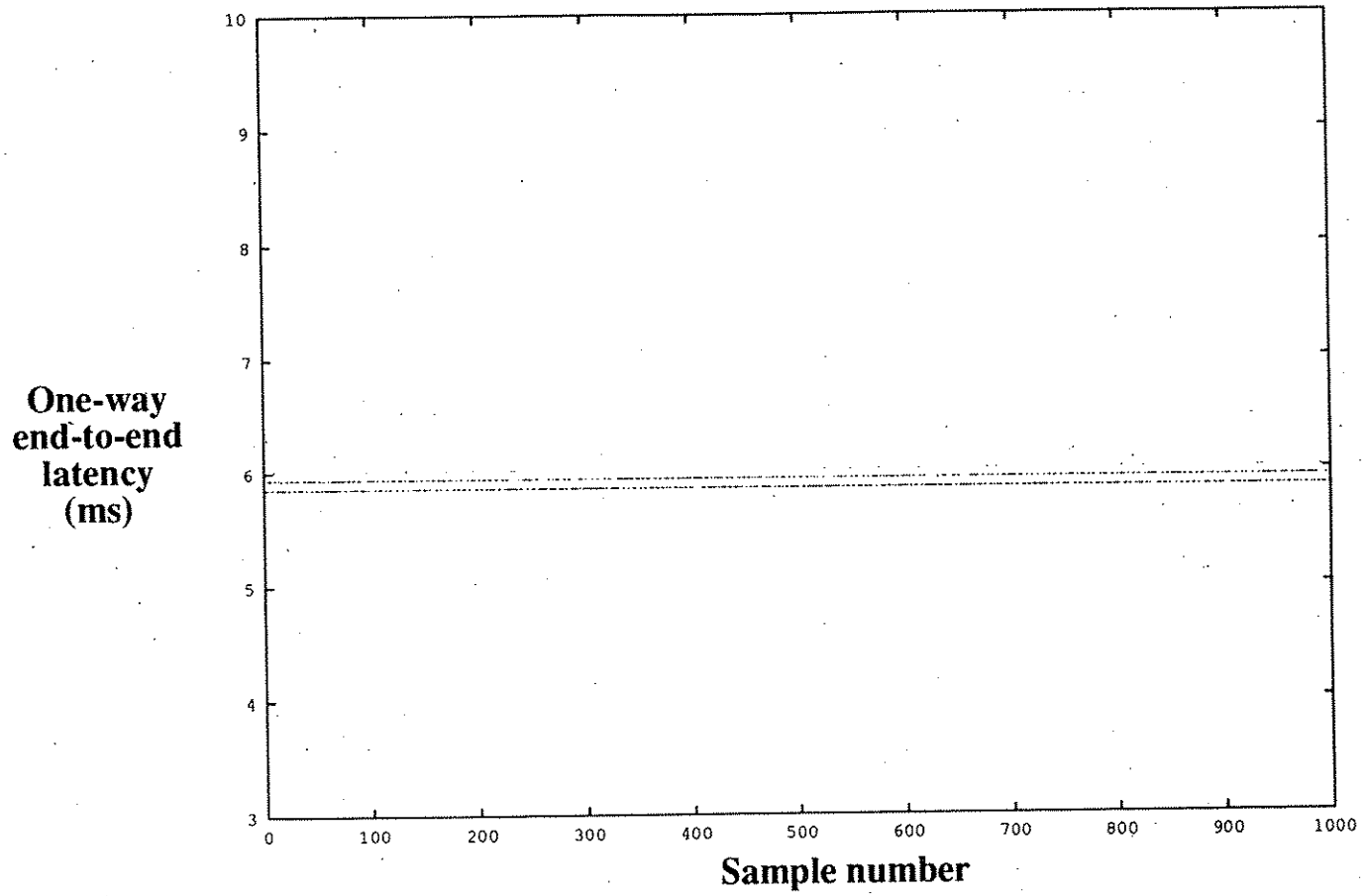
1000 samples
No asynchronous processor load
No background FDDI load
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 5.603 ms
99.9% threshold: 5.772 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 512 bytes



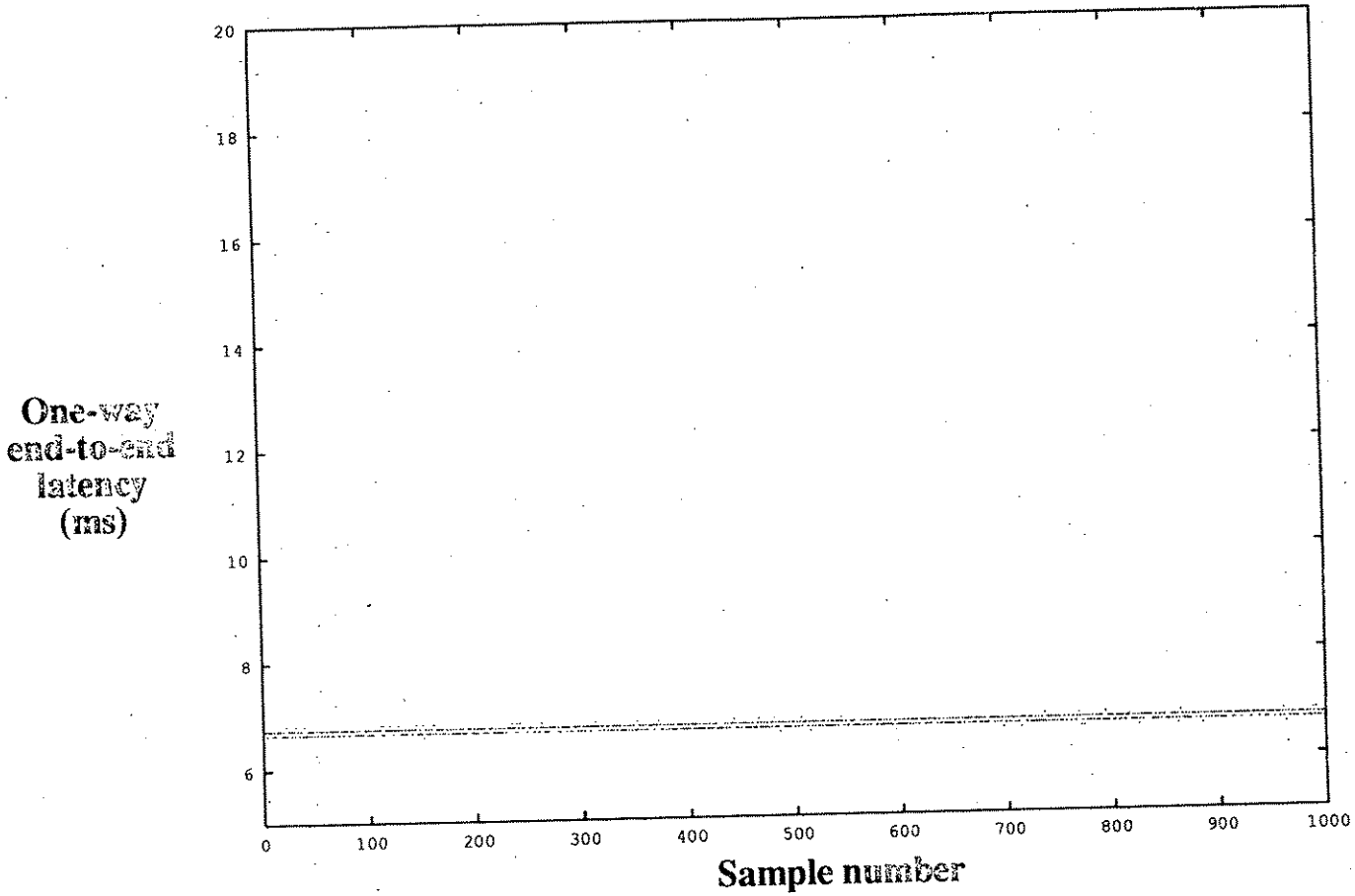
1000 samples
No asynchronous processor load
No background FDDI load
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 5.894 ms
99.9% threshold: 6.016 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



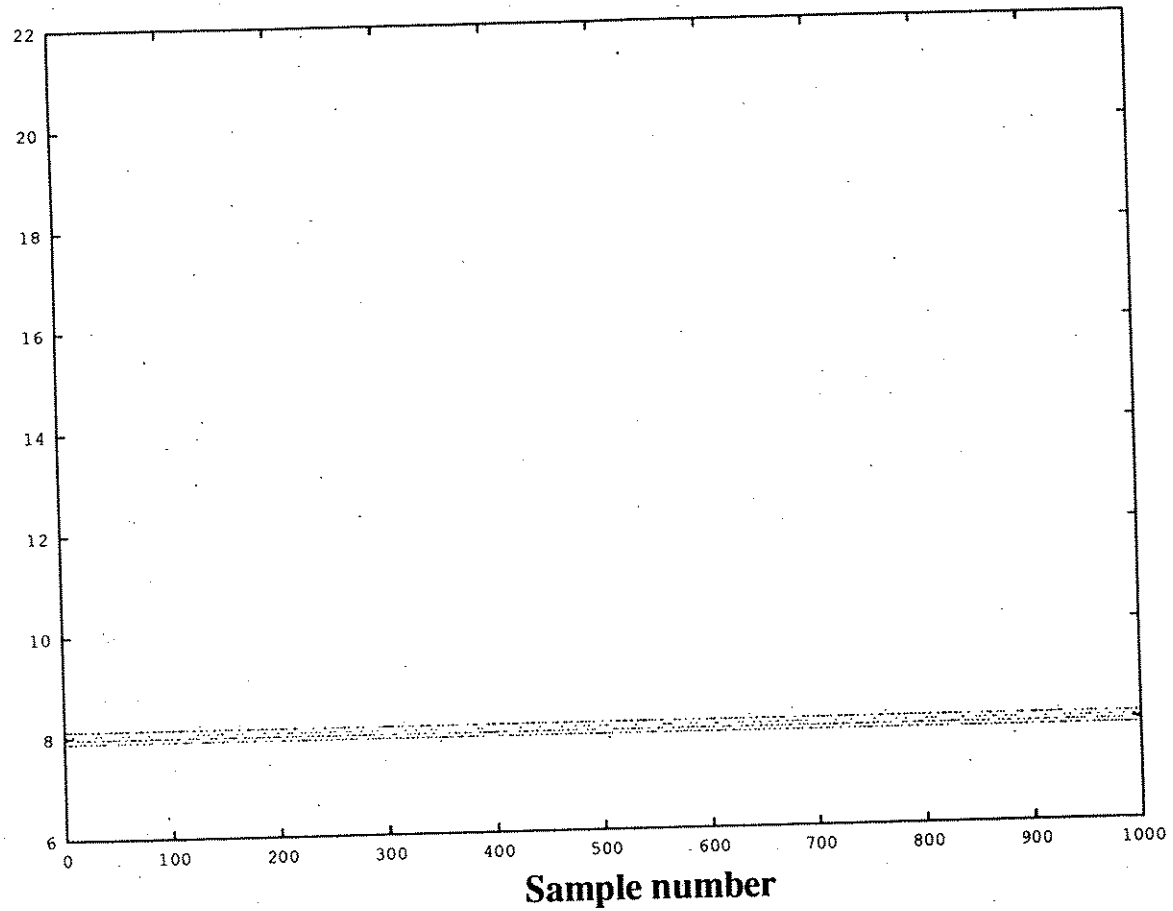
1000 samples
No asynchronous processor load
No background FDDI load
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 6.712 ms
99.9% threshold: 6.829 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



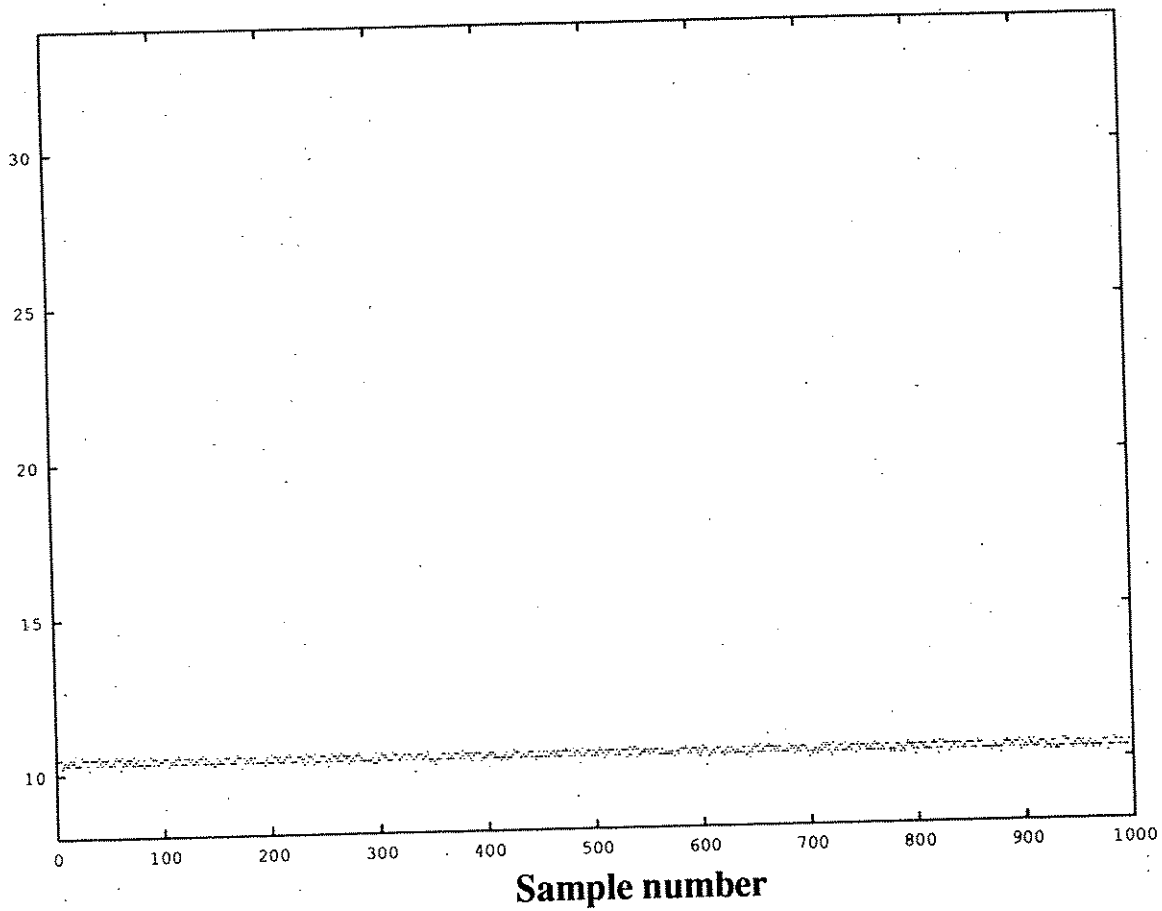
1000 samples
No asynchronous processor load
No background FDDI load
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 7.997 ms
99.9% threshold: 8.211 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



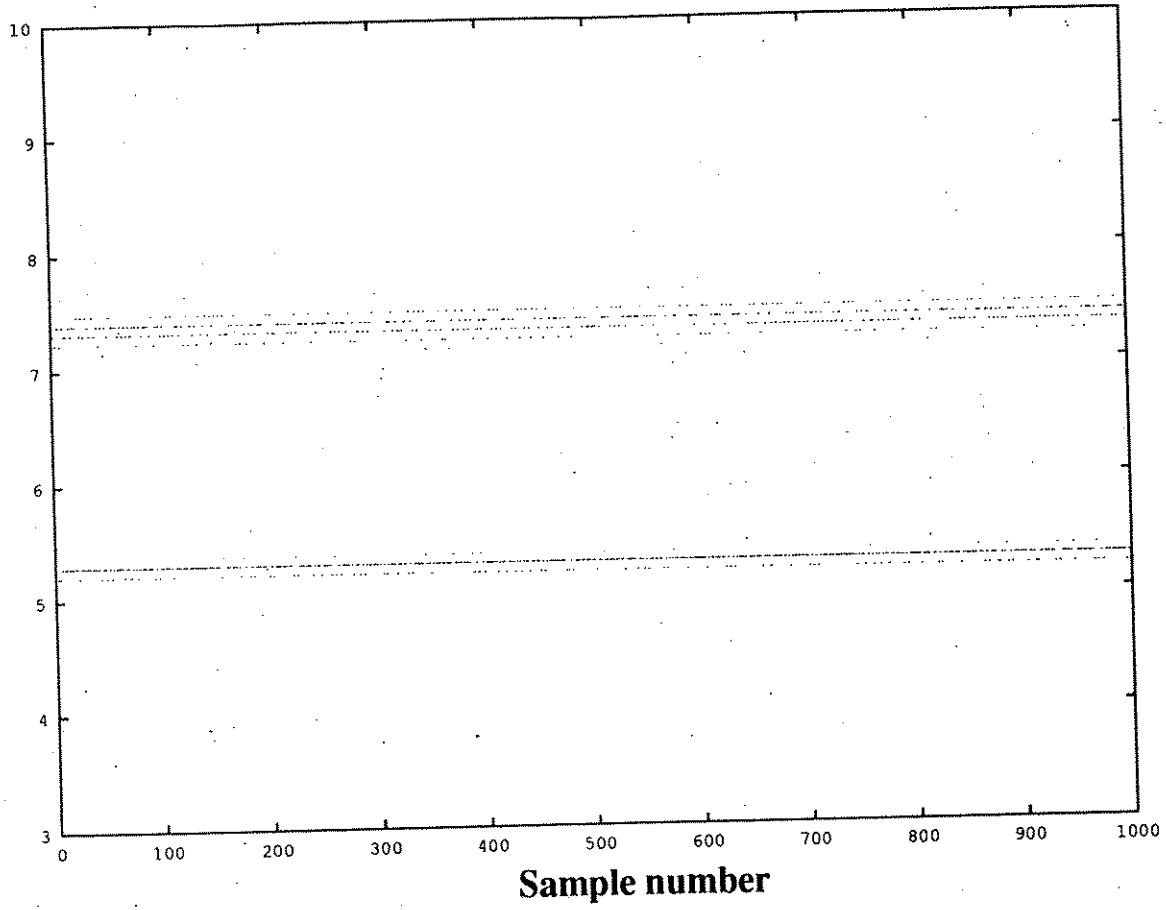
1000 samples
No asynchronous processor load
No background FDDI load
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: **10.402 ms**
99.9% threshold: **10.569 ms**

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 8 bytes



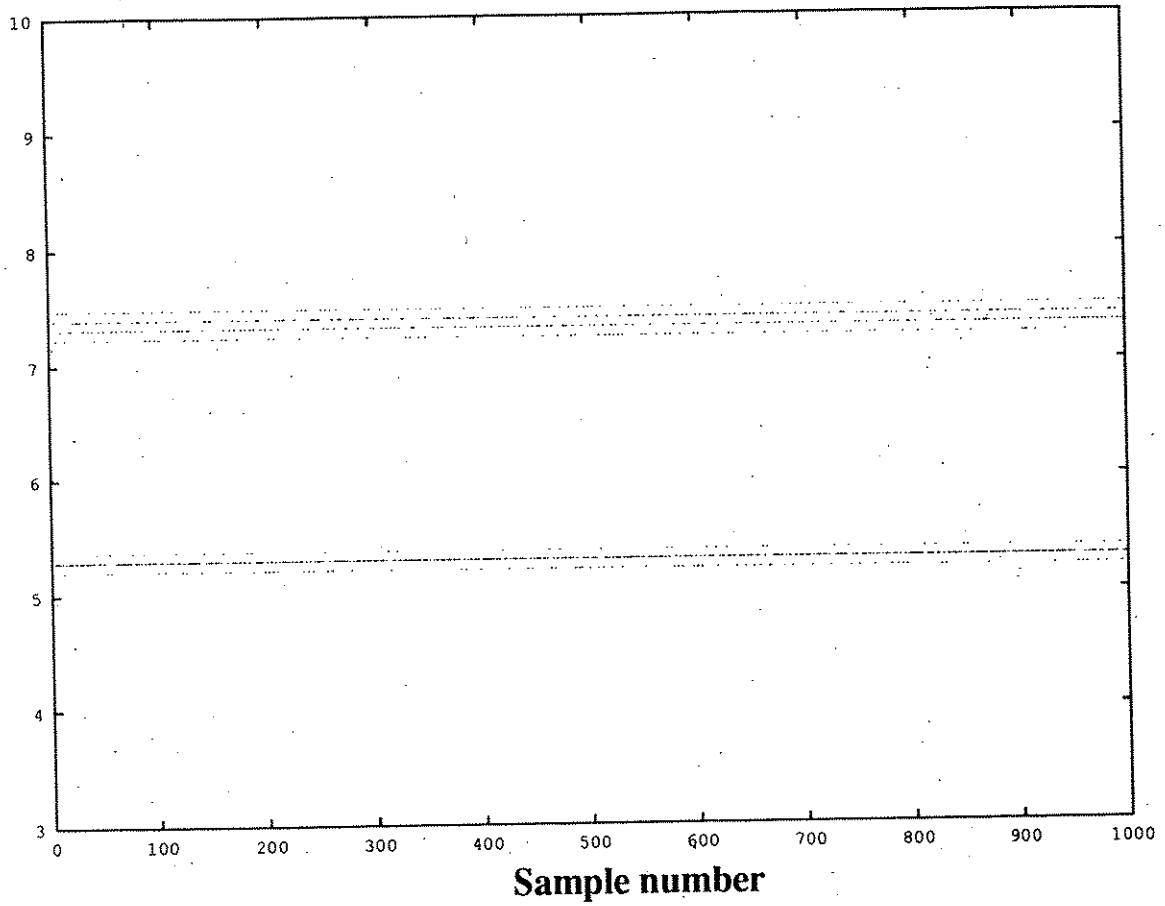
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 6.311 ms
99.9% threshold: 7.724 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 16 bytes



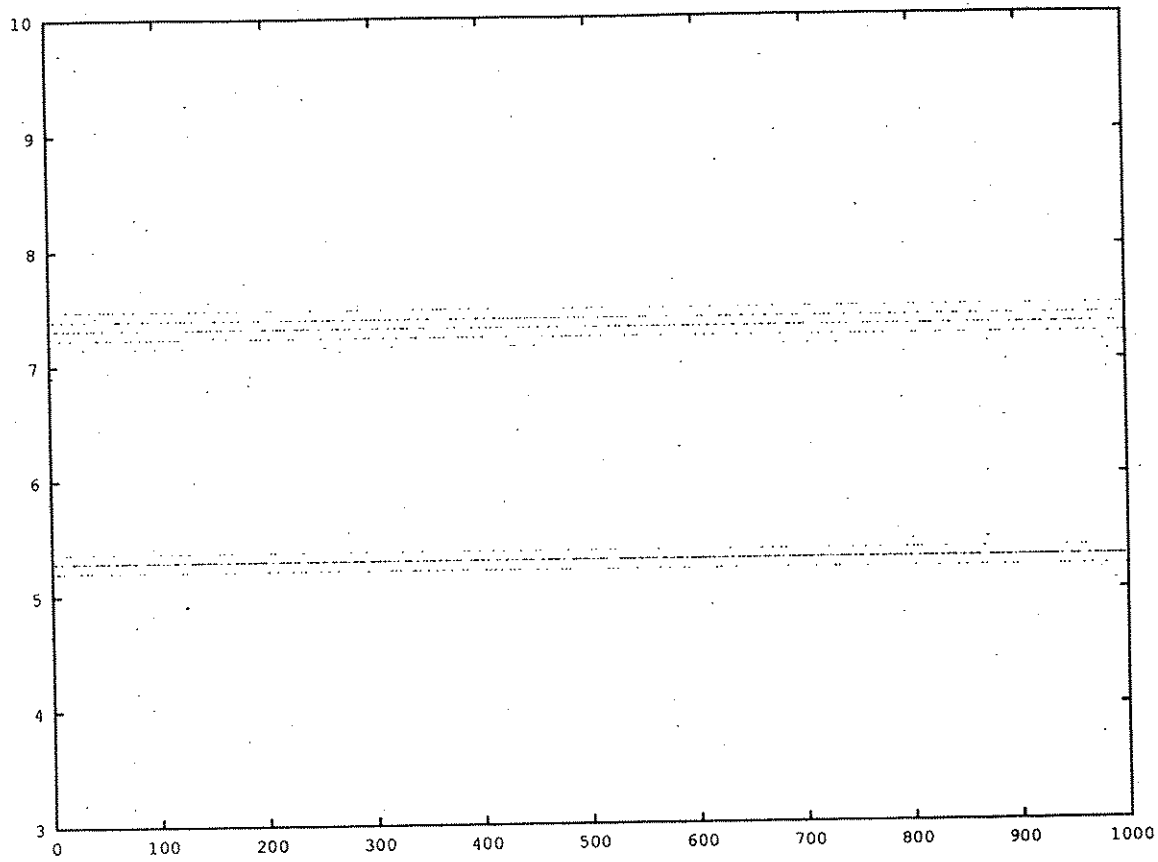
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 6.318 ms
99.9% threshold: 7.724 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 32 bytes



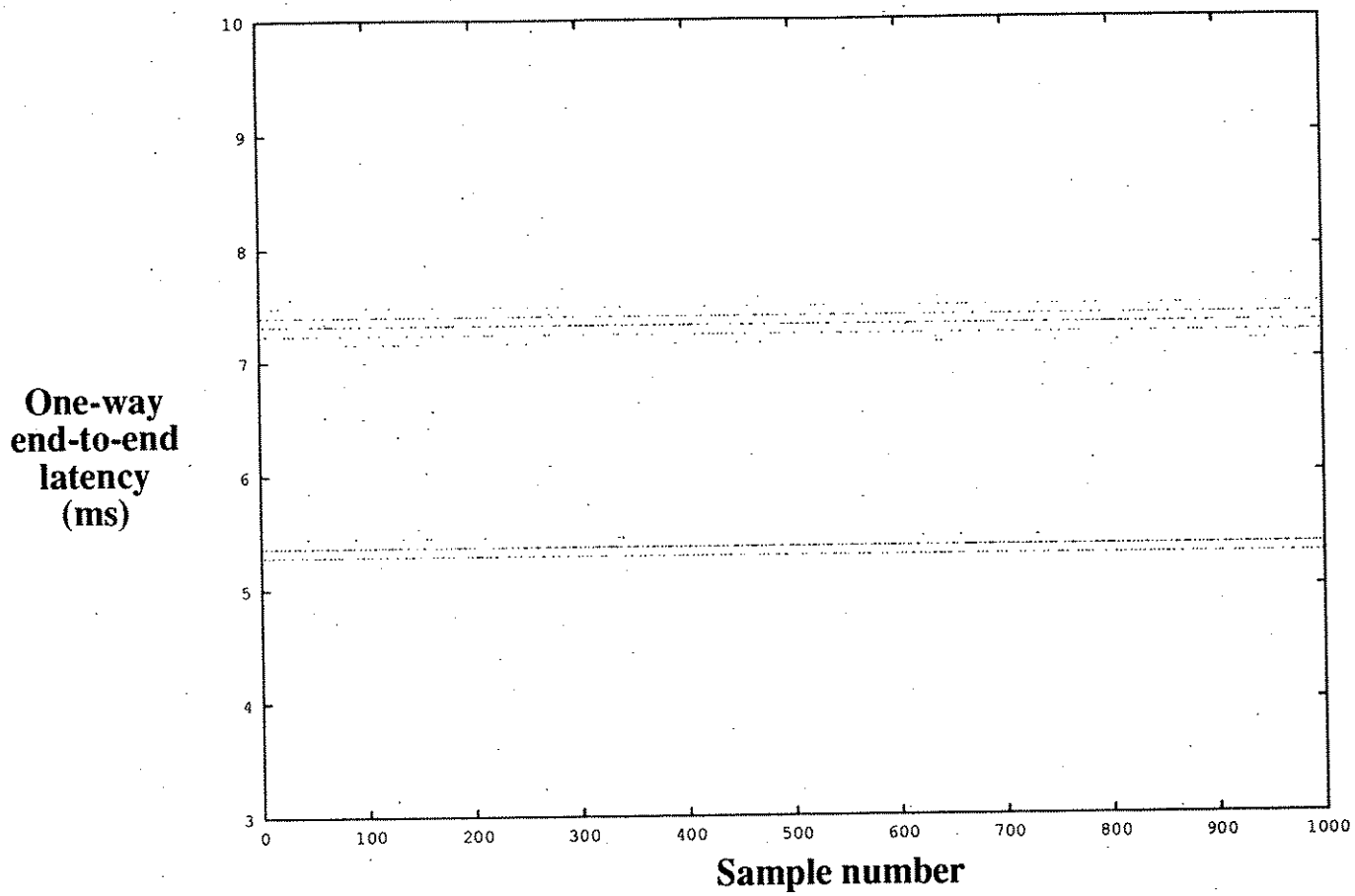
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 6.293 ms
99.9% threshold: 7.724 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 64 bytes



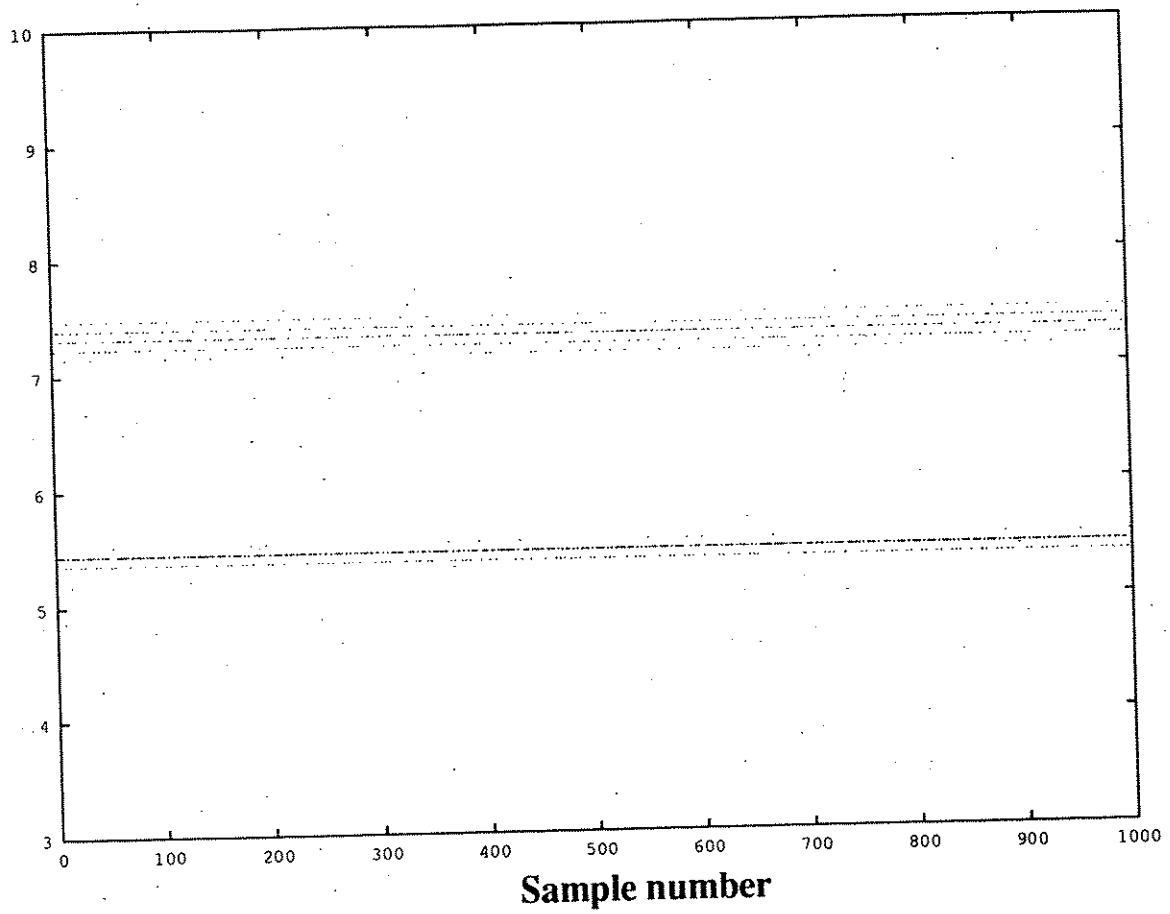
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 6.330 ms
99.9% threshold: 7.724 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 128 bytes



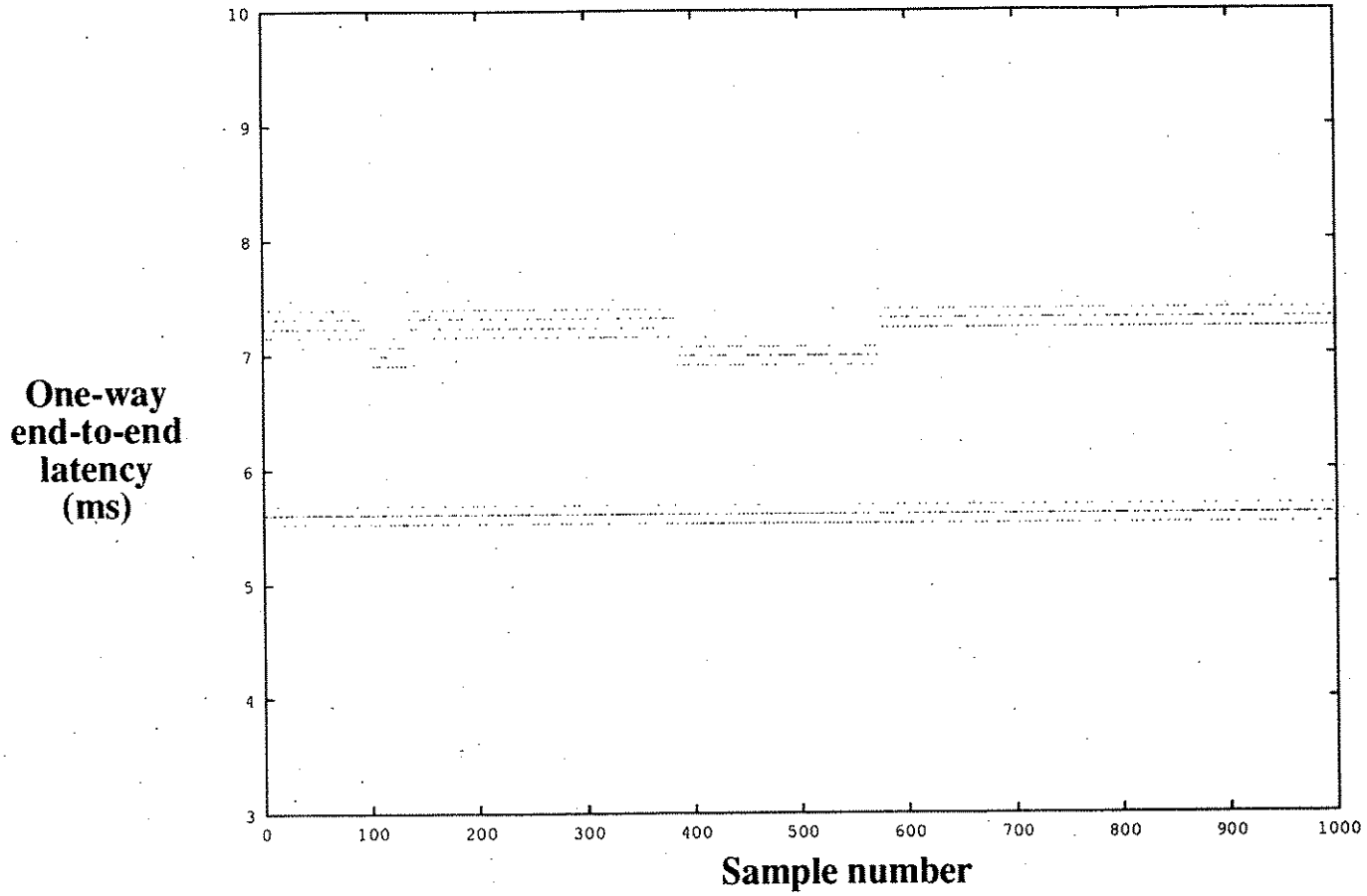
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 6.375 ms
99.9% threshold: 7.805 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 256 bytes



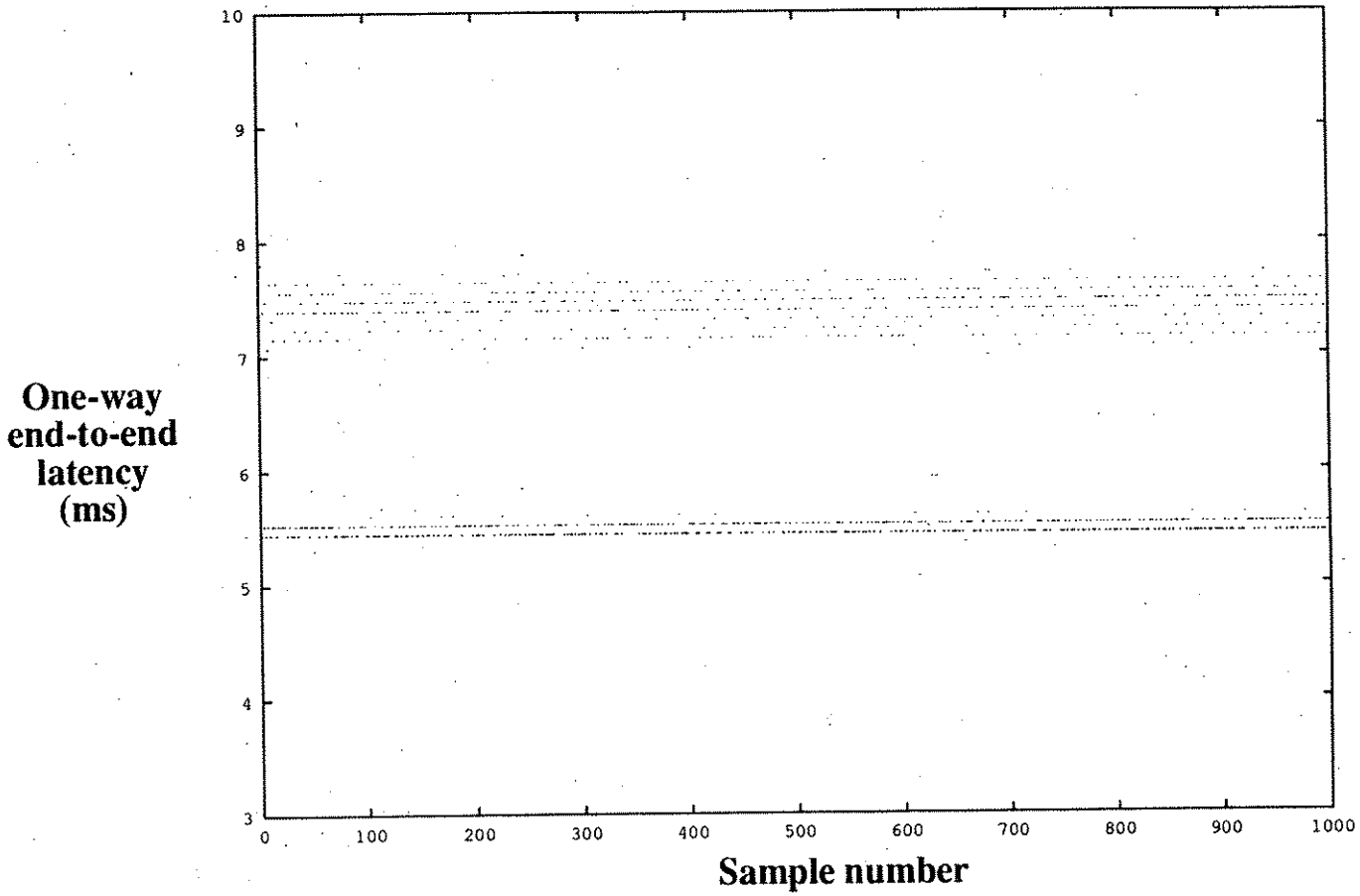
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 6.412 ms
99.9% threshold: 7.886 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 512 bytes



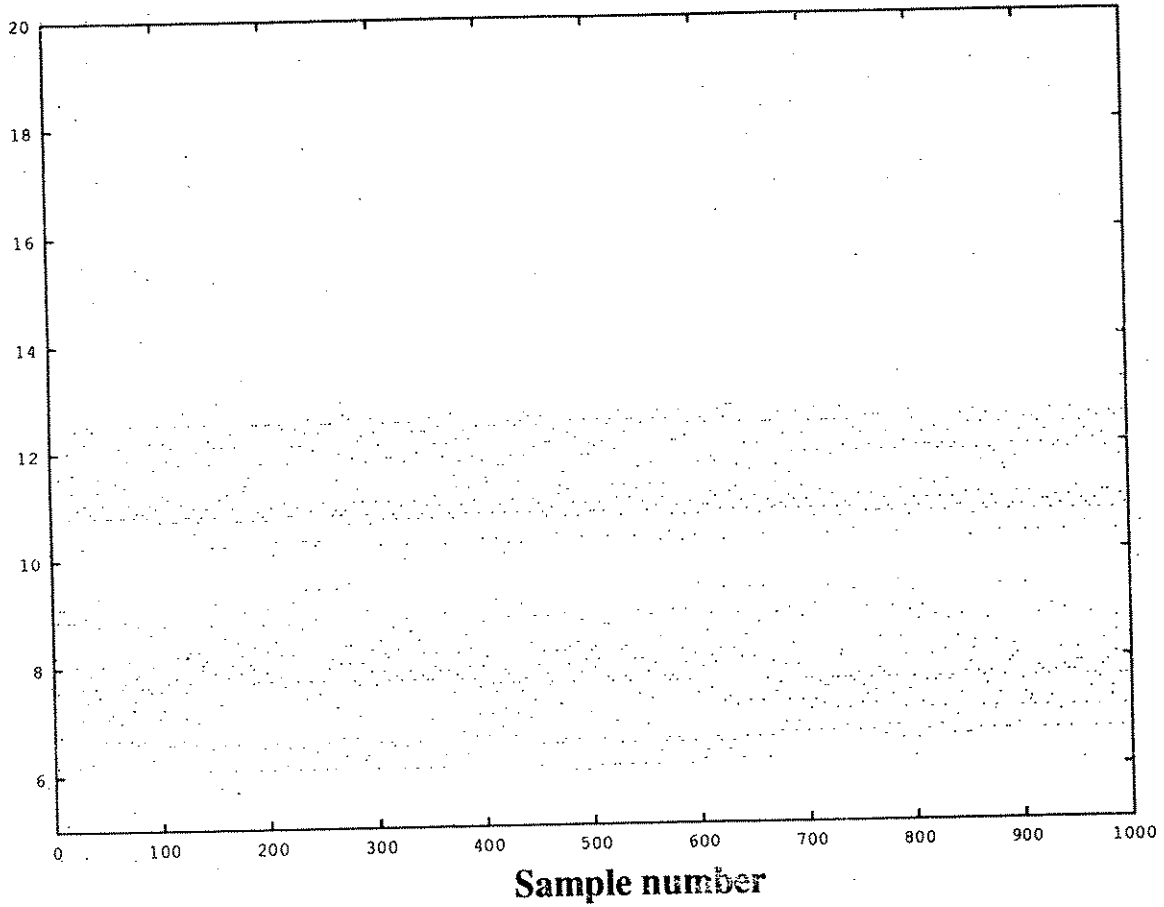
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 6.461 ms
99.9% threshold: 7.967 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



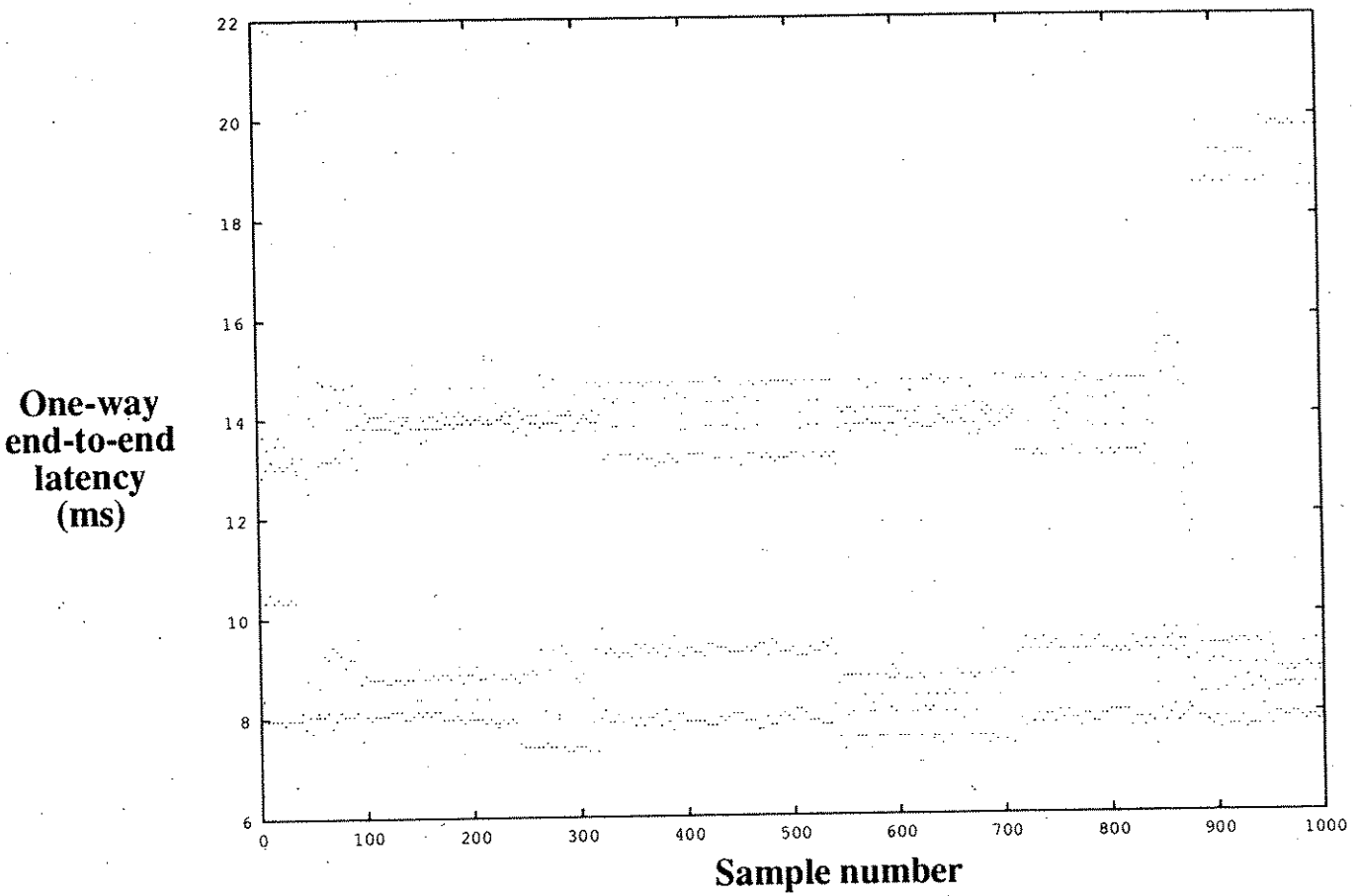
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 9.515 ms
99.9% threshold: 13.171 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



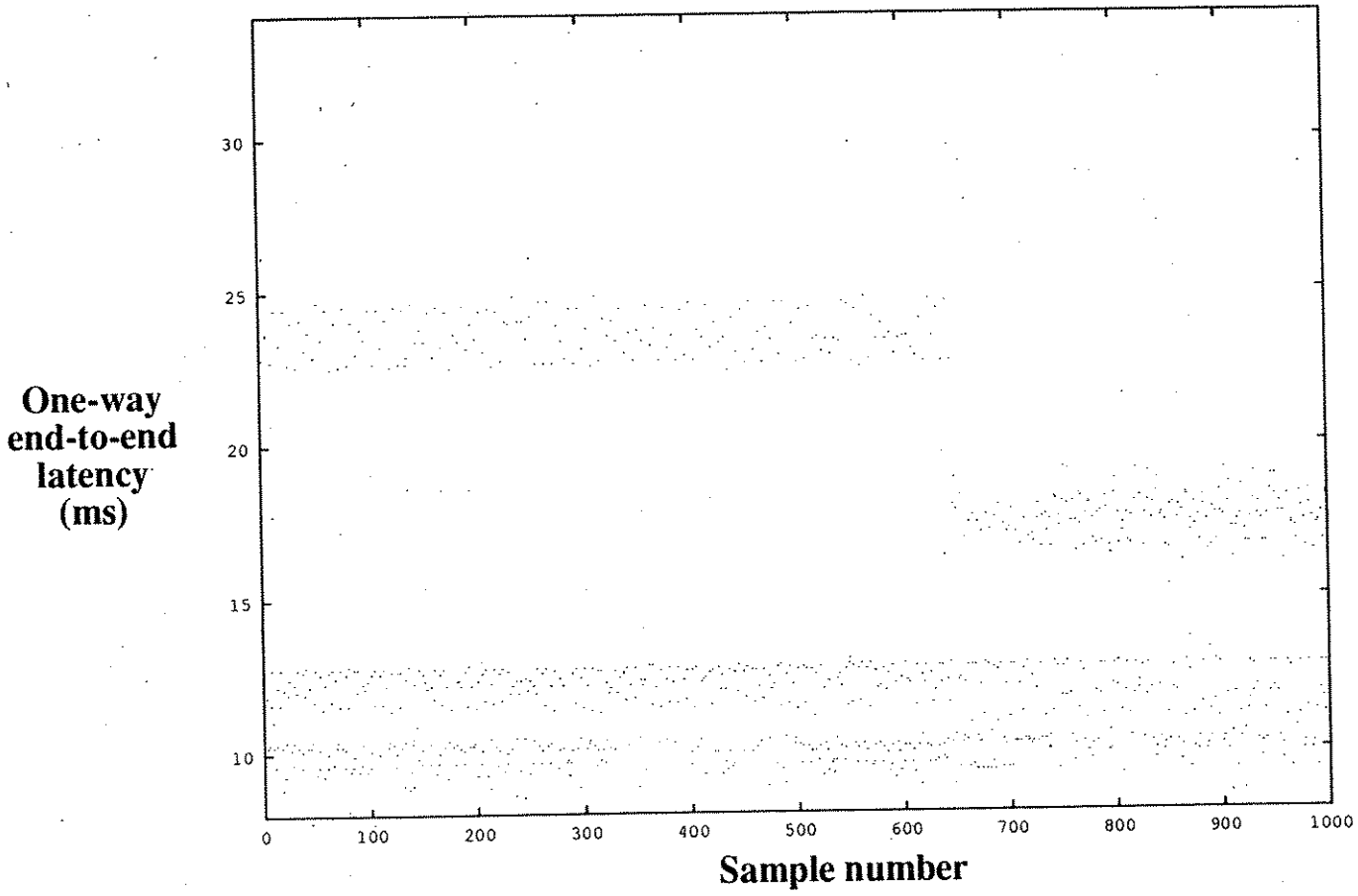
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 11.174 ms
99.9% threshold: 19.837 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



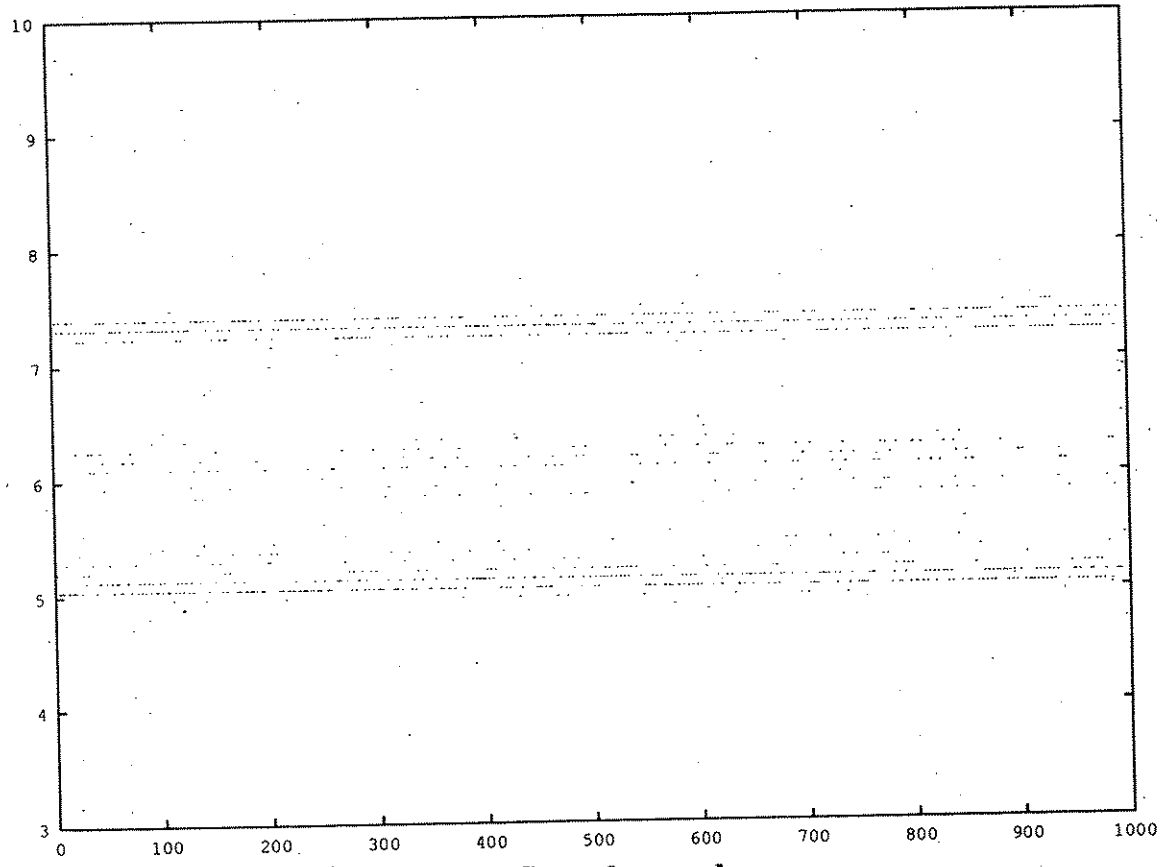
1000 samples
No asynchronous processor load
25 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 14.134 ms
99.9% threshold: 24.878 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 8 bytes



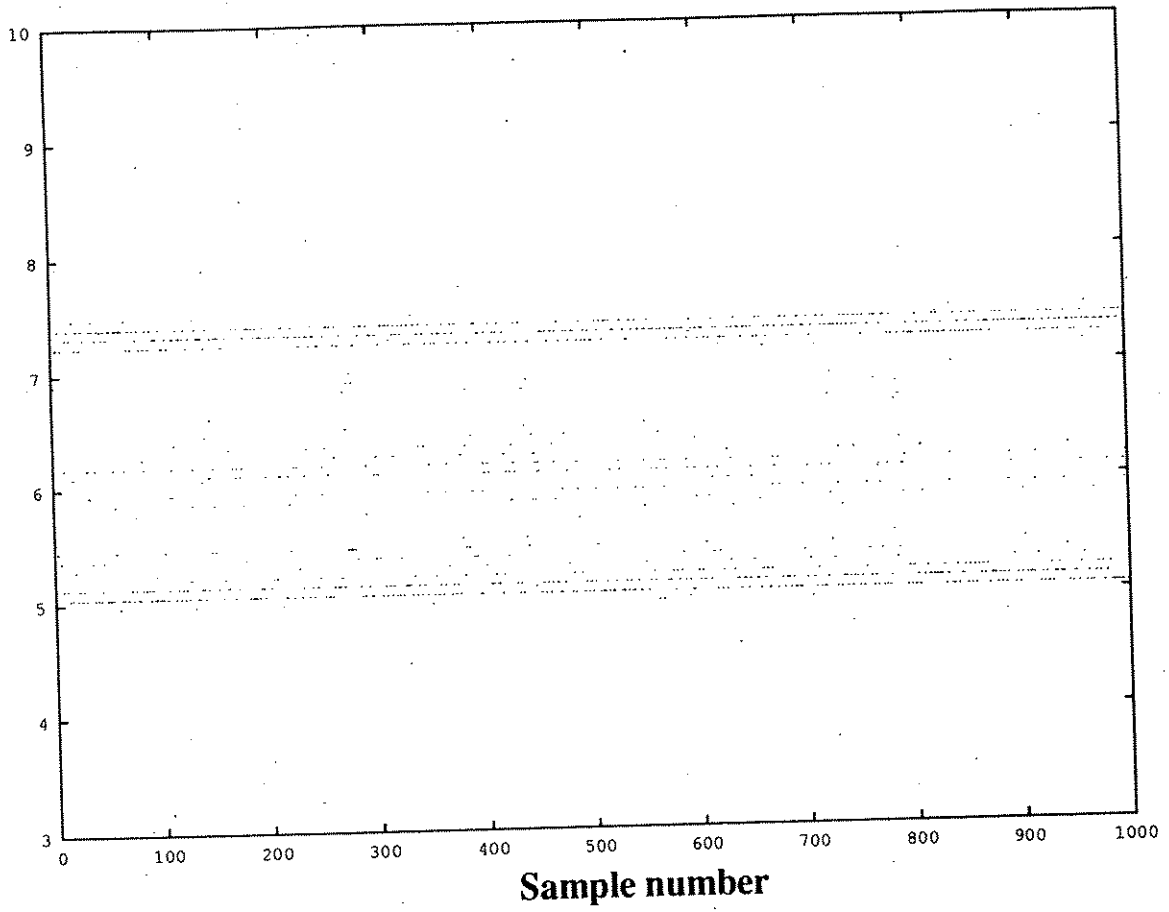
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 6.209 ms
99.9% threshold: 7.724 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 16 bytes



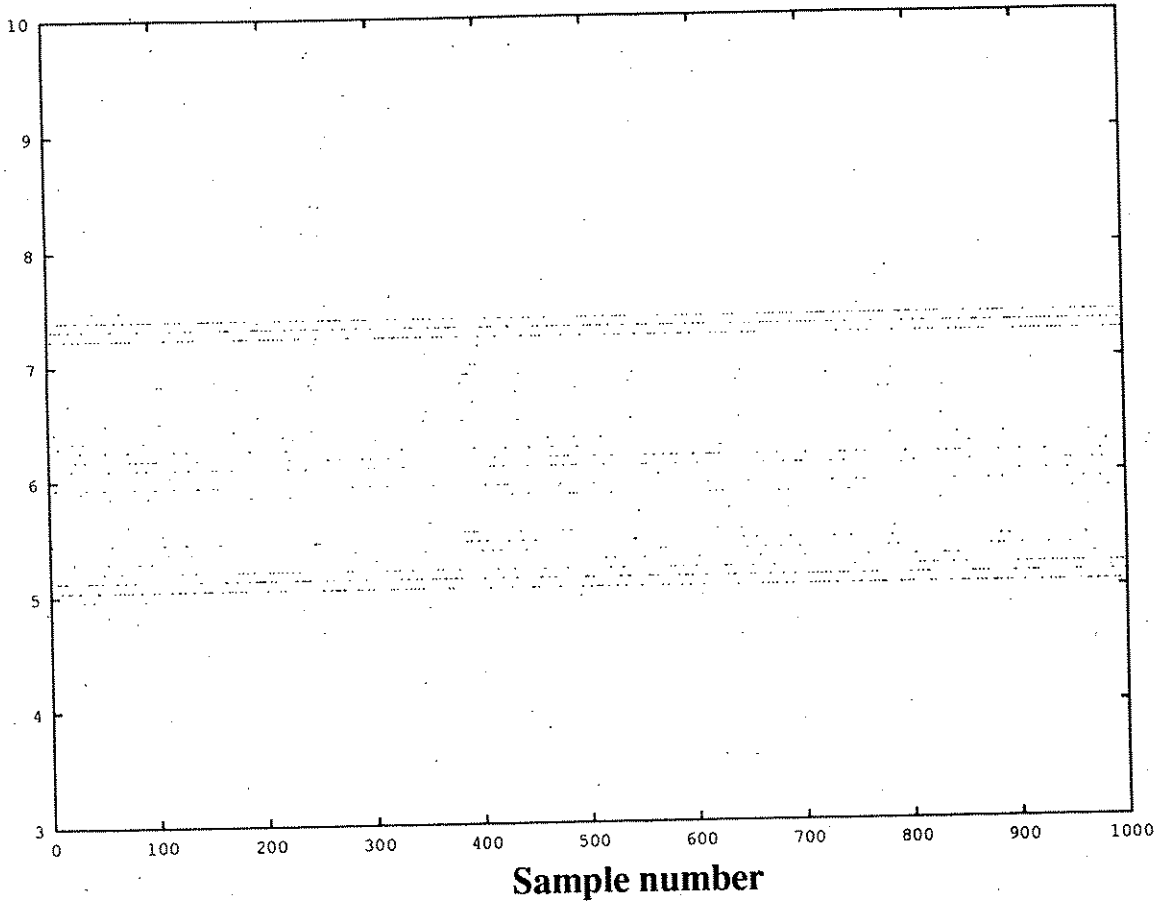
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 6.213 ms
99.9% threshold: 7.724 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 32 bytes



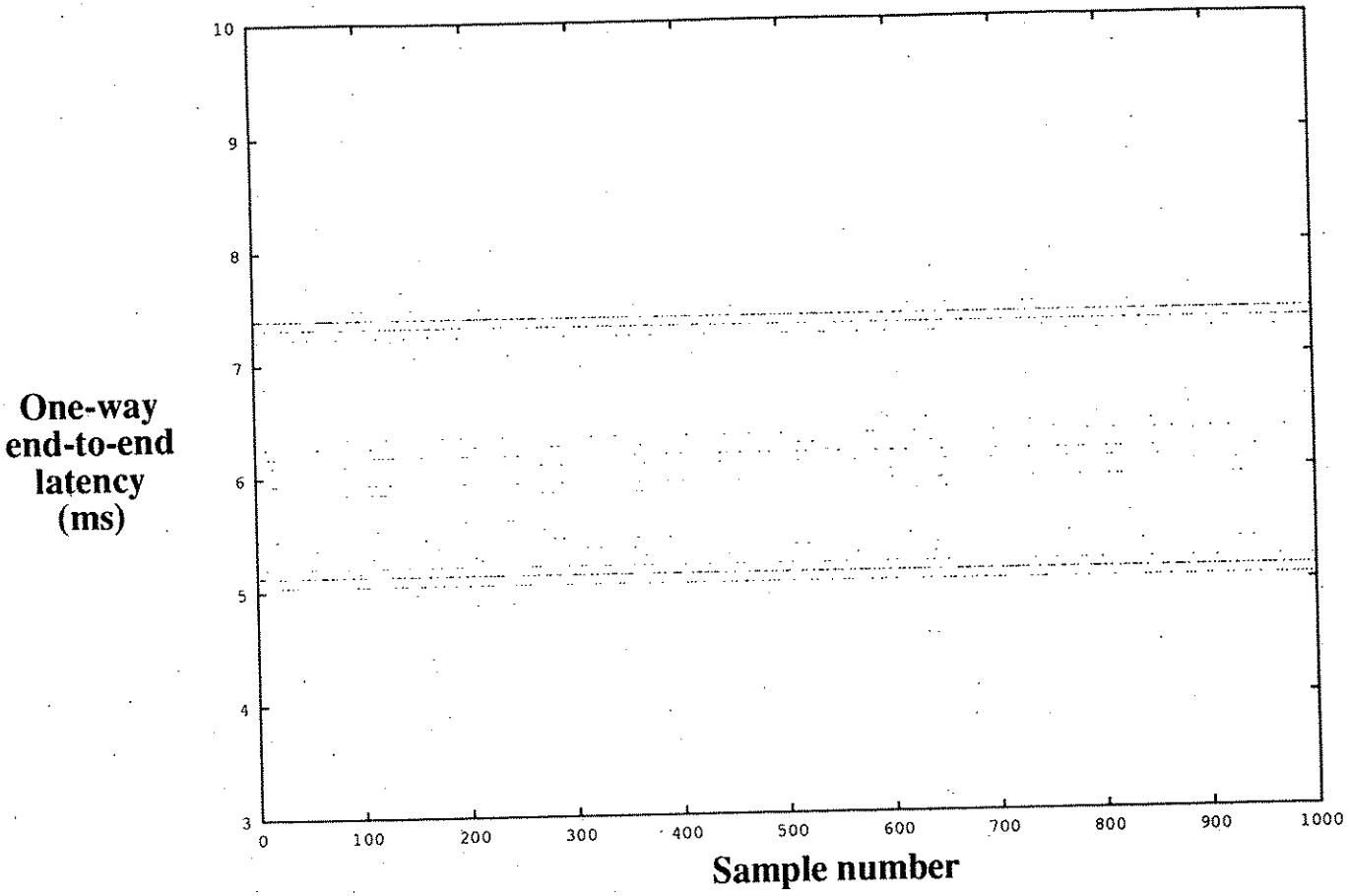
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 6.240 ms
99.9% threshold: 9.756 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 64 bytes



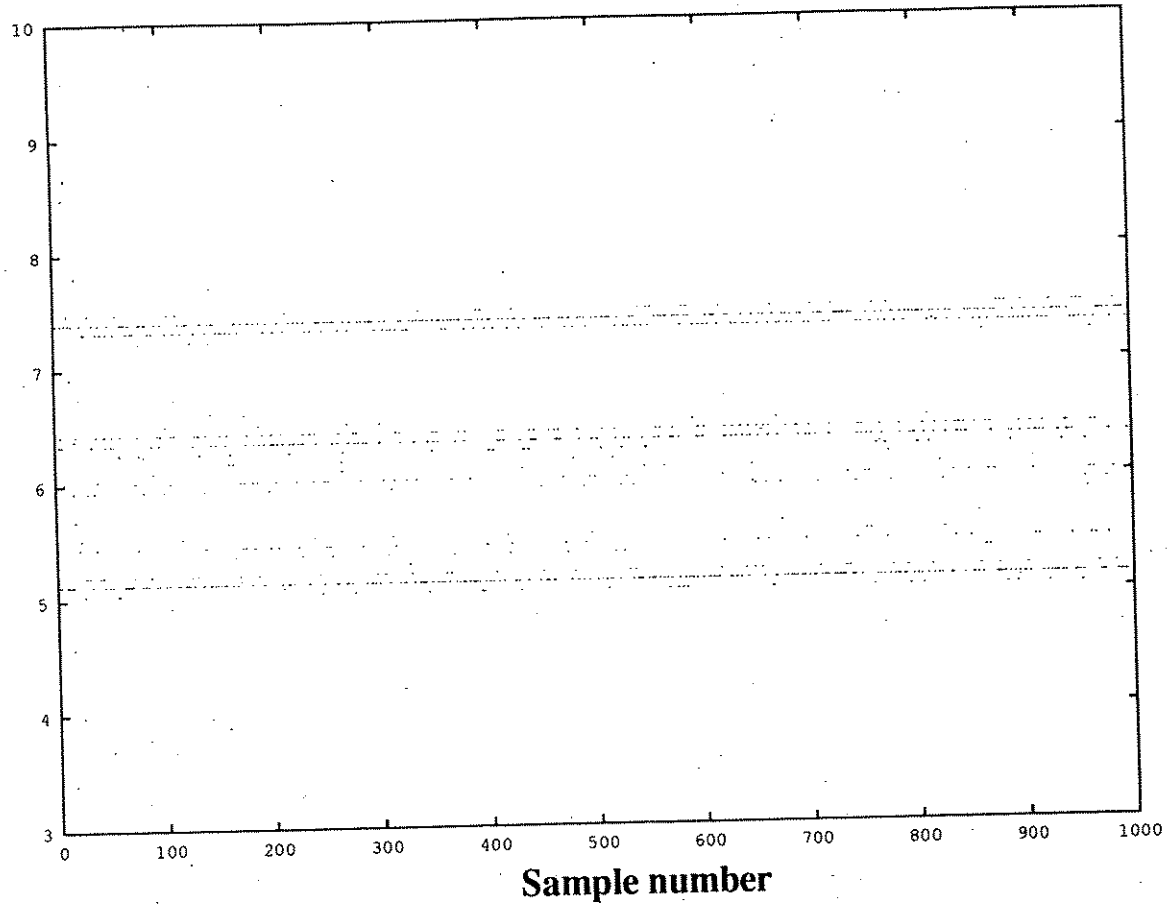
Average latency: 6.247 ms
99.9% threshold: 7.480 ms

1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 128 bytes



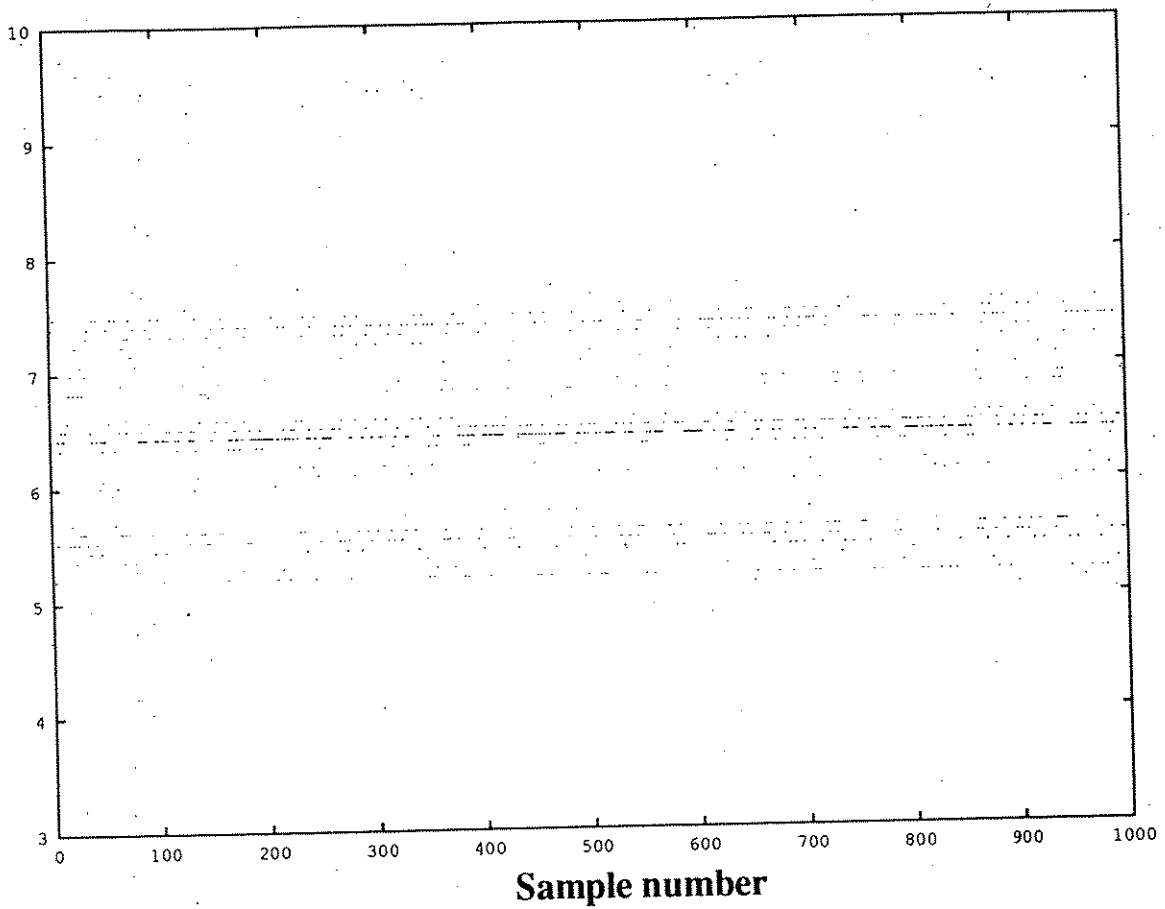
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 6.290 ms
99.9% threshold: 7.805 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 256 bytes



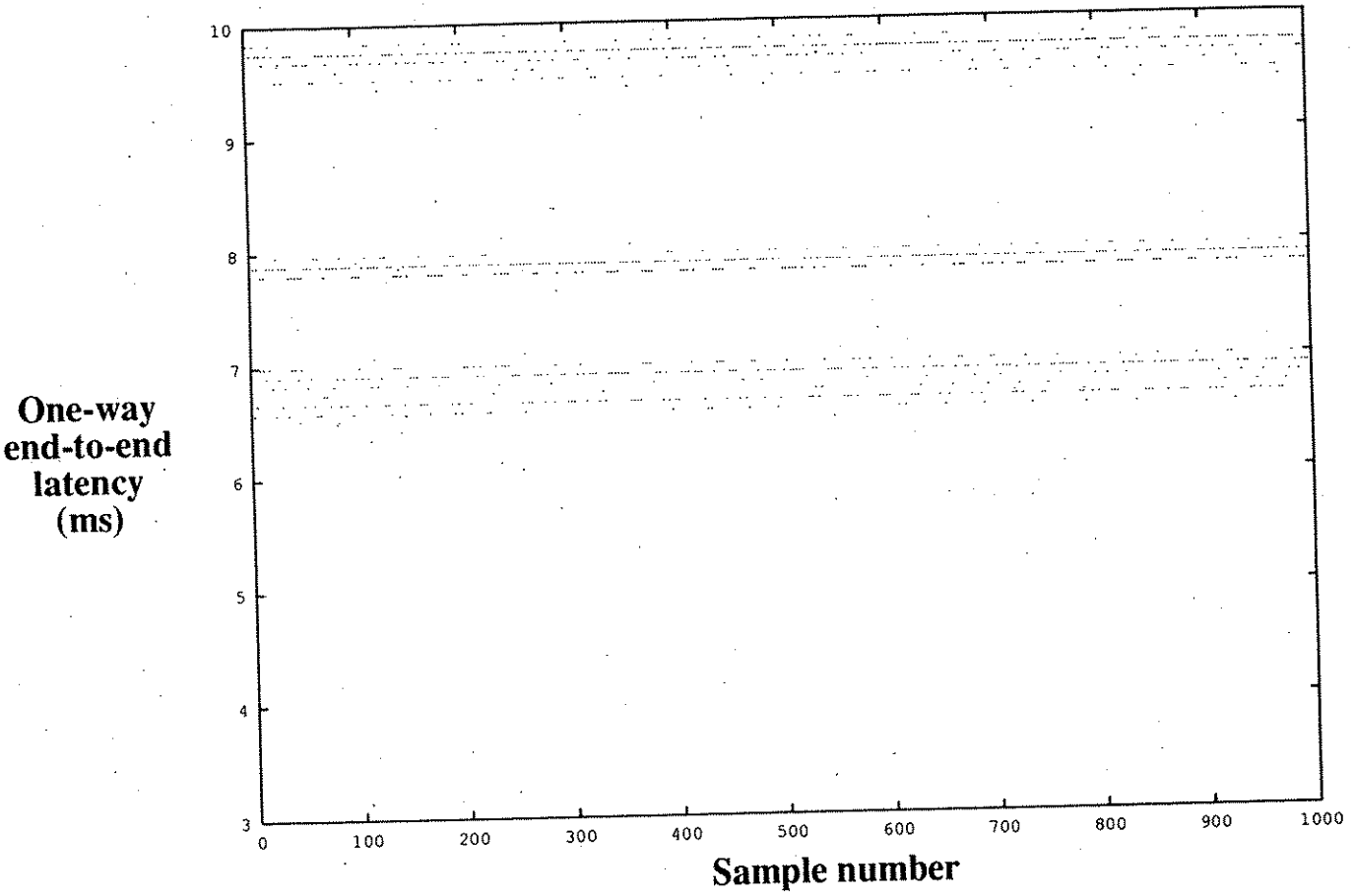
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 6.609 ms
99.9% threshold: 10.488 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 512 bytes



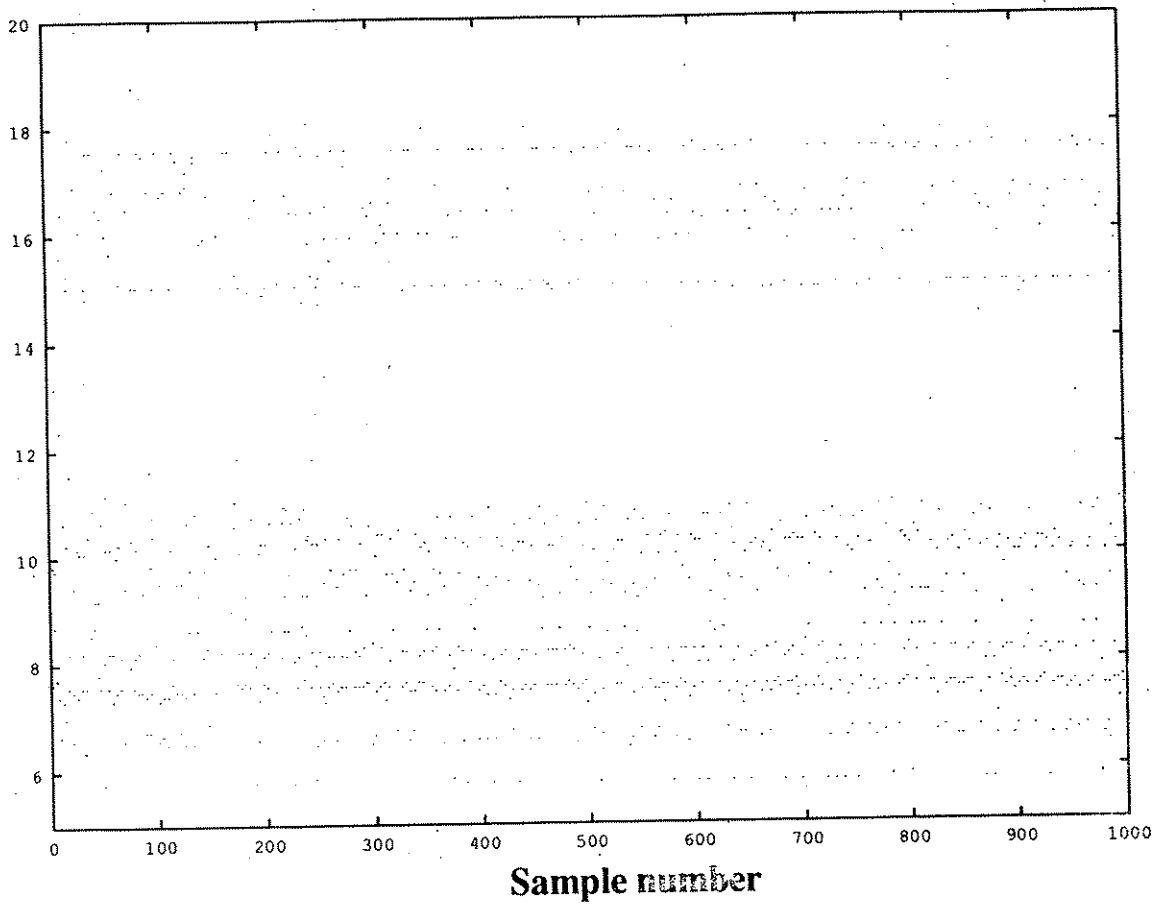
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 8.136 ms
99.9% threshold: 10.163 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



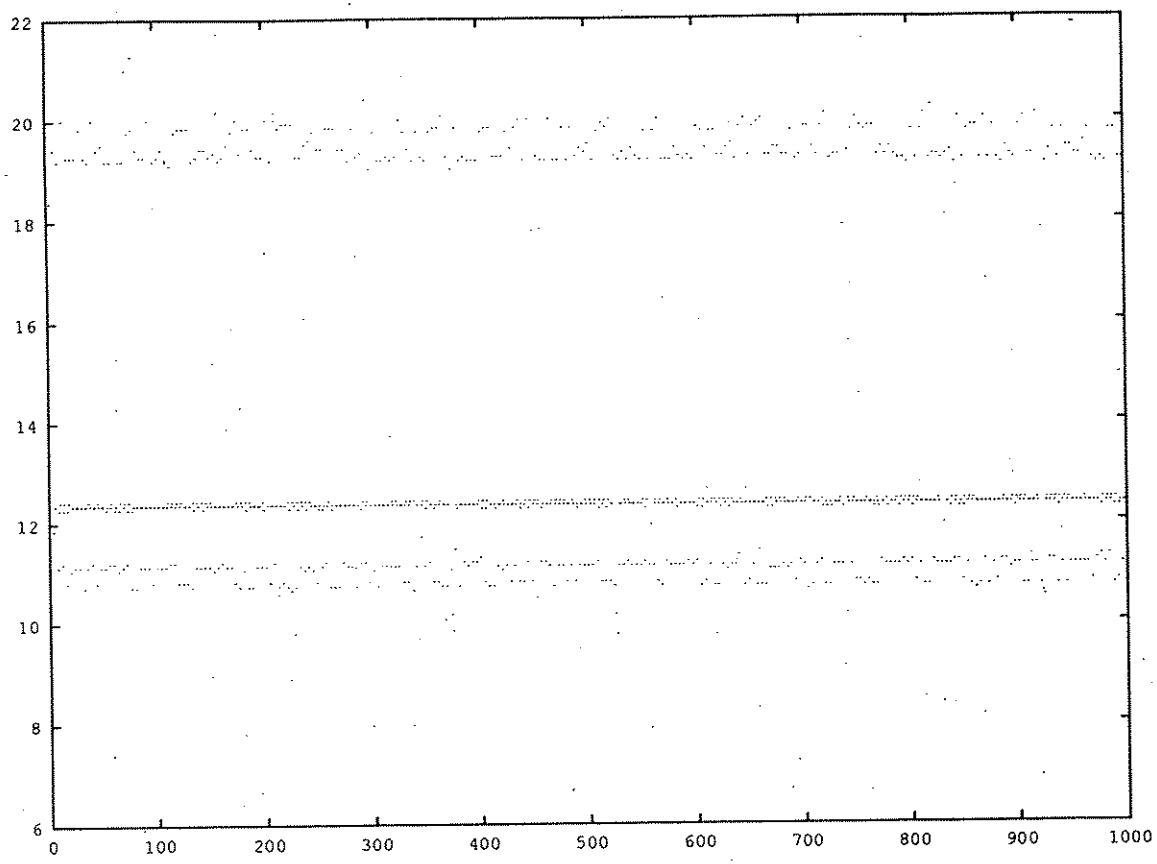
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 10.460 ms
99.9% threshold: 17.967 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



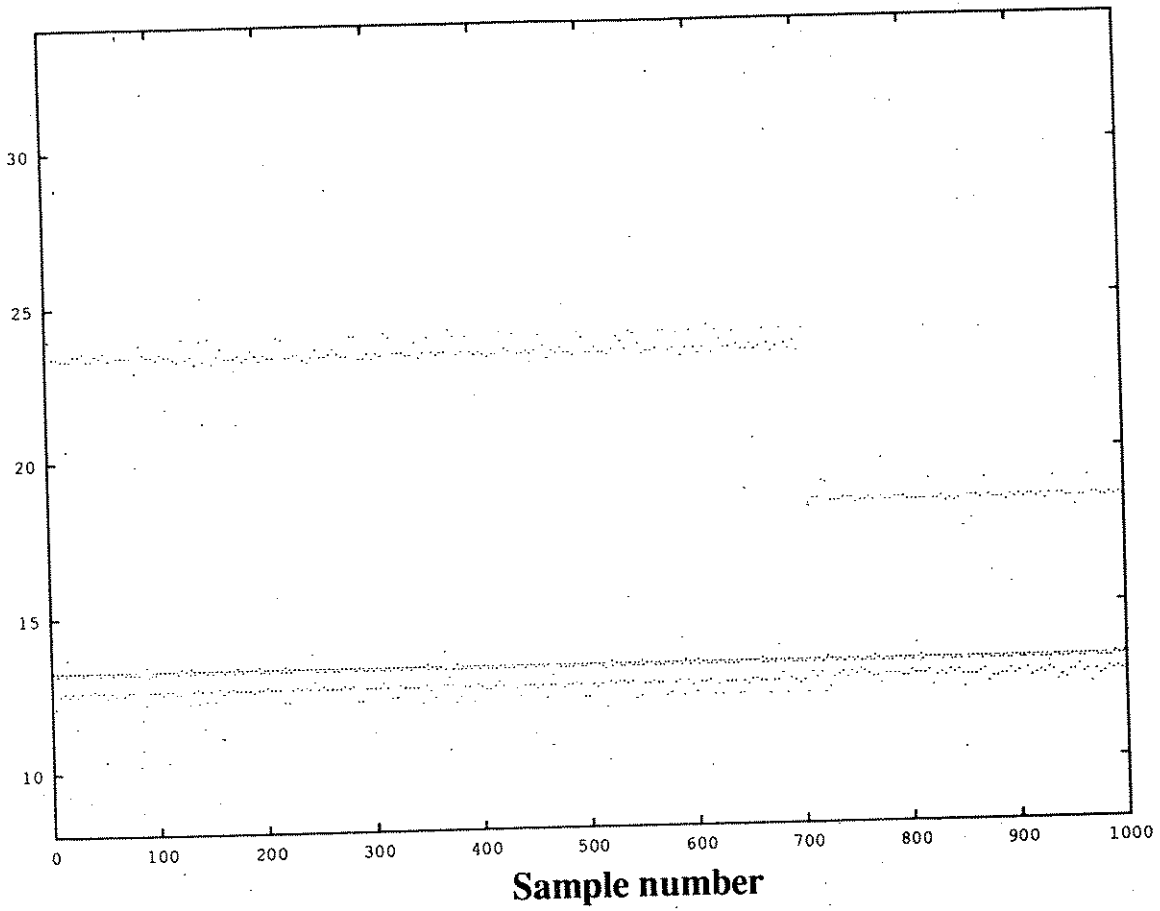
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 13.761 ms
99.9% threshold: 20.081 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



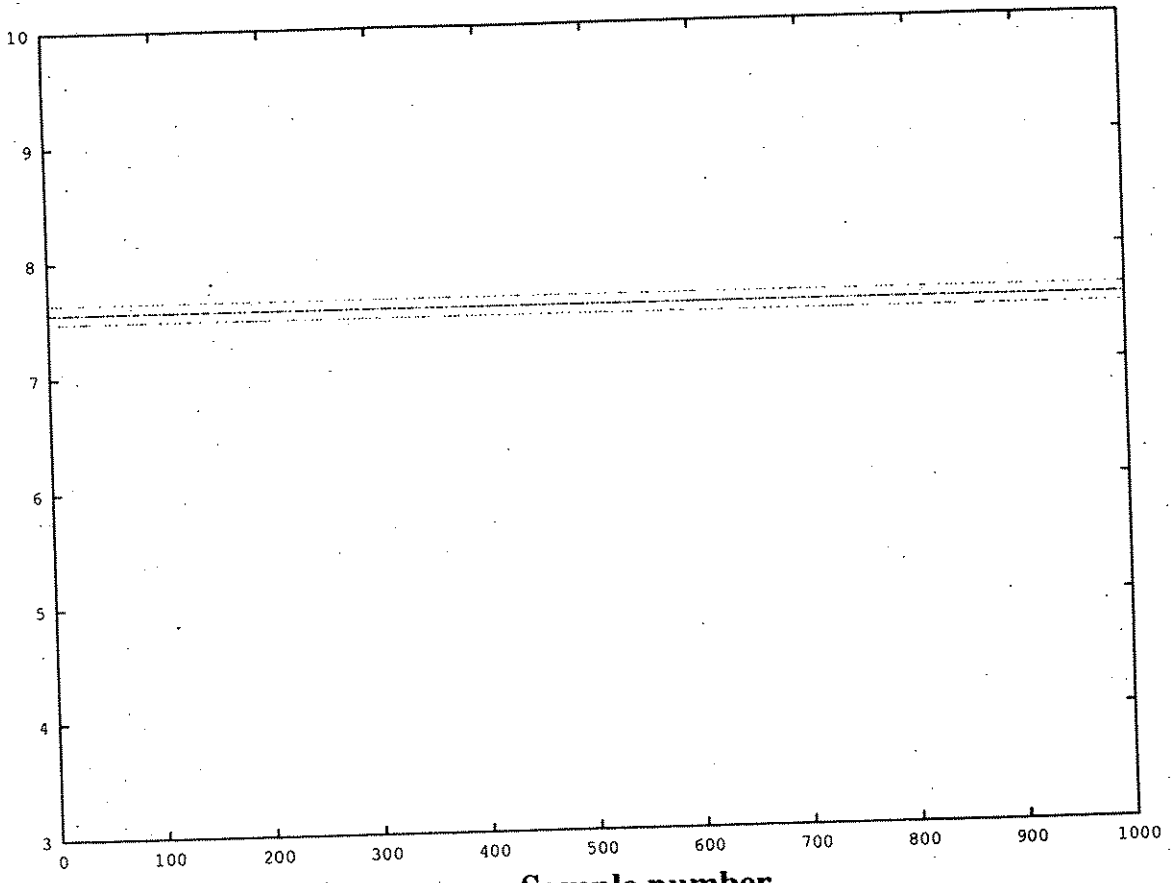
1000 samples
No asynchronous processor load
50 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 15.247 ms
99.9% threshold: 24.146 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 8 bytes



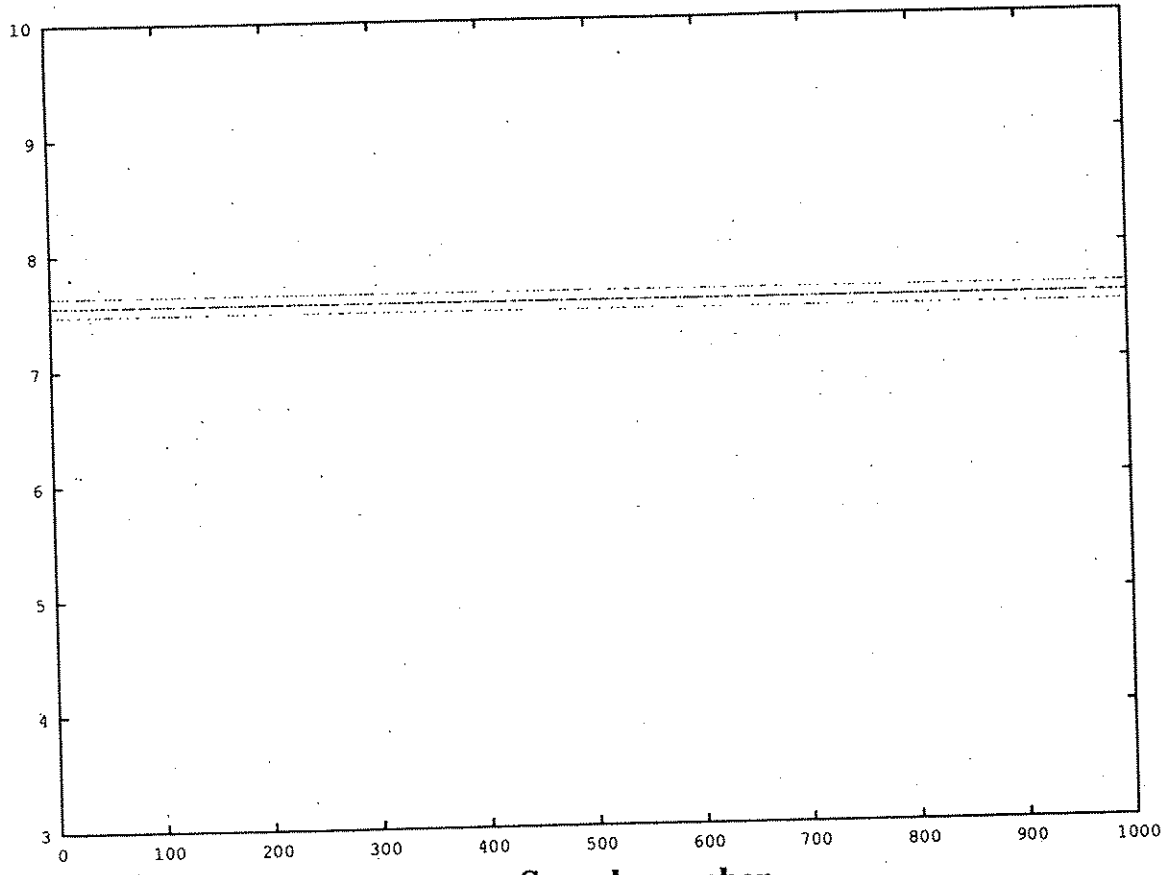
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 7.559 ms
99.9% threshold: 7.805 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 16 bytes



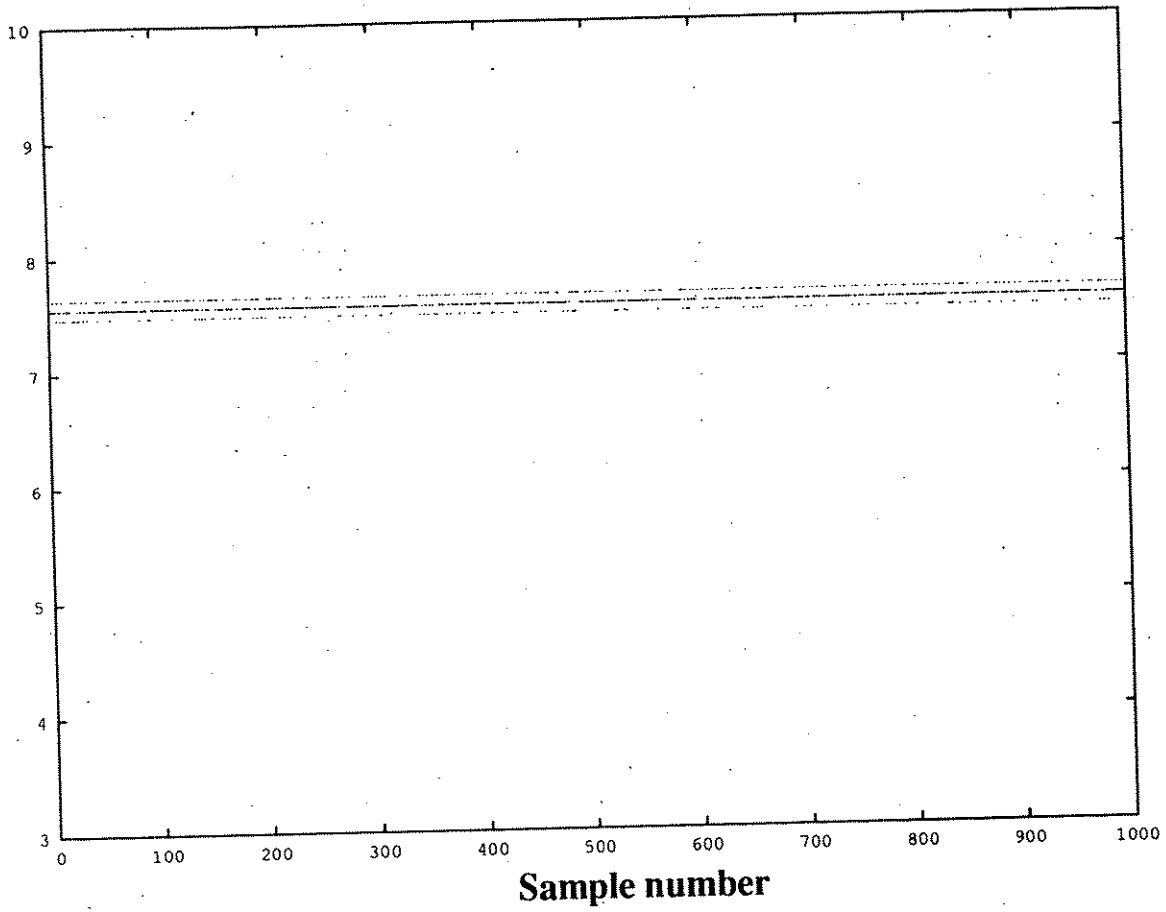
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 7.564 ms
99.9% threshold: 8.537 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 32 bytes



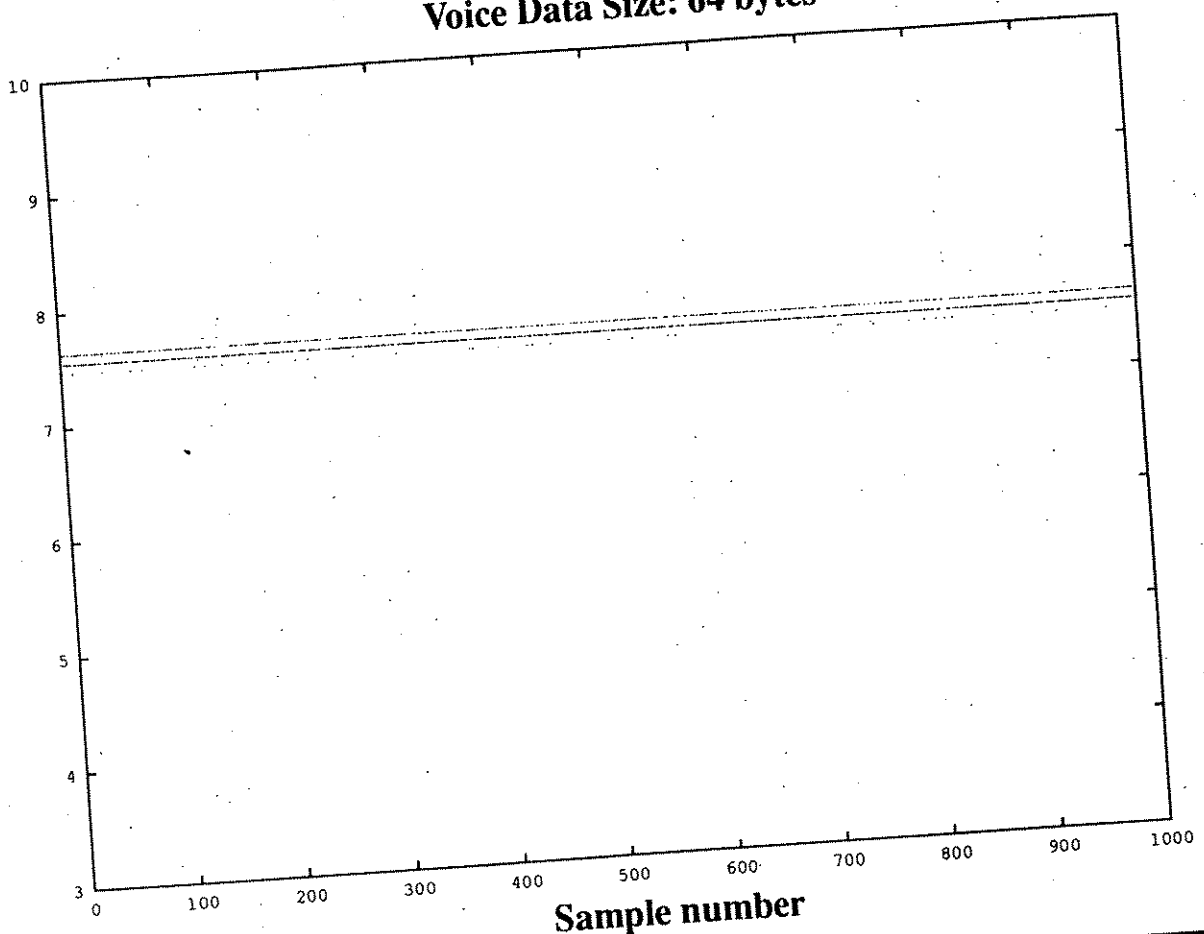
Average latency: 7.573 ms
99.9% threshold: 8.347 ms

1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 64 bytes



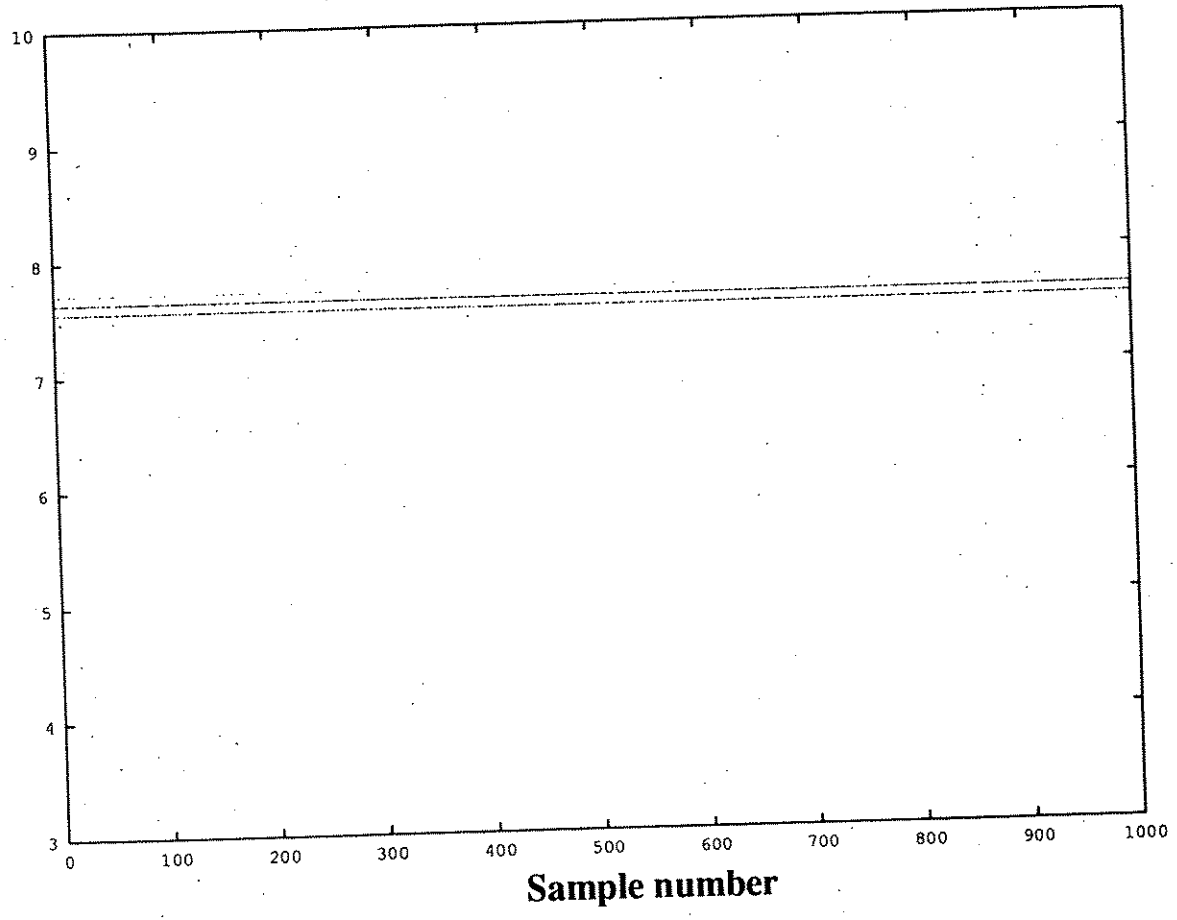
Average latency: 7.588 ms
99.9% threshold: 8.293 ms

1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 128 bytes



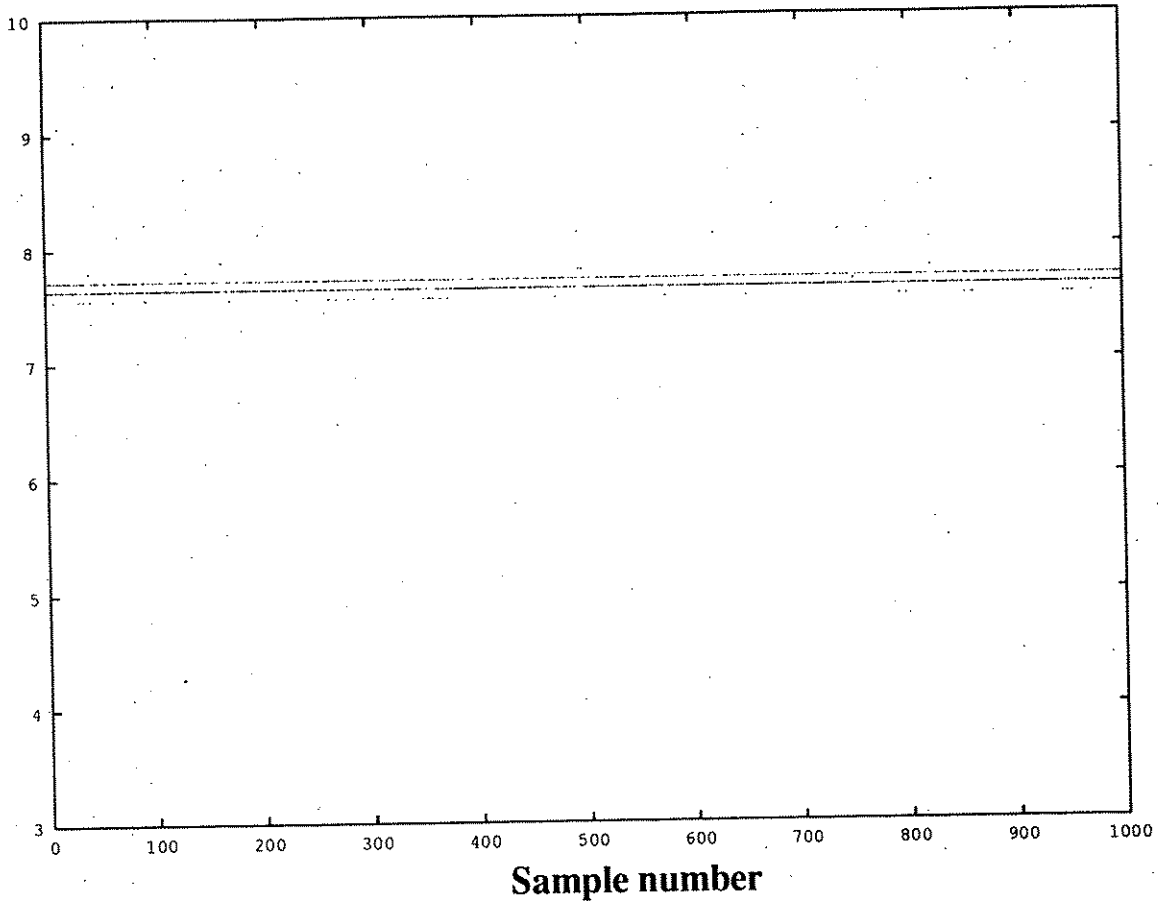
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 7.608 ms
99.9% threshold: 8.211 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 256 bytes



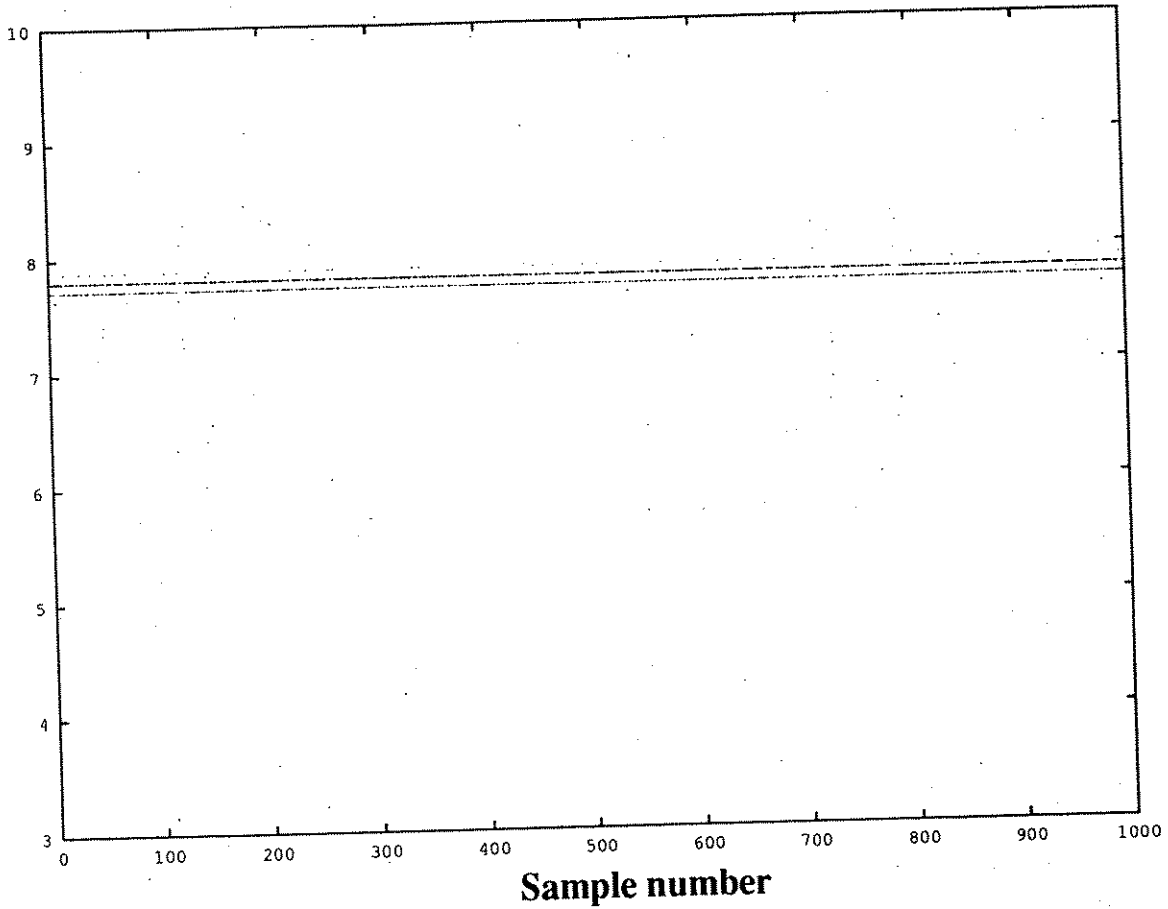
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 7.673 ms
99.9% threshold: 9.431 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 512 bytes



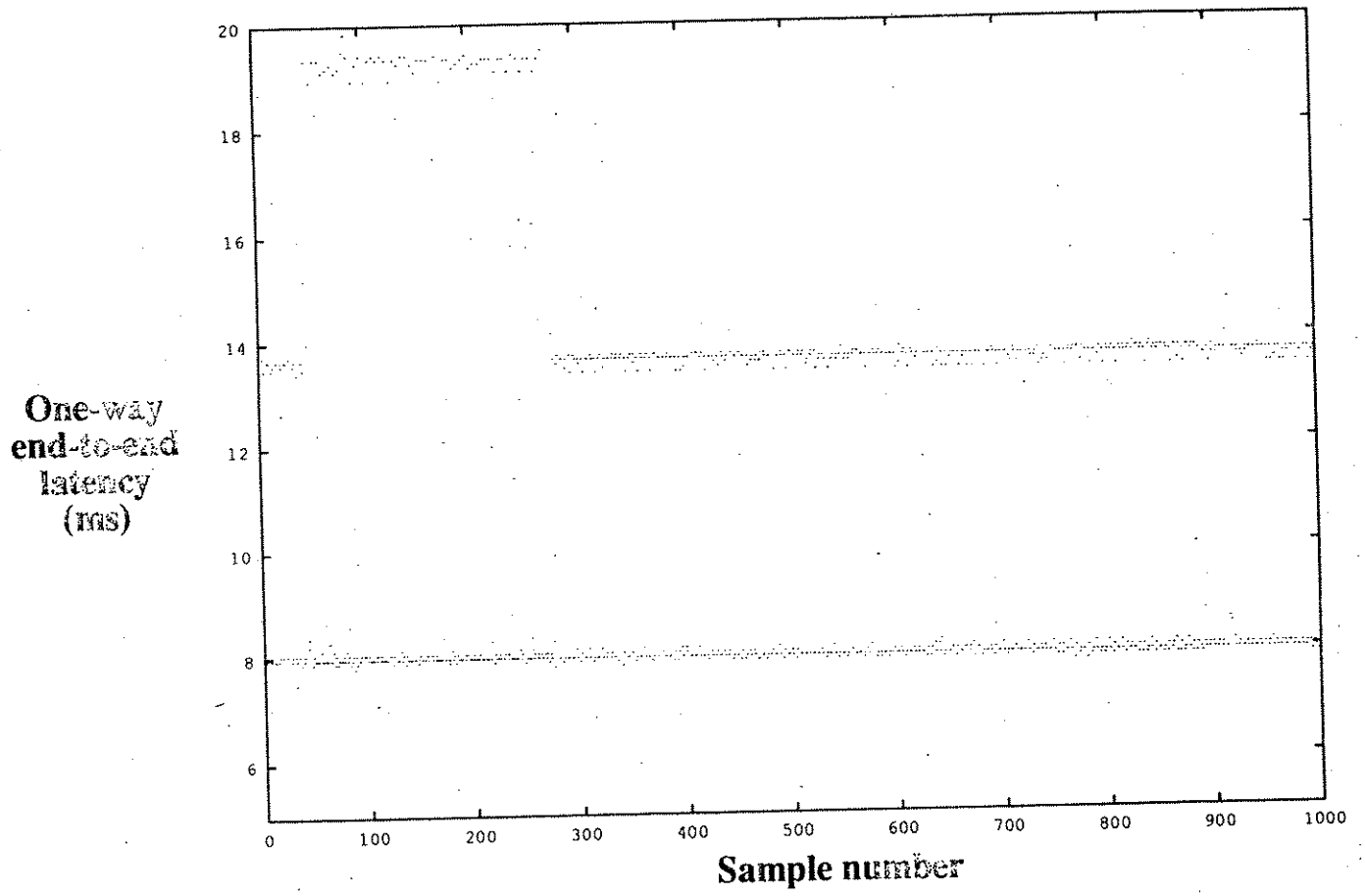
Average latency: 7.770 ms
99.9% threshold: 8.293 ms

1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 1024 bytes



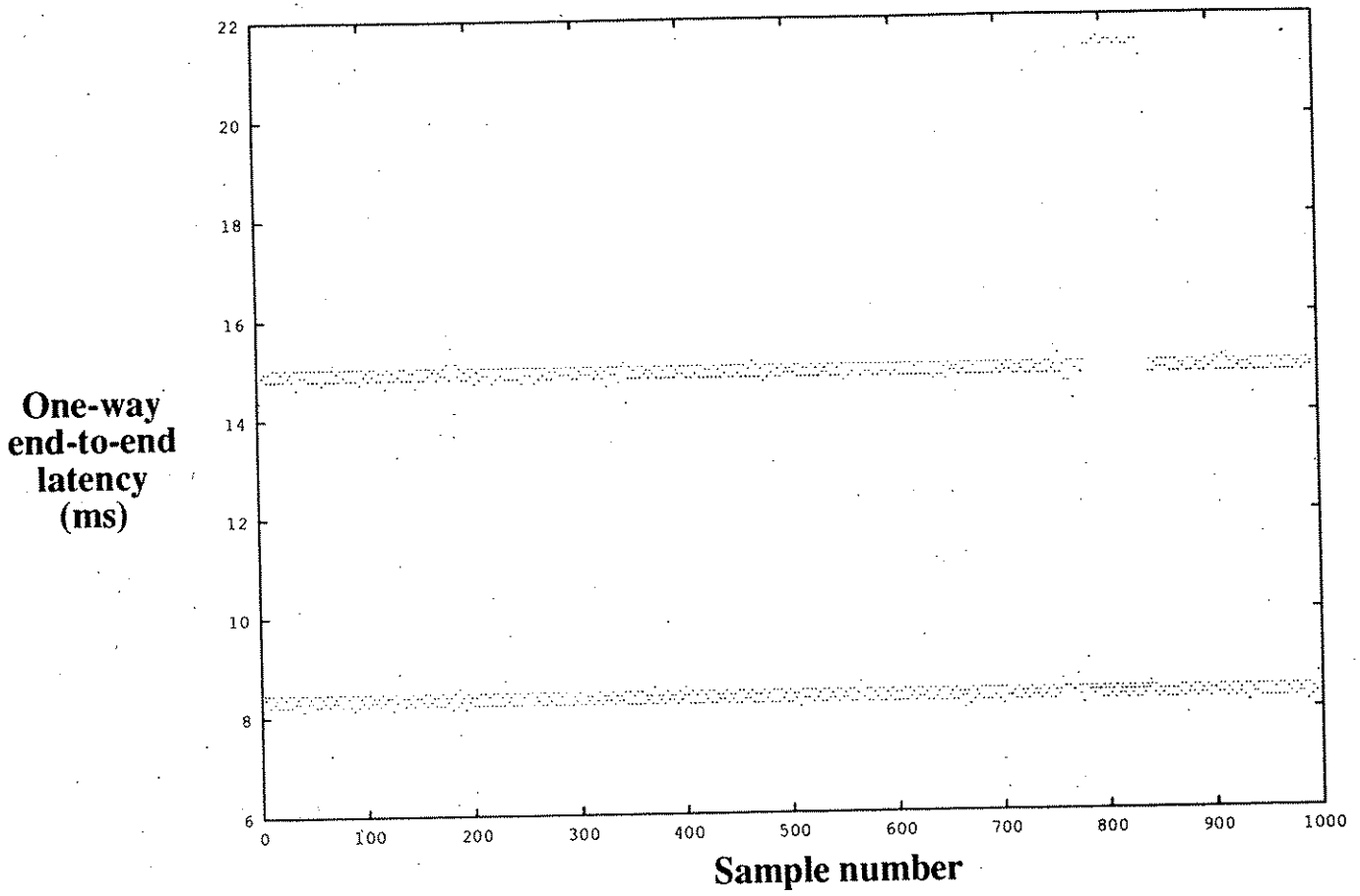
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 10.804 ms
99.9% threshold: 19.512 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 2048 bytes



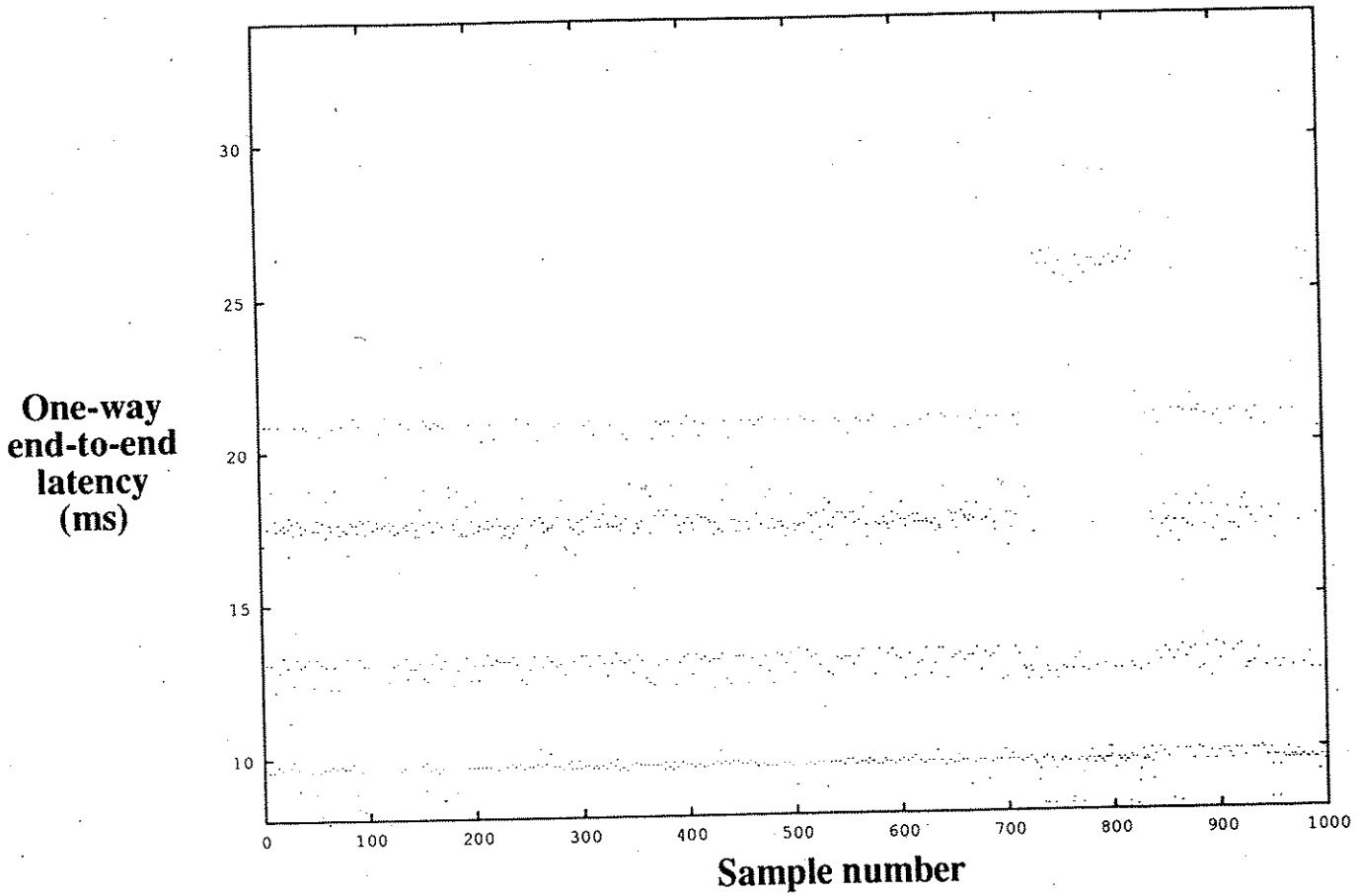
1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 11.608 ms
99.9% threshold: 21.463 ms

MULTICAST

JITTER MEASUREMENT

Voice Data Size: 4096 bytes



1000 samples
No asynchronous processor load
75 Mbits/sec background synchronous FDDI load (15 packets/token)
Multicast with 2 receivers
Voice data in FDDI synchronous class

Average latency: 14.575 ms
99.9% threshold: 33.984 ms

MULTICAST

Appendix H

FiberTap FDDI Network Monitor

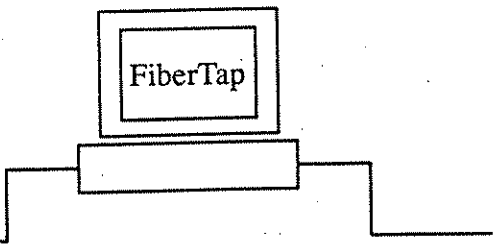
FiberTap

User's Guide

Version 1.1

Written By

James McNabb
Randall Atkinson



Introduction

FiberTap is a general purpose FDDI network diagnostic tool for system designers and debuggers who want real-time traces of network traffic, and for system maintainers who want a simple menu-driven user-interface, station monitoring capabilities, and impressive real-time graphic screens. FiberTap is even beneficial to the end-user who simply wants to visualize network traffic or quickly check the status of the network or of a remote node on the network.

The FiberTap Package

Your FiberTap package consists of one MS-DOS formatted distribution disk and this manual which is the *Fibertap User's Guide*. The distribution disk contains the program and all the files necessary to run FiberTap. The following are the names and a brief description of the files contained on the disk:

FIBERTAP.EXE	- the executable program.
MENU.FIL	- data for FiberTap's user-interface.
FILTER.FIL	- previously defined filters.
NAME.FIL	- previously defined ASCII names for addresses.
LITT.CHR	- Turbo C's little letter font.

Important: Make backups of this disk immediately! You should never tamper with the MENU.FIL or LITT.CHR files. They are critical to the operation of FiberTap and should never be manipulated directly. FILTER.FIL and NAME.FIL can be rebuilt by FiberTap but their deletion implies that all previously defined filters and names are lost.

Requirements

FiberTap runs on the IBM personal computer ATs and all 100% IBM PC/AT compatibles equipped with an EGA or VGA card and monitor. FiberTap requires DOS 2.0 or higher and should be run with 640K of RAM. An AMD FASTcard network interface board is also needed.

While not a requirement, a vast improvement in performance can be gained through the use of 80386-based machines running at 16MHz or faster using enhanced video BIOS. FiberTap uses screen I/O extensively and this class of PC provides improved video performance as well as faster CPU computation and bus bandwidth.

The FiberTap User's Guide

The *FiberTap User's Guide* introduces you to FiberTap, shows you how to manipulate the user interface, and describes the two modes of operation (Trace and Graphic) and their related features. The following is a breakdown of the sections in the *User's Guide:*

Getting Started: shows how to install and execute FiberTap.

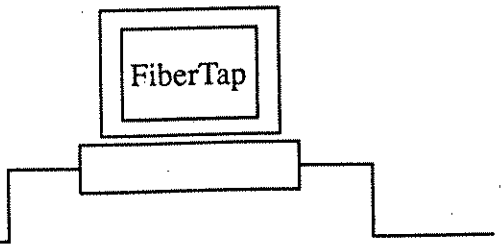
The User Interface: describes each type of menu screen and how to manipulate each individual menu selection within the user interface.

Trace Mode: introduces FiberTap's Trace mode and provides descriptions of its display and its commands. A detailed look at all of the Trace Mode features, such as diagnostic alarms, triggers, and the name filter is given.

Graphic Mode: introduces FiberTap's Graphic mode and describes its display graphs and statistics.

Station Mgmt Mode: introduces FiberTap's Station Management Mode and describes its display and use.

FiberTap Filters: provides a general discussion of FiberTap's filters. It shows how to express them and demonstrates their use.



Getting Started

In this section information relating to installing FiberTap on your system and starting up the program is presented.

Installation

At this point you should make backups of the distribution disk. FiberTap, during the course of normal execution, modifies the files contained on the distribution disk. To avoid accidental loss of a file and to preserve the initial FiberTap configuration, you must make backups.

Now that your backups are done, place one of them in drive **A**. Copy all the files on drive **A** to a subdirectory named FIBERTAP on your hard drive (see below). If you do not have a hard drive then you will execute FiberTap from drive **A**.

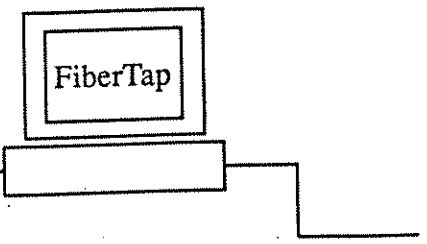
```
A> copy  *.*   c:\FIBERTAP
```

Executing FiberTap

Make sure you are in the directory that contains the FiberTap files before you begin. To execute FiberTap type **FIBERTAP** and press **Enter**. The screen should reveal FiberTap's user interface. You are now ready to investigate the user interface described in the next section.

```
A>  c:
C>  cd  FIBERTAP
C>  FIBERTAP
```

FiberTap's user interface is designed for simple, rapid navigation of its menus. All options of FiberTap are easily manipulated and the menu commands are consistent from menu to menu. This powerful user interface is a result of using pull-down menus. In this section, only the manipulation of FiberTap's pull-down menus are described; later sections discuss the meaning and significance of the various parameters and selections.



The User Interface

Pull-Down Menus

Pull-down menus are menus which overlap each other as you traverse down into lower levels. The menu overlapping all the others is your current active menu. Typically the main menu is a horizontal menu bar at the top of the computer screen. An example of pull-down menus is below:

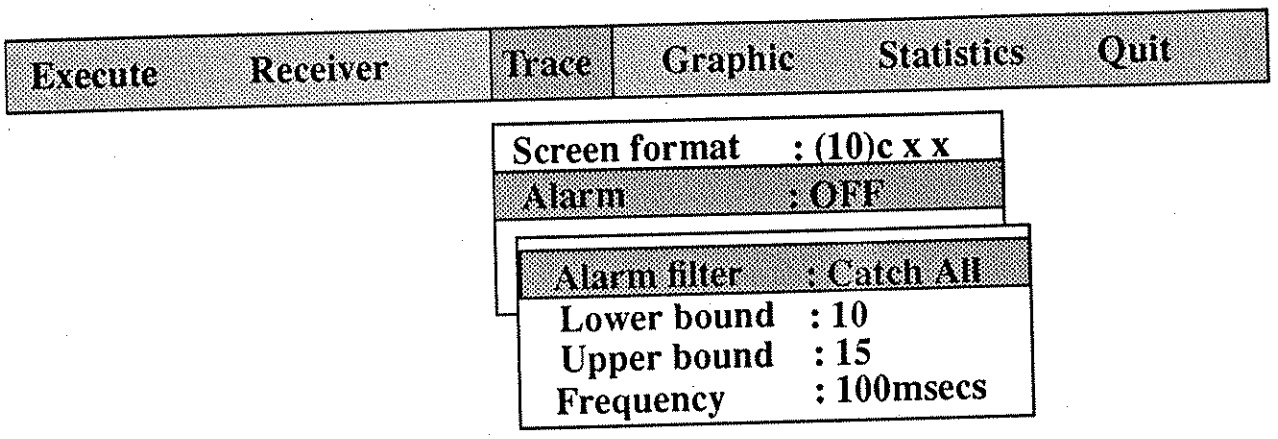


Figure 1: **Pull-Down Menus**

In Figure 1, the shaded bar at the top represents a horizontal menu bar. This is the main menu from which the other menus will pull-down, as shown by the other two menus. The first window is a result of selecting the third option, Trace, on the main menu. The second menu overlaps the first and is a result of selecting the second option, Alarm, on the first menu. These represent actual menus of FiberTap and will be discussed later.

Menu Navigation

Menu navigation is achieved through single key strokes. The following keys are the basic set used to navigate the menus:

Arrow Keys	move to desired selection within a pull-down menu or horizontal menu bar.
First letter of selection	move to that selection and select it. For example, pressing the letter **A** while in the menu that has Screen format as an option (refer to Figure 1) would produce overlapping menu.
RETURN	select current highlighted menu item.
ESC	exits current menu and returns to previous menu. This is a null operation when the main menu is the active menu.

Normal Menus

Normal menus are those that contain selections, each with their first letter highlighted. They are the basic type of menu and are generally referred to by name. FiberTap's first level normal menus are the Execute Menu, the Receiver Menu, the Trace Menu, and the Graphic Menu.

Edit Menus

Edit menus allow the user to input a specified option. They are represented by a box around the previous value of the option with the cursor positioned at the end. Within an edit menu, special edit commands are available. These are listed below:

HOME	positions the cursor at the beginning of the current edit menu.
END	positions the cursor at the end of the current value of the edit menu.
DELETE	deletes one character underneath the cursor.
←	moves the cursor to the left one character.
→	moves the cursor to the right one character.
RETURN	replaces the previous option value with the new value within the edit menu. It also exits the edit menu.
ESC	cancels the current edit and exits the menu.

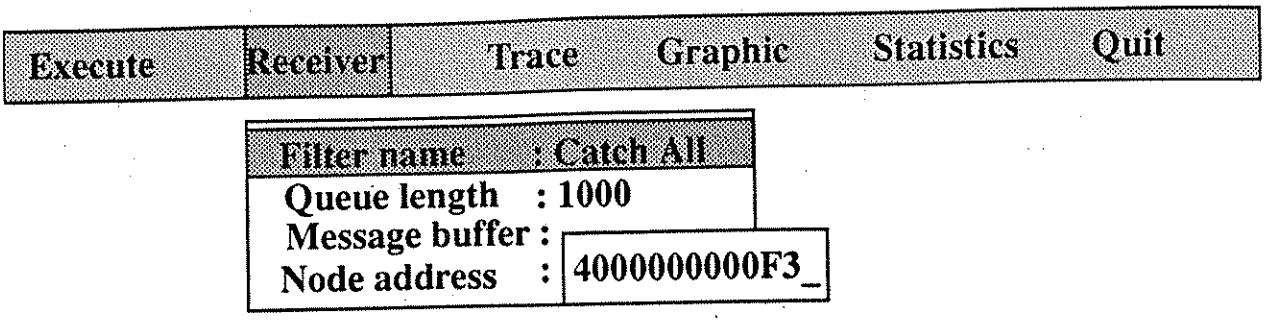


Figure 2: **An Edit Menu**

Figure 2 represents an edit menu. In this particular case, the option requested is a network address and its previous value was 4000000000F3. The edit menu was activated by selecting Node Address.

Display Menus

A display menu is used to present the user with information. There are no selections in a display menu. To exit the menu, press any key.

Toggle Menus

A toggle menu is used to toggle between different values of a selection. They are special in that they do not create an overlapping menu; they simply change the value of the current selection. FiberTap uses toggle menus for those selections that have only a few valid values, such as ON or OFF. To toggle the values for a given selection use the left and right arrow keys. If there is a range of valid values then the right arrow will progress toward greater values and the left arrow will progress toward smaller values. Both the right and left arrow will wrap once an extreme is reached.

The FiberTap Menus

Now that the various types of menus have been discussed, each FiberTap menu and its selections will be described. Remember that only manipulation of the user interface is being discussed in this section and a full description of FiberTap's modes and parameters are presented in later sections.

Execute Menu

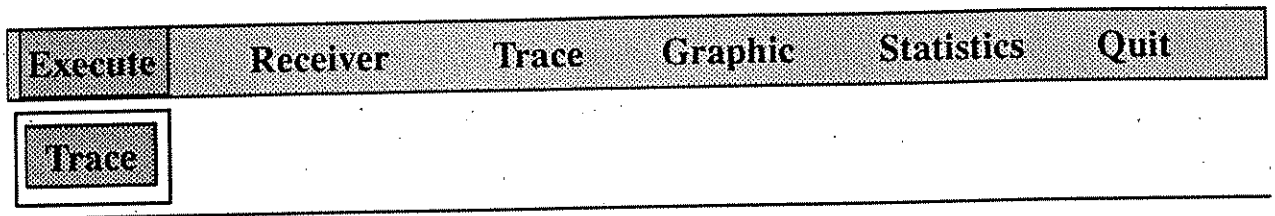


Figure 3: **Execute Menu**

The **Execute** Menu is used to begin either a Trace, Graphic, or Station Management diagnostic session This is a normal menu, but one with only a single selection. The selection is a toggle for which mode you wish to enter, trace, graphic, or station management. Once you have toggled to the desired value, pressing **RETURN** will enter the selected mode. Figure 3 represents the Execute Menu with the trace mode selected. Pressing **RETURN** at this point will start Fiber-Tap's trace mode.

Receiver Menu

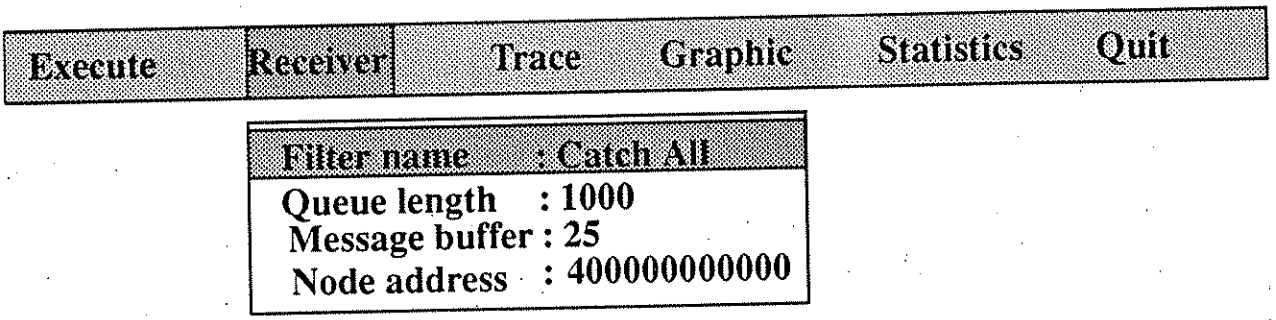


Figure 4: **Receiver Menu**

The Receiver Menu is used to customize the FiberTap receiver. It is a normal menu with selections enabling you to install a specific packet filter, change the queue or message buffer length, or assign FiberTap a different network address. If the combined values of Queue Length and Message Buffer require more memory than is available to Fibertap, you will be told when you try to execute and must then adjust the values to fit within available memory before you may begin execution.

Filter Menu

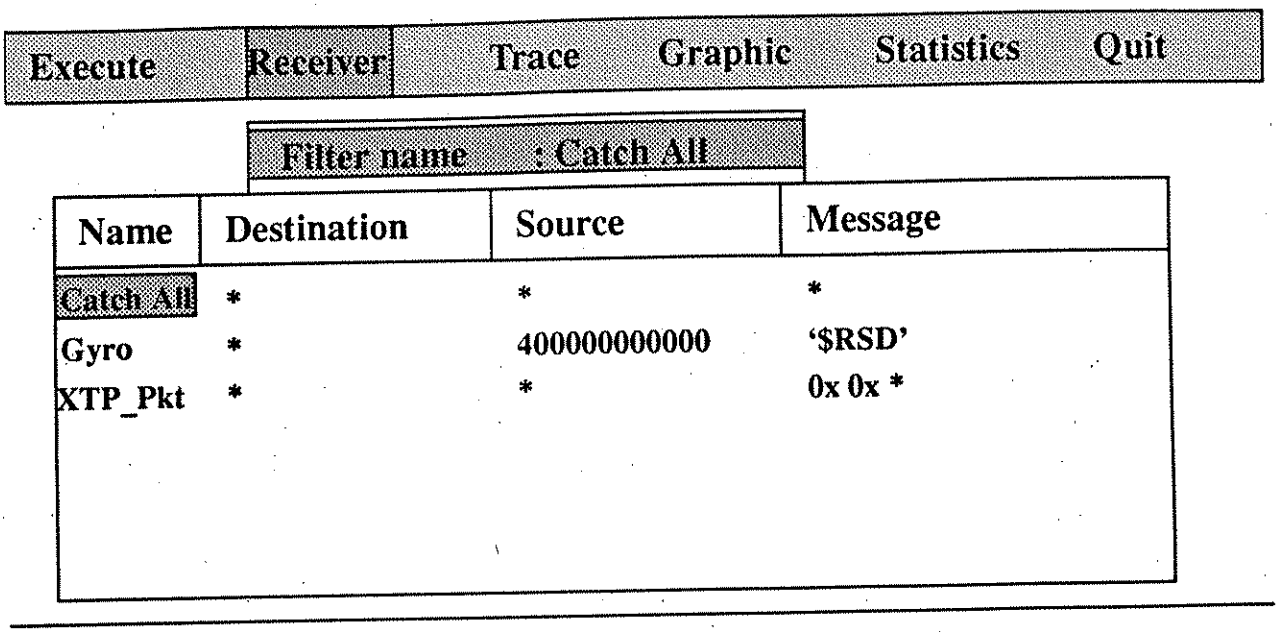


Figure 5: **Filter Menu**

Filter name

Specifies the packet filter the receiver will use during the next diagnostic session. If Filter name is selected the Filter Menu pulls down.

The Filter Menu is a library of previously defined filters. The name of each filter (listed alphabetically), destination address, source address and message field of the filter are displayed. The active filter will be at the top of the menu (highlighted) when it first opens. FiberTap's filter library can hold 100 different filters. With a large number of filters this menu will act as a window into this library-- the filters will scroll off the bottom and top of the menu. Figure 5 represents the Filter Menu with the CATCH ALL packet filter as the previous active filter.

From the Filter Menu you can create new filters, modify existing filters, delete unwanted filters, or select a new active filter. The keystrokes that achieve these commands follow:

RETURN pulls down the Insert Filter Menu filled in by the highlighted filter's attributes.

INSERT same as **RETURN**.

DELETE deletes the highlighted filter from the library.

ESC selects the highlighted filter to be the active filter.

Insert Filter Menu

The Insert Filter Menu is used to insert new filters or to modify an old filter. The highlighted filter's attributes (name, destination, source, message) from the Filter Menu are copied into the Insert Filter Menu's selections

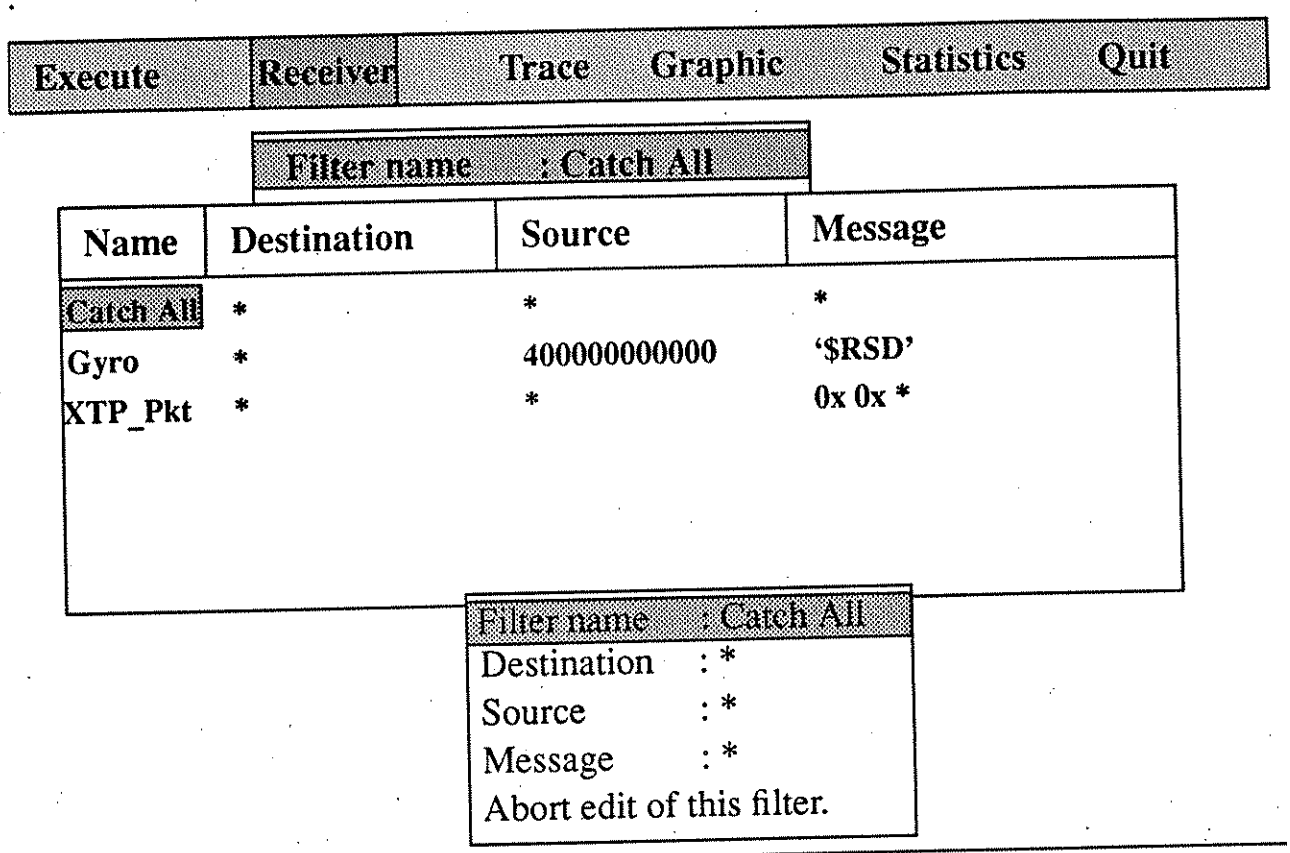


Figure 6: **Insert Filter Menu**

Filter Name: specifics the name of the filter. If you change this field then you are creating a new filter. If you only change the other fields and leave this one untouched, then you have modified this filter. Changes are accomplished via an edit menu.

Destination: specifies the destination address you wish to filter. It should be expressed as a twelve character representation of a hexadecimal number. For example, **4000A409C0FF** is a valid address. FiberTap also accepts the wildcard character * to mean all addresses and pre-defined ASCII names that correspond to network addresses. Changes are accomplished via an edit menu

Source: same as Destination except it applies to the source address.

Message: specifies the byte sequence within the information field to filter. FiberTap does a byte-by-byte comparison of these bytes to those of each packet's information field. The byte sequence is entered via an edit menu.

Abort: cancels the insertion or modification of the filter.

Important: In order to properly understand and use FiberTap's filters the **FiberTap Filters** section and the **Trace Name Filters** section must be read!

Queue Length

This represents the maximum number of packets that FiberTap will buffer in memory. FiberTap only buffers packets when its displays can not keep up with the network load. During bursts of traffic, FiberTap may have to buffer some of the incoming packets; when the network load subsides the queue empties and the displays catch up. Queue length is a toggle. The values range from 10 to 2000. A suggested starting value is 1000.

Message Buffer

The data field of a packet can range from 1 to 2K bytes. The Message buffer represents the maximum number of bytes that FiberTap should copy. For example, if the information you need from the packet data field does not exceed beyond the twentieth byte then the Message buffer can be set to 20. Because of the large range of values, the Message buffer is an edit menu.

Message buffer size is a very influential parameter and can cause unexpected results if not used properly. It affects the active filter, the memory requirement for the receive queue, and the Screen format (discussed later). For example, if you have set the Message buffer to 10 and have activated a filter that expects to filter on the 20th byte of the information field then FiberTap will not be able to filter correctly and the filtering of bytes 11 thru 20 will be ignored. Or if you set the Message buffer to 10 and you specify a Screen format that expects to display bytes past the 10th byte of the information field than those bytes will not be displayed. Also, the larger the Message buffer the more memory the receiver queue requires.

Node Address

Every node on an FDDI Ring has a unique network address. The Node address selection represents FiberTap's six digit hexadecimal network address. Changing the address is accomplished through the use of an edit menu. Note: If you specify a duplicate address (one which already exists on the network) FiberTap will not operate correctly.

Alarm Menu

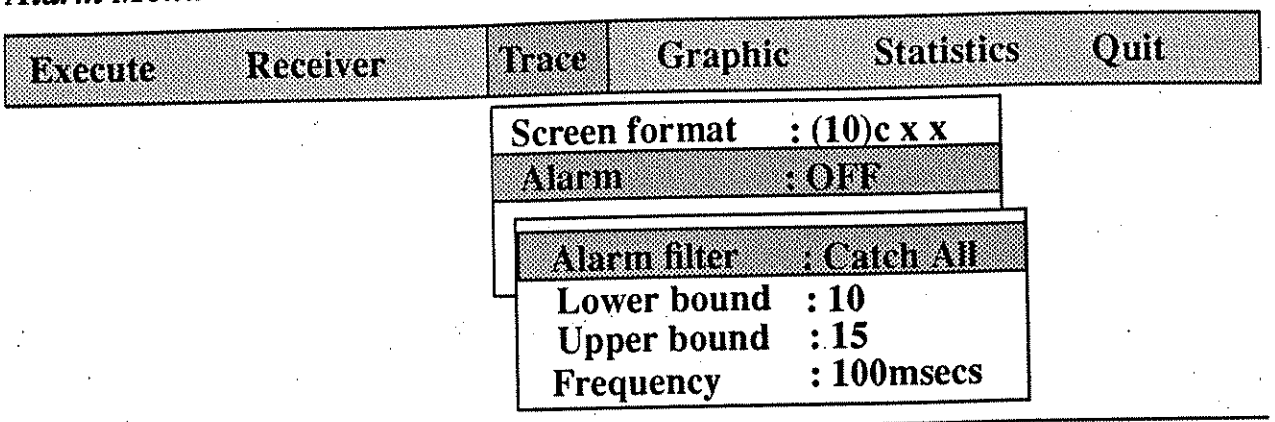


Figure 8: **Alarm Menu**

The Alarm Menu is used to select the specific packet (defined by a filter) you wish FiberTap to use as its alarm packet. The parameters for frequency alarms are also changed from here.

Alarm: specifies the name of the alarm filter. If Alarm filter is selected then the Filter Menu (discussed previously) pulls down. From the Filter Menu you can select a new alarm filter.

Lower bound: specifies the minimum number of alarm packets that FiberTap expects to arrive over the interval defined by Frequency. Changes are accomplished via an edit menu.

Upper bound: specifies the maximum number of alarm packets that FiberTap expects to arrive over the interval defined by Frequency. Changes are accomplished via an edit menu.

Frequency: specifies the interval of time that the Upper and Lower bounds apply to the arrival of alarm packets. If the number of alarm packets that arrive over this interval exceeds either bound then FiberTap triggers an alarm. Frequency values range from 50 msecs to 10 secs. To change these values use the toggle keys (left and right arrows).

Trace Menu

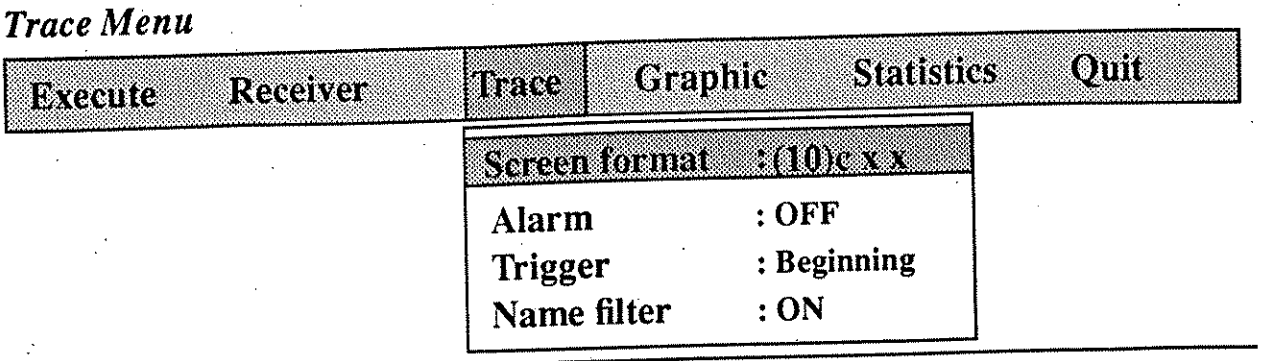


Figure 7: **Trace Menu**

The Trace Menu is used to customize FiberTap's trace mode. From this menu you can change the Screen format, activate diagnostic Alarms, set the Trigger position within the receive queue, and turn **ON** or **OFF** the Name filter.

Screen Format

This selection specifies the format in which each packet's data field is to be displayed on the Trace Display. Changing the format is accomplished through an edit menu.

Alarm

The Alarm selection allows you to activate diagnostic alarms within the trace mode. There are three possible settings - **OFF**, **Event**, and **Frequency**. To change among these values use the toggle keys ← and → (left and right arrows). Pressing **RETURN** while the Alarm selection is highlighted will pull down the alarm menu.

Trigger

The Trigger selection specifies the position of the trigger in the receiver queue. The trigger position represents the packet boundary between historical packets, those that arrived before the trigger, and future packets, those that will arrive. There are four possible settings- **OFF, Beginning, Middle,** and **End**. To change among these values use the toggle keys ← and → (left and right arrows).

Name Filter

The Name Filter selection is used to toggle FiberTap's name filter **ON** or **OFF**. With this option set to **ON**, The trace Mode will match previously defined names to hexadecimal addresses. This greatly enhances the readability of the trace mode. For example, suppose a file server on the network has the address 4AFDE323610A. It is easier to identify the ASCII name "File Server" than to decipher that address.

Graphic Menu

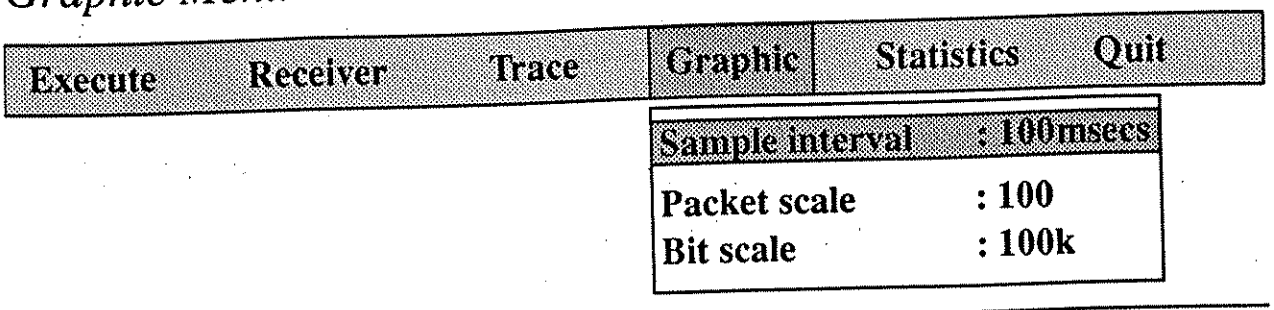


Figure 9: **Graphic Menu**

The Graphic Menu is used to customize FiberTap's graphic mode. From this menu you can adjust the sampling interval, the Packets/interval scroll graph scale, or the Bits/interval scroll graph scale.

Sample Interval

The Sample interval is the duration over which the graphic mode statistics and graphs are calculated. This selection is a toggle menu with values ranging from 50 msecs to 10sec.

Packet Scale

The Packet scale adjusts the Packet/interval scroll graph scale. This selection is a toggle menu with values ranging from 1 to 1G (1 billion) packets/interval.

Bit Scale

The Bit scale adjusts the Bits/interval scroll graph scale. This selection is a toggle menu with values ranging from 1 to 1G (1 billion) bits/interval

Statistics Display

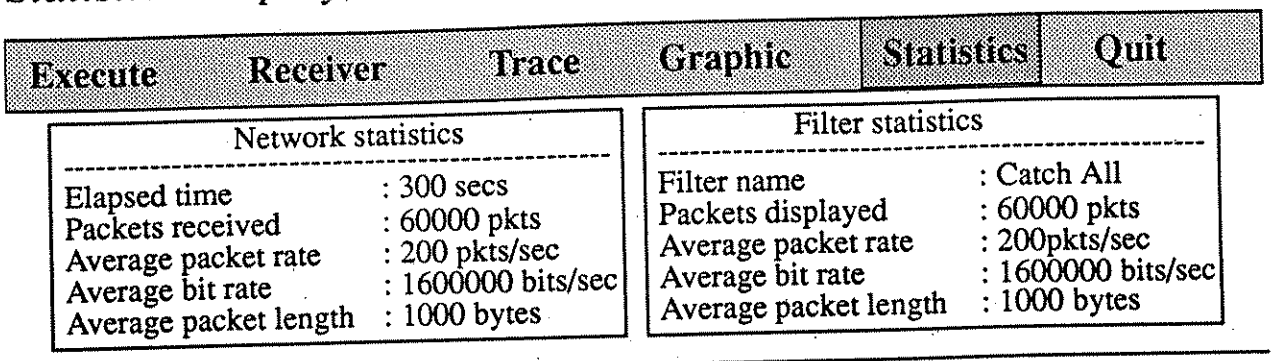


Figure 10: **Statistics Display**

The Statistics Display is a display menu which presents overall network statistics and statistics for the active filter. These statistics represent the most recent diagnostic session- trace mode or graphic mode.

Elapsed Time: duration of this diagnostic session in seconds.

Packets received: number of packets received during the elapsed time.

Packets displayed: number of packets processed and displayed during the elapsed time.

Average packet rate average packets/seconds during the elapsed time.

Average bit rate: average bits/second during the elapsed time.

Average packet length: average length of each packets data during this session.

Quit Command

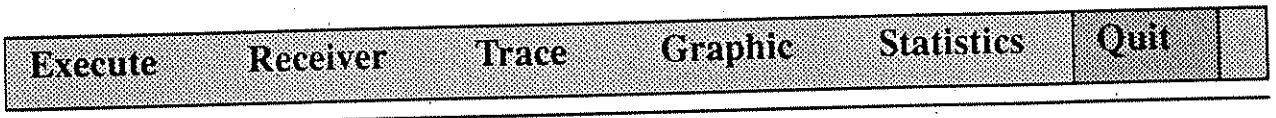
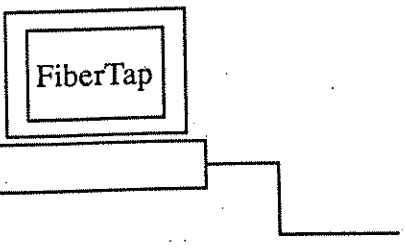


Figure 11: **Quit Command**

The Quit Command exits FiberTap and returns to MS-DOS.



Trace Mode

FiberTap's trace mode is designed for low-level diagnostics of the network traffic. With this mode you can observe all or any part of the contents of every packet that traverses the network, you can monitor the frequency of all or specific packets using diagnostic alarms, and you can set a trigger for an event and then browse through a trace of the network traffic that preceded and succeeded the event.

This section provides descriptions of FiberTap's Trace Display and Trace Commands. Also provided is a detailed look at the trace mode features, such as the screen format, diagnostic alarms, triggers, and the name filter

Trace Display

Elapsed:	25 secs
Packets received:	5
Packets displayed:	5
Screen format:	x x x
Trigger:	Beginning

Alarm:	OFF
Filter name:	Catch All
Destination:	*
Source:	*
Message:	*

Time	Destination	Source	SAP	Len	Message
[10.2]	[400000000000]	[4000AF000001]	[02:02]	[323]	00 B1 FF
[10.3]	[4000AF000001]	[400000000000]	[02:02]	[105]	FF 32 5F
[10.7]	[400000000000]	[4000AF000001]	[02:02]	[1024]	65 7C 42
[10.8]	[400000000000]	[4000AF000001]	[02:02]	[1024]	65 7D 20
[12.1]	[4000AF000001]	[400000000000]	[02:02]	[43]	7E 41 00

Figure 12: **Trace Display**

The Trace Display is divided into three windows: the *mode information window*, the *filter and alarm window*, and the *packet information log*.

Mode Information window

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

Elapsed time: time spent in the trace mode since last reset operation.

Packets received: number of packets that have traversed the network since the last monitor reset operation.

Packets displayed: number of packet that have been displayed in the *packet information log*.

Screen format: the current display format for the *packet information log*.

Trigger: the current setting for the trace trigger.

Filter and Alarm window

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

Alarm: current status of the alarm option.

Filter name: user-defined name of current active receive filter.

Destination: specification for the filtering of the destination.

Source: specification for the filtering of the source.

Message: specification for the filtering of the packet data field.

Packet Information window

The *packet information log*, located on the bottom two-thirds of the Trace Display, shows the actual contents of each packet that matches the filter described in the *filter and alarm window*. The time the packet was received, the destination and source node address of the packet, 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 screen format specification.

Trace Commands

Here are the special keystroke commands for the trace mode:

F1 or **R**: resets the trace mode. The statistics and elapsed time are set to zero.

F2 or **RETURN**: clears any status messages produced by FiberTap. Normally used when the receive queue overflows or an alarm occurs.

F3 or **SPACEBAR**: pauses the display. During bursts of traffic the *packet information log* scrolls traffic through the log very quickly. By pausing the screen you can observe the traffic within the log without it scrolling off the top.

F4 or **H**: transmits a test packet. This command will actually send a 100 bytes test message onto the network. This command is used to confirm that a network is operational.

F5 or **T**: causes a manual trigger. This command will bring up the Trace Trigger Display.

ESC: exits trace mode and returns to the user interface.

Screen Format

The screen format for the *packet information log* specifies the way in which each packet's data field is displayed. This feature allows you to tailor the trace display to show precisely what is wanted. The specification consists of a lists of byte types, each separated by a space. A byte type represents the display format for a particular byte. Byte types may also be preceded by a repetition count to specify that a series of bytes or of the same format. Below are the valid byte types:

c specifies the byte is to be displayed as an ASCII character.

d specifies the byte is to be displayed as a decimal number.

x specifies the byte is to be displayed as a hexadecimal number.

e specifies the byte is to be excluded (do not display it).

(n) specifies the repetition count for a byte type where **n** is the count.

> specifies *the rest*. This is used to apply a byte type to all the bytes from the current position to the end of the packet.

Format examples:

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

Diagnostic Alarms

FiberTap's alarms are used to notify you of a particular event that has occurred on the network. FiberTap supports both Event Alarms and Frequency Alarms, making it very flexible.

Event Alarm

An Event alarm notifies the user of the first and every subsequent occurrence of a packet that matches the alarm filter. This can be used to signal that a specific event or action occurred. The notification of the alarm occurs in the *filter and alarm window* with the time the alarm packet arrived and its destination and source address flashing. With each occurrence of the alarm packet this window is updated. To clear the most recent alarm notification press **F2** or **RETURN.**

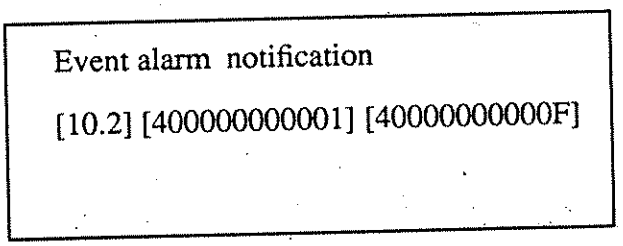


Figure 13: **Event Alarm Notification**

Figure 13 shows the *filter and alarm window* with an event alarm notification.

Frequency Alarm

The Frequency alarm notifies the user that the frequency of an alarm packet (defined by the alarm filter) is now outside a previously defined range. When specifying the alarm filter for a Frequency alarm, you must also specify the maximum and minimum frequency of packet arrivals for that particular filter. If FiberTap detects alarm packets that occur less often than the minimum frequency or more often than the maximum frequency an alarm is reported. Notification of a Frequency alarm includes the time the alarm packet arrived, its destination and source address, and the number of alarm packets that caused the alarm. This notification occurs in the *filter and alarm window*. To clear the notification press **F2** or **RETURN**.

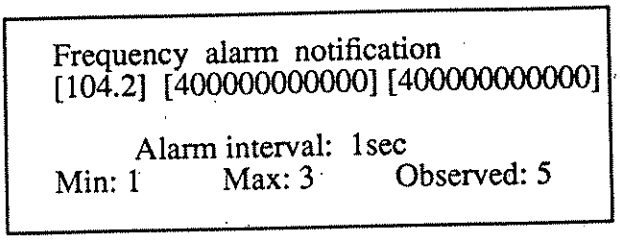


Figure 14: **Frequency Alarm Notification**

Figure 14 shows a Frequency alarm notification. The alarm interval for the alarm filter was 1 second with frequency bounds of 1 and 3; therefore alarm packets (those that match the alarm filter) can arrive with a frequency between 1 and 3 packets a second without causing an alarm. In the above notification, 5 packets were noticed over one of the 1 second intervals containing the displayed packet, thus causing the notification.

Triggers

The Trace Trigger is a sub-mode of the trace mode. It is used to observe packets that preceded and succeeded a particular event. This sub-mode is also used to store part or all of the receive queue to a disk file. There are two types of triggers: Manual and Alarm.

Trigger Position

The amount of history (preceding packets) is controlled by the trigger position. There are four settings for the trigger position- **Beginning, Middle, End** and **OFF**. **Beginning** represent a history of 10% of the receive queue, **Middle** a history of 50% of the receive queue, and **End** a history of 90% of the receive queue, while **OFF** disables the trace trigger. For example, if the receive queue length has been set to 1000 packets, then a trigger setting of **Beginning** implies that 100 packets and **End** implies that 900 packets will be kept as history.

Manual Trigger

The Manual trigger (as its name suggests) is activated by a keystroke from within the trace mode. Provided that the trigger position is not set to **OFF**, pressing **F5** or **T** will activate the manual trigger and bring up the Trigger Display. The trigger packet will be the most recent packet received by FiberTap.

Alarm Trigger

The Alarm trigger is activated when both the trigger position is not set to **OFF** and an alarm occurs. This is used to trigger on particular events described by alarms and provides a way to observe packet that arrived before and after the alarm. If the trigger position is set and an alarm notification occurs, FiberTap awaits any keystroke, then brings up the Trigger Display. The trigger packet will be the packet causing the alarm.

Trigger Display

The Trigger Display is a scrolling window into the receiver queue. For each packet in the receiver queue the queue position, time of arrival, destination and source address, packet length, and packet data field is displayed. The packet's data field is displayed in accordance with the screen format. A highlighted packet at the trigger position represents the boundry between packets that arrived before and after the trigger occurred. Figure 15 shows the Trigger Display with the trigger position at the 100th packet and the screen format set at **(4)x.** The first 99 packets (not shown) arrived before the trigger occurred, and packets 101 through 108 arrived after the trigger.

Queue	Time	Destination	Source	Length	Message
0100:	[10.3]	[400000000001]	[40000000000F]	[103]	00 54 42 DF
0101:	[10.4]	[4000000001AF]	[40000000000FF]	[661]	12 1F 22 A1
0103:	[14.6]	[400000000001]	[40000000000F]	[103]	00 54 42 DF
0104:	[16.7]	[40000000000F]	[400000000001]	[166]	92 AF 92 01
0105:	[17.5]	[4000000000FF]	[4000000001AF]	[1024]	12 FF 46 E3
0106:	[17.5]	[400000000001]	[40000000000F]	[103]	00 54 42 DF
0107:	[18.9]	[4000000001AF]	[4000000000FF]	[44]	00 45 FF 00
0108:	[19.0]	[400000000001]	[40000000000F]	[103]	00 54 42 DF

Figure 15: **Trigger Display**

Trigger Commands

Here are the keystroke commands for the Trigger Display:

Arrow Keys	scroll through receive queue one packet at a time.
PgUp, PgOn	scroll through receive queue one page at a time.
SPACEBAR	brings up File Menu.
INSERT	activates the Name Definition Display.
ESC	exits the Trigger Display and returns to the trace mode.

File Menu

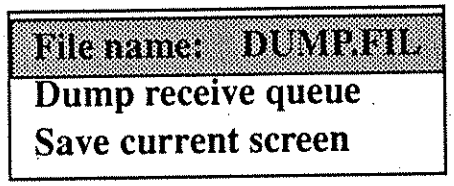


Figure 16: **File Menu**

The File Menu is used to store part of all of the receive queue to a file. From this menu you can change the name of the storage file, dump the screen to the current file (defined by the file name), or dump the receive entire queue to the current file. Pressing **ESC** exits menu and returns to Trigger Display.

File name:	edit menu selection used to change the name of the current file.,
Dump receive queue:	stores all packets contained in the receive queue to the current file.
Save current screen:	stores all packets displayed on the screen to the current file.

Trace Name Filter

The Trace Name Filter is a sub-mode of the trigger display. It is used to establish ASCII character names for network addresses. This feature allows you to customize the trace and Trigger Displays to show your own predefined names. For example, if your network has a file server, you can map its network address to a name such as "Server." If the name filter is active, then whenever FiberTap encounters that address it will replace it with the name "Server."

Name Definition Display

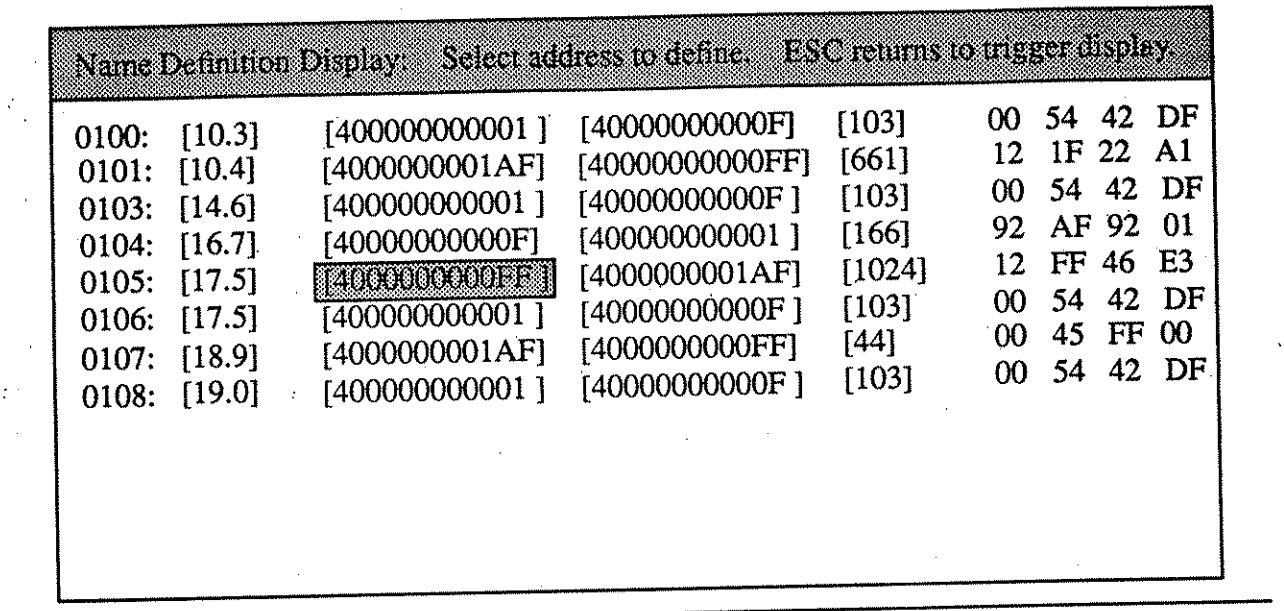
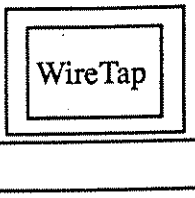


Figure 17: **Name Definition Display**

The assignment of user defined names to addresses is accomplished via the Name Definition Display. It is only accessible from the Trigger Display by pressing **INSERT**. This display resembles the Trigger Display, except that one of the addresses is highlighted. From this display you can move the highlighted bar to any address on the screen, select it, then enter an ASCII name that you wish associated with that particular address.

Commands:

Arrow Keys	move highlighted cursor to another address.
RETURN	select address to define. Brings up an edit menu.
ESC	exit Name Definition Display and return to the Trigger Display



Graphic Mode

FiberTap's graphic mode provides a visual representation of network traffic. It displays visual statistics about network utilization through the use of two scroll graphs and a histogram. The scroll graphs display graphical representations of the network load and the histogram shows the distribution of the packet sizes traversing the network.

In this section the scroll graphs and the histogram will be described. Also described will be the graphic modes's sample intervals and how they affect the scroll graphs.

Graphic Display

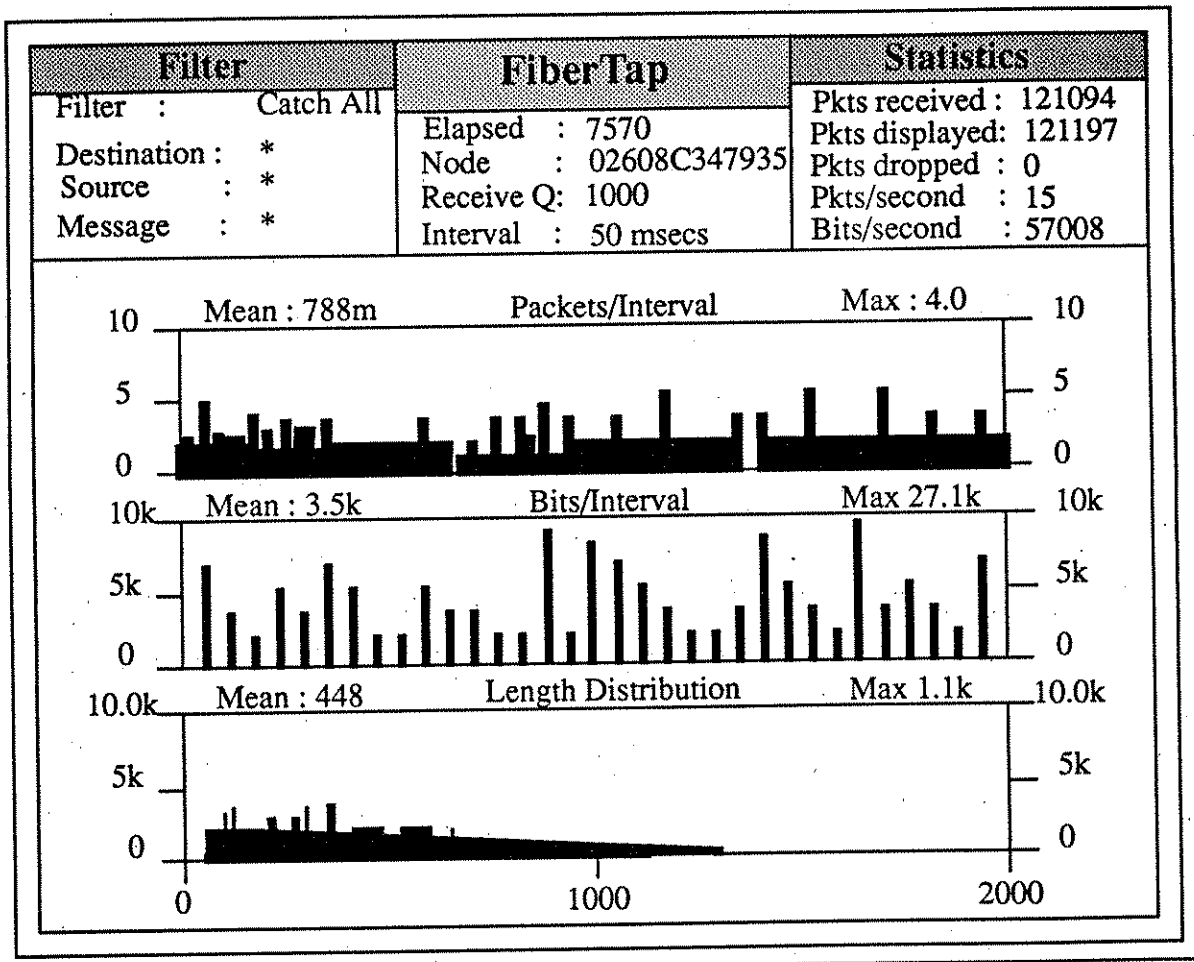


Figure 18: **Graphic Display**

The Graphic Display is divided into four areas: the *mode information window*, a *packet length histogram*, and two historical *scroll graphs*. These areas are shown in Figure 18.

Mode Information Window

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

Elapsed time:	time spent in the graphic mode since the last reset operation.
Interval:	sample interval for the scroll graphs and the statistics.
Filter:	user-defined name of current active receive filter.
Destination:	specification for the filtering of the destination address.
Source:	specification for the filtering of the source address.

Packet Length Histogram

The *packet length histogram* is located in the lower third of the Graphic Display and reports the distribution of packet sizes that have traversed the network since the monitor's last reset operation. The vertical scale represents the number of packets; the horizontal scale shows packet size. The vertical scale is automatically rescaled when a value breaches the top of the graph. The current maximum value of the top of the histogram is displayed next to the **Scale** label.

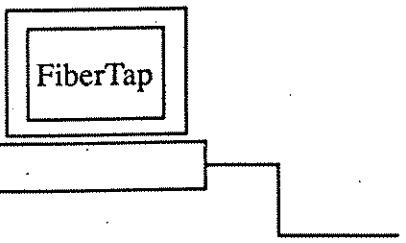
Scroll Graphs

The two *scroll graphs*, one reporting packets/interval and the other reporting bits/interval, are located in the upper two-thirds of the Graphic Display. These graphs update once every sample interval; 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 scroll graph reflects the network's most recent performance while the values to the left provide a summary of recent history. The scale of the scroll graphs, along with the current mean and maximum values recorded since the last monitor reset operation, are displayed at the bottom of each *scroll graph*.

Sample Interval

The sample interval is a very influential parameter of the graphic mode. It affects both scroll graphs and their respective statistics. The sample interval represents the duration of time FiberTap collects statistics for a specific data point of the scroll graphs. Thus the sample interval determines both the rate at which the scroll graph will be updated and the amount of history that the scroll graph represents. Also, the statistics at the bottom of each scroll graph are based on the sample interval.

For example, if the sample interval is set to 1 second then the scroll graphs represent 480 seconds of history. The scroll graphs are 480 pixels wide and each pixel width represents a sample interval. If the sample interval were set to 100 milliseconds and the **Mean:** under the packets/interval scroll graph displays 10.4, it is interpreted as 10.4 packets per 100 milliseconds or 104 packets per second. Likewise if the sample interval is set to 500 milliseconds and the **Max:** under the bits/interval scroll graph displays 449k, it is interpreted as 449k bits per 500 milliseconds of 898k bits per second.



Station Management

FiberTap's Station Management mode provides the user with the ability to observe the flow of packets through his station using the *Local FDDI Frame Counts* window and the ability to interrogate remote stations for their current configuration and frame counts using the *Remote FDDI Station Information* window.

In this section, both of these windows will be described. Also instructions will be given on how to use the Remote FDDI Station Information window to inquire of other stations on the network.

FiberTap

Ring Operational

Local FDDI Frames Received

Local Node Addr:	40000000011	Elapsed Time (sec):	14
Beacon MAC Frames:	0		
Claim MAC Frames:	1	Total MAC Frames:	1
		Total SMT Frames:	12
Synch LLC Frames:	6		
Asynch LLC Frames:	2	Total LLC Frames:	8
		Total Bad Frames:	1
		Total Frames:	22

Remote FDDI Station Management Information

Remote Node Addr:	422222222222	Remote MAC Index:	1
Remote Node Class:	Station	Total Remote MACs:	2
Downstream Addr:	422222222223	Sent Frames:	5
Upstream Addr:	422222222221	Received Frames:	5
T_Requested:	156	Total Bad Frames:	0
T_negotiated:	156	Total MAC Frames:	1

TAB to change field. **S to send SMT Request.**

Figure 19: **Station Management Display**

Station Management Display

The Station Management Display is divided into three areas: the *header*, the *Local FDDI Received Packets* window, and the *Remote FDDI Station Management* window. These are shown in Figure 19 above. Detailed information about the windows and fields on this screen follow.

Header

This is the top-most window on the *Station Management* screen. It has one line of message text which indicates the current ring state as perceived by the FDDI board present in the PC. If the ring is not operational, the message will indicate that and will also change color to red to alert you. FiberTap will automatically try to bring the ring back up whenever it goes down and the message text will change when it is successful.

Local FDDI Received Packets Window

This window describes the FDDI station on which the FiberTap software is running. It displays the local FDDI address, the elapsed time in seconds, and the counts of received FDDI frames. **Note:** in each case in this window, the number of frames received refers to the number of frames received during the *current* session of the Station Management mode of FiberTap.

Displayed Fields:

Local Node Addr:	The FDDI MAC address of the system running FiberTap
Elapsed Time (sec):	Number of seconds elapsed during the current session
Beacon MAC Frames:	Number of Beacon Medium-Access-Control frames received
Claim MAC Frames:	Number of Claim Medium-Access-Control frames received
Synch LLC Frames:	Number of Synchronous Logical Link Control frames received
Asynch LLC Frames:	Number of Asynchronous Logical Link Control frames received
Total MAC Frames:	Total number of Medium-Access-Control frames received
Total SMT Frames:	Total number of Station Management frames received
Total LLC Frames:	Total number of Logical Link Control frames received
Total Bad Frames:	Total number of invalid frames received
Total Frames:	Total number of frames received

Remote FDDI Station Management Information Window

This window describes the remote FDDI station that has been interrogated by the local FiberTap software. The user may indicate the *Remote Node Address* and *Remote MAC* to be interrogated. The **TAB** key is used to change the input field between the Remote Node Address and the Remote MAC Index fields. The S key is used to send a set of Station Management requests to the designated remote address and remote MAC.

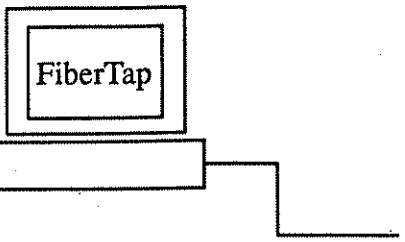
Note: All information in this window describes the remote node and remote MAC and its state at the time of the most recent SMT Request. Only the window above has information about the state of the local node on which FiberTap is executing.

Displayed Fields:

Remote Node Addr:	The FDDI MAC address of the remote node to interrogate
Remote MAC Index:	Which MAC at the Remote Node Address to interrogate (1 or 2)
Remote Node Class:	Whether the remote node is a Station or a Concentrator
Downstream Addr:	The remote node's downstream neighbor's address
Upstream Addr:	The remote node's upstream neighbor's address
T_requested:	The target token rotation time requested by the remote node
T_negotiated:	The target token rotation time being used by the FDDI ring
Total Remote MACs:	The number of MACs implemented (1 or 2) at the remote node
Sent Frames:	The number of frames sent by the remote node since it was last reset
Received Frames:	The number of frames received by the remote node since last reset
Total Bad Frames:	The number of invalid frames received since last reset
Total MAC Frames:	The number of MAC frames received since last reset.

Commands:

S	Send Station Management (SMT) requests to the specified address and MAC
TAB	Switch input field between the *Remote Node Addr* and *Remote MAC Index* fields.
ESC	Exit *Station Management* mode and return to main menu screen



FiberTap Filters

FiberTap is capable of filtering the network traffic so that it will only receive a particular type of packet. Filtering is accomplished by specifying a filter pattern consisting of the destination address specification, the source address specification, and the message field (packet's data field specification. Every packet received by FiberTap is compared byte-by-byte against the filter pattern. Packets that match the filter are recorded by FiberTap and those that so not are ignored.

The specification for the addresses can be one of three types- an actual address, and ASCII name previously defined through the name definition display, of the *"match all"* character. An actual address is the 6 digit hexadecimal value that represents the network address of interest. Using an ASCII name requires that the name has already been associated with an address through the name definition display. The *"match all"* character * suppresses the address comparison and allows all addresses to match. Below are example address specifications:

*	matches all addresses.
400010001234	matches only address 400010001234.
Broadcast	matches only the address previously associated with the name Broadcast.

The specification for the message field is represented by a list of byte descriptions. The byte descriptions specify the value that a particular byte of the packet must be in order for it to match the filter. A byte is described in either decimal (specified by a "**d**" after the value), hexadecimal (specified by a "**x**" after the value), or character (specified by ' ' surrounding the value). The *"match all"* character * suppresses the byte comparison and allows all values to match a particular byte. For repeated values, "(n)" specifies a repetition factor and the ">" symbol represents all remaining bytes of the message field. Below are example specifications:

100d 32d	match first byte of packet data is 100 (decimal) and the second byte is 5 (decimal).
0Ax FFx 'ABCD'	match if first byte is OA (hex), second byte is FF (hex), and the next four bytes are the ASCII characters "A", "B", "C", "D".
*** * * (10)123d**	ignore first three bytes of packet data and match if the next ten bytes are 123 (decimal).
*** 232d 1Ex>0d**	ignore the first byte and match if the first byte of packet data 232 (decimal), the second byte is 1E (hex), and all remaining bytes are zero (decimal)

The combination of the address specifications and the message specifications are used to create complete filters. Using the Insert Filter Menu in the user interface, you can build a library of filters, each consisting of a destination and source address specification, a message specification, and a name to identify it. Below are examples of complete filter specifications:

Filter name:	Catch All
Destination:	*
Source:	*
Message:	*

The Catch All filter will match all destination and source addresses and all message fields. This is the default filter for FiberTap and it will display all packets received.

Filter name:	Example1
Destination:	*
Source:	400000000001
Message:	'FIBERTAP'

The Example1 filter will match all packets from source address 400000000001 that have the first eight bytes of their data field equal to 'FIBERTAP'.

Filter name:	Example2
Destination:	File_Server
Source:	John
Message:	(10)* 'LOGIN'

The Example2 filter will match all packets to the address associated with the name File_Server from the address associated with the name John that have bytes 11-15 of their data field equal to 'LOGIN'.

Filter name:	Example3
Destination:	400011112222
Source:	400000003331
Message:	*

The Example3 filter will match all packets going from address 400000003331 to address 400011112222.

Appendix G

Experiment 7: Routers