Digitized Voice Distribution Using XTP and FDDI

Alfred C. Weaver
James F. McNabb

Computer Science Report No. TR-92-13
May 18, 1992

Digitized Voice Distribution Using XTP and FDDI

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

Phone: 804-982-2201
Fax: 804-982-2214
Internet: weaver@virginia.edu

ABSTRACT

The conventional way to distribute digitized voice over a computer network is to packetize some number of voice samples and transmit them as unacknowledged datagrams. If the number of voice samples per packet is small, then the loss of any one packet is unlikely to affect the perceived quality of the voice channel. The price paid for this simplicity is that, since this is a datalink protocol, the voice stream can not be routed over an internetwork. A resonable question to ask is whether a voice stream could in fact afford (in terms of latency) the higher quality service provided by transport and network layer protocols. If so, then the connection could be routed over internetworks and it could profit from the end-to-end reliability mechanisms inherent to transport protocols (e.g., transparent retransmission). And, depending upon the transport protocol, the voice stream might be able to utilize new services such as multicast. We experimented with this concept by building a multichannel voice distribution system using the Xpress Transfer Protocol running over an FDDI network. We made throughput, latency, and jitter measurements for a basic configuration and then for several variations: background synchronous traffic on FDDI of up to 75 Mbits/sec; packet loss rates of up to 10%; multicast distribution rather than unicast; and combinations of the above.

Digitized Voice Distribution Using XTP and FDDI

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

1. DIGITIZED VOICE

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
(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), voice samples can not be transmitted over a network 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 neces-

sary 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 more information from 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.

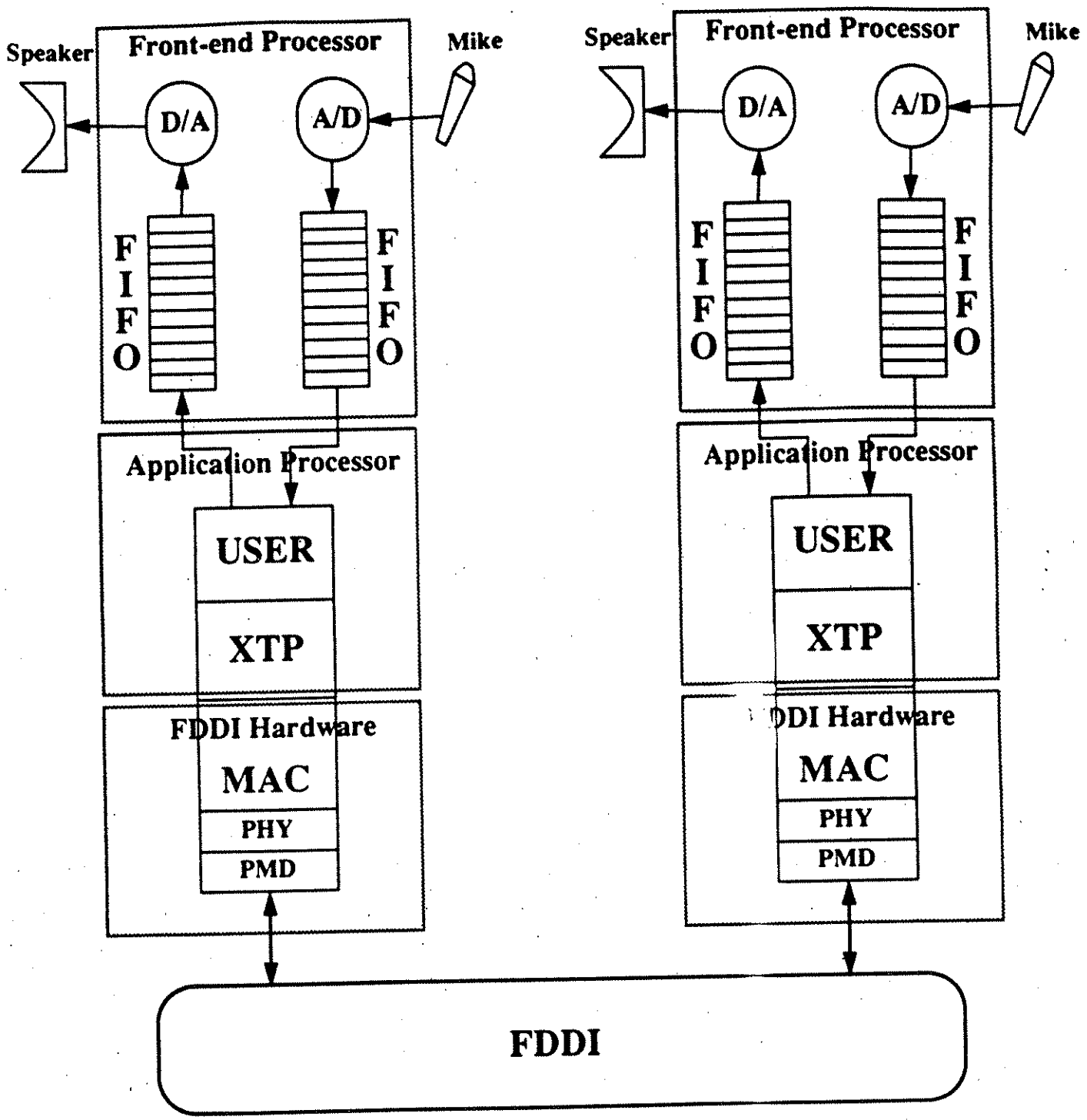

Figure 1.

Architectural Concept

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.

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 that 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 that 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, whereas 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 possible 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 (primarily 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 intermediate 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.

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), both written in Microtec C. While replacing selected system components could have increased overall performance (e.g., replacing the 68020 with a 33 MHz 68040, or choosing a faster FDDI board), we did not attempt such "heroic" measures to increase performance. The hardware suite is simple, commercial, standard, and readily available; thus our goal 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 later). LAN throughput measurements were made with *FiberTap*, our real-time FDDI network monitor [Weav89].

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. Throughout 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. Overview of Experiment

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). 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 emitted 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 others; 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 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.

4.3. 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. Since all our latency measurements are in units of milliseconds, the timer resolution was perfectly adequate for our needs.

5. EXPERIMENT 1: BASIC CONFIGURATION

5.1. 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 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 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 sta-

tions 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. 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
No background processor load
No background FDDI load
69 bytes framing overhead (44 bytes from XTP, 25 from FDDI)

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

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 0.14 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
No background processor load
No background FDDI load
69 bytes framing overhead

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

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

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

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.

6. EXPERIMENT 2: SYNCHRONOUS SPPT BACKGROUND FDDI LOAD

6.1. 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 333 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. Obvi-

ously, 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 4.

SPPT BACKGROUND LOAD GENERATION

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 4
SPPT Background Load Generation Parameters

6.2. 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.

Table 5 shows the number of voice channels which could be carried for a given voice data size and given background synchronous SPPT load on FDDI.

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 5
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.

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

Adding 25 Mbits/sec of background load increases the 99.9% threshold point by about 0.15 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.

7. EXPERIMENT 3: SYNCHRONOUS MPPT BACKGROUND FDDI LOAD

7.1. 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.

MPPT BACKGROUND LOAD GENERATION

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

7.2. Experiment 3: Analysis

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 others, generated after the burst begins, will suffer a smaller delay. 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 (not shown here) we observe that variance has increased substantially when compared to SPPT service. 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. The lower band around 4 ms corresponds closely to the results from the basic experiment in which there was no

background load.

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. 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.

8. EXPERIMENT 4: RETRANSMISSION

8.1. 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.

8.2. 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.

9. EXPERIMENT 5: MULTICAST

9.1. Experiment 5: Design

A unique feature of XTP is its capability for supporting a 1-to-many connection called *multicast.* 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 five we enlarged our network to include not only the two voice stations and the background traffic generator, but also an additional voice station that 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.

9.2. Experiment 5: Analysis

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

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 can be 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.

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.

10. CONCLUSIONS

Using the Xpress Transfer 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, 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 (is it worth using a transport protocol?) 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.

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 a special case), 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.

Finally, we repeat the point that we made no attempt to tune the system for higher performance. There are many obvious optimizations: improve the hardware, reduce the number of required acknowledgements, compress the voice data, don't transmit silence, etc. Had we tuned the system, we are confident that we could have doubled the number of voice channels carried.

11. ACKNOWLEDGEMENTS

This work was supported by E-Systems, Inc., Falls Church, Virginia, and was supervised there by Mr. Stephen Johnson and Mr. Carlos Burns. Co-funding was provided by the Institute for Information Technology of the Virginia Center for Innovative Technology, Herndon, Virginia. We are grateful to E-Systems and CIT for both their funding and technical input. The work was performed in the Computer Networks Laboratory at the University of Virginia, where we received valuable assistance from Mr. Robert Simoncic and Mr. John Fenton.

12. 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.

[Weav89] A.C. Weaver and J.F. McNabb, "A Real-Time Monitor for Token Ring Networks," MIL-COM'89, Boston, MA, October 16-18, 1989.