The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences[1]

Anindo Banerjea, Domenico Ferrari, Bruce A. Mah, Mark Moran
{banerjea,ferrari,bmah,moran}@CS.Berkeley.EDU
The Tenet Group
University of California at Berkeley
and
The International Computer Science Institute

Dinesh C. Verma
verma@watson.IBM.COM
IBM T J Watson Research Center

Hui Zhang
hzhang@CS.CMU.EDU
School of Computer Science
Carnegie Mellon University

TR-94-059

November 1994

ABSTRACT

Many future applications will require guarantees on network performance, such as bounds on throughput, delay, delay jitter, and reliability. To address this need, the Tenet Group at the University of California at Berkeley has designed, simulated, and implemented a suite of network protocols to support *real-time channels* (network connections with mathematically provable performance guarantees). The protocols, which constitute the prototype Tenet Real-Time Protocol Suite (*Suite 1*), run on a packet-switching internetwork, and can coexist with the popular Internet Suite. We rely on the use of connection-oriented communication, admission control, and channel rate control.

This protocol suite is the first complete set of communication protocols that can transfer real-time streams with guaranteed quality in packet-switching internetworks. Our initial development was done on a local-area FDDI network. We have since installed our protocols on the experimental wide-area internetwork of Project Sequoia 2000, where they have been running for several months. We have performed a number of experiments and demonstrations in this environment using continuous-media loads (particularly video). Our results show that our approach is both feasible and practical to build, and that it can successfully provide performance guarantees to real-time applications. This paper describes the design and implementation of the suite, the experiments we performed, and selected results, along with the lessons we learned.

---

1. This research was supported by the National Science Foundation and the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives, by AT&T Bell Laboratories, Digital Equipment Corporation, Hitachi, Ltd., Hitachi America, Ltd., Pacific Bell, the University of California under several MICRO grants, and the International Computer Science Institute. Bruce A. Mah and Mark Moran were supported by NSF Graduate Fellowships. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing official policies, either expressed or implied, of the U.S. Government or any of the sponsoring organizations.

1 Introduction

This paper describes the Tenet Real-Time Protocol Suite, a set of network protocols providing guaranteed-performance communication in heterogeneous packet-switching internetworks. This protocol suite is comprised of the first complete set of protocols capable of transferring real-time streams in packet-switching internetworks with guaranteed quality. The remainder of this section provides some background for our work. Section 2 outlines the overall architecture of the Tenet Suite. Section 3 and Section 4 respectively present the design and implementation of the individual protocols. In Section 5, we describe our experiments and measurements of the protocols. Section 6 provides a list of the lessons that we learned in the process of designing, implementing, and experimenting with our protocols, as well as some of the questions that remain unanswered. In Section 7, we summarize and describe future work.

1.1 A Brief History of the Tenet Suite

The Tenet Real-Time Protocol Suite was born out of pure curiosity. In 1987, one of the authors (Ferrari) was investigating the operating system support necessary for multimedia applications, and was repeatedly led to making the assumption that future networks would offer performance guarantees. While this assumption is true for circuit-switching networks, he felt that future integrated-services networks would have to be based, for economical reasons, on packet switching. This conjecture raised the question of whether a congestion-prone packet-switching network could economically and efficiently offer guaranteed throughput, delay, delay jitter, and reliability bounds.

Ferrari set out to build a scheme (a set of algorithms) that would offer a network's real-time clients the desired guarantees by providing *real-time channels*[2]. The basic ideas underlying the scheme were the real-time channel paradigm itself, admission control, connection-oriented communication, channel rate control, and deadline-based scheduling for real-time packets [1].

Later, he was joined by another of the authors (Verma), and, in 1989, the Tenet Group was founded with the primary objective of studying real-time communication protocols. Shortly thereafter, the remaining authors joined the group. The main activity of the group became, in Fall 1990 and Spring 1991, the design of a protocol suite embodying the Tenet scheme. Several simplifications were made to facilitate the development of the first suite: the most important simplification being the decision to provide only unicast real-time channels.

By Summer 1991, the suite had been specified in all its details, enriched with a distributed jitter control mechanism, extended to internetworks and non-deadline-based scheduling disciplines, and extensively simulated. We subsequently implemented, debugged, and tested the protocols on a local-area FDDI network. We then installed the Tenet protocols on the experimental wide-area network of Project Sequoia 2000 [2], where they have been running for several months, and performed various experiments and demonstrations with continuous media loads (especially video). The Tenet Suite is currently being ported to other testbeds, including XUNET 2 (a coast-to-coast ATM backbone connecting FDDI rings) and XUNET 3 West (a 1.5 kilometer serial HIPPI testbed); both XUNET 2 and XUNET 3 West are part of BLANCA, one of the five testbeds in the Gigabit Testbed Initiative [3]. Finally, a second generation protocol suite (*Suite 2* [4]) has been designed to support multi-party real-time communication (including multicast) and is currently being implemented.

1.2 Related Work

The literature on real-time communication contains many proposals for satisfying the requirements of continuous-media and other real-time applications. Most schemes, however, have not yet been used in designing real-time protocols. At best, paper designs of such protocols have been provided. So far, there have been very few implementations in this increasingly important area of networking research.

ST-II is a connection-oriented internetwork protocol [5] that has been implemented in several versions (e.g. [6], [7]). Although the protocol specification allows implementors to include a *FlowSpec* that indicates resource requirements, neither the contents of the FlowSpec nor algorithms for real-time admission control have been stipulated. To the best of our knowledge, the implementation by Delgrossi et al. [7] is the only one providing mathematically provable guar-

2. In this paper, we define a *real-time channel* to be a simplex connection that provides mathematically provable performance guarantees on data delivery.

antees. It uses admission control algorithms based on the Tenet scheme over Token Ring and FDDI networks [8]. However, neither the protocol specification nor this implementation provide a mechanism for performing and encapsulating reservations on multi-hop subnetworks (e.g. ATM networks).

Most recently, RSVP has been designed as a protocol for exchanging reservation messages [9]. It serves a role similar to, and shares many features with, our channel setup protocol. The major difference is that RSVP reservations are considered to be “soft state”, which must be periodically refreshed. This approach allows RSVP connections to adapt to network load and outages, but penalizes connections during normal operation, because it introduces additional protocol overhead and allows disruptions in real-time service due to underlying route changes. As the implementation of RSVP and associated admission control algorithms was in progress during this study, we could not compare its performance and utility to those of the Tenet Suite.

Some real-time protocols have been implemented for specific sub-networks (e.g. [10], [11]). However, to the best of our knowledge, these implementations are not applicable to heterogeneous internetworks. Standard signalling protocols for ATM networks have well-defined traffic and performance specifications and are expected to be capable of providing mathematically provable performance guarantees [12]. However, the corresponding admission control algorithms have not yet been specified.

2 The Architecture of the Tenet Suite

The Tenet scheme for real-time communication is based on some principles that dictated, or suggested, some of the architectural characteristics of the associated protocol suite [13]. First, all layers in a network’s architecture must be able to provide performance guarantees in order for any guarantees to be available to the network’s clients. If a layer is incapable of guaranteeing some performance bound, the layers above it cannot guarantee that bound either. For example, no real-time service offering mathematically provable delay guarantees (including probabilistic ones) can be built on top of the IEEE 802.3 (Ethernet) datalink layer protocol. Thus, a real-time communication service can only be built if the datalink layer is able to provide guaranteed-performance services to the network layer (as is the case of synchronous FDDI, ATM with a suitable signaling protocol and admission tests, 100VG-AnyLAN, and so on). Performance bounds can then be offered to applications by implementing real-time protocols at the network and transport layers. The Tenet Suite includes a network (or internetwork) layer protocol, the Real-Time Internetwork Protocol (RTIP), and two transport layer protocols, the Real-Time Message Transport Protocol (RMTP) and the Continuous Media Transport Protocol (CMTP).

In the Tenet scheme, real-time channels are set up in an establishment phase that precedes data transfer. An establishment message is issued from the source of the channel and travels to the destination, causing admission tests to be run in each node along the path. All forward-trip tests must be successful and an acknowledgment must reach the source for data transfer to begin. This separation between setup and transfer suggests that establishment (and teardown) be done by a control protocol distinct from the data delivery stack. This solution is not universally preferred (as in the cases of ST-II [5] and SRP [14]), but it has the advantage of allowing us to develop the control and data delivery protocols separately, in a more easily manageable, testable, maintainable, and portable fashion[3]. Control functions in the Tenet Suite are provided by the Real-Time Channel Administration Protocol (RCAP) [15] [16].

An important principle of the Tenet approach is that all real-time applications have requirements that can be expressed in terms of general performance or reliability bounds. As a consequence, there is no need for application-specific protocols (such as a video protocol, an audio protocol, and so on) at the network or transport layer, as one protocol at each of the layers will be sufficient. The Tenet Suite’s two transport protocols (RMTP and CMTP) differ from each other in their interfaces and services, rather than in the types of information media they can or should carry. CMTP is more oriented towards periodic traffic, and is time-driven instead of being based on the send-receive paradigm of RMTP.

The Tenet Suite is expected to offer a real-time service in an integrated-services network. Hence, we designed the Tenet protocols to coexist with the Internet protocols. Clients can send their non-real-time traffic using TCP or UDP

3. It is possible to use the data delivery protocols without the control protocol, and to get guaranteed performance from them whenever the load is known never to cause saturation. It is also possible to use the control protocols to limit admission (i.e. to avoid congestion) with non-Tenet delivery protocols, as long as sources are rate controlled.

and their real-time traffic using the Tenet protocols (after the appropriate connection setup). For this to be possible, code had to be inserted between IP and the network drivers that would properly schedule departing packets (giving priority to real-time packets), and demultiplex each arriving packet to the appropriate network protocol.

Finally, control must be exerted over data transfers to detect and recover from failures. Since this kind of control is not directly related to the connection management performed by RCAP, another protocol, the Real-Time Control Message Protocol (RTCMP), was introduced in the design of the suite, though it was not implemented due to our scarcity of programmer-power.

The suite that resulted from these considerations is shown in Figure 1. At the time of this writing, CMTP has been almost entirely implemented, but has not been tested, integrated with the other protocols, and experimented with. Its design is described in [17].

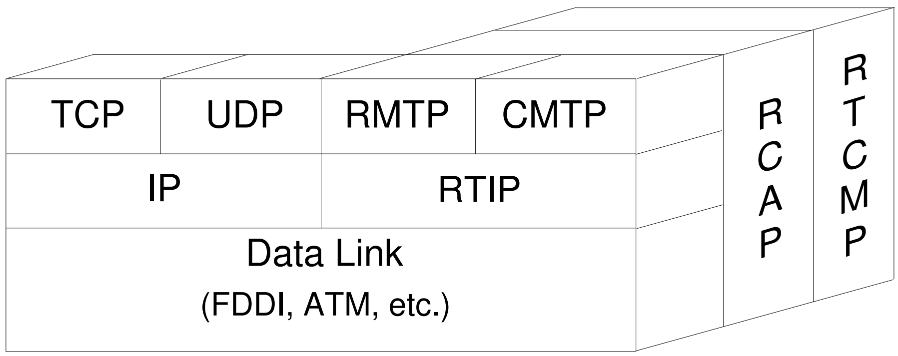

FIGURE 1. The Tenet Real-Time Protocol Suite. Also shown are the corresponding Internet protocols.

3 Design of the Tenet Protocols

This section describes the design of the various protocols in the Tenet Suite. We describe the channel administration protocol (RCAP), the network-layer protocol (RTIP) and one of the transport protocols (RMTP).

3.1 RCAP

Signalling and control services are provided in the Tenet Suite by the Real-Time Channel Administration Protocol (RCAP). The main functions of RCAP are the setup and teardown of real-time channels; it also supports status inquiries about established channels.

RCAP defines a set of messages to be passed between control entities along the path taken by a channel. Each message has an implicit direction of travel with respect to a given real-time channel, either "upstream" (towards the sender) or "downstream" (towards the receiver). RCAP assumes a reliable message delivery system; it does not provide explicitly for error recovery in the face of lost or corrupted control messages. The RCAP messages are shown in Table 1.

Channel establishment is performed in a single round trip. In the forward pass, an `establish_request` message is sent, hop by hop, from the source to the destination of the channel. Each RCAP entity maintains a routing table to compute the next hop of channel establishment. At each node along the way where RTIP runs[4], there is a local RCAP entity, which performs admission control tests [1] [19]. If the channel can be supported at this node, the necessary resources are tentatively allocated to the channel and the `establish_request` message (with local parameters appended) is forwarded to the next node. If any node determines that it cannot support the channel's performance requirements, it sends an `establish_denied` message back to the source; this message causes all the channel's tentatively reserved resources to be released.

4. RTIP and RCAP need not run within subnetworks that can themselves provide performance guarantees [18].

Message Name	Direction	Description
establish_request	Downstream	Request to establish a new channel, causing admission tests to be run at each hop and resources to be tentatively allocated for the channel.
establish_accept	Upstream	Indicates acceptance of a new channel, causing relaxation of resources and confirmation of resource allocation.
establish_denied	Upstream	Indicates that a channel establishment request was rejected, either inside the network or at the destination. Causes resources allocated to a channel to be released.
status_request	Downstream	Requests the status of a channel at each node.
status_report	Upstream	Returns data collected by a status_request message.
close_request_forward	Downstream	Message from source application to close an existing channel.
close_request_reverse	Upstream	Message from destination application to close an existing channel.

TABLE 1. RCAP messages.

Once the establish_request message reaches the destination, the destination application makes a final accept-or-reject decision on the channel. If it accepts the channel, the reverse pass of establishment begins. An establish_accept message is sent hop by hop back from the destination to the source, backtracking along the path. At each node on the path, the local RCAP entity may choose to relax the allocation of any resources that were over-reserved on the forward pass. The final resource allocation is computed, the RTIP data delivery agent is informed of the new channel, and the establish_accept message is passed to the next upstream node. When this message reaches the source application, data transfer on the channel can begin.

The client, when invoking RCAP, must specify its performance requirements (the so-called "quality of service" or QoS it needs) and a worst-case description of the traffic it will transmit over the new channel [1] [19]. The QoS parameters of the RCAP interface are listed in Table 2, the traffic parameters in Table 3. The parameters in these tables are part of the interface offered at the transport layer to the layer above. Note that the traffic parameters can also be viewed as QoS parameters, since they define the lower bounds on instantaneous and average throughput the network is being requested to provide. A positive answer returned by RCAP to the application may be interpreted as a contract: the network guarantees the performance bounds requested by the application, provided that the application obeys its traffic description at all times and that there are no failures in the network during the new channel's lifetime [20] [21]. This contract is valid until the channel is torn down.

Symbol	Description
D_{max}	Upper bound on end-to-end message delay
Z_{min}	Lower bound on probability of timely delivery
J_{max}	Upper bound on delay jitter (optional)
W_{min}	Lower bound on probability of no loss due to buffer overflow

TABLE 2. QoS parameters.

At any time, the sending application can request the status of the channel at each node. This status includes the channel state (e.g. partially established, fully established), the address of each node on the path, and the local resource reservations made. To gather this information, a status_request message is sent by the source of a channel. At each node, the local RCAP entity appends a record with the channel status and forwards the message to the destination. The destination converts the type of the status_request message to status_report; this message is

Symbol	Description
X_{min}	Minimum inter-message time
X_{ave}	Minimum average inter-message time
I	Averaging interval
S_{max}	Maximum message size

TABLE 3. Traffic parameters.

then transmitted hop by hop, unchanged, back to the sending application. Finally, either the source or the destination of a given real-time channel can request that the channel be closed. The `close_request_forward` and `close_request_reverse` messages, identical except for the "direction" of the messages, are used for this purpose. They are transmitted hop by hop along the path of a channel; each local RCAP entity releases all the resources allocated to the channel.

RCAP is designed to control real-time channels in a heterogeneous internetwork. In such an environment, different networks may have different scheduling policies or different resource models, which will require different admission control tests. This heterogeneity requires different parameters to be passed between nodes during channel establishment. However, it is desirable to hide these differences from the end-to-end channel establishment process. RCAP takes a hierarchical view of channel establishment in which the links and nodes in a subnetwork are abstracted into a single "logical link" in the internetwork. Such an approach allows RCAP to utilize the characteristics peculiar to an individual network in order to provide guarantees, yet hide the underlying details of that network whenever possible [15] [18].

3.2 RTIP

The Real-Time Internet Protocol (RTIP) is the network layer of the Tenet Suite. Its main function is to deliver packets in such a way as to meet the real-time requirements of the corresponding channel.

3.2.1 RTIP Overview

In contrast to the IP datagram service, RTIP is connection-oriented. Data forwarding of RTIP is unreliable since packets may be corrupted or lost due to buffer overflow (if $W_{min} < 1$, see Table 2). RTIP also performs rate control, jitter control, and scheduling based on the quality of service parameters of each connection. Since all packets on an RTIP connection follow the same path, packets are delivered to the destination in the same order in which they were transmitted by the source. Therefore, RTIP provides an unreliable, simplex, guaranteed-performance, in-order packet delivery service.

The RTIP header, shown in Figure 2, is fixed-size and allows rapid processing at each node. In order to inter-operate with IP, the first four bits of the RTIP header identify a packet as an RTIP packet. The other fields in the packet consist of a 16-bit local identifier, a 16-bit packet length field, and a 16-bit packet sequence number. In addition, RTIP carries a 32-bit timestamp, which indicates the time the packet was received by the RTIP module at the sending host. RTIP provides a checksum for its header, but does not provide a checksum for user data, since it does not guarantee data integrity.

<table>
<tr><th>0   3</th><th>7</th><th>15</th><th>23   31</th></tr>
<tr><td>RTIP</td><td>Version</td><td>unused</td><td>Local ID</td></tr>
<tr><td colspan="3">Packet Length</td><td>Packet Sequence Number</td></tr>
<tr><td colspan="4">Timestamp</td></tr>
<tr><td colspan="3">reserved</td><td>Header Checksum</td></tr>
</table>

FIGURE 2. RTIP packet header.

As an RTIP packet is received at a node, the only field in the packet that changes is the 16-bit Local ID; it is replaced by the Local ID for the channel used by the next node along the path. Changes in the RTIP header checksum can be pre-computed at connection establishment time to streamline packet processing.

The timestamp field in the RTIP header is used to maintain statistics about packet delays. The timestamp is initialized by the sending host to its local time. RTIP can use this timestamp to collect statistics about packets on a connection by measuring time relative to the timestamp of the first packet. Thus, meaningful statistics can be gathered even if the clocks in the network are not synchronized, as long as the clock rates in the different machines are roughly the same.

3.2.2 RTIP Packet Processing

The RTIP entities in each node ensure that packets are delivered to meet their performance requirements. When a connection is established by RCAP, RTIP is informed about the maximum delay d to be experienced by a packet at the node. Assuming that packets obey the worst case traffic description specified at connection establishment time, RTIP ensures that a packet arriving at a node at time T is transmitted from the node no later than $T + d$. In addition to these, RCAP also informs RTIP about the amount of buffer space to be allocated to the new connection, and about the minimum and average inter-packet intervals and the averaging interval for the channel (X_{min}, X_{ave}, and I)[5].

In order to ensure a connection's performance guarantees, each RTIP entity contains a rate control module and a scheduling module. The rate control module monitors the traffic on each connection and shapes it according to the traffic specification of that connection. In addition, the rate control module ensures that a connection's delay jitter requirements are satisfied. The scheduling module gives priority to packets with earlier delay bounds and ensures that all deadlines are met.

The rate control module marks each packet as ineligible when it arrives. If a packet was expected no earlier than time T_e, but arrives at an earlier time, then the packet becomes eligible for transmission at time T_e. A detailed explanation of the rate control module and the method to compute eligibility times can be found in [21]. After it becomes eligible, the packet is transferred to the RTIP scheduling module. The scheduling module is responsible for transmitting a packet within the interval d after it receives the packet. Any service discipline that can provide such a guarantee can be used with RTIP.

The RTIP entities also record statistics about packets arriving and departing on each connection.

3.3 RMTP

RMTP provides a message-based abstraction on top of RTIP. An instance of RMTP is needed at both endpoints of a real-time channel. RMTP was designed as a light-weight transport protocol, and resides at the same layer in the protocol stack as TCP. Unlike TCP, the service provided by RMTP is unreliable. Acknowledgments and retransmissions, the usual cure for loss and corruption, were deemed useless for those real-time applications with stringent delay bounds and for most continuous-media applications; thus, we decided that reliability should be provided to those applications that need and can afford it by a higher level protocol.

RMTP does not implement any kind of flow or congestion control scheme. It relies on RCAP to manage connections so that there is no congestion, and on RTIP to provide rate-based flow control through the RTIP rate control module. Furthermore, RMTP does not address the issue of packet reordering. Since all packets in a RTIP connection follow the same route, RMTP expects packets to be delivered to it in order.

The main service provided by RTMP is fragmentation and reassembly. Thus, the sending RMTP module can accept messages larger than the maximum RTIP packet size and break them into several RTIP packets. Each packet carries a message sequence number (in the RMTP header) and a packet sequence number (in the RTIP header). The receiving

5. In this discussion, we assume for simplicity that the traffic and QoS parameters of the network layer are so close to those of the transport layer (see Tables 2 and 3) that we can consider them identical and use the same symbols for both. This amounts to assuming (among other things) that no fragmentation of messages occurs in the transport layer.

RMTP module uses the message and packet sequence numbers to reassemble the received message. It also computes the checksum of the message header to verify the correctness of the assembled message.

The RMTP header is shown in Figure 3. In addition to fragmentation and reassembly, the header supports options to checksum user data and to collect end-to-end statistics about packet delays. The latter end-to-end statistics are computed on the basis of the timestamp carried in the RMTP message header.

0 **7** **15**	**23**	**31**
Timestamp		
Message length	Message Seq. #	Flags
Checksum		

FIGURE 3. RMTP message header.

4 Implementation of the Tenet Protocols

This section describes the first implementation of each of the currently-operational protocols in the Tenet Suite (RCAP, RTIP, and RMTP).

4.1 RCAP

The current RCAP implementation is divided into two parts: a library linked into each client application and a daemon process that runs on each node, independent of the applications [16]. These components, and the relationships between them, are shown in Figure 4. This implementation is extremely portable, compiling without modification on nearly all UNIX-like operation systems.

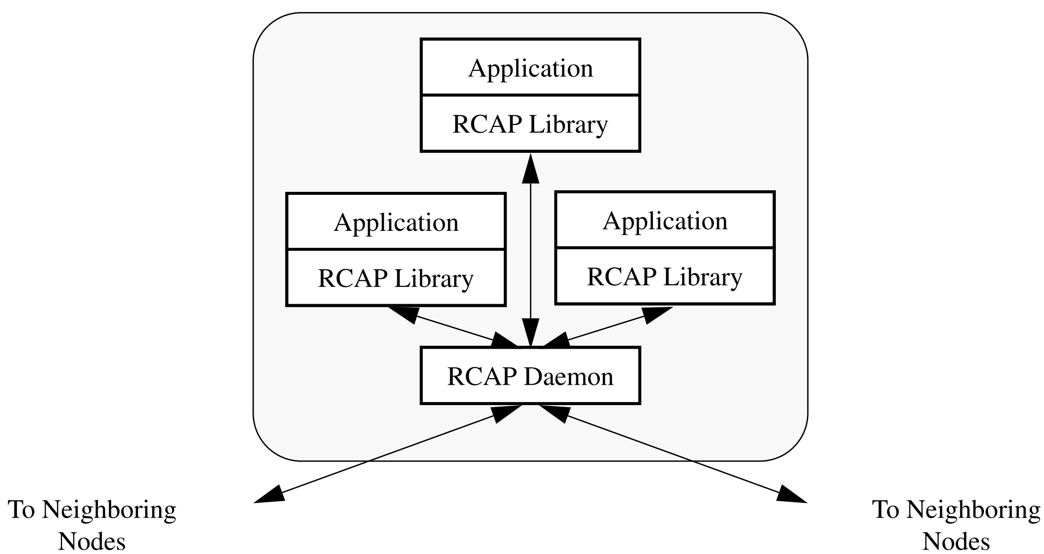

FIGURE 4. The RCAP library and daemon, within a single network node. Arrows represent RCAP control connections.

4.1.1 The RCAP Library

The RCAP library implements the network control API seen by the application writer. It exports a set of procedures that can be called within a user program to request various channel management functions. These procedures are listed in Table 4. A C language header file provides the function prototypes and data structure definitions needed by applications. In all current implementations, the RCAP library is a standard UNIX library that is linked to the application program.

Library Function	Description
`RcapEstablishRequest()`	Request to establish a new channel. Inputs are the traffic characteristics, performance requirements, and addressing information. Output is a channel identifier to be associated with a socket (if successful) or an error code (if denied). Analogous to BSD `connect()`.
`RcapRegister()`	Receiver uses this call to indicate its willingness to accept new connections. Analogous to BSD `listen()`.
`RcapReceiveRequest()`	Called by a receiving application to obtain an incoming channel establishment request.
`RcapEstablishReturn()`	Used by the receiver to indicate the final acceptance or rejection of a channel. The combination of `RcapReceiveRequest()` and `RcapEstablishReturn()` is similar to the BSD `accept()` system call.
`RcapUnregister()`	Used by the receiver to stop receiving new connections; does not affect existing real-time channels.
`RcapStatusRequest()`	Called by the source application to obtain the status of the channel at each of the intermediate nodes.
`RcapCloseRequest()`	Used by either sender or receiver to close a given real-time channel and release its resources. Similar to BSD `close()`.

TABLE 4. RCAP library exported procedures.

We made a conscious decision to keep very little state information in the RCAP library. We assumed that since each instance of the library resides in the address space of an application process, any such information is susceptible to corruption from the user program. The RCAP library is essentially a set of specialized remote procedure call (RPC) client stubs.

4.1.2 The RCAP Daemon

Information about the state of resources in the network is kept by an RCAP daemon process on each node. This process manages the resources (such as link bandwidth) associated with the local node and responds to requests for channel establishment and teardown. It maintains the state of each RTIP channel throughout the channel's lifetime.

Each daemon communicates with neighboring daemons and with local processes using control messages sent over TCP/IP connections. These connections play a role similar to that of the dedicated signalling channels found in ATM networks.

4.2 RMTP/RTIP

RMTP and RTIP have been implemented on various software and hardware platforms. One of the authors (Zhang) wrote the original implementations of RMTP and RTIP for HP/UX on HP 9000/700 workstations and for Ultrix on DECstation 5000 workstations. Tom Fisher and Hoofar Razavi ported these implementations to SunOS on SPARCstations. Keith Sklower built versions for IRIX on SGI workstations and BSDI BSD/386 on 80486-based PCs. All the software platforms are UNIX-like operating systems [22] with networking software derived from BSD UNIX [23] [24].

The software structure of the RMTP and RTIP implementations is shown in Figure 5. While RCAP is implemented in user space, RMTP and RTIP are implemented in the kernel and co-exist with TCP, UDP, and IP. A new type of socket, with a protocol type of `IPPROTO_RMTP`, is used by applications for sending and receiving data. RCAP controls the local establishment and teardown of channels through socket options applied to a special RTIP control socket.

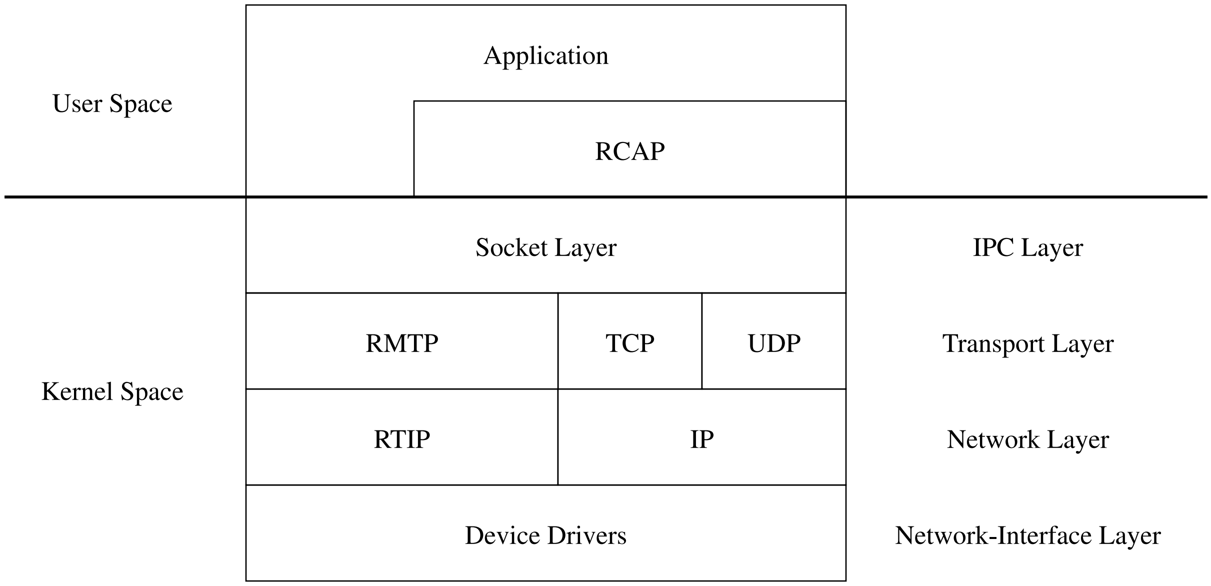

FIGURE 5. Software structure of RMTP/RTIP. The Internet protocols co-exist with the Tenet Suite.

Protocols previously implemented in BSD UNIX provide only best-effort service. In most implementations, there is a buffer pool shared among all the connections, and the queueing discipline used is First-Come-First-Served. To provide performance guarantees, we implemented priority queueing mechanisms, as well as per-connection buffer management and rate control.

4.2.1 Buffer Management

Buffer management in BSD UNIX is based on a system-wide central pool of free buffers called *mbufs*. To implement per-connection buffer management, where buffers are pre-allocated during channel establishment, we decided to use a special-purpose mbuf designed for the Network File System (NFS), which has associated with it a function pointer and a parameter. When the mbuf is released, the system does not return it to the central buffer pool, but calls the function with the parameter as an argument. For the purpose of RTIP buffer management, the function points to an RTIP buffer management function, and the parameter holds the Local Channel Identifier of the connection. The buffer management function, when called, returns the private buffer associated with the mbuf to the connection's buffer pool.

4.2.2 Service Disciplines

Some of the most important functions of RTIP are rate control, jitter control, and packet scheduling. These functions can be realized using rate-based service disciplines [25]. We have implemented a number of service disciplines in the prototype suite: Delay-Earliest-Due-Date [1], Jitter-Earliest-Due-Date [20] and Rate-Controlled Static Priority [26]. For the latter two, we implemented both rate-jitter controlling regulators and delay-jitter controlling regulators [25]. The implementation is modular, so that other types of regulators and schedulers can be easily implemented without affecting the rest of the system.

The operation of a rate-controlled service discipline involves three tasks: calculating the eligibility time of each packet, holding the packet if necessary, and enqueueing and dequeueing packets at the scheduler. The eligibility time of each packet can be calculated using the formulae defined in [25] [27].

Packet holding is implemented differently in gateways (or routers) than in the hosts. In a host, where packets are sent from some processes, holding packets can be implemented by putting the transmitting process into sleep. In a gateway, where packets are to be forwarded, there is no context or process associated with packets, and thus the sleeping mechanism cannot be used. Instead, the UNIX `timeout` procedure is used to delay a packet that arrives at a gateway earlier than its eligibility time.

Finally, the implementation of the enqueue and dequeue operations depends on the scheduling policy. All the measurement experiments described in the next section were performed using the EDD scheduler, which serves packets in order of their transmission deadlines.

5 Experiments with the Tenet Suite

The worst-case analysis used to derive the guarantees provided by the Tenet protocols is based on idealized models of the network components. However, the actual timing behavior of a workstation is complicated by considerations of process scheduling, interrupt handling, and CPU load. In addition, the calculations carried out by the admission control schemes depend on an accurate knowledge of the packet service times and transmission times, which depend on the implementation of the protocols and must be measured. Therefore, experimental measurement of our protocols is crucial, both to accurately calculate the parameters needed by the admission control tests and to verify the correct performance of the resulting system.

5.1 Preliminary Measurement of RMTP/RTIP Performance

To perform some preliminary measurements of RMTP/RTIP, we set up the network shown in Figure 6. The three workstations were connected by an FDDI ring, and routing tables were set up to form the logical topology shown in Figure 7. While `theorem` and `faith` had one FDDI interface each, `lemma` had two FDDI interfaces to allow it to act as a router in the logical topology. In some experiments, cross-traffic load was created by sending packets to a non-existent machine called `fake`.

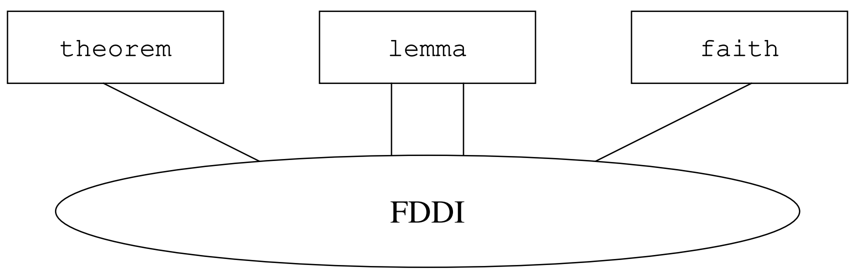

FIGURE 6. Physical topology of the local testbed. `theorem` and `lemma` are DECstation 5000/125s and `faith` is a DECstation 5000/240. Note that `lemma` has two FDDI interfaces. All machines run the Ultrix 4.2A operating system.

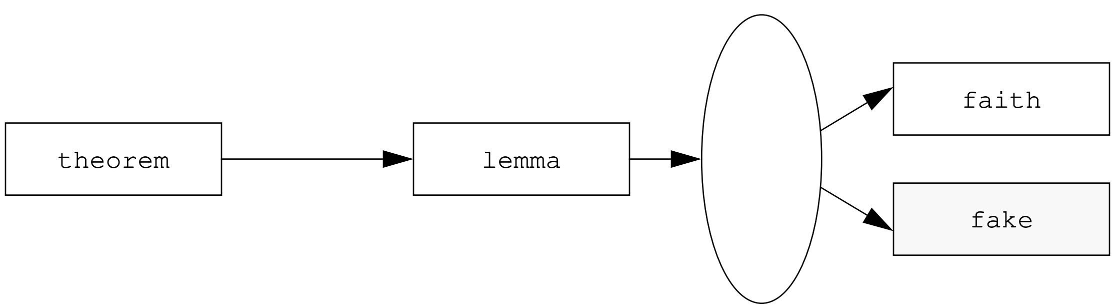

FIGURE 7. Logical topology of the local testbed. For some experiments, additional network load was created on `lemma`'s outgoing interface by sending additional network traffic to a non-existent machine (`fake`), shown in the shaded box.

Because our goal was to measure achievable performance, we did not perform admission control or channel setup through RCAP. Rather, we set the traffic descriptions and the local delay bounds at all the nodes individually. We also made no attempt to use realistic load patterns; the load was generated by processes sending fixed-size packets in a tight loop on the source machine. These experiments are described in more detail in [28].

5.1.1 Maximum Speed of RMTP/RTIP

Figure 8 compares the maximum achievable throughput of RMTP/RTIP with those of raw IP and UDP/IP. The experiment was performed with one process on `theorem` sending 10,000 packets of the same size in a tight loop to another process on `faith`, using `lemma` as a router. No additional load was applied to any of the machines. The packet size was varied to obtain the curves shown. For RMTP/RTIP, the traffic parameters were set so that the rate would be higher than the achievable output rate, in order to measure the maximum rate at which packets could be processed without introducing rate control. From the throughput curve in Figure 8, we can see that the performance of RMTP/RTIP is comparable to that of raw IP. This can be explained by considering the dominance of the data copying cost on protocol performance. Due to the overhead of UDP checksums, the throughput of UDP/IP is lower than that of raw IP or RMTP/RTIP.

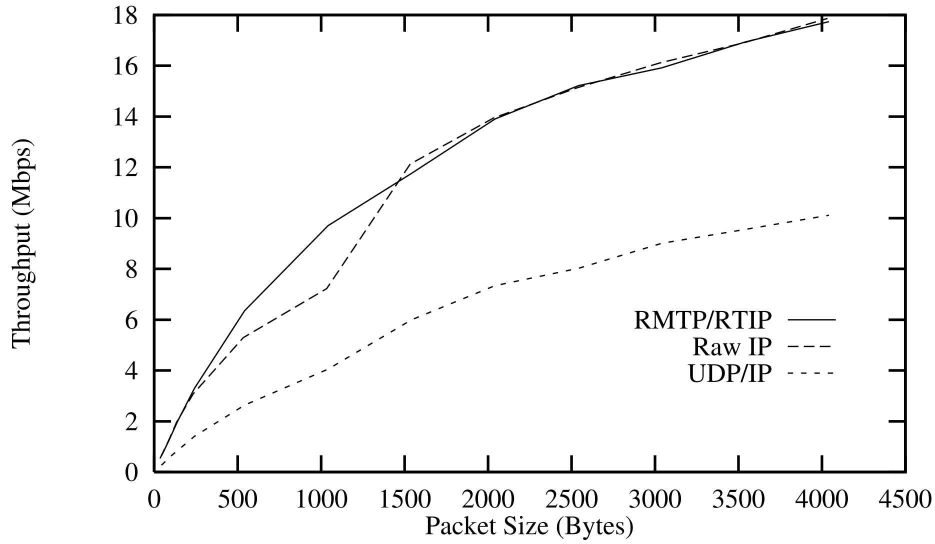

FIGURE 8. Throughput of RMTP/RTIP, UDP/IP, and raw IP.

A more detailed breakdown of the protocol processing costs is shown in Figure 9. These measurements were taken on a DECstation 5000/125. The figure shows that the protocol processing cost (i.e. the difference between the “Socket” curve and the “Socket + RMTP/RTIP” curve) is about 68 µs, and nearly independent of the packet size. The driver software processing time is also fixed at 114 µs per packet. The socket layer processing and transmission contributions dominate and scale with the packet size, since they involve data copying.

5.1.2 Rate Control Behavior

The next set of measurements shows the effect of rate control on RMTP/RTIP behavior. These graphs were obtained by sending data from a process on `theorem` in a tight loop to a receiving process on `faith`, which timestamped arriving packets. Figure 10 shows the arriving sequence numbers against time for channels with different (X_{min}, X_{ave}) combinations. I was 1 second for all the channels. As can be seen in the figure, the rate control mechanism forces adherence to the traffic characteristics specified in the contract, even though the user program is attempting to send in a tight loop.

5.2 Measurement of the Sequoia 2000 Implementation

The Tenet Suite implementation on the Sequoia 2000 wide area network offered us a unique opportunity to measure performance with realistic video-audio traffic and data loads. The measurements presented in this section were performed on the network topology shown in Figure 11. Video streams were sent using the video conferencing tool `vic`, developed by Steve McCanne, between machines at UC Berkeley and UC San Diego. Background data load was

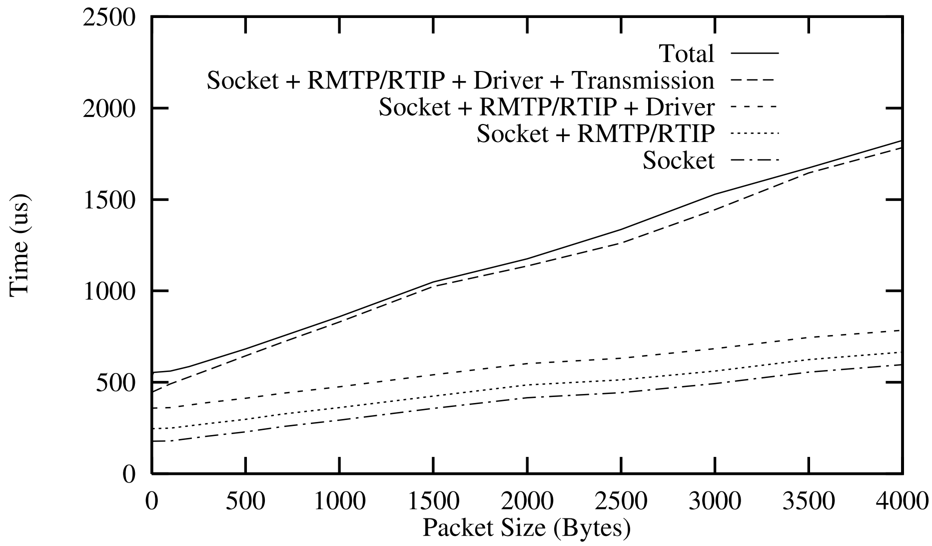

FIGURE 9. Breakdown of processing times in kernel for RMTP/RTIP `send`.

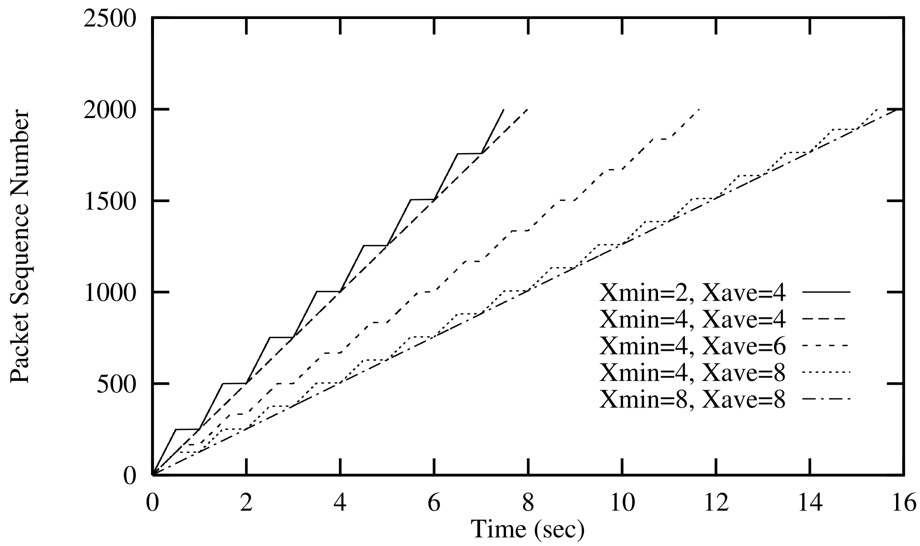

FIGURE 10. Effect of rate control at source on an RMTP/RTIP channel. The source process is attempting to send packets in a tight loop, but its rate is limited by the rate control mechanism.

added by transferring data from the machine `grayling` to a machine at UCLA. This caused competition for transmission resources on the T1 link between UC Berkeley and UC Santa Barbara.

`vic` is able to use either UDP/IP or RMTP/RTIP for data transmission. We compared the performances of a video session using each of the protocols by measuring queuing delays and queue lengths at the bottleneck node, by plotting the throughput and the frame inter-arrival time characteristics at the destination host, and by estimating the perceptual quality of the received data for end users. The Sequoia experiments are described in greater detail in [29].

5.2.1 Single RMTP/RTIP and Single UDP/IP Streams

We examined the queueing behavior of a single `vic` session using RMTP/RTIP at the bottleneck gateway (`sock`). Figure 12 shows the probability density function of queue lengths. The queue length is kept under control by the rate control and resource reservation mechanisms so that, even during periods of high network load, the number of out-

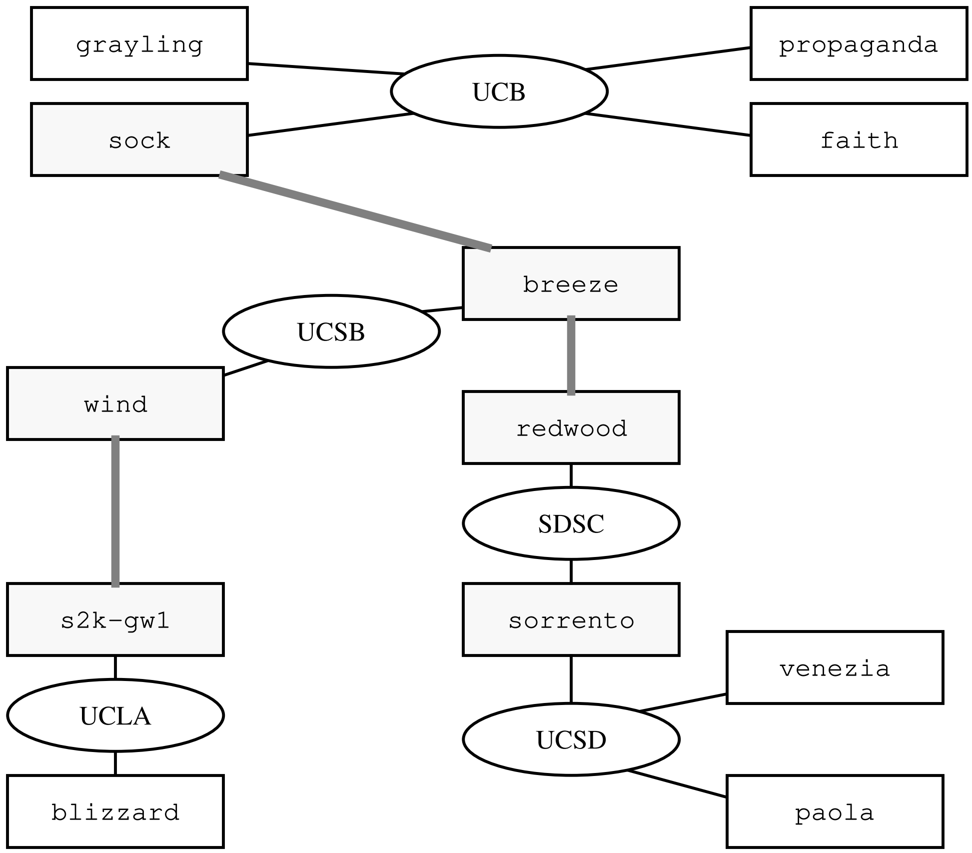

FIGURE 11. The Sequoia 2000 network. Shaded boxes represent routers; all other boxes represent hosts. Ovals represent FDDI local networks. Dashed lines stand for 1.5 Mbps T1 inter-campus links.

standing real-time packets to be served remains at or below five. Figure 13 shows the queue length as a function of time during the period of highest queue buildup. In Figure 14, which shows the delays encountered by packets during the experiment, we can see that the queueing delay at the bottleneck node is always below the guaranteed bound, which in this case was 70 ms.

The performance seen by the application is shown in Figure 15, which plots the throughput (in kilobits per second, averaged over 1-second intervals) over time. The received UDP/IP throughput is fairly smooth up to twenty seconds into the plot, when load is introduced into the network. At this time, performance suffers due to queue buildups, which cause delays and dropped packets. Even after the load is removed at thirty seconds, the transient emptying of the network buffers causes throughput variations for a few more seconds. Because of the reserved resources, the RMTP/RTIP throughput is unaffected by the increased network load.

5.2.2 Two RMTP/RTIP Streams

The previous experiment shows that the traffic on an RMTP/RTIP channel is protected from best-effort UDP/IP traffic. The next experiment demonstrates the effectiveness of the protection of RMTP/RTIP channels from other real-time (RMTP/RTIP) channels in the network. We set up two `vic` sessions from UC Berkeley to UC San Diego and observed their queueing behavior and arrival time characteristics. During the experiment, additional load was introduced from `grayling` as before. The throughput at the destination, shown in Figure 16, remains almost constant even when a substantial amount of load is introduced into the network.

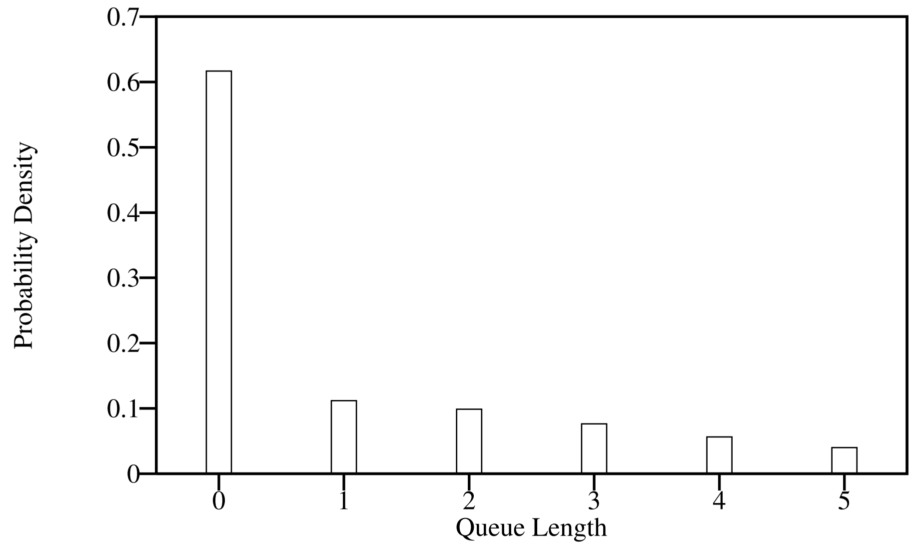

FIGURE 12. Probability density function of queue lengths, for a single RMTP/RTIP channel under congestion. The mean queue length is 0.964.

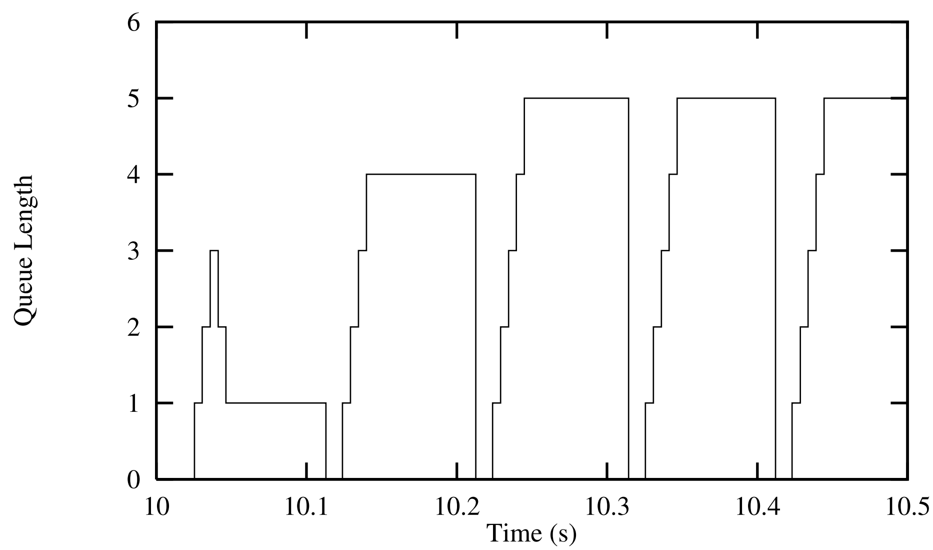

FIGURE 13. Queue length vs. time, for a single RMTP/RTIP channel under congestion.

5.2.3 Impact of Video Quality

Although the experiments in the previous sections clearly demonstrate that the protocols are providing their promised service, it is also important that a human user appreciate the difference between the quality of video transported by UDP/IP and that of video transported by RMTP/RTIP. The next experiment attempts to measure the impact of the improved performance on the viewer of a video session. In this experiment, two `vic` video sessions were run simultaneously from two machines at UC San Diego to two machines at UC Berkeley: one using UDP/IP from `venezia` to `propaganda`, the other using RMTP/RTIP from `paola` to `faith`. Both sessions had an average rate of 580 Kbps (roughly 10 frames per second). This rate was purposely chosen to allow both sessions to get equal performance in the absence of additional load on the network.

Fifty attendees of the University of California’s 1994 Industrial Liaison Program Conference viewed the two video streams and were asked to evaluate their quality under various levels of load in the network. The load was introduced from a machine at UCLA using UDP packets to congest the link between UC Santa Barbara and UC Berkeley. The viewers were asked to score the videos without prior knowledge of which protocol suites were in use.

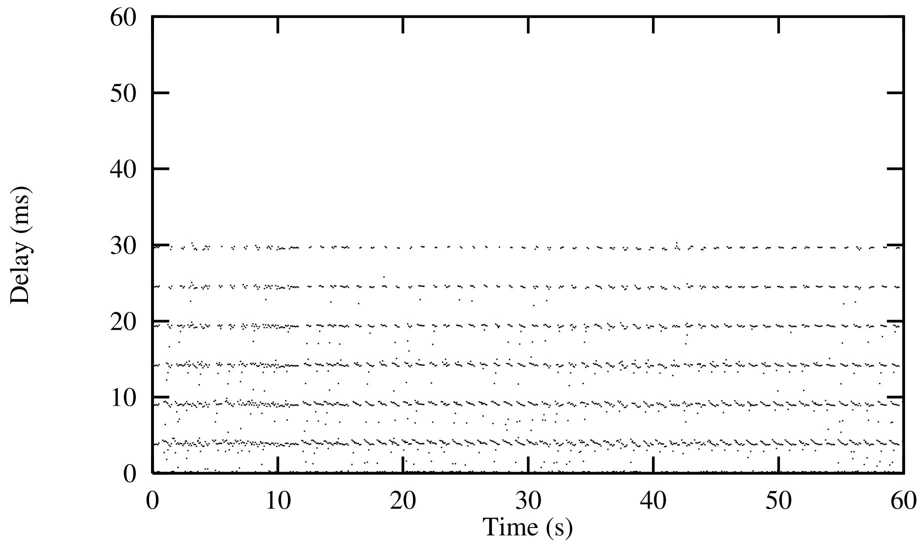

FIGURE 14. Packet delays over times, for a single RMTP/RTIP channel under congestion. The mean delay is 7.37 ms.

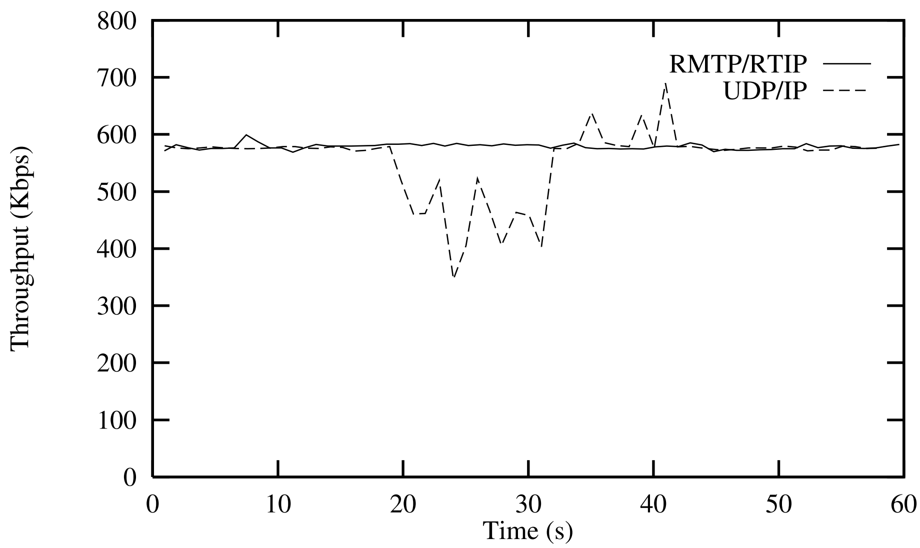

FIGURE 15. Throughputs of RMTP/RTIP and UDP/IP streams under load.

With a 99% confidence interval, the mean opinion score of the RMTP/RTIP video session did not change during the experiment. However, with the same confidence interval, the mean opinion score of the UDP/IP session dropped by 54% when the same load was introduced into the network. This result validates our claim that the use of the Tenet protocols prevents visible degradation of video quality when congestion occurs in the network.

5.2.4 Connection Setup Latency

The final experiment was conducted to investigate the time to perform admission control tests in the connection establishment phase. One concern expressed about resource reservation and admission control is that the time to perform establishment tests might be a significant cost in connection establishment. To address this issue, we measured the establishment process to find the round-trip delay, the per-hop latencies, and the portion ascribable to admission control tests.

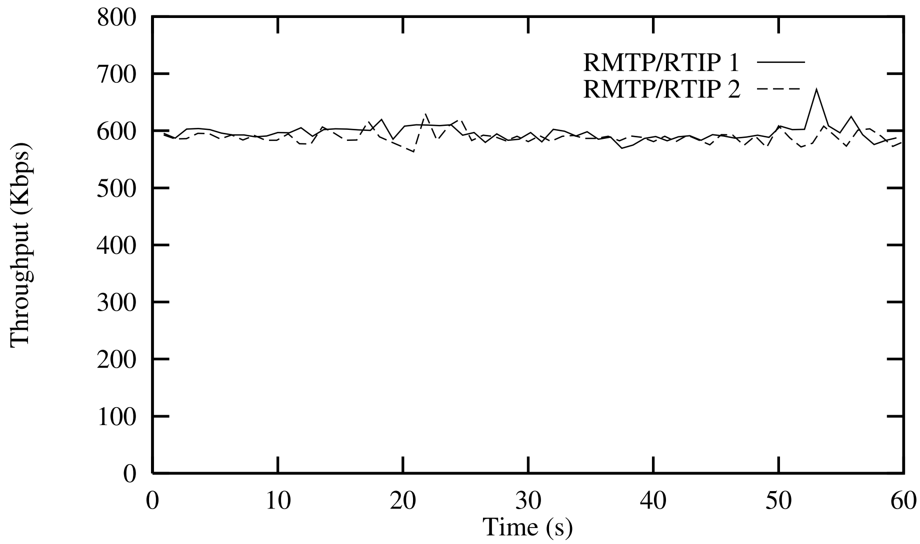

FIGURE 16. Throughputs of co-existing RMTP/RTIP channels.

The path used in the previous experiments, from `faith` at UC Berkeley to `paola` at UC San Diego, involves six machines, three FDDI rings, and two T1 links. The time to perform the end-to-end round trip establishment is about 90 ms. This latency does not change appreciably with the number of RTIP connections established for up to 30 connections. The main factor which determines the latency is the path length. For example, for a shorter path from `faith` to `redwood` at the San Diego Supercomputer Center, the end-to-end setup latency is about 70 ms.

The breakdown of latencies into per-machine and per-link costs for a typical run (from `faith` to `redwood`) is shown in Table 5. This path involves four machines, one FDDI ring and two T1 links. The node latencies are shown separately for the forward and reverse directions, while the link latencies are averages of the forward and reverse latencies.

Node	Forward (ms)	Reverse (ms)	Link (ms)
0	3.906	3.907	3.674
1	2.056	1.832	11.9
2	1.637	1.825	9.779
3	1.853	1.958	—

TABLE 5. Node and Link latencies for channel establishment

Node 0 (a DECstation 5000/125) was slower than the others (DECstation 5000/240s), which explains the higher node latency. We found that the times to perform the admission control tests themselves were always less than 0.5 ms per node. Thus, the establishment latency is dominated by propagation delays and message transport overheads.

6 Lessons

Many lessons were learned from designing, implementing, and experimenting with the Tenet Real-Time Protocol Suite. None of these conclusions could have been reached with the same level of certainty, or at all, if we had just published a few papers on our admission control algorithms and moved on to something else. In this section, we have highlighted what we feel are the most important lessons we have learned and, based on our experience, some of the most important questions still to be answered.

Proof of concept: The most important result we obtained was a demonstration that guaranteed performance data delivery can be provided in a packet-switching environment using the Tenet approach. Significantly, our experiments indicate that it is possible to build signalling protocols that establish real-time channels rapidly enough to satisfy most clients, and to build real-time data delivery protocols that attain the speeds achievable by UDP/IP.

Admission tests must be improved: Even though we did not run experiments intended to investigate this aspect, one can show that the admission control tests we used in the Tenet Suite were too pessimistic. Overly-pessimistic algorithms are undesirable because they reduce the *real-time* capacity of the network[6]. We now have more optimistic tests for both deterministic [30] and statistical [31] guarantees, and plan to experiment with them in Tenet Suite 2.

Multi-party support is needed: The absence of multi-party support has been a real problem when trying to run even small multimedia conferences using Suite 1. This problem has two aspects: without multi-party support, connection setup for an N-way conference is cumbersome and amounts of resources reserved are high. Again, this problem is being solved in Suite 2 by providing multicast channels, resource sharing between related channels (of the same conference) [32], and higher-level abstractions to simplify connection setup for multi-party communication [4].

A variety of establishment paradigms is useful: Experience with applications and conversations with other researchers have convinced us that some applications are best served by *source-initiated* connection establishment (e.g. preplanned conferences in which most participants are known ahead of time), some by *receiver-initiated* establishment (e.g. seminar distribution)[7], and some by *duplex* establishment (e.g. videophone). Although RCAP was designed and implemented for source-initiated establishment, the Tenet establishment procedure could be extended trivially to these other cases as well.

Separation of control and delivery protocols facilitates development: Among the lessons we learned specifically from the design of Suite 1, perhaps the most important is the advantages of separating RCAP from RMTP/RTIP. This separation, and the decision to implement RCAP completely in user space, facilitated the debugging and porting of the implementations. We also discovered that it is sometimes useful to install and run RMTP/RTIP without RCAP and vice versa.

Many additional lessons were learned during the implementation: One lesson was that the use of an existing, reliable transport protocol (TCP) for delivering RCAP messages greatly simplified and speeded implementation, though perhaps another mechanism for intra-machine communication would have been more efficient. Another lesson was that the use of blocking functions in the API made the interface unfriendly to both application programmers and users[8]. An asynchronous approach is being used in the implementation of Suite 2. Thirdly, in debugging these protocols, we found that keeping the protocol entities in a consistent state across the network was difficult. Therefore, we had to develop mechanisms to ensure consistency and/or facilitate its restoration. More generally, we learned that basic network management tools are *necessary* for debugging and deploying real-time protocols.

Remaining questions: A number of questions still need to be answered, and we plan to use this suite and its successor (Suite 2) to shed some light on them. Some are small and detailed (e.g. are the representations for time and probability values chosen for RCAP suitable?). Others are more far-reaching (e.g. what is an appropriate traffic representation, neither so rich as to be too difficult to use, nor so simple as to fail to accurately model resource requirements?). Other questions are fundamental to real-time networking research: Is distributed delay jitter control really useful? Are statistical guarantees really usable? Are deterministic guarantees really needed? Unfortunately, answering the more fundamental of these questions requires real clients with real applications; therefore, we will have to wait until real-time protocols (ours and others) are deployed in a production environment.

7 Conclusions and Future Work

While we made a number of simplifying shortcuts in both the design and the implementation of the Tenet Suite, we feel that the project proved the correctness, feasibility and practicality of the Tenet approach to real-time communica-

---

6. Near-full utilization of the network can still be obtained by the addition of non-real-time traffic.

7. By *receiver-initiated* establishment, we mean only that the direction of establishment is from the receiver to the source: the traffic requirements still are specified by the source.

8. Blocking procedure calls cause the user interface of an application to “freeze up” until the procedure call returns, unless a multi-threaded design is used to avoid this problem.

tion. The result is the first suite of communication protocols capable of transferring real-time streams on packet-switching internetworks with guaranteed performance.

Much work has already been done to extend the Tenet approach and protocol suite. The next generation scheme and suite (*Scheme 2* and *Suite 2*, respectively) have been designed, as mentioned above, and implementation is underway. Suite 2 supports multi-party communication, including routing and establishment of multicast connections and resource sharing [4]. A key difference between the resource sharing in Scheme 2 and other proposals is that Scheme 2 uses simple, application-specific hints to share resources among related connections without violating the mathematical guarantees of the Tenet approach. Analysis and simulation results indicate that resource sharing can greatly improve resource utilization for most multi-party conferences [32]. Suite 2 also supports partitioning of network resources into independent "virtual networks" and supports advance reservation of resources for future connections [33].

In addition, several improvements have been made to the unicast scheme and suite. Besides the already mentioned enhancements to the admission control tests, mechanisms for Dynamic Connection Management (DCM) have been developed to support dynamic adjustment of traffic and performance parameters and/or routes of existing connections. These mechanisms have been experimented with in the context of Suite 1 [34]. DCM has also been used and augmented to support fault tolerance and fault recovery [35]. Future work includes implementation and experimentation with these new algorithms and mechanisms.

8 References

[1] D. Ferrari, "Real-Time Communication in Packet Switching Wide Area Networks," TR-89-022. International Computer Science Institute, Berkeley, May 1989.

[2] D. Ferrari, J. C. Pasquale, and G. Polyzos, "Network Issues for Sequoia 2000," *Proc. CompCon Spring'92*, San Francisco, CA, February 1992.

[3] "Gigabit Network Testbeds," *IEEE Computer 23*, 9, September 1990.

[4] A. Gupta, W. Heffner, M. Moran, and C. Szyperski, "Network Support for Realtime Multi-Party Applications," *Proc. Fourth International Workshop on Network and Operating System Support for Digital Audio and Video*, November 1993.

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

[6] C. Partridge and S. Pink, "An Implementation of the Revised Internet Stream Protocol (ST-2)," *Internetworking: Research and Experience 3*, 1, March 1992.

[7] L. Delgrossi, R. G. Herrtwich, and F. O. Hoffman, "An Implementation of ST-II for the Heidelberg Transport System," IBM ENC TR-43.9303, To appear in *Internetworking: Research and Experience*.

[8] R. G. Herrtwich, personal communication, November 1992.

[9] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network*, September 1993.

[10] A. Lazar and C. Pacifici, "Control of Resources in Broadband Networks with Quality of Service Guarantees," *IEEE Communication Magazine*, October 1991.

[11] I. Cidon, I. Gopal and R. Guerin, "Bandwidth Management and Congestion Control in PlaNET," *IEEE Communication Magazine*, October 1991.

[12] ATM Forum, ATM User-Network Interface Specification Version 3.0, September 1993.

[13] D. Ferrari, A. Banerjea, and H. Zhang, "Network Support for Multimedia: A Discussion of the Tenet Approach", *Computer Networks and ISDN Systems 26*, special issue on Multimedia Networking, 1994.

[14] D. P. Anderson, R. G. Herrtwich, and C. Schaefer, "SRP: A Resource Reservation Protocol for Guaranteed Performance Communication in the Internet," TR-90-006, International Computer Science Institute, Berkeley, CA, February 1990.

[15] A. Banerjea and B. Mah, "The Real-Time Channel Administration Protocol," *Proc. Second International*

Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg, Germany, November 1991.

[16] B. Mah, "A Mechanism for the Administration of Real-Time Channels," MS Report, University of California at Berkeley, March 1993.

[17] B. Wolfinger and M. Moran, "A Continuous Media Data Transport Service and Protocol for Real-Time Communication in High Speed Networks," *Proc. Second International Workshop on Network and Operating System Support for Digital Audio and Video,* Heidelberg, Germany, November 1991.

[18] D. Ferrari, "Real-Time Communication in an Internetwork," *Journal of High Speed Networks 1*, 1, 1992.

[19] D. Ferrari and D. C. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE Journal on Selected Areas in Communications 8*, 3, April 1990.

[20] D. Ferrari, "Distributed Delay Jitter Control in Packet-Switching Internetworks," *Journal of Internetworking: Research and Experience 4*, 1, 1993.

[21] D. C. Verma, "Guaranteed Performance Communication in High Speed Networks," PhD dissertation, University of California at Berkeley, December 1991.

[22] D. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Communications of ACM 7*, 7, July 1974.

[23] S. J. Leffler, M. K. McKusick, M. J. Karels, and J. S. Quarterman, *The Design and Implementation of the 4.3 BSD UNIX Operating System*, Addison-Wesley Publishing Company, 1989.

[24] S. J. Leffler, W. N. Joy, R. S. Fabry, and M. J. Karels, "Networking Implementation Notes: 4.3 BSD Edition," *Unix System Manager's Manual,* USENIX Association, April 1986.

[25] H. Zhang and D. Ferrari, "Rate-Controlled Service Disciplines," To appear in *Journal of High-Speed Networks*, 1994.

[26] H. Zhang and D. Ferrari, "Rate-Controlled Static Priority Queueing," *Proc. IEEE INFOCOM '93*, San Francisco, CA, September 1993.

[27] H. Zhang, "Service Disciplines for Integrated Services Packet-Switching Networks," PhD Dissertation, UCB/CSD-94-788, University of California at Berkeley, November 1993.

[28] H. Zhang and T. Fisher, "Preliminary Measurement of the RMTP/RTIP," *Proc. Third International Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, CA, November 1992.

[29] A. Banerjea, E. Knightly, F. Templin, and H. Zhang, "Experiments with the Tenet Real-Time Protocol Suite on the Sequoia 2000 Wide Area Network," *Proc. ACM Multimedia '94*, San Francisco, CA, October 1994.

[30] H. Zhang and D. Ferrari, "Improving Utilization for Deterministic Service in Multimedia Communication," *Proc. 1994 IEEE International Conference on Multimedia Computing and Systems*, May 1994.

[31] H. Zhang and E. Knightly, "Providing End-to-End Statistical Guarantees Using Bounding Interval Dependent Stochastic Models," *Proc. ACM SIGMETRICS '94*, May 1994.

[32] A. Gupta, W. Howe, M. Moran, and Q. Nguyen, "Scalable Resource Reservation for Multi-Party Real-Time Communication," *Submitted for publication*, August 1994.

[33] D. Ferrari and A. Gupta, "Resource Partitioning for Real-Time Communication," *Proc. IEEE Symposium on Global Data Networking*, Cairo, Egypt, December 1993.

[34] C. Parris, H. Zhang, and D. Ferrari, "A Dynamic Management Scheme for Real-Time Connections," *Proc. INFOCOM '94*, June 1994.

[35] A. Banerjea, C. Parris, and D. Ferrari, "Recovering Guaranteed Performance Service Connections from Single and Multiple Faults," TR-93-066, International Computer Science Institute, Berkeley, CA, November 1993.