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

Anindo Banerjea,[2] Domenico Ferrari, Bruce A. Mah,
Mark Moran, Dinesh C. Verma,[3] and Hui Zhang[4]

The Tenet Group
Computer Science Division
University of California at Berkeley
Berkeley, CA 94720-1776
and
The International Computer Science Institute
1947 Center St., Suite 600
Berkeley, CA 94704-1105

ABSTRACT

Many future applications will require guarantees on network performance, such as bounds on throughput, delay, delay jitter, and reliability. To address this need, we have 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 protocol suite. We rely on the use of connection-oriented communication, per-channel admission control, channel rate control, and priority scheduling.

This protocol suite is the first set of transport and network-layer communication protocols that can transfer real-time streams with guaranteed quality in packet-switching internetworks. We

---

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., 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.

2. Now with Philips Research, 4005 Miranda Avenue, Suite 175, Palo Alto, CA 94304.

3. Now with the IBM T. J. Watson Research Center, Yorktown Heights, NY 10598.

4. Now with the School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA 15213.

have performed a number of experiments and demonstrations on multiple platforms 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 some of the lessons we learned.

1 Introduction

This paper describes the Tenet Real-Time Protocol Suite, the first set of network and transport layer protocols capable of transferring real-time streams in heterogeneous packet-switching internetworks with guaranteed quality. The design, implementation, and experiments described herein served two purposes: 1) to investigate whether such guarantees can be provided in a packet-switching internetwork, and 2) to add to the knowledge of the field through a systematic study of the complexity and performance of such a scheme.

The remainder of this section provides some background for our work. In Section 2, we briefly state some of the principles behind the Tenet Suite; more details, as well as justifications, can be found in [1][2][3][4]. We then describe the architecture, which arises from a consideration of these principles. Section 3 and Section 4 present the design and the 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 during this endeavor. 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 in 1987, when 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.

Ferrari set out to build a scheme (a set of algorithms) that would offer a packet-switched network's real-time clients the desired guarantees by providing *real-time channels*[5]. The basic ideas underlying the scheme were the real-time channel abstraction itself, admission control, connection-oriented communication, channel rate control, and deadline-based scheduling for real-time packets [1]. He was later joined by the other authors in designing a protocol suite embodying the Tenet scheme.

Several simplifications were made to facilitate the development of the first suite. Most notably, we decided to provide only unicast real-time channels, and to introduce multicast capabilities in a later version of the suite after studying the service requirements of multi-party applications [5]. We also postponed the study of routing algorithms for real-time channels.

By Summer 1991, the suite had been specified in all its details and extensively simulated. We subsequently implemented, 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 [6], and performed various experiments and demonstrations with continuous-media loads. A second generation protocol suite (*Suite 2*) has been designed to support multi-party real-time communication (including multicast) and is currently being implemented [7].

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 [8] is a connection-oriented internetwork protocol that has been implemented in several versions (e.g., [9], [10]). Although the protocol specification includes a *FlowSpec* that indi-

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

cates resource requirements, the algorithms for real-time admission control have not been specified. To the best of our knowledge, the implementation by Delgrossi *et al.* [10] is the only one providing mathematically provable guarantees. It uses admission control algorithms based on the Tenet scheme over Token Ring and FDDI networks [11]. ST-II follows many of the same principles as the Tenet Suite; it is a very rich protocol and can support multicast streams. However, neither the protocol specification nor the implementation of [10] provide a mechanism for performing and encapsulating reservations on multi-hop subnetworks (e.g., ATM networks).

More recently, RSVP has been designed as a protocol for exchanging reservation messages [12]. A subject of current work within the Internet Engineering Task Force, RSVP 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 flows to adapt more easily to network load and outages, but introduces some refresh overhead and allows quality of service disruptions due to routing changes or lost refresh messages. In contrast, the Tenet suite uses "hard" state to provide consistent performance during normal operation and explicit recovery in case of network faults [13]. 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 networks (e.g., [14], [15]). To the best of our knowledge, these implementations are not applicable to heterogeneous internetworks. Standard signalling protocols for ATM networks can support mathematically provable performance guarantees [16]. However, the corresponding admission control algorithms have not yet been fully 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 our protocol suite [2]. In this section, we

briefly recall some of these principles. More details, as well as basic motivations, have been presented and discussed elsewhere [1] [2] [3] [4].

First, all layers in a network's architecture, in particular, the datalink layer, must be able to provide guaranteed-performance services. Datalink layers such as synchronous FDDI, ATM with a suitable signaling protocol and admission tests, and 100VG-AnyLAN meet this requirement. By contrast, no real-time service offering mathematically provable delay guarantees can be built on top of the IEEE 802.3 (Ethernet) datalink layer. Performance bounds can be offered to applications by implementing real-time protocols at the network and transport layers, on top of a real-time datalink protocol. 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. In this phase, admission control tests are run at each node along the channel's path, all of which must succeed 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 [8] and SRP [17]), but it had the advantage of allowing us to develop, test, and maintain the control and data delivery protocols separately. Control functions in the Tenet Suite are provided by the *Real-Time Channel Administration Protocol* (RCAP) [18] [19].

An important principle of the Tenet approach is that all real-time applications have requirements expressible by general performance or reliability parameters. Therefore, there is no need for media-specific protocols at the network or transport layers, as one general-purpose protocol at each layer will suffice. The Tenet Suite's two transport protocols (RMTP and CMTP) differ from each other in their interfaces and services; CMTP is oriented towards periodic traffic, and is time-driven instead of being based on the send-receive model used by RMTP.

The Tenet Suite is expected to offer a real-time service in an integrated-services network.

Hence, we designed our protocols to coexist with the Internet protocols. Clients can send their non-real-time traffic using TCP/IP or UDP/IP and their real-time traffic using the Tenet protocols.

Finally, a control facility is required 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. This protocol computes new routes and resource allocations in the event of network failures [13].

The suite that resulted from these considerations is shown in Figure 1, which represents our proposed architecture for an integrated services internetwork. At the time of this writing, CMTP has been almost entirely implemented, but has not been tested and integrated with the other protocols. Its design is described in [20].

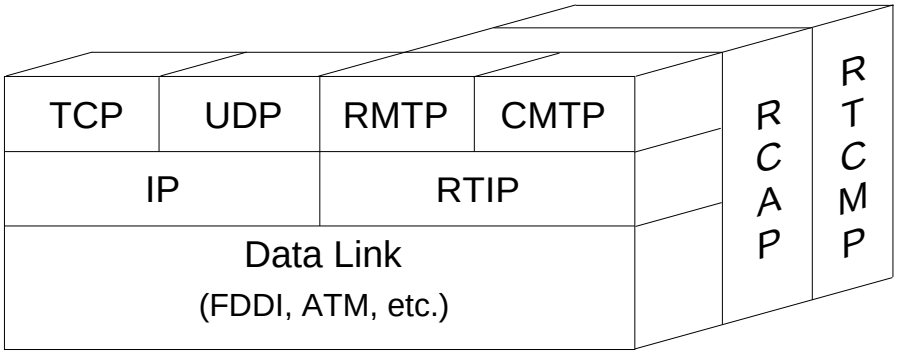

FIGURE 1. The Tenet Real-Time Protocol Suite. Also shown are the corresponding Internet protocols.

3 Design of the Tenet Protocols

This section discusses the design of our signalling protocol (RCAP), our network layer protocol (RTIP), and one of our 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. The RCAP messages are shown in Table 1.

Message Name	**Direction**	**Description**
`establish_request`	Downstream	Request to establish a new channel.
`establish_accept`	Upstream	Indicates acceptance of a new channel.
`establish_denied`	Upstream	Indicates that a channel establishment request was rejected.
`status_request`	Downstream	Requests channel status at each node.
`status_report`	Upstream	Returns data collected by a `status_request`.
`close_request_forward`	Downstream	Message from source to close a channel.
`close_request_reverse`	Upstream	Message from destination to close a channel.

TABLE 1. RCAP messages.

Channel establishment is performed in a single round trip in order to keep the setup time small. More complex client negotiations and/or resource balancing may be added using a method similar to DCM [21]. In the forward pass, an `establish_request` message is sent, hop by hop, from the source to the destination of the channel. At each node along the path where RTIP runs, a local RCAP entity performs admission control tests [1] [3]. If the channel can be supported, the necessary resources are tentatively allocated and the `establish_request` message is forwarded to the next node.

Once this message reaches the destination, an `establish_accept` message is sent hop by hop back from the destination to the source, backtracking along the path. Each local RCAP entity may relax the allocation of any resources that were over-reserved on the forward pass. It then informs the local RTIP entity of the new channel and passes the `establish_accept` message upstream. When this message reaches the source application, data transfer may begin.

If any node determines that it cannot support the channel's performance requirements, it sends an `establish_denied` message hop by hop back to the source; this message causes the channel's tentatively reserved resources to be released.

During establishment, the client must specify its performance requirements (the *Quality of Service,* or *QoS,* it needs) and a worst-case description of the traffic it will transmit over the new channel [1] [3]. The QoS parameters of the RCAP interface are listed in Table 2, the traffic parameters in Table 3. Note that two of the four QoS parameters in Table 2 are probabilities to support statistical bounds on delay and loss. These services allow bandwidth and buffer space to be "overbooked", appreciably increasing the utilization of these resources [3].

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.

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.

Establishment of a channel is analogous to the signing of 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 channel's lifetime [22] [23]. This contract is valid until the channel is torn down.

Finally, either the source or the destination of a given channel can request that it be closed, using the `close_request_forward` and `close_request_reverse` messages. They are transmitted hop by hop along the channel route and cause the channel's resources to be released.

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. Interoperability is obtained by deriving network-specific parameters and bounds from the end-to-end client-specified parameters (such as those in Table 2 and Table 3). RCAP abstracts these network-specific bounds into network-independent bounds that can be used by the end-to-end establishment process. Such an approach allows RCAP to utilize the specific characteristics of an individual network in order to provide guarantees, yet hide the underlying details of that network whenever possible [4] [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 to meet the channels' real-time requirements. In contrast to the IP datagram service, RTIP is connection-oriented. Data forwarding in RTIP is unreliable since packets may be corrupted or lost due to buffer overflow (if $W_{min} < 1$). RTIP performs rate control, jitter control, and scheduling based on the QoS parameters of each connection. Since all packets on an RTIP connection follow the same path, and we assume that subnets do not reorder packets, packets arrive at the destination in the same order they were sent 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 to simplify processing at each node. In order to coexist with IP[6], the first four bits of the RTIP header identify a packet as an RTIP packet. The other fields in the packet consist of a local channel ID, a packet length field, and a packet sequence number used in message reassembly (see Section 3.3). In addition, each packet carries a timestamp, which indicates the time the packet was received by the RTIP module at the sending host, and is used by our distributed jitter-control mechanism [22]. RTIP includes a checksum for

6. Since RTIP was designed for experimentation under as few constraints as possible, we did not try to maintain IP compatibility, but merely ensured that the two protocols could co-exist. RTIP packets may be encapsulated in IP packets to simplify wide-scale deployment over networks where routers do not implement RTIP; however, no real-time guarantees can be made in such a scenario.

its header, but does not provide a checksum for user data.

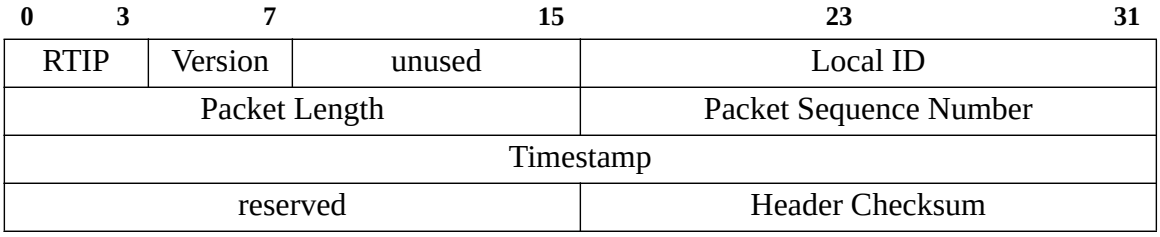

FIGURE 2. RTIP packet header.

Each RTIP entity contains a regulator and a scheduler. The regulator monitors the traffic on each connection and shapes it according to the traffic specification of that connection. The regulator also ensures that a connection's delay jitter requirements are satisfied [22]. The scheduler ensures that no packet (or a bounded fraction of the packets) remain in the node longer than their local delay bounds.

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 resides at the same layer in the protocol stack as TCP, but is lighter-weight. For example, the service provided by RMTP is unreliable. Acknowledgments and retransmissions, the usual cure for loss and corruption, were deemed useless for real-time applications with stringent delay bounds and for most continuous-media applications. RMTP relies on RCAP to manage connections so that there is no congestion, and on RTIP to provide rate-based flow control. Finally, RMTP does not perform packet reordering, since RTIP provides in-order packet service.

The main service provided by RMTP 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. The receiving RMTP module uses sequencing information carried in the packet headers to reassemble the received message. The RMTP header is shown in Figure 3.

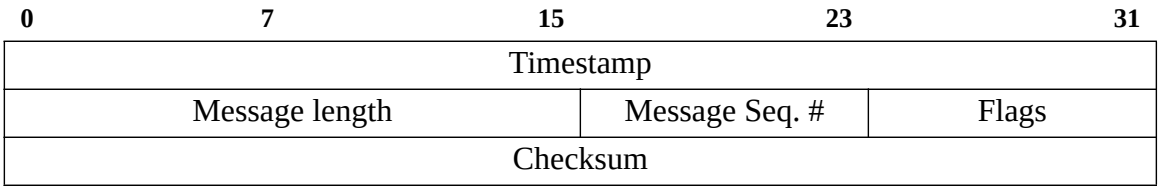

FIGURE 3. RMTP message header.

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 [19]. 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.

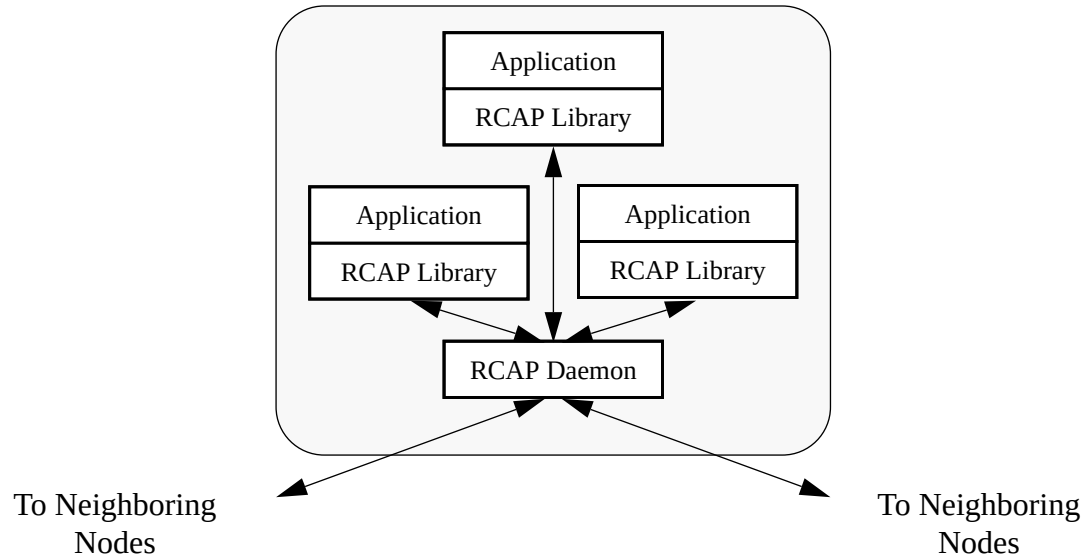

FIGURE 4. The RCAP library and daemon, within a single network node. Arrows represent RCAP control connections.

The RCAP library implements the network control API. It exports a set of procedures that can be called within a user program to request various channel management functions. These pro-

cedures are listed in Table 4. 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.
`RcapRegister()`	Receiver uses this call to indicate its willingness to accept new connections.
`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.
`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.

TABLE 4. RCAP library exported procedures.

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 node, and responds to requests for channel establishment and teardown. It maintains the local 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 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 9000/700s and for Ultrix on DECstation 5000s. Tom Fisher and Hoofar Razavi ported these implementations to SunOS on SPARCstations. Keith Sklower built versions for IRIX on SGI workstations and BSDI BSD/OS on 80486-based PCs. All the software platforms are UNIX-like

operating systems [24] with networking software derived from BSD UNIX [25] [26].

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. A new type of socket, with protocol type `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.

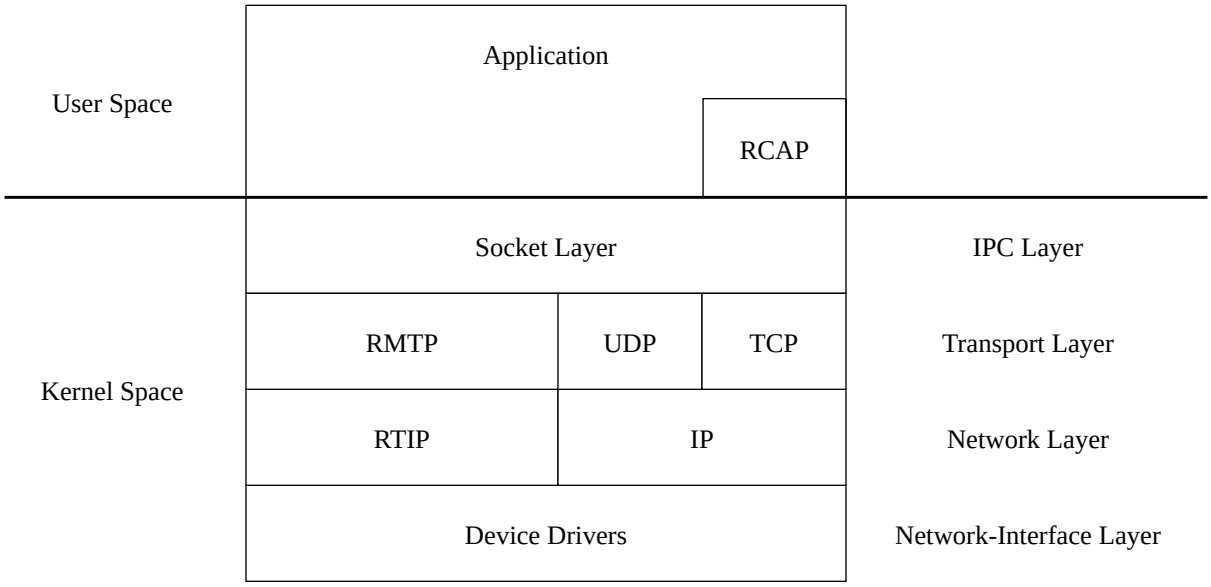

FIGURE 5. Software structure of RMTP/RTIP. The Internet protocols co-exist with the Tenet Suite.

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 per-channel buffer management, as well as priority queueing mechanisms and rate control.

We implemented per-channel buffer management using a special-purpose mbuf originally used by the Network File System (NFS). When an mbuf used by a channel is released, it is returned to the set of buffers reserved for that channel instead of to the system-wide buffer pool.

A number of rate-based service disciplines [27] have been implemented in our suite: Delay-Earliest-Due-Date [1], Jitter-Earliest-Due-Date [22] and Rate-Controlled Static Priority [28]. For the latter two, we implemented both rate-jitter and delay-jitter controlling regulators

[27]. The implementation is modular, so that other regulators and schedulers can easily be used.

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 [27] [29]. Packets can be held in a host by putting the transmitting process to sleep, and in a gateway router by using the UNIX `timeout` procedure. 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

Experimental measurements of our protocols is crucial. 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, timer granularity, 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 must be measured from implementations.

5.1 Preliminary Measurement of RMTP/RTIP Performance

For our initial experiments, we set up the network shown in Figure 6. 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, best-effort load was created by sending packets to a non-existent machine called `fake`.

Because the goal of this experiment was to measure achievable performance, we did not use RCAP's admission control or channel setup; we set the traffic descriptions and local delay bounds individually. We also made no attempt to use realistic load patterns; the load was gener-

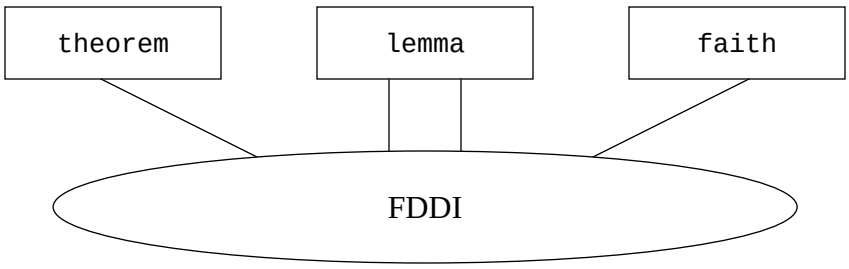

FIGURE 6. Physical topology of the local testbed. `theorem` and `lemma` are DECstation 5000/125s and `faith` is a DECstation 5000/240. Note that `lemma` has two FDDI interfaces. All machines run the Ultrix 4.2A operating system.

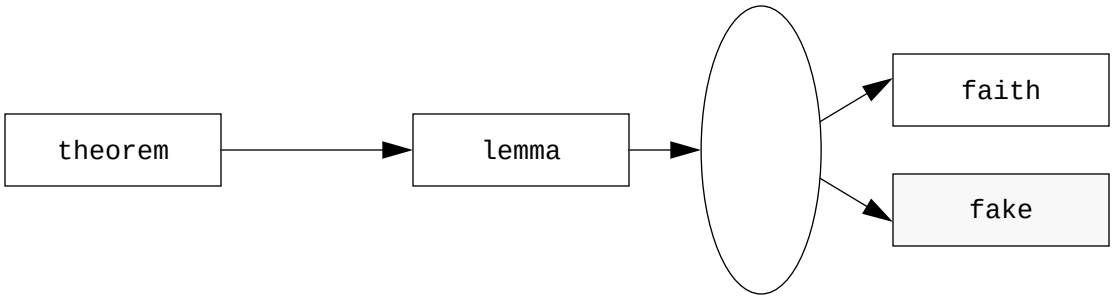

FIGURE 7. Logical topology of the local testbed. For some experiments, additional network load was created on `lemma`'s outgoing interface by sending additional network traffic to a non-existent machine (`fake`), shown in the shaded box.

ated by sending fixed-size packets in a tight loop. More details can be found in [30].

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`, through `lemma`. The packet size was varied to obtain the curves shown. The RMTP/RTIP parameters were set to allow traffic at higher than the achievable output rate, in order to measure the maximum rate at which packets could be processed. 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.

A detailed breakdown of the RMTP/RTIP overheads on a DECstation 5000/125 is shown

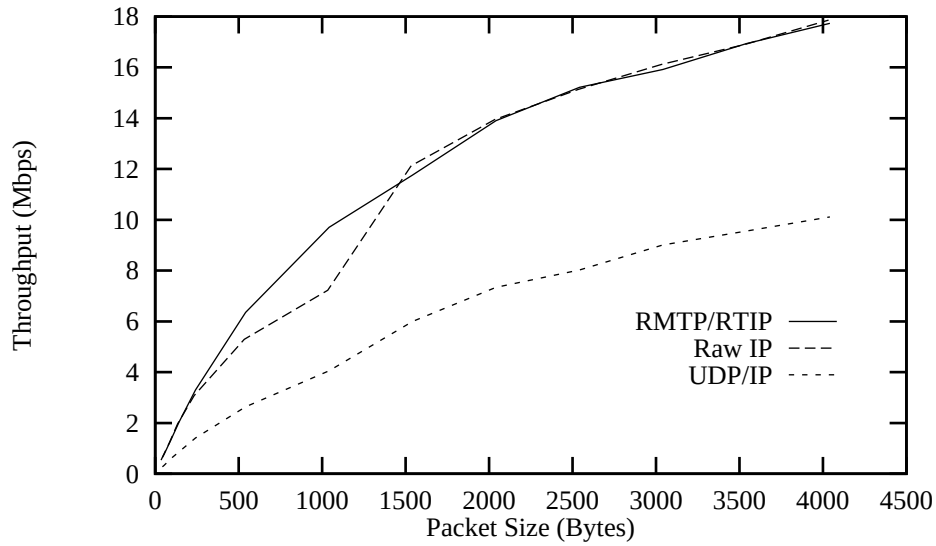

FIGURE 8. Throughput of RMTP/RTIP, UDP/IP, and raw IP.

in Figure 9. The protocol processing cost 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.

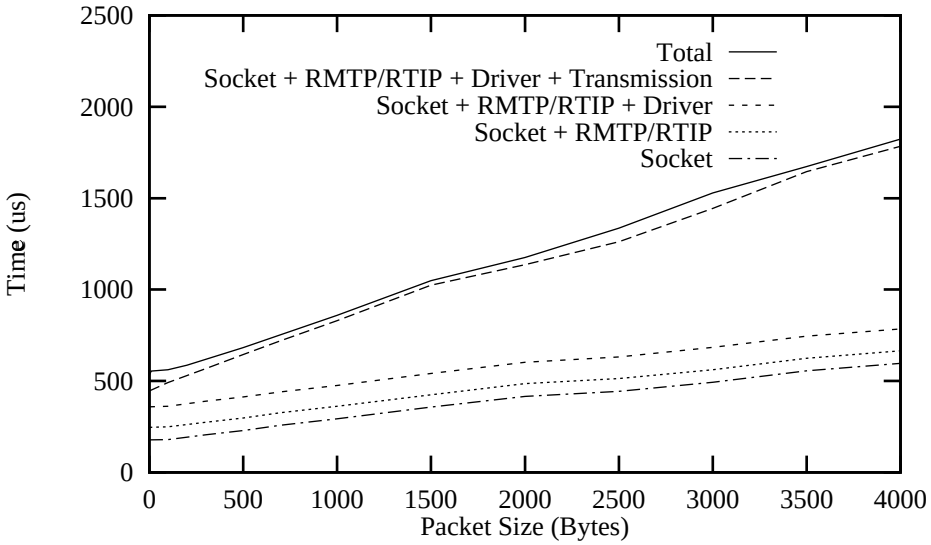

FIGURE 9. Breakdown of processing times in kernel for RMTP/RTIP `send`.

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 every channel. As can be seen in the figure, the rate control mechanism forces adherence to the traffic specification, even though the source is attempting to send in a tight loop.

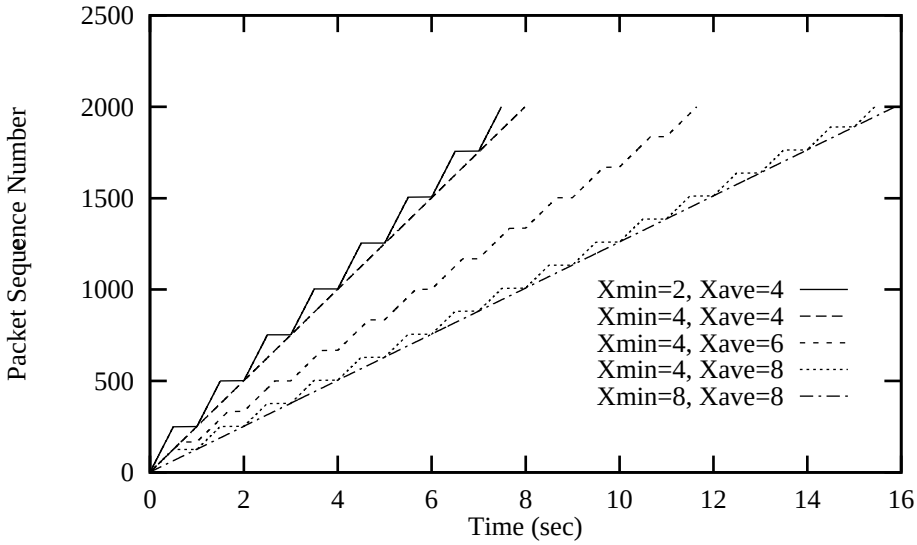

FIGURE 10. Effect of rate control at source on an RMTP/RTIP channel. The source process is attempting to send packets in a tight loop, but its rate is limited by the rate control mechanism.

5.2 Measurement of the Sequoia 2000 Implementation

The Sequoia 2000 wide-area network offered us an 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 traffic was added by transferring data from `grayling` to `blizzard`. This traffic was used to congest 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 in experiments summarized in this section. Our Sequoia experiments are described in greater detail in [31].

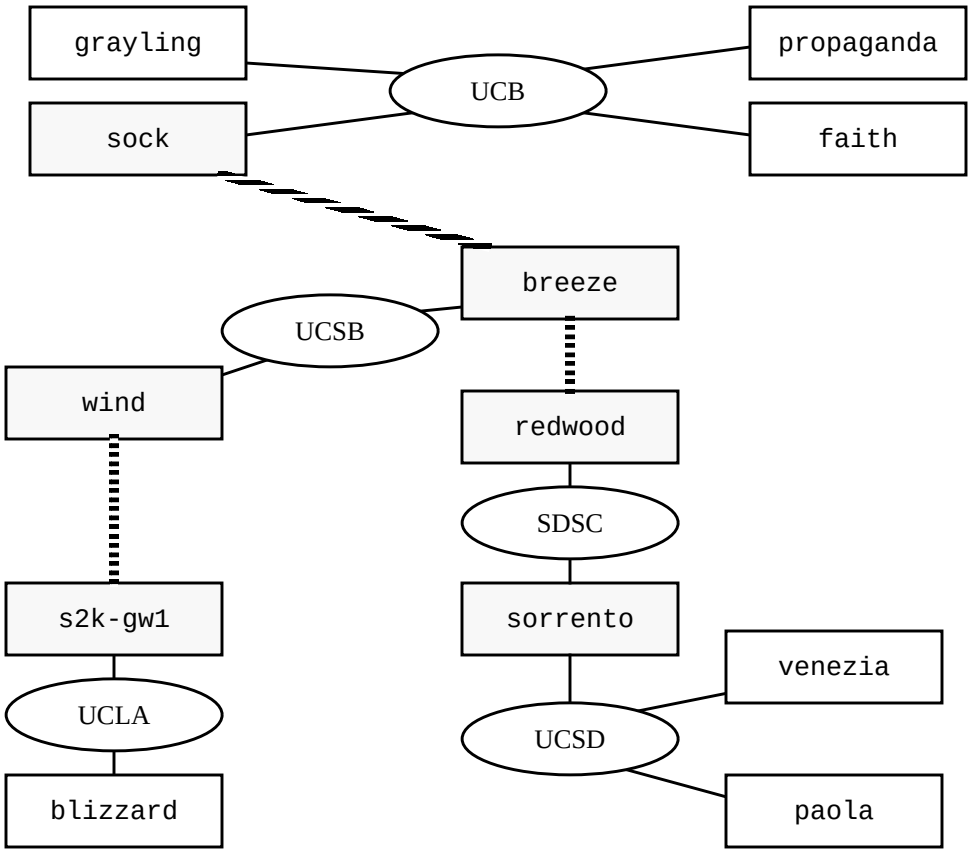

FIGURE 11. The Sequoia 2000 network. Shaded boxes represent routers; all other boxes represent hosts. Ovals represent FDDI local networks. Dashed lines stand for 1.5 Mbps T1 inter-campus links.

5.2.1 Single RMTP/RTIP and Single UDP/IP Streams

We used RCAP to establish an RMTP/RTIP channel for a `vic` session. The channel parameters given to RCAP set peak and average rates of eleven RMTP messages per second. Since the peak and average rates were equal, the averaging interval was not relevant. Each message corresponded to a single motion JPEG frame, with a maximum size S_{max} of 7200 bytes[7]. We examined the queuing behavior at the bottleneck router (`sock`). Figure 12 shows the probability density function of queue lengths. The queue length is curbed by the rate control and resource reservation mechanisms. Even under a non-real-time load sufficient to saturate the outgoing link, the number of outstanding real-time packets to be served remains at or below five[8]. In Figure 13,

7. These RMTP parameters translated to RTIP layer parameters of 88 packets per second (average and peak rate), with a maximum packet size of 1000 bytes.

which shows the delays encountered at `sock`, 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 delays in Figure 13 concentrate around several discrete values at distances that correspond to a packet's transmission time.

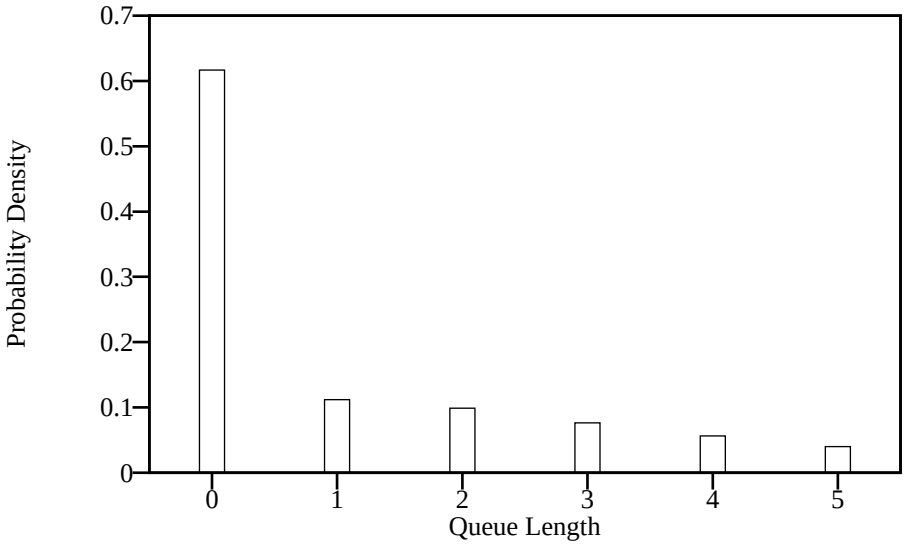

FIGURE 12. Probability density function of queue lengths, for a single RMTP/RTIP channel under non-real-time traffic congestion. The mean queue length is 0.964.

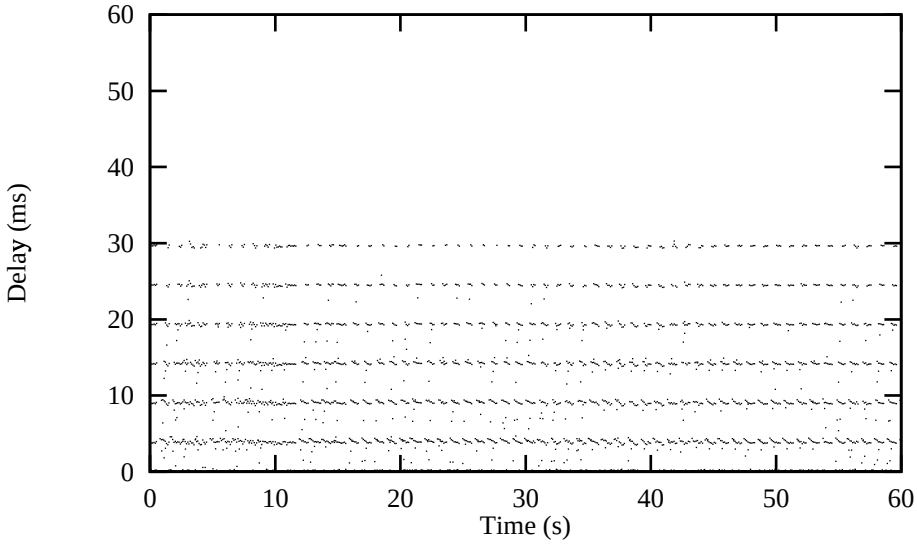

FIGURE 13. Packet delays vs. time, for a single RMTP/RTIP channel under non-real-time traffic congestion. The mean delay is 7.37 ms.

8. The queue length reaches as high as five because the effect of timer granularity on the RTIP rate control implementation.

Figure 14 plots the received throughput (in kilobits per second, averaged over 1-second intervals) over time. The 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 build-ups, which cause delays and dropped packets. Even after the load is removed at thirty seconds, the transient emptying of the network buffers causes variations for a few more seconds. Because of its higher priority, however, the RMTP/RTIP stream is protected from the background traffic.

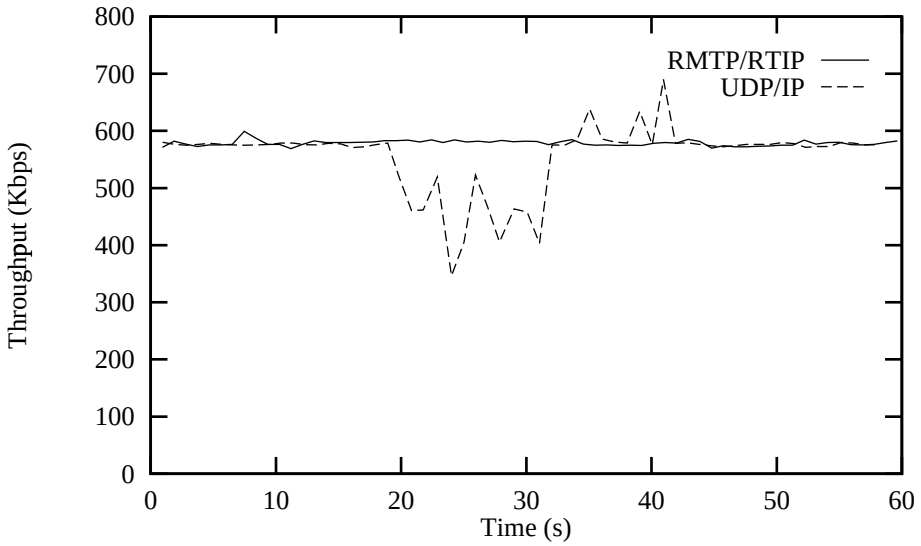

FIGURE 14. Throughputs of RMTP/RTIP and UDP/IP streams under non-real-time traffic load.

5.2.2 Two RMTP/RTIP Streams

This experiment demonstrated the protection of multiple RMTP/RTIP channels from each other. We set up two `vic` sessions from UC Berkeley to UC San Diego and introduced from additional load from `grayling` as before. The throughputs, shown in Figure 15, remain almost constant even when a substantial amount of load is introduced into the network.

5.2.3 Impact on Video Quality

Although the experiments in the previous sections clearly demonstrate that our protocols provide their promised service, it is also important that a human user appreciate the difference between the quality of video transported by UDP/IP versus RMTP/RTIP. In this experiment, two

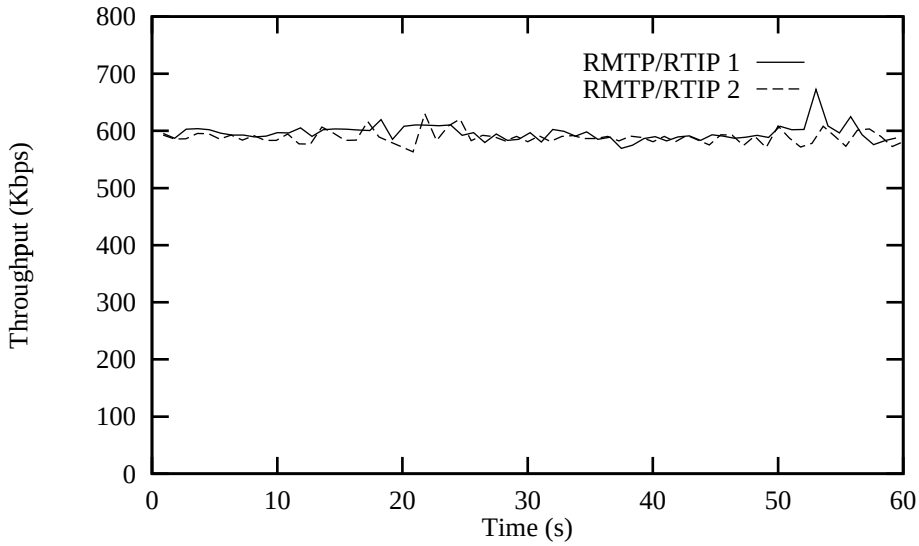

FIGURE 15. Throughputs of co-existing RMTP/RTIP channels under non-real-time traffic load.

`vic` video sessions were run simultaneously from UC San Diego to 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 ten frames per second), a rate chosen to allow both sessions to get equal performance on an unloaded network.

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

5.2.4 Connection Setup Latency

The final experiment was conducted to investigate the time taken to establish connections. One concern expressed about connection-oriented services and admission control is that the setup phase might introduce significant delays, especially due to the need to perform admission tests in

each node. To address this issue, we measured the various delays incurred during establishment.

The path used in the previous experiments, between `faith` at UC Berkeley and `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 is roughly constant for up to thirty established RTIP connections. The main factor which determines the latency is the path length. For example, for a shorter path from `grayling` 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 `grayling` 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)
`grayling`	3.906	3.907	3.674
`sock`	2.056	1.832	11.9
`breeze`	1.637	1.825	9.779
`redwood`	1.853	1.958	—

TABLE 5. Node and link latencies in channel establishment

`grayling` (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, our setup latency is dominated by propagation delays, protocol processing, and control message transport overheads (note that the control messages are transported over TCP/IP).

6 Lessons

We learned many lessons from designing, implementing, and experimenting with the Tenet Real-Time Protocol Suite. None of these could have been discerned 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 highlight 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. Of course, our work has not been performed in isolation, and we have benefitted and learned from continual interactions with others investigating the same or similar problems.

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 deterministic 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[9]. We now have more optimistic tests for both deterministic [32] and statistical [33] guarantees, and plan to experiment with them in Tenet Suite 2.

Multi-party support is needed: As we expected, the absence of multi-party support has been a hindrance to 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 the amounts of resources reserved are high. This problem is being solved in Suite 2 by providing multicast channels, resource sharing between related channels in the same conference [34], and higher-level abstractions to simplify connection setup for multi-party communication [7].

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., pre-planned conferences in which participants are known in advance), some by *receiver-initiated* establishment (e.g., seminar distribution)[10], and some by

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

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

duplex establishment (e.g., videophone). Although RCAP was designed for source-initiated establishment, our establishment procedure can be extended to these other cases as well.

Separation of control and delivery protocols facilitates development: The separation of our signalling and data-delivery protocols, and the decision to implement RCAP completely in user space, facilitated the debugging and porting of the implementations.

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 of RCAP parameters suitable?). Others are more far-reaching (e.g., what is an appropriate traffic representation?). Other issues are fundamental to real-time networking research (e.g., are deterministic and statistical guarantees necessary and usable, and how do they translate to charges for network usage?) Unfortunately, answers to the more fundamental questions require real users and applications, and will have to wait for real-time protocols (ours and others) to be deployed in production environments.

7 Conclusions and Future Work

While we made some simplifications in both the design and implementation of the Tenet Suite, we feel that the project proved the feasibility and the practicality of the Tenet approach to real-time communication. The result is the first suite of network and transport-layer protocols capable of transferring data 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; a prototype implementation is almost complete, and multicast channel establishment has already been demonstrated. Suite 2 supports multi-party communication, including routing and establishment of multicast connections and resource sharing [7]. 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 while preserving the

mathematical guarantees of the Tenet approach [34]. Suite 2 also supports partitioning of network resources into independent “virtual networks” [35] and supports advance reservation of resources for future real-time channels [36].

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 [21]. DCM has also been used and augmented to support fault tolerance and fault recovery [13] [37]. Future work includes implementation and experimentation with these new algorithms and mechanisms.

8 Acknowledgments

We are grateful to all past and current members of the Tenet Group for their contributions to the design and implementation of Tenet Suite 1, to the experiments performed on it, and to the contents and readability of this paper. We are also indebted to Lixia Zhang and the anonymous referees for their helpful comments on this manuscript and suggestions for its improvement.

9 References

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

[2] 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.

[3] 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.

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

[5] M. Moran and R. Gusella, “System Support for Efficient Dynamically-Configurable Multi-Party Interactive Multimedia Applications,” *Proc. Third International Workshop on Network and Operating System Support for Digital Audio and Video*, San Diego, CA, November 1992.

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

[7] 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 (NOSSDAV '93)*, Lancaster, England, November 1993.

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

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

[10] L. Delgrossi, R. G. Herrtwich, and F. O. Hoffman, "An Implementation of ST-II for the Heidelberg Transport System," IBM ENC TR-43.9303, *Journal of Internetworking: Research and Experience, 5*, 2, June 1994.

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

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

[13] A. Banerjea, "Fault Management for Realtime Networks," Ph.D. dissertation, University of California at Berkeley, December, 1994.

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

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

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

[17] 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.

[18] 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 (NOSSDAV '91)*, Heidelberg, Germany, November 1991.

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

[20] 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.

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

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

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

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

[25] 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.

[26] 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.

[27] H. Zhang and D. Ferrari, “Rate-Controlled Service Disciplines,” *Journal of High-Speed Networks 3*, 4, 1994.

[28] H. Zhang and D. Ferrari, “Rate-Controlled Static Priority Queueing,” *Proc. IEEE INFOCOM’93*, San Francisco, CA, September 1993.

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

[30] 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 (NOSSDAV‘92)*, San Diego, CA, November 1992.

[31] 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.

[32] 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.

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

[34] A. Gupta, W. Howe, M. Moran, and Q. Nguyen, “Scalable Resource Reservation for Multi-Party Real-Time Communication,” *Proc. INFOCOM‘95*, April 1995.

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

[36] D. Ferrari, A. Gupta, and G. Ventre, “Distributed Advance Reservation of Real-Time Connections,” *Proc. Fifth International Workshop on Network and Operating System Support for Digital Audio and Video, (NOSSDAV‘95)*, Durham, NH, April 1995.

[37] A. Banerjea, C. Parris, and D. Ferrari, “Recovering Guaranteed Performance Service Connections from Single and Multiple Faults,” *Proc. GLOBECOM‘94*, San Francisco, CA, November 1994.