Protocols for Providing Performance Guarantees in a Packet-Switching Internet

Carlyn M. Lowery

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

ABSTRACT

As advances in technology enable us to implement very high speed computer networks, we expect to use our networks for more diverse applications. While the Internet was designed with textual data processing in mind, future networks will carry information such as voice, music, images, and video, along with textual data. Many new applications will have real-time performance requirements, where the timing of data arrival is crucial to its usefulness.

This paper describes a methodology developed at the University of California at Berkeley to support such applications, reviews related research work, and proposes a real-time delivery system, composed of a new protocol for administration of real-time connections, combined with modifications to the Internet Protocol (IP) to support such connections. Transport protocol requirements are also discussed. This work is intended to facilitate experiments with real-time communication over the Experimental University Network (XUNET).

1. Introduction

As advances in technology enable us to implement very high speed computer networks, we expect to use our networks for more diverse applications. While the Internet was designed with textual data processing in mind, future networks will carry information such as voice, music, images, and video, along with textual data. Many new applications will have real-time performance requirements, where the timing of data arrival is crucial to its usefulness.

This paper describes a methodology developed at the University of California at Berkeley to support such applications, reviews related research work, and proposes a real-time delivery system, composed of a new protocol for administration of real-time connections, combined with modifications to the Internet Protocol (IP) to support such connections. Transport protocol requirements are also discussed. This work is intended to facilitate experiments with real-time communication over the Experimental University Network (XUNET).

1.1. Background

The performance requirements of new real-time applications are still being explored, and they have been discussed in the literature ([ChMe88] [Fe90] [WrTo90]). The requirements include (but may not be limited to): delay bounds - limits on the maximum delay experienced by a packet; loss bounds - limits on the likelihood of packet loss; and jitter bounds - limits on the variability of the delay. Applications differ regarding the relative importance of these factors. For example, for video-conferencing, delay and jitter bounds are stringent, but some packet loss may be tolerated. For one-way transmission of CD-quality music, the delay bound may be long, while the jitter and loss bounds are small. For video browsing of still images or for shared document editing, the delay bound may be short and the loss bound stringent, but the jitter bound unnecessary.

Such requirements are not met by the current packet-switching Internet, which employs the Internet protocols IP [Po81a], TCP [Po81b], UDP [Po80]. The network protocol, IP, provides no guarantees about timely, reliable, or ordered packet delivery. Packets sent onto the Internet may traverse different routes, and they may or may not arrive at their destinations. The most commonly used transport protocol, TCP, adds reliability by employing end-to-end acknowledgment and retransmission. Such reliability is inappropriate for many real-time applications; if a video frame for interactive display does not arrive on time (or nearly so), it may as well never arrive. In this case, retransmission is a waste of resources. The connectionless transport protocol, UDP, provides unreliable delivery using IP.

Circuit switching can be used to provide the necessary guarantees, but it does not provide an optimal solution. Circuit-switching is the standard method for providing real-time performance guarantees in the current commercial environment. In a circuit-switching network, an application must reserve the maximum bandwidth which it intends to use. Bandwidth which is not used is wasted. This works adequately when traffic is fairly constant, but when traffic is bursty, it results in severe under-utilization of resources. Additionally, circuit-switching is inappropriate for short transmissions, where the connection setup time creates excessive overhead.

A modified packet-switching network could serve both real-time traffic, with its associated temporal guarantees, and best-effort traffic, which does not require such guarantees, and in many

This material is based upon work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

cases does not even require connections. Such a network is discussed herein.

1.2. Methodology

The methodology for providing real-time performance guarantees is based upon connection establishment with resource reservation, as described in [FeVe89] and [FeVe90]. A brief review follows. End-to-end performance guarantees for a conversation are provided by creating a connection with fixed routing and performance bounds for each node along the path from source to destination. Such a connection may be called a real-time channel.

To establish a real-time channel, a client specifies its performance requirements and describes its traffic characteristics. An establishment message is sent from the client's host (the source) to the destination. At each node along the path, the best performance (e.g. delay and loss bounds) which can be guaranteed is computed, and tentative resource reservations are made. The offered guarantees are based upon the node's processing power, buffer capacity, and line bandwidth, as well as the already existing real-time channels and their performance requirements. Information about the new channel is retained, such as the local performance requirements, local resource reservations, and the next node on the path.

The destination host determines whether the accumulated guarantees meet requirements, and if so, finalizes the channel establishment. It does this by determining the amount by which performance requirements have been exceeded, and deciding which nodes may reduce their guarantees, and by how much. A confirmation message is sent back along the same path, nodes reduce their resource reservations to match the reduced requirements, and the client is notified of successful channel establishment.

Failure of channel establishment may occur at any node. If a node is unable to provide guarantees which meet the minimum requirements, a failure message is sent back immediately.

Upon notification of channel establishment, the client begins sending packets along the real-time connection. All nodes are responsible for meeting their guarantees, giving priority to real-time packets and serving best-effort packets as time permits. Distributed rate control is used to prohibit clients from exceeding their specified loads and interfering with other channels; each node monitors traffic and appropriately penalizes offending channels.

A real-time channel is viewed as a legal contract between the client and the network. The client promises not to exceed its specified load, and the network promises to meet its guarantees, except in the event of hardware failure. The network ensures that it can meet its guarantees by exercising admission control, refusing to establish new real-time channels when it cannot provide the necessary performance.

1.3. Environment

The environment for which our real-time delivery system has been designed is an internetwork whose gateways implement the system, and whose subnets have performance characteristics which can be bounded. In order to provide guarantees, it must be possible to control performance in each gateway (by resource reservation) and across the interconnecting links. For example, if two gateways are connected by a point-to-point link, transmission time is determined by gateway processing (including queueing), link bandwidth, and propagation delay. However, if two gateways are connected by an Ethernet, the transmission time cannot be bounded, since the medium access time is unbounded. Therefore, performance guarantees cannot be made.

Our experimental environment includes an FDDI ring connected to an ATM network (XUNET II). On an FDDI ring, as long as a node does not generate more synchronous traffic than its reserved synchronous bandwidth, delay can be bounded, because media access delay is bounded by twice the Target Token Rotation Time [Ro86]. Such behavior allows the reservation of bandwidth in order to guarantee performance. The ATM network will participate as a subnet which provides guarantees.

2. Protocol Design

2.1. Design Goals

Our real-time delivery system has been designed with multiple goals in mind. First, interoperability with existing Internet protocols is desirable. Many applications will not have real-time requirements, and those applications will continue to desire the services provided by TCP/IP and UDP/IP. Also, hosts which do not provide real-time services should be able to transmit ordinary IP packets. Second, an iterative approach to development is desirable; for early implementations, minimal essential features will be included, and existing Internet services will provide valuable support. For example, routing services, control services (ICMP), and transportation protocols (TCP and UDP) do not all need to be replaced simultaneously.

Separate protocols are used for channel administration and data delivery. This clean separation was inspired by the design of circuit-switching systems such as CCITT Signalling System 7 and Datakit [FrMa87]. Frazer noted that ‘‘experience indicates that considerable flexibility is required in the service definition for virtual circuit administration.’’ In these systems, this principle has been taken to the extreme, and separate control computers are used for administration. While we will not necessarily implement the control protocol on a separate processor, keeping the protocols separate will make enhancement of the control protocol more straightforward. Additionally, this approach minimizes the overhead processing on data packets, and it minimizes changes to the standard IP header.

2.2. Protocol Description

2.2.1. Channel Administration

The Real-Time Channel Administration Protocol (RCAP) provides management services, such as channel establishment, teardown, and modification.

Control packets are delivered by the network protocol, the Real-Time Internet Protocol (RTIP), and are identified by a unique protocol number in the RTIP header. For experimental purposes, an unallocated number will be used. For general Internet usage, a unique protocol number will have to be obtained from the Internet Assigned Numbers Authority.

Each control packet begins with six fields, as follows:

Type	Sequence Number
Channel Source Address	
Channel Destination Address	
Source Channel ID	Checksum

The Type is a 16-bit field indicating the type of the control message. The Sequence Number is a 16-bit field specified by the sending node to uniquely identify the control packet, which supports acknowledgment and retransmission. The Channel Source and Channel Destination Addresses are the 32-bit Internet addresses of the hosts to be connected by the channel. The Source Channel ID is a 16-bit identifier chosen by the source, which uniquely identifies the channel on that host. The Source Channel ID combined with the Channel Source Address uniquely identifies the channel in the Internet. (This identifier is used for channel control messages which may be sent to arbitrary nodes on the path. For data transfer, a distinct Local Channel Identifier is used at each node on the path to support efficient forwarding.) The 16-bit Checksum applies to the entire control message and is computed similarly to the TCP checksum.* The contents of the remainder of the message depend upon the message type.

Control functions can be divided into three categories: basic functions, additional features, and fault handling. These functions and the messages which support them are described in the next three sections.

2.2.1.1. Basic Functions

The basic functions of the control protocol are channel establishment and channel teardown. In the case of standard operation with no failures, the protocol will function as described in this section.

Five message types support these functions: establish_request, establish_accept, establish_denied, close_request, and close_confirm.

Channel establishment is initiated by the source host by preparing an establish_request message and forwarding it to the first node en route to the destination. The fixed path for the channel is selected as the establishment message works its way towards the destination. The source host provides the following global parameters:†

Parameter	Notation	Size (in bytes)	Description
		Traffic Characteristics	
MinInterarrival	$x_{\min}$	1	minimum interarrival time in milliseconds
AveInterarrival	x_{ave}	1	average interarrival time over a duration I in milliseconds
Duration	I	1	duration in seconds over which x_{ave} is specified
MaxSize	$s_{\max}$	2	maximum packet size in octets
		Performance Requirements	
DelayMax	$D_{\max}$	2	maximum delay in milliseconds
DelayProb	$Z_{\min}$	1	probability that a packet's delay will be lower than $D_{\max}$, expressed in hundredths (100 for a deterministic requirement)
JitterMax	$J_{\max}$	1	maximum delay jitter in milliseconds
JitterProb	$U_{\min}$	1	probability that a packet's jitter will be lower than $J_{\max}$, expressed in hundredths (100 for a deterministic requirement)
NoDropProb	$W_{\min}$	1	probability that a packet is not dropped due to insufficient buffer space, expressed in hundredths

Each node determines the next node en route to the destination, computes the guarantees it can offer, places tentative resource reservations, and selects a Local Channel Identifier to be used by the previous node at the time of data transfer. (The Local Channel Identifier may be any

* The checksum field is the 16 bit one's complement of the one's complement sum of all 16-bit words in the message. For purposes of computing the checksum, the value of the checksum field is all zeros.

† The ''Notation'' column specifies the formal notation which is used in the previously referenced papers about our methodology, [FeVe89], [Fe90], and [FeVe90].

available number other than zero, which is used to identify express datagrams.)

The node saves appropriate state for the channel. This data must be accessible by Channel Source Address and Source Channel ID, for processing of future control messages, as well as by Local Channel ID, for processing requests from the RTIP module. The necessary values include: Channel Source Address, Source Channel ID, Local Channel ID, next node address, previous node address, local guarantees, buffer reservation, and the state of the channel (i.e., idle, connecting, connected, enhancing).

The node adds appropriate information to the message, recomputes the checksum, and forwards the message towards the destination. The data provided by each node is:

Parameter	**Notation**	**Size (in bytes)**	**Description**
NodeAddr	–	4	Internet address of node
MaxDelayI	$d_{i,n}$	2	maximum delay at this node in milliseconds
LateProbI	$P_{do,n}$	2	probability of deadline overflow at this node, expressed in ten-thousandths
MaxJitterI	$j_{i,n}$	2	maximum delay jitter at this node in seconds$*10^{-4}$
JitterProbI	–	2	probability of exceeding the jitter bound at this node, expressed in ten-thousandths
NoDropProbI	$w_{i,n}$	2	probability that a packet is not dropped at this node due to insufficient buffer space, expressed in ten-thousandths
BuffAllocI	$b_{i,n}$	4	buffer space allocated at this node, in bytes

Inclusion of this detailed data in the channel establishment request message allows experimentation with different algorithms for resource relaxation by the destination host. The format of the establish_request message is displayed below; the source may indicate that certain performance parameters are irrelevant by using values of all ones.

<table>
<tr><td colspan="4">standard control packet header</td></tr>
<tr><td>MinInterarrival</td><td>AveInterarrival</td><td>Duration</td><td>DelayProb</td></tr>
<tr><td colspan="2">MaxSize</td><td colspan="2">DelayMax</td></tr>
<tr><td>JitterMax</td><td>JitterProb</td><td>NoDropProb</td><td>Padding</td></tr>
<tr><td colspan="4">NodeAddr</td></tr>
<tr><td colspan="2">MaxDelayI</td><td colspan="2">LateProbI</td></tr>
<tr><td colspan="2">MaxJitterI</td><td colspan="2">JitterProbI</td></tr>
<tr><td colspan="2">NoDropProbI</td><td colspan="2">Padding</td></tr>
<tr><td colspan="4">BuffAllocI</td></tr>
<tr><td colspan="4">data about multiple additional nodes</td></tr>
</table>

The destination host verifies that performance requirements are met, and then prepares an establish_accept packet. The packet includes the performance parameters to be provided (which may surpass those requested), the required performance for each node (which may not surpass that which was offered), and a Local Channel Identifier to be used by the previous node when sending data packets along the channel. (Algorithms for determining required performance for each node in the event of excess resource reservation are specified in [FeVe89] and [FeVe90].) This packet is sent back along the path to the previous node.

Each node relaxes resource reservations as appropriate, and provides necessary information to the local RTIP module. This information consists of: the Local Channel Identifier; the next node address; the delay, jitter, and throughput requirements; and the next node's Local Channel Identifier. RCAP then adds its Local Channel Identifier to the message, for utilization by the previous node, and forwards the packet to the previous node.

The format of the establish_accept message is:

standard control packet header			
MinInterarrival	AveInterarrival	Duration	DelayProb
MaxSize		DelayMax	
JitterMax	JitterProb	NoDropProb	Padding
NodeAddr			
MaxDelayI		LateProbI	
MaxJitterI		JitterProbI	
NoDropProbI		Local Channel ID	
BuffAllocI			
data about multiple additional nodes			

A channel establishment request may be denied by any node along the route, if it cannot accommodate the channel. Rather than forwarding the establish_request message, the node sends an establish_denied message back towards the source. Each node releases resources which were tentatively reserved for the channel, and transmits the message towards the source. The establish_denied packet contains the address of the node which halted the establishment and a code indicating the reason for failure. Its format is:

standard control packet header	
Node Address	
Failure Code	Padding

The last two bytes are padded with zeros so the control message ends on a 32-bit boundary.

The close_request packet is generated by the source host in order to close the connection, and it has no contents. It is passed to each node, so resources may be freed.

The close_confirm packet is generated by the destination host, and it is sent directly to the source host, to advise it that the connection has been closed. It has no contents.

2.2.1.2. Additional Features

Additional features include channel modification and status requests.

A client may modify its channel parameters during a connection, changing the definition of its load or its performance requirements. The source host determines whether the change is an enhancement or relaxation, and it generates an enhance_request or relax_request message as appropriate.

The enhance_request message is identical in format to the establish_request message, and operations at the nodes are similar. Service offers are based upon the use of resources already allocated to the channel combined with additional available resources. The destination allocates requirements and returns an enhance_accept message, which is identical to an establish_request message except that the Local Channel Identifier is not necessary. As in channel establishment, a channel enhancement request may be denied by any node along the route, if it cannot accommodate the new requirements. Instead of forwarding the enhance_request message, it generates an enhance_denied message which is propagated towards the source, causing all nodes to release the additional resources which were tentatively reserved to support the enhancement.

The relax_request message is identical in format to the establish_request message, but its use is somewhat different. Since the source knows all of the nodal guarantees (having received them in the establish_confirm message), it generates the entire request message with all the per-node information, and then sends it directly to the destination host. The destination host relaxes requirements as in an ordinary channel establishment and returns a relax_accept message along the path.

Status messages are used to enable the source to obtain the current status of all the nodes participating in a connection, to facilitate debugging and perhaps for other purposes. A status_request message may be generated by the source and sent along the path to the destination. Each node adds its address, current state, and performance guarantees. The destination changes the type of the message to status_report and returns the message to the source. The format of the status_request and status_report messages is:

standard control packet header	
NodeAddr	
MaxDelayI	LateProbI
MaxJitterI	JitterProbI
NoDropProbI	State
BuffAllocI	
data about multiple additional nodes	

2.2.1.3. Fault Handling

Since RCAP cannot depend upon the data link layer for reliable packet transfer, RCAP includes subnet enhancement features. Reliable delivery of control packets between two nodes is obtained by use of a sequence number and acknowledgement for each packet, combined with timeout and retransmission. Sending nodes assign sequence numbers in increasing order. Upon receipt of a control packet, RCAP computes the checksum and sends an acknowledgment if it is valid. Otherwise, the acknowledgement is not sent, so timeout and retransmission occurs. If the packet is a duplicate (probably due to loss of an acknowledgement), it is discarded.

Multiple errors can also occur due to hardware or software failures, so the protocol includes features to handle these situations.

If a node crashes, real-time communication cannot continue over the same path, and corrective action must be taken. Routing software is responsible for monitoring the availability of nearby nodes and links, and upon discovery of a failure, it notifies the local RCAP module. Upon notification of a failure, RCAP terminates all real-time channels using that path, including those that are under establishment. (Future versions of RCAP may attempt to reroute the channels; for an initial implementation, an ungraceful close is a straightforward solution.)

A node initiates termination of a broken channel by sending a reset_request message to the source of the channel. The source immediately notifies RTIP to stop sending data across the channel, and RTIP returns an appropriate error code to the client at the time of the client's next ‘‘send’’ call. If the channel was fully established, the source is aware of all of the nodes along the path, due to the information in the establish_accept message. The source sends kill_request messages to all nodes along the path (since it can no longer rely on sending messages over the path). The nodes which are alive release resources allocated to the channel and respond with kill_confirm messages. If a node receives a kill_request message for an inactive channel, it also responds with a kill_confirm.

If a source receives a reset_request for a channel which has not been fully established, the situation is more complex, since the source does not know the entire path. The source attempts to notify as many nodes as possible, by sending a message along the path and asking the destination to send a message in the reverse direction. The source sends a close_request message along the path, causing nodes before the failure to release resources. The source also sends a dest_close_request to the destination, which propagates the request along the channel's reverse path, causing nodes after the failure to release resources. If establish_request has already reached the destination, this is sufficient. Otherwise, the destination is not able to send the message back, and some nodes are never reached. In this event, per-node timers will eventually notice a problem, as described at the end of this section. The four message types which support channel reset (reset_request, kill_request, kill_confirm, and dest_close_request) have no contents.

If a node has a lesser failure which results in loss of channel states, it will be unable to respond properly to either real-time data packets or control messages. Similarly, protocol failure can result in a node receiving a request which is inappropriate for the state of a channel. If a node receives a data packet with an inactive Local Channel Identifier, it returns an error message to the previous node, with an appropriate failure code. The previous node then initiates a channel termination. Upon receipt of an inappropriate control message, a node returns an error message to the node which sent the message, and appropriate action is taken. In an initial implementation, this action will be channel termination, since a failure has apparently occurred somewhere along the path.

The channel source maintains timers for all of its requests, and it is responsible for taking appropriate action if responses are not received. Appropriate lengths for these timeouts will need to be experimentally determined. In the case of idempotent requests such as relax_request and kill_request, the requests are retransmitted; after a limited number of retransmissions, RCAP gives up. In the case of an outstanding establish_request or enhance_request, lack of a response may be interpreted as a channel failure, and the channel closed. In the case of a close_request, kill_requests are disseminated as described above.

Timers are also maintained at the nodes, so that resources are not held indefinitely in the event of failure. For example, if the source crashes during channel establishment or during a conversation, the channel should be closed and resources released. If a node times out during channel establishment or observes a lack of traffic over an established channel for an extended period of time, it sends a source_status_request to the channel source. Upon receipt of such a request, the source checks on the state of the channel and responds with a source_status_report. An appropriate status is provided, indicating that the channel is idle, the channel is under establishment, or it is established. If the source indicates that the channel is idle, or if no response is received after a few attempts, the resources are released.

2.2.2. Data Transfer

Once a channel has been established, data is transmitted via the network-layer protocol, RTIP, which is based upon IP. RTIP has the additional features of fixed routing, deadline-based scheduling, and rate control.

RTIP packets are nearly identical to standard IP packets. Two additional header options are defined, the Local Channel Identifier and Jitter Correction Factor.

All real-time packets include a Local Channel Identifier. The inclusion of a non-zero Local Channel Identifier in a packet header alerts a node that the packet belongs to a real-time channel, and the identifier is used as an index into a data structure containing information about the channel's performance requirements and the next node. The packet is then processed in accordance with those requirements. A Local Channel Identifier of zero indicates that the packet is an express datagram, and it is prioritized behind packets belonging to real-time channels, but ahead of ordinary datagrams.

Real-time packets belonging to channels with jitter bounds also include a Jitter Correction Factor, an option which is used to convey information between intermediate nodes in order to limit delay jitter.

The option formats are:

11100001	00000100	Local Channel ID
Type=225	Length=4	

11100010	00000100	Jitter Correction Factor
Type=226	Length=4	

In the event that RTIP fragments a datagram, these options must be copied into the headers of the fragments. Each option appears at most once in a datagram. The options are class 3 (which is reserved for future use in the IP specification), numbers 1 and 2.

Non-real time packets are identical to IP packets; the additional header options are not used.

RTIP also includes mechanisms for rate control, both at the host and at the nodes. At the host, RTIP ensures that the transmitted traffic does not exceed the load specified by the client. A leaky bucket approach is appropriate for applications which are not concerned about preserving the temporal patterns of their data generation. The leaky bucket serves to smooth out variations in packet arrivals. For clients who wish to preserve temporal patterns, an approach which tracks transmissions and only intervenes when load specifications are exceeded is more appropriate. Such an approach is described in [Fe89], and this approach is also appropriate for rate control at the nodes. Rate control requires RTIP to retain additional state information for each channel.

2.3. Client Interface

A client interface to RCAP could be implemented in the 4.3 BSD UNIX† environment as described below. The client interface to RCAP is not strictly part of the protocol definition, but it is an essential part of a complete system design. A client interface is provided here to facilitate understanding of the operation of the protocol.

The proposed client interface is very similar to the ordinary Unix communications interface. An application creates a normal socket using the ''socket'' call. In then makes a ''connect'' call, specifying the destination of the communication. If the transport layer protocol supports connections, a transport layer connection is created. Otherwise, the address provided is used as the destination for all data packets.

The client then makes an EstablishRTChannel call, specifying its traffic characteristics and performance requirements. At this time, RCAP attempts to establish a network-layer connection with the desired parameters. It provides the obtained parameters, or it returns an error code if the requirements cannot be met.

† UNIX is a trademark of Bell Laboratories.

The client transmits data over the socket using normal system calls. If the client wants to change the parameters of the conversation at some point, the client makes a ModifyRTChannel call, specifying the new traffic characteristics and performance requirements. RCAP attempts to make the change, and it provides the obtained parameters or returns an error code if the new requirements cannot be met. When communication is complete, the client closes the real-time channel with the CloseRTChannel call, and then closes the socket.

A destination waiting to accept a real time connection creates a socket of the appropriate type and waits for input, just as in ordinary socket communication. If a connection-oriented transport protocol is being used, the destination uses the ‘‘listen’’ and ‘‘accept’’ system calls; if a datagram transport protocol is being used, the destination uses the normal system calls for reading from datagram sockets.

As an alternative to establishing a real-time channel with guaranteed performance and reserved resources, the client may use the EstablishExpress call to request that packets sent over the socket be prioritized ahead of ordinary datagrams, but behind real-time packets.

The call parameters and their types are summarized below:*

```
struct realTimeParameters {
        short MinInterarrival;
        short AveInterarrival;
        short Duration;
        short MaxSize;
        short DelayMax;
        short DelayProb;
        short JitterMax;
        short JitterProb;
        short NoDropProb;
};

error = EstablishRTChannel(socketNumber, desiredParameters, obtainedParameters);
int error, socketNumber; realTimeParameters *desiredParameters, *obtainedParameters;

error = ModifyRTChannel(socketNumber, desiredParameters, obtainedParameters);
int error, socketNumber; realTimeParameters *desiredParameters, *obtainedParameters;

error = CloseRTChannel(socketNumber);
int error, socketNumber;

error = EstablishExpress(socketNumber);
int error, socketNumber;
```

3. Transport Protocol Issues

Many of the standard transport layer functions are provided by real-time channels. Transport protocols normally provide error recovery, sequencing, and end-to-end flow control. When RCAP is used to set up a real-time channel at the network layer, a bound is placed on loss due to

* The notation is consistent with that used in the 4.3BSD Unix documentation; calls are displayed as they would be made by a user, and the types of the variables are documented below.

congestion, sequencing is achieved by use of a fixed-route channel, and the maximum transmission rate is agreed upon at the time of channel establishment. New transport protocols will be required to effectively support real-time communication in this environment.

Clients who require real-time performance guarantees can be grouped into three categories. Some real-time clients will also desire end-to-end flow control, acknowledgments, and retransmission, such as clients transferring bulk data within a time limit. Most real-time clients will not require these features, but desire a real-time connection close to that provided by RCAP. Some clients will wish to transfer express datagrams. Three types of transport protocols to address these needs are discussed in this section.

3.1. Real-time Connections with End-to-End Flow Control and Retransmission

Clients requiring absolutely complete and correct data delivery will continue to desire protocols which support acknowledgement and retransmission, since real-time channels cannot eliminate bit errors or packet loss due to failure. Some data receivers may not be able to guarantee processing of incoming data at any particular rate, and may wish to be able to continue to use end-to-end flow control.

Much of the research on transport protocols for high-speed networking has focussed upon designing protocols which provide reliable delivery and do not fail at high data rates. These protocols can be used with RCAP if real-time services are desired. The interaction of end-to-end flow control with network rate control is an area requiring further exploration.

Protocols which have reverse traffic for acknowledgment and control will require real-time channels in both the forward and reverse directions. Different bandwidths will be required for data and acknowledgments, but the delay bounds are likely to be the same.

3.2. Real-time Connections

Real-time connections may be supported with little functionality at the transport level. A protocol which provides end-to-end transport-layer connections (for the purpose of fixing the end points for conversations) and provides options for error-handling would be desirable. For initial experimentation, real-time communication may be accomplished by using UDP with RCAP on top of RTIP.

Flexible error-handling options are desirable since real-time requirements typically do not allow time for retransmission. RCAP allows the user to specify a maximum packet loss rate due to buffer overflow, but the possibility of transmission errors cannot be reduced to zero, due to line noise. A transport protocol should detect missing or damaged information (probably by using checksums and sequence numbers) and respond appropriately. When a loss occurs, some clients may want to know how much information was lost, and at what position in the transmission. Others may wish to receive data which contains errors; they may be able to make some use of it. Still others may want the transport protocol to provide error correction services. Such services could be implemented by transmission of duplicate information or by Forward Error Correction. Software implementations of Forward Error Correction do not meet real-time requirements, but hardware solutions are being explored [Mc90].

3.3. Express Datagrams

Express datagrams may be sent by using a dedicated Local Channel Identifier (zero) which results in prioritization ahead of ordinary datagrams, but behind packets belonging to real-time channels. Such datagrams will receive better service than ordinary datagrams, but specific performance guarantees will not be provided. This service may be provided by using UDP combined with the EstablishExpress call to RCAP, which specifies that a socket is an express socket.

RCAP sets the RTIP option for the socket so that a Local Channel Identifier of zero is automatically sent with each data packet.

While guaranteed-performance datagram service could be provided by multiplexing datagrams across a pre-established channel, such an approach would be likely to result in significant resource waste. The traffic patterns of real-time datagrams would be difficult to estimate, and maintenance of such a channel would reserve resources and thereby prevent establishment of other real-time channels. Further research is necessary to determine whether guaranteed-performance datagrams are required.

4. Related Work

A number of other research efforts are aimed at providing performance guarantees through the use of connections with fixed routing and resource reservation. These efforts are described below.*

4.1. Flow Protocol (FP)

Zhang [Zh89, Zh90] has proposed a rate-based network control system which provides guarantees for average throughput in a packet switched network. A ‘‘VirtualClock’’ mechanism is used at each node to monitor and regulate the average transmission rate over each connection. This approach far exceeds TCP/IP in terms of meeting diverse throughput requirements; simulations have demonstrated that competing traffic is properly regulated such that throughput guarantees are met.

Zhang’s research has focussed upon the design and simulation of the VirtualClock algorithm to be used by each node for scheduling data packet transmission and monitoring transmission rates. An overview of the essential features of the algorithm follows. During connection establishment, a client requests a desired average transmission rate, a minimum average transmission rate, and the interval over which that average will be maintained (the Average Interval). The network reserves resources at all nodes along the path and guarantees a throughput within the acceptable range, or, if it cannot provide the service, it rejects the request. (Clients also specify user expected delay, and the network provides the estimated delay. However, no guarantees about delay are made.) To monitor transmission rates, the node associates a variable called the VirtualClock with each connection. Upon arrival of the first data packet on a connection, the VirtualClock is set to the actual time. Throughout the duration of an Average Interval, upon each packet arrival, the VirtualClock is increased by the amount of time it would have taken for that amount of data to arrive, if the source were transmitting at its average rate. Thus, if a source transmits faster than its specified throughput, its VirtualClock value exceeds the actual time. If a source transmits slowly, the VirtualClock falls behind the actual time. At the end of an Average Interval, the VirtualClock is compared with the actual time. If it exceeds the real time by too much, control actions are taken. (The source is advised to slow down, and if the misbehavior continues, further traffic over that connection receives lowest priority in handling.) Otherwise, the VirtualClock is replaced with the current time, and a new Average Interval begins. An auxiliary VirtualClock (auxVC) is used to assign timestamps to packets to order transmission. Upon packet arrival, the auxVC is set to the larger of the real time and the current auxVC, the auxVC is increased in the same fashion as the VirtualClock, and the packet is stamped with the auxVC.

* A reader of original papers on these efforts will note differences in terminology. This area of research is new enough that terms have not been standardized. The terms ‘‘channel’’, ‘‘flow’’, ‘‘congram’’, and ‘‘stream’’ all refer to connections with fixed routing, guaranteed performance, and reserved resources. Use of the term ‘‘stream’’ has the appealing attribute that one can refer to ‘‘upstream’’ and ‘‘downstream’’ nodes. For consistency in discussion herein, the terms ‘‘channel’’ and ‘‘connection’’ are used.

Packets are ordered for transmission by the timestamps. The rationale for replacing the auxVC with the real time if the real time is larger is to prevent bursty flows from interfering with others. Without this mechanism, a sender could increase the priority of its packets by maintaining a period of silence and then sending a burst.

This work is distinguished from ours in that it does not provide delay or jitter bounds, or limits on packet loss due to buffer overflow. Under load, the VirtualClock mechanism schedules packets in order to regularize the data flows according to average throughput, rather than scheduling them to meet end-to-end deadlines. As a result, bursty flows experience higher variance in queueing delay, and lower throughput flows experience both higher queueing delay and higher variance in queueing delay. In contrast, our deadline-based scheduling approach decouples average throughput requirements from delay bounds; deadlines are based upon the desired end-to-end delay, rather than average throughput. Another distinction is that the Flow Protocol is intended to replace IP, rather than to be compatible with it.

A prototype of FP has been implemented by modification of IP under SunOS [Kh90]. A significant implementation issue was scheduling of data packet processing. This required modification of the network interface code, as well as the IP code. The standard mode of operation under Unix is for IP to process packets in a FIFO fashion. When a packet for IP arrives from a network device, a hardware interrupt causes it to be placed on a queue for IP, a software interrupt is raised, and IP is subsequently run to service the queue. IP processes all packets on the queue, in FIFO order, and places each packet at the tail of the appropriate outgoing queue. Similarly, when a process has a packet to send via IP, it makes a system call, and IP is run to service the packet. In the prototype implementation, Khale modified the network interface to support addition of packets at any point in the outgoing queue, rather than at the head or tail only. This enhancement also required modification of other modules which access the network queue directly, such as ARP.

4.2. Session Reservation Protocol (SRP)

The Session Reservation Protocol (SRP) has been proposed in combination with the DASH resource model [AnHeSc90]. SRP is a resource management protocol which was designed to be independent of transport protocols, to be compatible with IP, and to allow a host implementing SRP to benefit from its use even when communicating with hosts not supporting SRP. The SRP effort differs from ours primarily in terms of the load model and offered performance parameters, the method for allocating excess performance guarantees, the method of identifying real-time data packets, and the mechanism for communication among control entities.

The load model employed by SRP is based upon ‘‘linear bounded arrival processes.’’ A linear bounded arrival process is categorized by a maximum message size (M), maximum message rate (R), and a workahead limit (W). In any time interval of length T, the number of messages arriving may not exceed $W+TR$. On the basis of these factors, logical arrival times for messages may be determined. The logical delay of a message between two points is defined as the difference between the logical arrival times at the two points. Clients specify their traffic in these terms, and SRP uses this model to provide guarantees about logical delays, rather than actual delays. SRP but does not attempt to provide jitter bounds, but expects the destination application to compensate for jitter. SRP does not accept a loss bound from the client, but instead reserves sufficient buffers to ensure no packet loss due to buffer overflow.

Relaxation of resources in SRP is accomplished by use of an algorithm based on economics. On the forward pass of channel establishment, each node determines its cost function for delay bounds, and adds its portion to a cumulative function. This cost function indicates the savings which would be attained by relaxing the offered delay bound. This savings is not necessarily in terms of money; for example, it may be in terms of a metric reflecting system load. On the return pass, the excess delay is absorbed by those nodes with the steepest cost functions.

SRP was designed for compatibility with IP, in the sense that the IP header does not need to be changed to support SRP connections. (Implementation of SRP and changes to IP are necessary on gateways and hosts offering connections with real-time guarantees.) Since the IP header is unchanged, a channel identifier cannot be included in the header, and special techniques which are dependent upon the higher-level protocols are necessary to identify real-time packets. Upon receipt of a data packet, SRP examines the IP header to determine what the upper level protocol is, and then calls a procedure for that protocol to identify its connection and determine whether a real-time connection exists. For example, a TCP connection is identified by the source and destination addresses contained in the IP header and the ports contained in the TCP header. In order to avoid examination of all packets in this fashion, real-time packets have the low-delay bit set in the IP header. Therefore, packets which do not have the low-delay bit set may be treated as ordinary datagrams.

For communication among control entities, SRP uses the Sun Remote Procedure Call (RPC) protocol. This means that SRP, as currently defined, may only be used in environments supporting this protocol.

An SRP prototype has been implemented under SunOS [De90]. The implementation focussed primarily upon the control protocol. Deadline-based scheduling of data packet transmission was not implemented. Under low load, deadline guarantees are met. However, admission control is not effective; SRP allows numerous real-time channels to be established, resulting in excess delays. Additionally, interrupts resulting from actions such as moving the host's mouse result in disruption of packet delivery. Delgrossi identified CPU scheduling and network access scheduling as major issues in providing performance guarantees under Unix.

4.3. Multipoint Congram-Oriented High Performance Internet Protocol (MCHIP)

The Multipoint Congram-Oriented High Performance Internet Protocol (MCHIP) is being developed at Washington University as part of a very high-speed internetworking project [Pa90], [PaTu90]. This project is aimed at providing guaranteed performance for communications across subnets with diverse capabilities (including connectionless wide area networks), where the subnets are described by a set of performance parameters. Subnets may have designated resource servers, which keep track of the established channels and available resources. In addition to providing simplex channels between hosts, MCHIP will provide multicast capabilities. The project also involves development of high-speed gateways with all data delivery functions implemented in hardware (e.g. fragmentation, reassembly, rate control, resequencing), and all routing management and channel administration functions implemented in software on separate control processors. The gateways will process packets in FIFO order (with the exception of buffering due to resequencing). A prototype FDDI-ATM gateway is under development, but details of the methodology for channel management and resource reservation are not yet available.

4.4. Stream Protocol (ST-II)

The Stream Protocol, Version II, is an experimental Internet Protocol developed by a team from Bolt Beranek and Newman and the USC Information Sciences Institute [Top90]. The published information consists of the protocol specification; implementation issues, such as mechanisms for packet transmission scheduling and resource reservation, are not discussed. ST-II is intended to run alongside IP, having additional higher level protocols and routing facilities.

The ST-II protocol differs from ours in a number of areas, the most significant being multicast support and the approach towards resource reservation. ST-II supports multicast, which adds considerable complexity to the protocol. Mechanisms are provided for functions such as adding and removing destinations from a connection, negotiating unique connection identifiers among multiple nodes, and having multiple ‘‘branches’’ of a connection undergoing modification simultaneously. While the ST-II specification does not provide details regarding models for resource reservation, the messages exchanged during channel establishment and the parameters contained in those messages impact the possible methodologies. The traffic load and performance parameters differ from ours, and the algorithms for their use are not specified. For example, resource reservations are made in a one way pass along the path to the destination. The means by which this may be done is not clear, given that each node on the forward pass does not know how long the path is, and thus what its appropriate share of the load is. Further comments on the ST-II specification are provided in Appendix A.

Lessons learned from implementing a prototype of an earlier protocol, ST, contributed to the development of the ST-II specification.

4.5. Connection-Oriented IP Working Group

The Internet Activities Board has recognized the need for a connection-oriented service, and an Internet Engineering Task Force Connection-Oriented IP (IETF-COIP) Working Group has been created. The members of the group are engaged in a collaborative effort to design, implement, and experiment with a connection-oriented internetworking protocol and resource management algorithms. Current plans are to base the work upon ST-II, MCHIP, and FP. The project is in an early stage, and considerable opportunity exists for new participants to influence the final product.

5. Areas for Further Research

Further research areas include routing, multicast, and synchronization. A more robust channel administration mechanism would try alternate routes in the event of establishment denial. This would require the support of a new routing system which contained information regarding alternate paths. Support for multicast connections would facilitate such applications as conferencing among several sites. Synchronization of multiple channels would support applications requiring synchronized audio and video.

Current time-sharing operating systems may be inadequate for real-time communications, and alternatives should be explored. A multiprocessor combined with a real-time operating system would provide a solid foundation for making firm guarantees. Such systems are under development; Tokuda is developing a real-time operating system based on the Mach kernel which will support specific scheduling guarantees [Tok90].

6. Conclusion

This paper has described the central core of a methodology for supporting real-time communications, proposed a real-time delivery system with channel control and network protocols which may be used as a basis for experimentation, and discussed related efforts.

7. Acknowledgements

I would like to thank my advisor, Domenico Ferrari, as well as the members of the Tenet research group, for their ideas and support. Their work provided the foundation of new networking concepts which made this project possible, and their advice has been invaluable.

References

[AnHeSc90] D. Anderson, R. Herrtwich, and C. Schaefer, ''SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet,'' Rept. No. TR-90-006, International Computer Science Institute, Berkeley, February 1990.

[ChMe88] T. Chen and D. Messerschmitt, ''Integrated Voice/Data Switching,'' *IEEE Communications Magazine* 26(6), June 1988, pp. 16-26.

[De90] L. Delgrossi, personal communication, September 1990.

[Fe89] D. Ferrari, ''Real-Time Communication in Packet-Switching Wide Area Networks,'' Rept. No. TR-89-022, International Computer Science Institute, Berkeley, May 1989.

[Fe90] D. Ferrari, ''Client Requirements for Real-Time Communication Services,'' Rept. No. TR-90-007, International Computer Science Institute, Berkeley, March 1990; also, Proc. International Conference on Information Technology, October 1-5, Tokyo, Japan, and *IEEE Communications Magazine* 28(11), November 1990.

[FeVe89] D. Ferrari and D. Verma, ''A Scheme for Real-Time Channel Establishment in Wide-Area Networks,'' Rept. No. TR-89-036, International Computer Science Institute, Berkeley, May 1989; also, *IEEE J. on Selected Areas in Communications* 8(3), April 1990, pp. 368-379.

[FeVe90] D. Ferrari and D. Verma, ''Buffer Space Allocation for Real-Time Channels in a Packet-Switching Network,'' Rept. No. TR-90-022, International Computer Science Institute, Berkeley, June 1990.

[FrMa87] A. Fraser and W. Marshall, ''Data Transport in a Byte Stream Network,'' Technical Memorandum, AT&T Bell Laboratories, Murray Hill, NJ, April 28, 1987.

[Kh90] A. Khale, personal communication, October 1990.

[Mc90] A. McAuley, ''Reliable Broadband Communication Using a Burst Erasure Correcting Code,'' *Proceedings of SIGCOMM '90.*

[Pa90] G. Parulkar, ''The Next Generation of Internetworking,'' *ACM SIGCOMM Computer Commun. Rev.* 20(1), January 1990.

[PaTu90] G. Parulkar and J. Turner, ''Towards a Framework for High-Speed Communication in a Heterogeneous Networking Environment,'' *IEEE Network Magazine*, March 1990, pp. 19-27.

[Po80] J. Postel, ''User Datagram Protocol,'' RFC 768, August 1980.

[Po81a] J. Postel, editor, ''Internet Protocol; DARPA Internet Program Protocol Specification,'' RFC 791, September 1981.

[Po81b] J. Postel, editor, ''Transmission Control Protocol; DARPA Internet Program Protocol Specification,'' RFC 793, September 1981.

[Ro86] F. Ross, ''FDDI - A Tutorial,'' *IEEE Communications Magazine* 24(5), May 1986, pp. 10-17.

[Tok90] H. Tokuda, personal communication, November 1990; also ''The Impact of Priority Inversion on Continuous Media Applications,'' in ''First International Workshop on Network and Operating System Support for Digital Audio and Video,'' Rept. No. TR-90-062, International Computer Science Institute, Berkeley, November 1990.

[Top90] C. Topolcic, editor, ''Experimental Internet Stream Protocol, Version 2 (ST-II),'' RFC 1190, October 1990.

[WrTo90] D. Wright and M. To, ''Telecommunication Applicaitons of the 1990s and their Transport Requirements,'' *IEEE Network Magazine,* March 1990, pp. 34-40.

[Zh89] L. Zhang, ''A New Architecture for Packet Switching Network Protocols,'' PhD thesis, Massachusetts Institute of Technology, July 17, 1989.

[Zh90] L. Zhang, ''Virtual Clock: A New Traffic Control Algorithm for Packet Switching Networks,'' *Proceedings of SIGCOMM '90.*

Appendix A - Comments on ST-II Specification

We firmly support the methodology of fixed routing and resource reservation to support performance guarantees in packet-switching networks. The ST-II specification is a significant step towards this goal, but it is very complicated, and it should be expected to undergo modification as implementation issues are addressed.

While the specification does not provide details regarding models for resource reservation, the messages exchanged during channel establishment and the parameters contained in those messages impact the possible methodologies. The mechanism by which resource reservation is accomplished requires further investigation. According to the specification, nodes are expected to reserve appropriate resources and offer appropriate guarantees upon receipt of a CONNECT message, which contains global performance requirements and previous node guarantees. However, it is not clear how a node knows what its requirements are. For example, if a maximum delay is specified, and the length of the path is unknown, the portion of the delay which may be absorbed by that node is also unknown. An alternate approach is for nodes to offer the best possible performance, and for the destination to appropriately relax requirements in the ACCEPT message.

The specification states that excess resource allocations are not released as a result of relaxed requirements in the ACCEPT message, and that sources are expected to send explicit CHANGE messages if they do not wish to be charged for excess allocations. Instead, excess allocations should be released. Otherwise, resources are reserved but unused, resulting in wasted resources due to rejection of other streams which would otherwise be accepted. Additionally, it is improper to charge a client for services which are not provided.

It is not clear how the Flow Specification may be used to provide the necessary guarantees. For example, the sender of a CONNECT request specifies an end-to-end delay bound. Each node provides information regarding mean delay and delay variance. However, this information is insufficient to determine whether the end-to-end delay bound will be met. We are interested in seeing algorithms by which the information in the Flow Specification is intended to be used, and we recommend that the approach which we have proposed for providing bounds on delay, loss, and jitter be considered as an alternative.

Problems caused by excessively large messages could be alleviated by supporting fragmentation in ST. If this were done, large control messages would not have to be subdivided by the control protocol, and data packets would not have to be dropped if a connection were rerouted over a path with a smaller Maximum Transmission Unit (MTU).

The methodology of limiting stream packet sizes to the smallest MTU has impacts which are not addressed. First, if a node lowers the maximum packet size during stream setup, additional resources will have to be reserved at upstream nodes to make the same guarantees; they will have to handle packets with greater frequency, and the required bandwidth will increase, due to additional header overhead. These factors may change the service guarantees which they can offer. A more complex approach to resource reservation is necessary to handle this eventuality. Alternatively, nodes using small-MTU subnets could fragment packets and reassemble them at the other end, including the time necessary for fragmentation and reassembly in their delay guarantees.

The interaction of ST with the source and destination applications is not clear. It is not clear why the destination application should be given the opportunity to modify stream parameters. The ST module at the destination has adequate information to determine whether requirements are being met, to reduce node requirements if excess guarantees have been made, and to accept or reject the stream. (It seems reasonable to give the destination the ability to accept or reject, but not to change the parameters.) Additionally, the source does not need to be consulted regarding the acceptability of the actual parameters; either they meet the requirements or they don't. The source should be advised of the parameters, since they will fall somewhere between the desirable and the minimal.

The specification states that the stream name is normally used in control messages for stream identification. However, Virtual Link Identifiers (VLIs) are often used, apparently in order to provide faster access to information during control communication. This creates additional implementation complexity, and it is not clear that the value is worth the complexity.

A clearer distinction between routing functions and stream control functions would be beneficial. Normally, routing systems monitor the state of adjacent gateways, and take appropriate actions when they become unreachable. The use of HELLO messages in ST is redundant if proper routing facilities are implemented.