Providing Performance Guarantees in an FDDI Network

Darrell D. E. Long,[†] Carol Osterbrock[‡]
Computer and Information Sciences
University of California, Santa Cruz

Luis-Felipe Cabrera
Computer Science Department
IBM Almaden Research Center

Abstract

A network subsystem supporting a continuous media file system must guarantee a minimum throughput, *a maximum* delay, *and a maximum* jitter. *We present a transport protocol that provides these guarantees.*

To support different types of service, our protocol is built from modules selected to meet the requirements of each communication session. A buffering technique is used to provide jitter guarantees. To provide throughput and delay guarantees, network performance is optimized based on the required transfer rate.

The effects of controlling transmission rate and packet size are presented. The resulting transport protocol is modeled on a simulated FDDI network and the results are analyzed. We show that the protocol provides the required guarantees for the anticipated types of traffic.

1 Introduction

Digital multimedia data is characterized by high data volume that must be transferred with strict performance requirements. Each component of a multimedia system must cooperate by providing guarantees that it can sustain the required transmission rate.

A multimedia file system must also address the problem that not all the components of the system can provide the same performance. In particular, current disk technology is insufficient to provide multimedia transfer rates. Thus, it is necessary to distribute disk I/O over several storage units that can operate in parallel to achieve high transfer rates.

[†]Supported in part by the National Science Foundation under Grant NSF CCR-9111220, by the Institute for Scientific Computing Research at Lawrence Livermore National Laboratory and by the Office of Naval Research under Grant N00014–92–J–1807.

[‡]Supported in part by the Institute for Scientific Computing Research at Lawrence Livermore National Laboratory and by a CE/CIS Domestic Graduate Research Assistantship

Swift [1] is an architecture for a distributed file system to support the storage and processing of multimedia data. It uses multiple storage repositories and processing units to provide the required performance, and uses a local area network for communication between the clients and the storage units. To provide satisfactory service the architecture exposes parameters to be guaranteed during client sessions.

The network subsystem must address two problems. The first is meeting throughput requirements. Although the storage of data may be divided among several repositories, all of the data must pass through the network. Recent advances in network technology, in particular *FDDI* (Fiber Distributed Data Interface) [3], begin to provide the necessary data rates. The second is guaranteeing that the delay between the sender and the receiver will not exceed the required maximum. This can be achieved using known properties of FDDI networks, and using buffering and rate control to insure that jitter, or variability in delay, will fall within the desired limits.

We present a brief overview of the Swift architecture and then focus on the network transport protocol that we have developed for it. In §2 we present background information, including a brief description of the Swift architecture and related research. We discuss which performance parameters can be guaranteed in §3. In §4 we discuss how the transport protocol was designed and how the design decisions were made. The components of the protocol are described in §5. A simulation of the protocol and the performance results obtained using it are presented in §6.

2 Background

There are three main limitations to storing, retrieving, and processing multimedia data: the speed of mass storage devices, processors, and communications. Processors are now fast enough to manipulate the data at the required rates. Disk technology has improved, but the data rates provided are still insufficient. With FDDI, network tech-

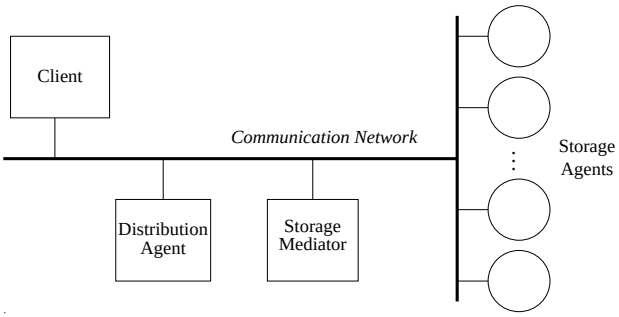

Figure 1: Components of the Swift Architecture

nology is approaching the point of providing sufficient data rates.

2.1 The Swift Architecture

The goal of the Swift I/O architecture [1] is to support high data rates in a general purpose distributed system. It addresses data rate mismatches between the application, the storage devices, and the interconnection network. Swift provides high data rates by using a high-speed interconnection medium and using multiple storage devices operating in parallel. Swift can use any appropriate storage technology including high-performance disk arrays. It can thus adapt to technological advances to support ever increasing I/O demands. The components of the Swift architecture are depicted in figure 1.

Swift objects are managed by *storage agents*. An implementation operates as follows: when a client issues a request to access an object, a *storage mediator* reserves resources from all the necessary storage agents and the communication subsystem. The storage mediator then presents a *distribution agent* with a *transfer plan*. Swift assumes that sufficient storage and data transmission capacity can be preallocated and thus will be available. It supports negotiations between the client (which can behave as a *data producer* or a *data consumer*) and the storage mediator to perform resource preallocation. A request that cannot be satisfied is denied, but may be reissued later when more resources are available. The distribution agent follows the transfer plan to store or retrieve client data at the storage agents.

2.2 Network Requirements

There are several constraints on the delivery of data. First, data must be delivered at a nearly constant rate for the duration of the transmission. This requires the network to guarantee that a nearly constant fraction of the bandwidth will be available, and that there will be no delays in the transmission which exceed the requirements of the application.

Second, messages must be kept in sequence, and message loss must be minimized. In order to meet time constraints it may be necessary to allow part of a stream to be discarded instead of waiting for it to be received correctly. However, there are some types of communication, such as file transfer, where losing part of the data is unacceptable, and timing guarantees are not required.

Third, available bandwidth may affect decisions made by clients. For instance, if there is insufficient bandwidth to transmit an uncompressed stream, the client may decide to use compression, even though that may degrade the quality of the data. Alternatively, the client may decide to wait, or change the parameters of the transmission.

2.3 Types of Data Communication

The performance requirements of the protocol depend on the type of data transfer. Ferrari [6] examines different types of data transfers and their associated requirements. The primary types of transfer that need to use large amounts of network resources are audio, video, and large file transfers.

One type of interactive audio or video session involves two clients transmitting voice or video streams to each other, for instance in teleconferencing. These types of transmission require that the delay be as small as possible, so that conversations take place naturally.

A second type of interactive service involves the editing and combining of audio and video streams. This second type of session is a primary goal of Swift. The important constraint in this type of session is that the delays within the system should be as small as possible, because the user will want to be able to stop and start streams with imperceptible delay.

A non-interactive continuous media session consists of data being recorded or played back. In this case, an initial delay can be tolerated, but the amount of jitter, or variability in the delay, must be minimized. In addition, both voice and video have high bandwidth requirements and it may be necessary to synchronize streams which may not have been created together.

Compression adds an additional complication to the transport service as the amount of compression that will occur for a given data stream is not known in advance and usually varies over the length of the stream. This means that the exact rate of data transfer will be unknown in advance and may vary over the duration of the session.

Necessary control messages occur unpredictably and must be delivered quickly and reliably. These short messages use the available bandwidth left after the reserved traffic has been allocated. Examples of control messages are the messages used to activate and deactivate connections and acknowledgments used for flow control and reliability. Because of the nature of the FDDI channel some of the available bandwidth is always available for unguaranteed messages, even when all of the guaranteed bandwidth is being used.

The delivery time of control messages cannot always be controlled as no channel is completely reliable. When a control message is lost the sender retransmits it. Thus delivery time may be longer than expected. The Swift transport protocol operates with the assumption that control messages will be received within a bounded time, but if they are not the protocol can still proceed correctly.

2.4 Related High-speed Protocols

Several transport protocols have been developed to take advantage of recent advances in data communication performance. They are called "light-weight" protocols because they use as little bandwidth and processing as possible for the protocol. None of them provides a satisfactory protocol for our needs given the diversity of traffic that need to be accommodated. Some light-weight transport protocols in use include *Delta-t*, *NETBLT*, *Datakit*, and *XTP* which are compared and analyzed by Doeringer, *et al.* [4]. Some of their common characteristics are: reduced overhead to establish and break connections; minimal packet exchange for flow control; use of rate control to reduce contention; and fixed-size headers.

Delta-t (Δt) [8, 13] is a transport protocol for high-speed networks based on a timer mechanism. Both sender and receiver maintain a timer, along with the rest of the information required for a sliding window protocol. The timer is used to decide when the connection is closed, and the connection is initiated by the first data packet. By using data packets to open connections and timers to determine when connections are closed, Delta-t avoids the problem introduced by using non-data packets for handshaking. The sliding window can be moved without explicit acknowledgment messages.

NETBLT (*NETwork BLock Transfer*) [2] is a bulk data transfer protocol designed for use on the Internet. Since this protocol may encounter links with varying data capacity, it uses rate control to minimize network congestion and multiple buffering to minimize errors and to provide high throughput. This same rate control algorithm is used in Swift. Briefly, a transmission is divided into buffers of a size agreed upon by the sender and receiver. The buffer transmission rate is determined. The protocol then calculates the rate at which to transmit the network data packets that will be assembled from the buffers. It also calculates the *burst size*, the size of the largest continuous stream of packets, and *burst rate*, the effective rate of the largest burst. Based on these calculations, the protocol determines the correct routing, packet rate, and buffer size to complete the transfer. By thus regulating packet rate network congestion is controlled.

3 What FDDI can Guarantee

Fiber Distributed Data Interface (FDDI) is a specification for a high-speed physical and data link protocol to be used with fiber optic cable, although it can also be implemented on copper wire. FDDI uses a *timed token ring* protocol to control access. Sevcik and Johnson [12] have analyzed the properties of the FDDI protocol and found that the upper bound on the token cycle time, and therefore on the time between two transmissions of synchronous frames, is less than twice the target token rotation time (TTRT) when the synchronous allocation is done correctly.

By using the FDDI synchronous mode the Swift transport protocol can operate with a known upper bound on the delay of any packet. This information is used in two ways. First, the protocol uses timers to decide if the connection is valid or if a packet has been lost. Since the upper limit on delay is known, it is possible to use the strictest possible bound. Second, maximum packet delay is used to set buffer sizes so as to guarantee the jitter seen by the application.

4 Goals of the Transport Protocol

We use the following terms to describe the functions of the protocol, the data structures, and the procedures involved. A *transport service data unit* (TSDU) is the basic unit of data used by the application. In the case of a video stream, a TSDU is a video frame. In the case of an audio stream, a TSDU may be a sample or a sequence of samples. For file transfer applications, the TSDU is defined by the application. The TSDU may also be called the block size or frame size. A *transport protocol data unit* (TPDU) is the basic unit of data used by the transport protocol, also called a packet. It may be the same size as a TSDU, larger or smaller. In an FDDI network, the TPDU

size is generally kept as large as possible since it has been shown that the best utilization of resources occurs when the packets are large [5].

4.1 The Swift Environment

The transport level protocol was designed with the assumption that the other layers of the protocol stack also provide guarantees. It also assumes that the higher layers deliver data at a known rate. In the case of real-time devices such as microphones or cameras the data rate is known. For storage devices the data rate is controlled by the storage subsystem [10]. In general, the data rate is controlled according to the throughput requirements specified by the application.

An important function of the protocol is insuring that the delivery rate at the receiver is guaranteed within the bounds specified. The rate that must be controlled is the rate of transmission of TPDUs across the network. The transport of TPDUs is controlled by a timer-based protocol [8]. Rate control has been shown to be effective in reducing contention in large file transfers [2], and is well suited to the constant data flow that is characteristic of continuous media.

Because the transfer rate is nearly constant, and the senders and receivers have guaranteed that they can provide the resources to handle the rate specified, and because the protocol assumes that the network will deliver the data reliably, little flow control is used. The nature of the data will often make it preferable to simply ignore a missing TPDU rather than attempt to have it retransmitted. The strength of the delivery guarantee depends on the nature of the data. For example, losing a video frame is less noticeable than losing an audio sample.

When flow control is necessary it must be as transparent as possible. The method chosen is similar to the Delta-t [13] simplex connection method. This is a window-based flow control where acknowledgments are assumed if a negative acknowledgment is not received. The maximum wait is computed from knowledge of the network. Acknowledgments may also be sent, but the protocol is designed so that the sender never has to wait for an acknowledgment to continue. If a packet is lost, the sender will receive a retransmission request.

Buffering is used to minimize variations in delay, to avoid data starvation at the receiving end, and to avoid queuing delays at the sending end. The method of allocating buffers to minimize jitter is discussed in §4.2. Because the buffers may hold large amounts of data the time it takes to copy data to and from buffers can be significant. To minimize the copying done the application loads data directly into the buffers and then releases them to the transport service. To enforce the guaranteed rate constraints the application waits for a buffer if none are available.

The storage mediator is responsible for allocating the bandwidth of the network to clients. If a request cannot be granted it may be possible to reschedule it for a later time or to relax the requirements. This must be negotiated by the storage mediator, sender, and receiver. If a client request does not include a bandwidth requirement, then the bandwidth allocation is set by the storage mediator. The bandwidth that is guaranteed is the greatest amount that the session can use so clients must request their maximum burst rate.

4.2 Delay and Jitter Guarantees

The variation in transmission delay is called *jitter*, and cannot be tolerated in continuous media streams. When high-quality output is desired, as in audio transmissions, jitter control can be more important than the actual amount of delay. Ferrari [7] presents an analysis of jitter in high-speed networks, and a method for jitter control on an internetwork.

The goals of providing a guaranteed maximum jitter, a guaranteed maximum delay, and a guaranteed minimum data rate conflict. By using buffers large enough to hold all the data that is being transferred, and filling them at the beginning of the session, jitter can be eliminated. However, the initial delay may be unacceptable, and the storage requirements could be impossible to meet.

4.2.1 Guaranteed Maximum Jitter for Large Frames

To guarantee that jitter will be minimized for large frames, it must be guaranteed that the TPDUs used to make the frames will be transmitted in a timely fashion. In order to make this guarantee, the following parameters must be considered: f_n, the time when frame n is expected to be completely received; r_n, the time when all TPDUs that make up frame n have arrived at the receiver; $d_{\max}$, the maximum possible delay over the network; $d_{\min}$, the minimum possible delay over the network; $j_{\text{net}} = d_{\max} - d_{\min}$, the maximum jitter due to network delays; p, the size of the data in a packet, or TPDU; b, the size of the data in a frame, or TSDU; τ, the maximum jitter due to rate control and process switches; ρ, the arrival rate of TSDUs at the sender's transport layer; and $j_{\max}$, the guaranteed maximum jitter.

The total jitter perceived by the transport layer will be $j_{\text{net}} + \tau$. The goal of the transport service is to allocate the

buffers in such a way that this jitter can be smoothed, so that for every frame k,

$$\frac{1}{\rho} - j_{\max} \leq r_{k+1} - f_k \leq \frac{1}{\rho} + j_{\max} \qquad (1)$$

It is already known that the TSDUs will be arriving at the sender at rate ρ. They will then be split into packets of size p, and transmitted to the receiver. The delay given by $r_{k+1} - f_k$ is the maximum allowable delay between the earliest time that frame k could have been used and the latest time that the last packet that makes up frame $k+1$ could arrive. It is assumed that the application layer will use packets at the constant rate ρ whenever possible, and if a packet is too late then the next packet will be used earlier in order to compensate for the late packet. It can be seen that as long as the maximum jitter provided is $j_{\max}$, this procedure will insure that the application receives packets within the jitter guarantees.

With this assumption consider what constraint (1) really means. The start time for f_k should always be $T_s + k \times 1/\rho$. The desired value of f_{k+1} is $T_s + (k+1) \times 1/\rho$, so the difference between them is just $1/\rho$. Therefore, the constraint (1) can also be stated as:

$$|r_{k+1} - f_{k+1}| \leq j_{\max}. \qquad (2)$$

In other words, the difference between the actual time the buffer is ready and the time it should ideally be ready cannot be greater than the guaranteed maximum jitter. In order to insure that this is always true, there should be enough buffers so that packets arrive early enough to compensate for network delays. In addition, packets that arrive earlier than required will be stored until they are needed, and not discarded.

However, at no time can the bandwidth used exceed the bandwidth allocated, as was discussed in §4. If the exact bandwidth required is allocated, and the only delay allowed is the delay required to fill the first buffer, then the requirements can be met only if $j_{\text{net}} + \tau$ is less than $j_{\max}$, and there are no losses. For the rest of this discussion, we will assume that there are no losses, since FDDI is in general very reliable. In order to minimize jitter, we look at what happens when extra bandwidth is allocated.

It is possible to send each frame slightly faster than the session requires. Depending on how much extra bandwidth is present and how large the buffers at the receiver can grow, each frame can arrive earlier than it is needed. For example, say that the frame is released to the sender at time t. If the session is only allowed the guaranteed data rate r, then it will take at least b/r seconds to transmit the entire frame. However, if a factor Δr is added, then the time will be $b/(r + \Delta r)$, which is a shorter time interval. This means that the frame will be available at the receiver earlier and thus the maximum jitter in the transmission can be smaller.

On the other hand, it is not possible or desirable to allocate more buffers than are necessary. There are several reasons for this. First, when there are more buffers, the delay at the beginning of the session becomes larger. Second, the size of the buffers is limited by the amount of memory that can be allocated to the session. Third, the data arrives at a constant rate, so if there are more buffers allocated than necessary, the extra buffers will never be used.

The method used to allocate buffers is to find the desired number of buffers such that the jitter constraint can be met and the initial delay constraint is not violated. Once the number of buffers has been found it is possible to compute the burst rate [9] required to keep these buffers ready to be delivered at the receiver when needed.

5 Operation of the Protocol

The Swift transport protocol was designed to provide high-speed, light-weight service with delivery guarantees. There are three main functions of the transport protocol: establishing the connection, transmitting the data, and removing the connection. Each is discussed below.

5.1 Establishing the connection

When the storage mediator notifies the sender and receiver that a session has been initiated they each initialize the necessary modules to complete the connection with the given parameters. The sender starts the application interface for the type of application that is being initiated, and the packet builder that will construct the packets, either by splitting the buffers or by combining multiple buffers. The receiver starts the application interface for the correct type of application, and the packet builder that will perform the inverse operation of the sender's packet builder.

At the time the connection is established, the values of some of the parameters of the session are used to set up the protocol modules. The TSDU size is fixed, and is either set by the request or computed by the storage mediator. It is the same for both sender and receiver. The maximum number of buffers that can be active at each station is also fixed by the storage mediator when the request is analyzed. The calculated throughput is communicated from the storage mediator along with the maximum possible network delay.

5.2 Transmission Protocol

The data transmission protocol consists of the sender protocol and the receiver protocol. The protocol relies on the fact that the application will be providing or using data at a nearly constant rate, and therefore the transport service needs to have buffers ready at that same rate in order to keep the flow of data within the bounds established when the connection is established.

5.2.1 Sender

The sender protocol state diagram is shown in figure 2.

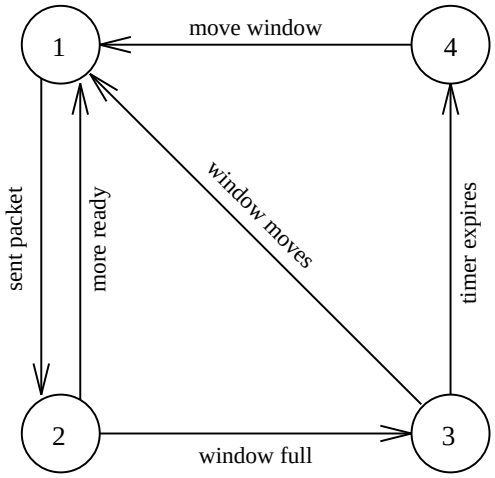

Figure 2: State diagram for sender protocol

State 1 is the normal state of the protocol. The sender is ready to send TPDUs, but waiting for the TTRT timer T to expire. When the timer expires, the sender sends packets. After each packet is sent, it enters state 2.

State 2 is reached after each packet is sent. If there are more packets ready to send, the protocol returns to state 1. If there are no more available buffers, the sender proceeds to state 3.

State 3 is reached when no more buffers are available. From state 3, either the timer expires or a positive acknowledgment or negative acknowledgment is received. The protocol moves the buffer window past the acknowledged packets, and returns to state 1. If the timer expires for a packet, the packet is assumed to be acknowledged implicitly, since no negative acknowledgment was received. However, a packet can only be acknowledged if all the packets before it have been acknowledged.

In state 3, if the timer for all unacknowledged packets expires, the protocol proceeds to state 4.

State 4 is reached if the timer has expired for all of the active packets. The sender proceeds as if all the packets have been acknowledged since the receiver will take whatever action is necessary to continue the communication. The protocol then returns to state 1.

5.2.2 Receiver

The receiver protocol state diagram is shown in figure 3.

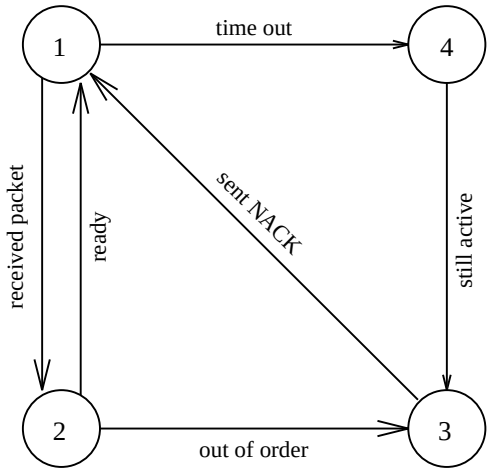

Figure 3: State diagram for receiver protocol

State 1 is the normal state of the protocol. The receiver is waiting to receive the next packet. When a packet is received the receiver enters state 2, while if the timer expires it enters state 4.

In state 2, a packet has been received. If the packet is expected, it is added to the buffer and the protocol returns to state 1. If the packet is out of order, then an earlier packet was not received, and the protocol enters state 3, to request retransmission of the earlier packet. If the received packet is a duplicate, it is ignored and the protocol returns to state 1.

In state 3, a packet must be retransmitted. First, the packet's timer is compared to the round trip time. If the timer will expire before the packet can be received no negative acknowledgment request is sent. If there is time to retransmit the packet then a request for retransmission is sent to the sender. Whether or not the request is sent the protocol returns to state 1.

In state 4 nothing has been received in the time-out interval. The receiver must request retransmission of all packets starting with the next one that is expected up to the window end. It enters state 3 to request retransmission.

5.3 Terminating the Connection

The connection can be terminated by the sender, the receiver, or the storage mediator. The storage mediator sends a message to both the sender and receiver to inform that a connection is terminated. When the receiver does not receive any packets after the computed Δt for the connection, the connection is implicitly closed and the receiver sends a termination message to the storage mediator.

When the sender receives a connection termination message it immediately signals to the application that the connection is no longer available and stops transmitting packets. It then releases all the buffers it held and sends a message to the storage mediator that the session is closed. The sender then completes. If the termination message arrives while there are still packets to transmit the sender opens a new connection with the same parameters as the previously existing connection.

When the receiver receives a connection termination message it places a mark after the last buffer that was completely received indicating the end of the transmission. It then discards any of this session's packets that it receives after the termination message. When the end-of-transmission marker has been placed a message is sent to the storage mediator that the session is closed. After all of the received buffers have been processed by the application the receiver releases all the buffers allocated to the task.

After the storage mediator has sent messages to the sender and receiver, and received session close confirmation messages, it removes the session from the list of active sessions.

6 Simulation

We have simulated the performance of the protocol using CSIM [11]. CSIM is designed for simulation of networks and contains tools for setting up processes and communication links as well as many tools for generating test events and gathering statistics.

The protocol simulation allows us to specify any number of audio, video, or general data sessions. The simulation assumes that the incoming data is arriving as guaranteed, that is, that it is being provided at the rate that was requested by the storage mediator. The simulation also assumes that each session is using a separate processor.

To test the behavior of the network under realistic load constraints, various combinations of audio, video, and block transfers were tested. For the simulation, video streams have a data rate of 3600 kilobytes/second, which is 30 frames/second and 120 kilobytes/frame. Audio streams have a data rate of 8 kilobytes/second, which is a typical audio sampling rate. The file transfer rate was set at 100 kilobytes/second. The tests combined different types of streams, or used all the same type. The simulation time varied from 10 to 60 seconds. In addition, the number of stations on the ring was varied from 2 to 100. The stations were assumed to be evenly distributed around the ring with the distance between the senders and receivers chosen randomly.

The protocol simulation takes the buffers as they are passed in and either splits them or combines them to form network packets. The size of the packets that are built is dependent on the type of network being simulated (for FDDI, packets are 4500 bytes). These packets are then passed to the network layer according to the rate control constraints necessary to fulfill the guarantees.

The performance parameters of interest are measured throughput for each stream, maximum delay, and jitter. The maximum delay is reported directly. Throughput is found by measuring the duration of the session and dividing that time into the number of bytes transmitted. To find the maximum jitter, the "gaps" between buffers were measured. A gap is the time between reception of the last packet of buffer n and the last packet of buffer $n-1$. There is a target value for these gaps, which is the arrival rate of the buffers at the sender. The jitter measured is the maximum variation over the target arrival time. This is because a buffer available early will not cause jitter, since it goes unused until it is needed.

The assumptions for these measurements are that throughput is guaranteed for the simulated load and that the maximum delay and maximum jitter for each session are guaranteed to be twice the target token rotation time ($2\times$TTRT). The measured throughput, maximum delay, and maximum jitter were plotted as a function of the number of senders and the load.

6.1 Results

The results of the simulations are shown in figures 4, 5, and 6 for a TTRT of 2 milliseconds. The loads chosen were various combinations of audio, video, and block data. The points were plotted by finding the total load due to all the sending stations. Each run has a duration of 60 seconds. The largest load that can be allocated to guaranteed traffic is slightly less than 12.5 megabytes/second, which is the total capacity of the FDDI channel. The exact amount that

can be allocated depends on the length of the channel and the number of stations, and the TTRT.

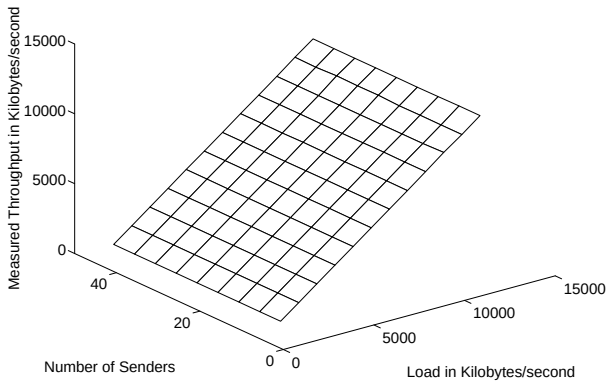

Figure 4: Throughput: TTRT 2 milliseconds

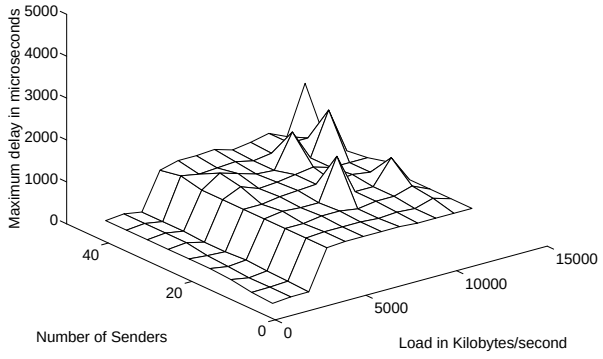

Figure 5: Maximum Delay: TTRT 2 milliseconds

It can be seen that the throughput is consistently equal to the load. This is consistent with the analysis of FDDI by Sevcik and Johnson [12]. In addition, the delay and jitter fall within the guaranteed maximum, which is 4 milliseconds for a TTRT of 2 milliseconds.

In all the simulations the maximum jitter was the same as or less than the maximum delay. This is because the protocol is designed to have packets arrive earlier than they are needed by sending them at a slightly faster rate than the exact data throughput rate. In most cases when a buffer was needed it was already present and the jitter was zero.

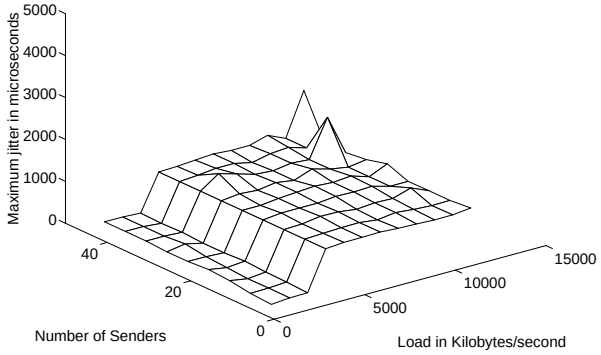

Figure 6: Maximum Jitter: TTRT 2 milliseconds

The only time jitter greater than zero resulted was when the delays of the packets making up the buffer combined so that the last packet arrived later than the expected use time. This happened mainly on block file transfers where the buffer size was equal to the packet size, so any delay in packet delivery is also jitter for the application.

It can be seen that the FDDI network has no trouble providing the throughput required. This is true even at the highest loads that were simulated. The network also provided the guaranteed maximum delay and jitter. In most of the experiments the maximum jitter was less than the TTRT. This is once again because the protocol reserves extra bandwidth and uses multiple buffers in order to send the data before it will be needed.

There are several data points where the maximum jitter was above the TTRT but it was still under the guaranteed maximum of 4 milliseconds. These are isolated points where the simulated load was heavy and therefore the traffic level was high. Since the simulations were run several times for each combination of load and number of senders, it is not unexpected that in one experiment there may be a value of jitter that is higher than the rest. In no experiment did the value exceed the guaranteed maximum.

These results indicate that the protocol can provide the required performance guarantees when the underlying network is a FDDI network capable of providing synchronous transmission mode, the application is able to deliver the data in guaranteed buffer sizes at a guaranteed rate, and the operating system is able to guarantee fair access to processing cycles and shared memory.

7 Conclusions

The Swift transport protocol provides mechanisms to establish guarantees for the speed of the transfer of multimedia data and for delays in delivery. The protocol is customized for the application when a connection is established, so that the necessary services can be provided.

The Swift protocol is a light-weight protocol that insures high bandwidth utilization. Swift uses rate control to ensure that all sessions can provide guaranteed service. In addition, when necessary, it uses sliding window flow control to make sure that all the data is delivered in order. The protocol is designed to provide the type of service requested by the application which may be continuous media or file transfer.

Continuous media data is guaranteed a bandwidth and a maximum jitter. If requested a maximum end-to-end delay can also be guaranteed. Data is delivered as reliably as possible, but if a packet is lost, reliability is sacrificed in order to maintain the bandwidth and jitter guarantees.

File transfer data is also guaranteed a bandwidth. This is not strictly necessary, but since other sessions need to reserve bandwidth, it is necessary that any session using a significant portion of the bandwidth must do so via reservations. Since there are no delay or jitter requirements data is transferred reliably with the guarantee that no data will be lost.

The protocol was simulated under realistic FDDI network conditions. Performance was found satisfactory and to provide good resource utilization. The protocol relies on the network providing guarantees of delivery time and delay. The simulations show that the protocol will behave correctly when the assumptions about the characteristics of the other components of the architecture are satisfied.

Acknowledgements

We are grateful to M. Long, B. Montague, K. B. Sriram, K. Taylor, M. Thakur, and A. Varma for their many helpful comments.

References

[1] L.-F. Cabrera and D. D. E. Long, "Exploiting multiple I/O streams to provide high data-rates," *Computing Systems*, vol. 4, no. 4, 1991.

[2] D. D. Clark, M. L. Lambert, and L. Zhang, "NETBLT: A bulk data transfer protocol," RFC 1112, Network Working Group, 1987.

[3] Digital Equipment Corporation, *A Primer to FDDI: Fiber Distributed Data Interface*. Digital Equipment Corporation, 1991.

[4] W. A. Doeringer *et al.*, "A survey of light-weight transport protocols for high-speed networks," *IEEE Transactions on Communications*, vol. 38, Nov. 1990.

[5] D. Dykeman and W. Bux, "Analysis and tuning of the FDDI media access control protocol," *IEEE Journal on Selected Areas in Communications*, vol. 6, July 1988.

[6] D. Ferrari, "Guaranteeing performance for real-time communicaton in wide-area networks," Technical Report, University of California at Berkeley, 1990.

[7] D. Ferrari, "Design and applications of a delay jitter control scheme for packet-switching internetworks," in *Network and Operating System Support for Digital Audio and Video*, Nov. 1991.

[8] J. G. Fletcher and R. W. Watson, "Mechanisms for a reliable timer-based protocol," in *Computer Networks 2*, North-Holland Publishing Company, 1978.

[9] T. Little, "Protocols for bandwidth-constrained multimedia traffic," in *Proceedings of the* 4th *IEEE International Workshop on Multimedia Communications*, 1992.

[10] D. D. E. Long and M. N. Thakur, "Scheduling real-time disk transfers for continuous media applications," in *Proceedings of the* 12th *Symposium on Mass Storage Systems*, IEEE, Apr. 1993.

[11] H. Schwetman, "CSIM reference manual (revision 15)," Tech. Rep. ACT-ST-252-87, Microelectronics and Computer Technology Corporation, 1991.

[12] K. C. Sevcik and M. J. Johnson, "Cycle time properties of the FDDI token ring protocol," *IEEE Transactions on Software Engineering*, vol. 13, Mar. 1987.

[13] R. W. Watson, "Delta-t protocol specification," Tech. Rep. UCID-19293, Lawrence Livermore National Laboratory, 1983.