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.

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

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