A Framework for Interactive Multicast Data Transport in the Internet

by

Suchitra Raman

B.Tech. (Indian Institute of Technology, Madras, India) 1996
M.S. (University of California, Berkeley) 1998

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Steven R. McCanne, Chair
Professor Randy H. Katz
Professor Kenneth Goldberg

2000

The dissertation of *Suchitra Raman* is approved:

Chair Date

Date

Date

University of California at Berkeley

2000

A Framework for Interactive Multicast Data Transport in the Internet

Abstract

A Framework for Interactive Multicast Data Transport in the Internet

by

Suchitra Raman
Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Steven R. McCanne, Chair

The remarkable growth of the Internet as the a data transmission medium has in part been enabled by the simplicity and scalability of the core Internet Protocol (IP), which is used for addressing and routing unicast data packets through the network. The IP service model does not provide any packet delivery guarantees, but rather provides a "best-effort" contract, and leaves it to higher layers to provide enhanced services using this basic service. Today, the *de facto* transport protocol on the Internet is the Transmission Control Protocol (TCP) [109, 128]. TCP was designed primarily for applications such as *telnet*, a remote terminal application, and *ftp*, a file transfer application, which require data to be delivered reliably and in an ordered manner. While the TCP abstraction and protocol are indeed invaluable for transporting simple data types in *telnet* and *ftp*, as well as other applications with straightforward reliability requirements, two notable changes make the TCP approach inappropriate both with respect to its restrictive delivery semantics as well as its internal algorithms for loss recovery. First, TCP is not compatible with extensions to the basic IP service model for providing network-layer multicast. Second, the emergence of rich media types and applications creates a need for transport protocol requirements that are not satisfied by TCP's restricted semantics. The newer applications are interactive and handle these special media types in special ways. For example, an image viewer that does not rely on TCP may handle JPEG image data delivered out of order and reconstruct missing portions using interpolation techniques, thereby enhancing the interactivity to the end user. Such applications require sophisticated delivery semantics and are not best served by an overly restrictive protocol such as TCP. What is required here is a transport protocol whose semantics can be tailored by the application for efficient network transmission.

Our approach to solving these issues is a soft state-based interactive multicast data transport protocol framework. We present a model for "soft state" as an end-to-end construct that enables loose state synchronization between sender and receivers. We treat protocol control state at the end points as "soft" by not requiring that it be perfectly consistent at all instants. This allows us to avoid tight sender-receiver synchronization, as in TCP-like instantaneous receiver acknowledgements. Our soft state-based transport protocol provides a *relaxed reliability*, instead of TCP-like deterministic reliability.

Second, to accommodate heterogeneity among receivers and network paths, we allow receivers to tailor the semantics of reliability. Hence, a receiver incapable of or uninterested in processing portions of the data stream may refrain from receiving it reliably. We do not rely on the transport-level sequence space, but rather, use application-specific namespaces to express receiver

preferences while requesting retransmissions. This application-level namespace is exposed to the transport protocol and is used by the receiver to selectively retrieve specific data items. The use of such a common "vocabulary" to describe data puts the application in control of loss recovery.

Finally, since many new data types including certain image formats can be processed and rendered out of order at the receiver, we do not enforce a TCP-like delivery order on the data stream. Instead, we provide out-of-order delivery to the receiving application and demonstrate its benefits for image delivery. This specific technique is also applicable to unicast transmission and we design and implement an interactive image transmission protocol for use in the World Wide Web.

These techniques form the bases of the new transport protocol framework for interactive multicast data transport. Our transport protocol is layered on top of UDP [108] in the protocol stack, and we have implemented it as a user-level library called *libsstp*, a library for soft state-based transport protocol. We also present probabilistic analyses of the performance of our protocol in terms of the performance of the basic algorithms for loss recovery, using "slotting and damping," as well as the tradeoffs involving consistency and bandwidth consumption.

Professor Steven R. McCanne
Dissertation Committee Chair

To my parents, Lakshmi and Harihar Raman

Contents

List of Figures

List of Tables

Acknowledgements

This thesis was made possible by the help, support and guidance of several people. First, I would like to thank my advisor Steven McCanne for giving me a chance to work with him during the past four years. It has been my pleasure and honor to be associated with him right from the days when he was a young Professor at Berkeley to his savvy Internet entrepreneur days. Steve has been a constant source of encouragement and guidance at every step in my graduate career. He is an excellent researcher and maintains the highest standards for himself and others surrounding him. I hope that I will inherit his high standards from him and live up to them! I owe Steve a special thanks for supporting me when I decided to move to Cambridge.

I was most fortunate to be advised by a stellar dissertation committee: Professors Steven McCanne, Randy Katz, and Kenneth Goldberg. I am grateful to them for their advice and support during the latter stages of my thesis research. This thesis would not be possible but for Randy's generosity, when he accepted me into Berkeley as a research assistant four years ago. Ken Goldberg offered invaluable insight into my analysis of "soft state" and pointed out some interesting related work in the field of Bayesian analysis. Professor Joseph Hellerstein served on my qualifying examination committee and I have enjoyed my conversations with Joe. He has always given his 100% to the task at hand no matter how small or big. His advice has helped me to sharpen my work and writing.

In addition to my committee members, I had the good fortune of working with Professor John Guttag at MIT. John was a "father figure" and a constant source of wisdom to me. He has always encouraged me to be bold when picking research problems to tackle and has provided critical feedback on my work. I would like to thank him for supporting me during the last two years of my graduate career by providing me with an office here in LCS and more importantly, introducing me to a most vibrant and stimulating atmosphere.

I spent two summers of graduate school in industry internships. In addition to being a welcome hiatus from the routine of graduate school, each of these was an ideal chance to interact with top-notch researchers and practitioners from the industry. During the summer of 1997, I was a research intern with Scott Shenker and Lee Breslau at Xerox PARC. I also gratefully acknowledge Scott Shenker for participating in my qualifying exam committee and providing me with insightful comments on the soft state model. Scott is a passionate researcher and has a keen modeling mind that he applies very well to real and relevant problems. Some of my own affinity for modeling and analysis is acquired from my interaction with him. Scott's modesty and humility are rare in a person of his stature. Lee Breslau was always extremely helpful to me, especially as I learned my way around the ns-2 simulator.

Again in the summer of 1999, I took a break from graduate school to learn more about multicast routing. I was most fortunate to work with Radia Perlman at the Sun Laboratories Boston Center for Networking, who at the time was deeply involved in a most interesting debate on what the wide-area multicast protocol for the Internet should be. I was fortunate to witness and participate in this debate at such close quarters and that has strengthened my understanding of the Internet Protocol and extensions to the basic service model and had the opportunity to experience IETF and its amazing processes. I would also like to thank Dah-Ming Chiu, Miriam Kadansky, Phil Rosenzweig, and Joseph Wesley for making my summer at Sun Laboratories a fun and fruitful experience.

An integral part of the Berkeley systems student's graduate career is the semi-annual research retreat at either Lake Tahoe or Monterey Bay. Under workshop-like settings at these retreats, we presented works in progress. Jean-Chrysostome Bolot, Steve Casner, Steve Deering, Deborah Estrin, Sally Floyd, Mark Handley, Kevin Mills, Srini Seshan, Brian Smith, and other MASH retreat participants provided immensely useful feedback on various aspects of the MASH system and protocols, in the early conceptual stages when it mattered the most.

Some of the wonderful times I had in graduate school were outside of the EECS department! I would like to thank Professor Kenneth French of the MIT Sloan School for his masterful lectures in finance theory that broadened my understanding of capital markets, and provided food for thought, especially in the crazy times we are in. Ken and my brother Sundaresh have converted me into a markets aficionado by exposing me to two very different views of the markets in our long and inspiring discussions. It is indeed fascinating how markets are so *human,* and show the same exaggerated reactions that we do!

The years I spent in graduate school were most enriched by my day-to-day interactions with fellow graduate students — Elan Amir, Yatin Chawathe, Gene Cheung, Todd Hodes, Ketan Mayer-Patel, Matt Podolsky, Sylvia Ratnasamy, Cindy Romer, Angela Schuett, Wilson So, Andrew Swan, Teck-Lee Tung, Helen Wang, Tina Wong, and Kristin Wright. I would also like to thank Brian Shiratsuki for ensuring that our systems and networks were functioning 24x7 and for being very responsive every time I called him with a complaint from MIT. I would also like to thank Murari Srinivasan a friend and colleague with whom I worked on the image transport protocol for JPEG. His expertise in image processing and my familiarity with network protocols were combined well in our successful collaboration.

This thesis is a bicoastal production and I thank several people both at Berkeley and MIT for helping me with administrative matters. Kathryn Crabtree and Peggy Lau, who were in-charge of graduate student matters at Berkeley, executed their duties most smoothly, making my interaction with the department and the graduate overseeing committees hassle-free. I especially thank them both for taking such good care of my innumerable petitions sent to them from afar! Thanks also to Theresa-Lessard Smith and Bob Miller, who managed our research group matters with utmost efficiency. I would also like to express my thanks to Marilyn Pierce, Dan Engelhardt and Dan Wilson on the east coast, and the MIT LCS/EECS administration for making their resources available to me.

My parents Harihar and Lakshmi Raman and my brother Sundaresh have given me their love, affection and support at all times and have encouraged me to aim high. They have taught me the importance of virtue and provided me with the "lenses" to view the world and its people with, for those alone matter. Sundaresh and his loving family gave me much support during the last several years and I thank them for this. I would like to thank my husband Hari Balakrishnan and the Balakrishnan family for all their help and support during the last several years. Hari's positive energy has been an endless source of inspiration and enthusiasm for me, especially during difficult times. Hari, I owe this to you, and I hope that I can continue to give you my undivided love and support.

Suchitra Raman
May 2000

Chapter 1

Introduction

I will finish what I sta

— Bart Simpson's chalkboard exercise,
The Simpsons, Episode 8F05

1.1 Motivation

The last decade has seen a significant amount of research and commercial activity on the Internet. By every measure, e.g., the number of hosts, the number of registered domains, and the number of bytes transferred, the Internet is experiencing a tremendous expansion in size. One recent report estimates that the number of hosts on the Internet has grown from about 15 million in September 1996 to about 70 million in February 2000 [92]. In addition to the rapid growth in size, the Internet has also seen an expansion in the number and types of applications in use. Indeed, the power of the Internet is, to a large extent, due to the variety of applications it supports. While conventional applications such as electronic mail, file transfer and remote login continue to be widely used, applications such as the World Wide Web (WWW) [144], audio/video delivery, and collaborative applications are increasing in popularity.

There are several factors that have fueled this remarkable growth of the Internet as a data transmission medium. One of the main ones is the simplicity and scalability of the core Internet Protocol (IP), which is used for addressing and routing unicast data packets through the network. A packet transmitted by a host is forwarded along a path of routers and eventually reaches the destination. Each IP router looks up the destination address in the header of a packet and forwards it appropriately towards the destination based on its routing table that contains reachability information. IP routers behave as nodes that "store-and-forward" packets, but since router memory available for queueing packets is limited, incoming packets may sometimes be dropped. The IP service model does not provide any packet delivery guarantees, but rather provides a "best-effort" contract. Under this contract, when the network accepts a datagram, it provides no guarantee that packets will be successfully delivered to a receivers in the order in which they were sent or delivered in a timely manner, or that exactly one copy of the packet will be delivered.

IP leaves it to higher layers to provide refined versions of this basic service. For example, the end-to-end transport layer of the protocol stack is responsible for shielding the application from packet loss by providing loss recovery, congestion control and bandwidth management, as well as connection management in the case of connection-oriented unicast transport. Today, the *de facto* transport protocol on the Internet is the Transmission Control Protocol (TCP) [109, 128]. TCP was designed primarily for applications such as *telnet*, a remote terminal application, and *ftp*, a file transfer application which require data to be delivered reliably and in an ordered manner to the receiving application. TCP is a connection-oriented unicast transport protocol that provides a reliable, byte-stream abstraction in which the bytes are delivered to the receiving application in the same order that they were transmitted at the sender. TCP is responsible for connection management, loss recovery, and flow and congestion control. The TCP sender uses positive acknowledgement messages (ACKs) transmitted by the receiver to detect and retransmit lost segments. TCP also infers network congestion from packet loss and adapts to it by reducing its sending rate. The TCP abstraction and protocol is valuable for transporting simple data types, as in *telnet* and *ftp*, as well as for applications with straightforward reliability requirements.

However, TCP has significant limitations as a universal data transport protocol both with respect to its restrictive delivery semantics as well as its internal algorithms for loss recovery. We discuss two important factors that interfere with TCP's effectiveness as a data transport protocol.

First, TCP is incompatible with extensions to the basic IP service model for providing network-layer multicast. The core Internet Protocol service model was extended in 1989 to accommodate efficient multi-point communication, i.e., one-to-many and many-to-many communication. In this model, the network delivers a packet from a source to an arbitrary number of receivers. As in unicast IP, the IP multicast service model is "best effort" and provides no delivery guarantees. In multicast, the network delivers a packet from a source to an arbitrary number of receivers by forwarding a copy of that packet along each link of a distribution tree. Senders simply send their packets to an abstract "group address" and receivers express their interest in receiving these packets by joining the corresponding multicast group through a group membership protocol. The collection of senders and receivers exchanging data over a common multicast group is often called a *multicast session* or simply a *session*.

Since the introduction of IP Multicast [30] almost a decade ago, a great deal of innovation has occurred in the area of multicast-based applications. A vast array of real-time video [41, 135, 84] and audio [64, 54] conferencing as well as playback, shared whiteboard [63, 133], and large-scale file or software distribution [42] applications have been developed, that benefit from the extended service model. Many of these applications require a transport protocol that provides an effective loss recovery scheme. Merely extending the ACK-based TCP retransmission algorithm to provide this reliability causes multiple ACKs, one per receiver, to be sent back to the sender, imploding the source and congesting the routers on the path toward the source. Worse, the severity of ACK implosion increases with the size of the multicast group to which the sender is transmitting. Since the performance benefits of the underlying multicast distribution mechanism are, in general, greater when group sizes are larger, an ideal multicast transport protocol must work well for large group sizes.

The second key factor that impedes TCP is the dominant trend in the current Internet towards the use of richer and more diverse data types and applications to handle them. The emergence of such rich media types and applications creates a class of new transport protocol requirements that not satisfied by TCP's semantics. For instance, there are about 200 MIME types within eight main categories registered with the Internet Assigned Numbers Authority (IANA) alone. In addition to these registered types, more than 75 unregistered MIME types proliferate today's Web sites [59].

Each of these represents a distinct media type or format ranging in complexity from plain text to layered JPEG images. The newer applications that handle such media types do so in special ways. An example of this is an image viewer that can handle JPEG image data delivered out of order and reconstructs missing fragments of an image using interpolation techniques. Such applications require sophisticated delivery semantics and may not be best served by an overly restrictive protocol such as TCP. What is required here, is a transport protocol whose semantics can be tailored by the application for efficient network transmission.

1.2 The Problem

Several issues render the multicast transport problem more challenging than the corresponding unicast transport problem. The conventional approach to providing reliability for unicast relies on *tight synchronization* between sender and receiver. One way to achieve such close synchronization between sender and receivers in a multicast session is for the receivers to regularly send feedback messages to the sender reporting the delivery status of transmitted data. This single receiver approach does not, however, work well in the multiple receivers case. For example, suppose we extended unicast TCP [109, 128] in a straightforward manner to the multicast scenario by having each receiver send a feedback message to the source. For example, in Figure 1.1, packet 1 transmitted by sender A is lost. Receivers B, C, and D detect this loss and respond by transmitting a NACK or a request for repair back to the sender every time a packet is detected as lost. However, this results in multiple copies of the same message being transmitted from each receiver back to the sender. In large groups, this synchronized behavior by the receivers causes an "implosion" of feedback messages at the source, increasing its packet processing overhead and also causing congestion on the path. Hence, a truly scalable multicast transport protocol must refrain from requiring tight synchronization between the sender and every receiver in the group.

A second important challenge for a scalable multicast transport protocol is robust operation in the face of network failures. One of the fundamental design goals of the IP architecture has been to ensure that end to end flows are relatively unaffected by the failure of individual portions of the network. This has considerably influenced the design of the underlying network and has lead to today's packet switched Internet. In packet switched networks, all address information required to route a packet is carried in the packet headers. IP routers do not maintain any flow-specific state that is critical to the flow. Therefore, while unicast transport was designed to be "survivable" in the event of router failures, not much attention was paid to surviving end point failures. This was justified, since unicast communication is meaningless when one or the other communicating party has failed.

Robustness and fault tolerance are in general desirable goals; but they are especially important in the case of scalable multicast protocols since large groups have inherently dynamic membership [2] and are prone to individual host failures. If p_f is the probability of host failure or network failure that disconnects a host from the rest of the session, the probability that an ensemble with N hosts operates successfully is $(1 - (1 - p_f)^N)$. As shown in Figure 1.2, this probability reaches close to 1 even for moderately sized groups and failure probabilities. Hence, a robust and scalable multicast transport protocol designed to work well with large groups should be insensitive to dynamically changing group membership as well as host failures.

The other important aspects of a transport protocol are the semantics of reliability and

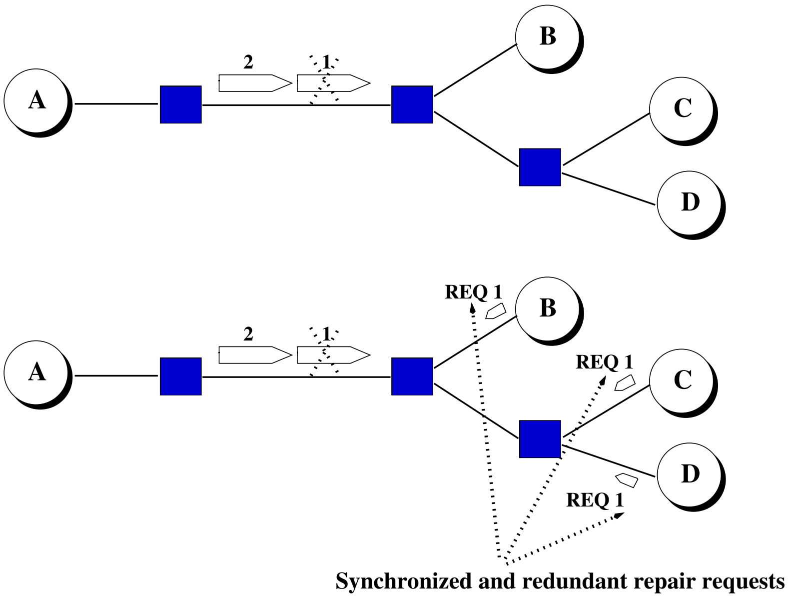

Figure 1.1: Large sessions suffer from the feedback implosion problem if each receiver transmits a feedback message instantaneously. Here, receivers B, C, and D each detect the loss of packet 1 and transmit a request for retransmission.

ordering provided to the application. The traditional approach is to provide sender-based complete reliability, i.e., the sender ensures that all its data is received. The sender's transmission can be tailored to the requirements of the only receiver in unicast, and as a result both are said to have share a common "goal" for the transmission. However, extending this single-point sender-based control does not scale well to the multicast case. It is hardly acceptable to impose a single sender-driven behavior in such a heterogeneous session composed of diverse hosts (from powerful desktops to impoverished "thin" clients), or networks paths ranging from low bandwidth, loss-prone wireless links to high bandwidth satellite links. For example, how must the round-trip time be estimated for such a diverse session? Should we adapt to network congestion and throttle the sender down to the least well-connected member? In addition, depending on user preferences and host capabilities, members in a multicast session may have different reliability requirements. A scalable transport protocol must support the wide range of heterogeneity in network paths and end host capabilities, as well as differences in the semantics of reliability within the context of a single multicast session. Hence, merely extending sender-driven control algorithms does not work well for multicast because the "fate sharing" inherent in unicast communication does not scale well to multicast.

One of the main contributions of this work is to show how receivers within a single multicast session can tailor the sender's transmission to reliably retrieve only those portions of the data stream that are relevant, thereby reducing the amount of wasted bandwidth. We first observe that this flexibility in fine-grained control over reliability semantics is not possible to achieve using the traditional layered protocol architecture in which the application and transport layers do not share a common vocabulary to define data items within the sender's transmission sequence. For example, when a receiver detects that bytes 1456 – 2912 of a transmission are lost, it does not know the

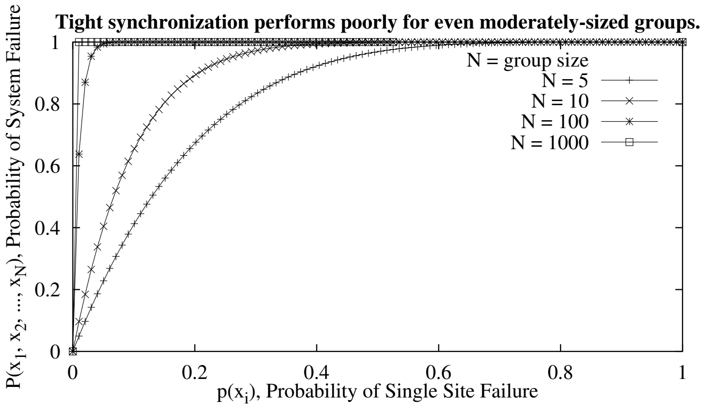

Figure 1.2: A transport protocol that uses tight synchronization between end points is highly sensitive to failures of individual sites.

corresponding application objects. This makes it impossible for the receiver to tailor its requests for retransmission based solely on transport level identifiers. We propose that the application use a generic namespace defined by application-level boundaries and expose it to the transport layer when naming data units. A receiver uses these application-level boundaries and corresponding names to request retransmission of lost data. Since data are no longer named as a sequence range relative to a specific source's sequence space, this approach also has the added benefit of distributing the responsibility of retransmission to all eligible members of the session. Hence, any member that has the requested data available to them is eligible to respond to the request, thereby making the system robust against sender failures.

The emergence of the World Wide Web has led to the appearance of diverse data types available through it, for example, images constitute a significant and growing fraction of traffic on the World Wide Web. According to a recent study, JPEG (Joint Photographic Experts Group) format images account for 31% of bytes transferred and 16% of documents downloaded in a client trace [46]. The HyperText Transport Protocol (HTTP) [37] uses TCP [109] to transmit images on the Web. While the use of TCP achieves both reliable data delivery and sound congestion control, these come at a cost—interactive latency is often significantly large and leads to images being rendered in “fits and starts” rather than in a smooth way. The reason for this is that TCP is ill-suited to transporting latency-sensitive images over loss-prone networks where losses occur because of congestion or packet corruption. When one or more segments in a window of transmitted data are lost in TCP, later segments often arrive out-of-order at the receiver. In general, these segments correspond to portions of an image that may be handled upon arrival by the application, but the in-order delivery abstraction imposed by TCP holds up the delivery of these out-of-order segments to the *application* until the earlier lost segments are recovered. As a result, the image decoder at the receiver cannot process information even though it is available at the lower transport layer. The image is therefore rendered in bursts interspersed with long delays rather than smoothly.

Earlier work on the scalable reliable multicast protocol (SRM) [40] recognizes the problem of ACK-implosion and proposes a distributed and randomized control algorithm to limit the amount of feedback traffic generated in a multicast session. In addition, this work also introduced the notion of a *protocol framework* to perform receiver-driven reliability, where receivers control which data items should be fetched. However, while the paper argues in favor of such a receiver-driven reliable transport framework, no specific mechanisms were put forth to realize it.

The TCP-like in-order delivery abstraction is appropriate for image encodings in which incoming data at the receiver can only be handled in the order it was transmitted by the sender. Some compression formats are indeed constrained in this manner, e.g., the Graphical Interchange Format, GIF [44] which uses lossless LZW compression [72, 142] on the entire image. However, while some compression formats require fully reliable and in-order delivery, several others do not. Notable examples of formats that encourage out-of-order receiver processing include JPEG [141, 104] and the emerging JPEG2000 standard [67]. In these cases, a transport protocol that facilitates out-of-order data delivery allows the application to process and render portions of an image as they arrive, improving the interactivity and perceived responsiveness of image downloads. Such a protocol also enables the image decoder at the receiver to implement effective error concealment algorithms on partially received portions of an image, further improving perceived quality. Ideally, the transport protocol must provide a basic set of loss recovery and data delivery semantics that can be customized for specific applications. Our proposal for such a multicast transport protocol is similar in spirit to earlier work on application-level framing (ALF) [23] that advocates greater application control over traditional transport functions, including loss recovery and delivery order. However, in this work, we present a more complete solution by providing a generic transport protocol and show how its mechanisms can be customized to achieve desired application behavior.

In summary, the emergence of multicast as well as the emergence of new types of data at the application layer have rendered TCP or TCP-like mechanisms ineffective as a transport protocol both in terms of abstraction (delivery order and reliability semantics) as well as mechanism (feedback management) on the Internet. In this dissertation, we develop a truly receiver-reliable protocol framework for effectively transporting interactive data via multicast on the Internet. We address issues of data naming, data delivery semantics and protocol robustness. We also address the challenges posed by new data types and show how data type-specific optimizations may be supported in a generic transport protocol framework.

1.3 Our Solution: Interactive Multicast Transport Using Soft State

Our approach to solving the aforementioned issues for interactive multicast data transport based on “soft” protocol state. We present a model for “soft state” as an end-to-end construct that enables loose state synchronization between sender and receivers. We propose this as a basic building block for constructing a robust and scalable transport framework for interactive multicast applications. Here, we take a different approach from TCP and treat protocol control state as soft state. To accommodate heterogeneity among receivers and network paths, we allow receivers to tailor the semantics of reliability. Hence, a receiver incapable of or uninterested in processing portions of the data stream may refrain from receiving it reliably. We do not rely on the transport-level sequence space, but instead, use application-specific namespaces to express receiver preferences while requesting retransmissions. Finally, since many new data types including certain image formats can be processed and rendered out of order at the receiver, we do not enforce a TCP-like delivery order on the data stream. Instead, we provide out-of-order delivery to the receiving application and demonstrate its benefits for image delivery.

These techniques form the bases of the new soft state-based transport protocol framework (SSTP) for interactive multicast data transport. Our transport protocol is layered on top of UDP [108] in the protocol stack, and implemented as a user-level library called *libsstp*, a library for soft state-based reliable multicast, as shown in Figure 1.3. In the remainder of this section, we present more details on our approach.

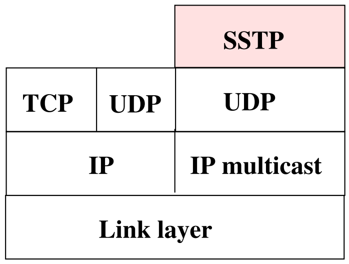

Figure 1.3: libsstp is a user-level implementation of soft state-based reliable multicast that runs over UDP.

1.3.1 Soft State as a Data Transport Primitive

The notion of soft state has been in popular use among routing protocol designers since the advent of multicast routing. Multicast routers maintain membership information on behalf of hosts or other networks downstream within their multicast routing tables. This membership information, ultimately generated by receivers signalling to their upstream routers, is used to make forwarding decisions for multicast data flows. The traditional approach to state management treats the state as "hard" or assumes that the state is valid unless and until explicitly deallocated by end hosts. However, it is crucial for routing protocols to design components within the network that are "forgiving" and self-healing in the face of buggy end system implementations. In multicast routing, routers defend against buggy hosts that fail to signal to the routers when they leave a group. Instead of using the hard state approach, multicast routers periodically expire the membership state in their tables, unless explicitly re-registered by a downstream router or host. While this notion of soft state is generally agreed upon as a defensive measure by routing protocol designers, its properties are not well-understood.

In our work, we first propose a formal model for soft state that treats soft state as an end-to-end construct. We then use it to maintain loose synchronization between the sender and receivers. The soft state model is simple: state at the end hosts is treated as "soft" and relies on periodic refreshes of this state by the session members. Our basic data model is an abstract table of {key, value} pairs. We also discuss how this data model can be extended to represent large repositories of data scalably organized hierarchically. We define a probabilistic metric called *consistency* to evaluate and compare the different soft state-based protocols. Based on our model, we evaluate the performance of different variants of soft state-based protocols and show how the bandwidth dedicated to control state refreshes affects the overall performance of the protocol. We theoretically analyze our model for the simple open-loop announce/listen protocol. Based on this model, we systematically characterize data consistency and performance tradeoffs of our soft state model under a range of workloads and network loss rates for the simple open-loop case and its variants. We also extend the open-loop variant of announce/listen by adding receiver feedback to enhance data consistency and performance without increasing network resource consumption. Based on our model, as well as the observation that several protocols have inherently "soft" or periodically changing data, e.g., route advertisements [86, 55, 78], DNS updates [88], MBone session directories [53], stock quote or general information dissemination services [107], we propose a soft state-based transport protocol framework. This framework provides a parameterized spectrum of reliability semantics all derived from one framework — from simple announce/listen communication to feedback-based reliable transport. The framework also optimally allocates bandwidth based on packet loss rates and application workload to maximize consistency. The result is a parameterized framework that can be tuned to provide one of a continuum of "reliability levels".

We explore the use of feedback from receivers to enhance the consistency provided by sender-driven soft-state refreshes. However, since instantaneous and uncontrolled feedback can implode the sender, we investigate a rate-controlled receiver feedback. Since we do not rely on instantaneous positive feedback, our soft state communication entails a *probabilistic delivery model* with relaxed reliability. This is in contrast to the deterministic guarantees provided by TCP, which uses instantaneous feedback messages from the receiver.

1.3.2 Receiver-driven Reliability

The traditional approach to providing reliability, *a la* TCP, is to recover all lost transmissions. As described in Section 1.2, this sender-based TCP approach does not extend well to the multicast case. The sender-based approach is also better suited for the data types and applications, e.g., *ftp* and *telnet*, prevalent during the design of TCP, which did not have the ability to process partially received data. However, modern applications such as image browsers, distributed shared interactive whiteboards, and interactive directory services, do not require *all* data to be reliably delivered, but rather require a transport protocol that allows the receiver to tune its reception.

Receiver and network heterogeneity are more significant in the context of multicast, where a single session may simultaneously span multiple hosts with varying levels of processing capacity and network paths. Here, each receiver must tailor a sender's data to reliably deliver only useful portions. Consider the example of the "thin" client application that runs on a device with a 2-bit gray-scale display. Such a device is unable to process and render high-resolution GIF images and does not require that portion of the transmitted data to be recovered reliably.

Transport layer sequence numbers, the traditional construct used to name and identify data items at the sender and receiver, are not enough to provide the richer and more flexible reliability semantics required for such applications. If a piece of data is lost in transit, the receiver has no means to discover to which portion of the application data stream the corresponding sequence number gap maps. Hence, more relaxed reliability semantics based on application requirements are difficult to architect. What is required is a richer naming structure that is shared between the transport and application. Such a naming structure must also support large data sets in a scalable manner. In our protocol framework, we show how to name data at the transport layer in a manner that allows the transport and application layers at the application to cooperatively decide if retransmissions are required.

The sender and receivers use an application-specific namespace to describe data. In addition to transmitting the payload, the sender also sends a piece of *meta data* or a signature or summary of the corresponding data describing the name, type, and other application-defined attributes. Since the semantics of the namespace and meta-data are generated in an application-specific fashion and are under application control, it is possible for the receiving application to decide, based on the meta-data information, whether the corresponding data item is to be recovered completely.

Receivers use the meta data and namespace information to schedule requests for retransmissions, and the effectiveness of their decisions is greatly improved when the namespace information is received without losses from the sender. Here, negative acknowledgement messages are generated in a scalable manner using randomized timers. This randomized timer technique is otherwise also referred to "slotting and damping" and has been used in other contexts ([21, 36, 4]) for limiting feedback in a large group. When a receiver decides to request a lost transmission from the sender, it uses a loss recovery algorithm based on "slotting and damping" similar to the SRM protocol. Feedback messages are controlled damping their transmissions and suppressing duplicate copies of the same message generated by different receivers. To ensure that the meta data and namespace information is delivered reliably, we use the same underlying slotting and damping-based loss recovery algorithm. While slotting and damping was previously applied to recover lost application data, we extend its use here to perform loss recovery on the naming data as well.

Hence, a richer naming system combined with an efficient name distribution mechanism allows receivers to tune their retransmissions and solves the end-point heterogeneity problem.

1.3.3 Out-of-Order Processing

Another important aspect of multicast as well as unicast transport protocols is the order in which data is delivered to the application running at the receiver. Simple data types such as `telnet` and `ftp` that lack structure within the application perform equally well with any delivery semantics. However, as we observed in Section 1.1, the number of different media types and formats is on the rise on the Internet. In addition, several of them are designed specifically for network transmission, making it possible to handle partially received data. This feature of network-optimized data formats, impacts our design of the application-transport interface. First, it dictates the framing boundaries for datagrams transmitted on the network, and more importantly, it enables receivers to process partially received data in an out-of-order fashion.

Even though the issue of data delivery order is important in unicast as well as multicast transport, we focus on the specific case of JPEG images in the context of unicast. Our choice here is motivated by the immediate application of our techniques to WWW transfers. However, the techniques presented for out-of-order delivery easily transition to the multicast case as well.

We first highlight the disadvantages of using TCP or a TCP-like in-order delivery protocol for image downloads. The main drawback of using TCP for image downloads is that its in-order delivery model interferes with interactivity. To demonstrate this, we conducted an experiment across a twenty-hop Internet path to download a 140 KByte image using HTTP 1.1 running over TCP. The loss rate experienced by this connection was 2.3%, 3 segments were lost during the entire transfer, and there were no sender retransmission timeouts.

Figure 1.4 shows a portion of the packet sequence trace obtained using `tcpdump` running at the receiver. We see a transmission window in which exactly one segment was lost, and all subsequent segments are received, causing the receiver to generate a sequence of duplicate ACKs. There are ten out-of-sequence segments received in all waiting in the TCP socket buffer, none of which is delivered to the image decoder application until the lost segment is received via a (fast)

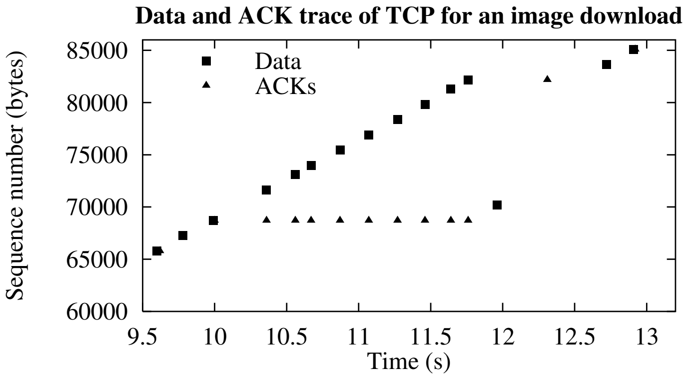

Figure 1.4: Portion of packet sequence trace of a TCP transfer of an image.

retransmission almost 2.2 seconds after the loss. During this time, the user sees no progress, but a discontinuous spurt occurs once this lost segment is retransmitted to the receiver, and several kilobytes worth of image data are passed up to the application.

To understand how ordering semantics influence the perceptual quality of the image, we conduct another experiment where the image is downloaded over TCP. We study the evolution of image "quality", as measured by peak signal-to-noise ratio (PSNR) [121] with respect to the original transmitted image. Figure 1.5 shows this for a transfer that experiences a 15% loss rate. We find that the quality remains unchanged for most of the transfer, due to an early segment loss, but rapidly rises upon recovery of that lost segment. A more gradual evolution in PSNR, as in the "ideal" transfer which does out-of-order delivery is desirable for better interactivity.

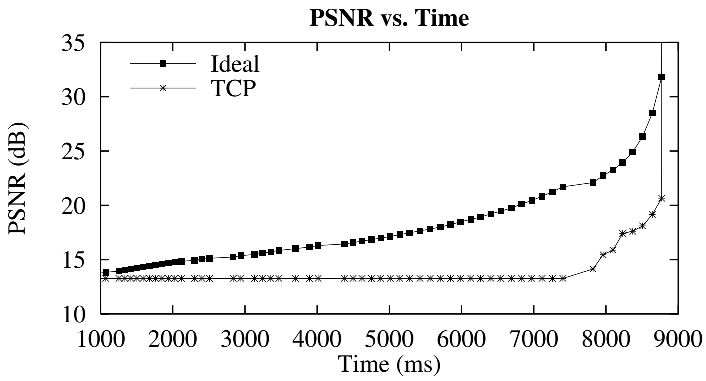

Figure 1.5: PSNR evolution of the rendered image at the receiver for a TCP transfer with 15% loss rate.

We observe that a design in which the underlying transport protocol delivers out-of-sequence data to the application might avoid the perceived latency buildup. In order to do this, the transport "layer" (or module) must be made aware of the application framing boundaries, such that each data unit is independently processible by the receiver. Hence, out-of-delivery combined with application-level framing can vastly improve the perceptual quality of the received images.

1.4 Contributions of Dissertation

The core focus of this dissertation is the design, development and analysis of techniques for an ALF-based transport protocol for interactive multicast applications. The key contributions of this thesis are as follows.

- **A Model for Soft State Transport:** We have developed an abstract mathematical model and provided a solution for soft state based communication that is applicable to a variety of applications. Using the model, we show how the performance of basic announce/listen protocols can be improved using special scheduling techniques at the sender as well as by the addition of receiver feedback. Through this model, we have unified several previous attempts, e.g., SAP [49] and SRM [40] at solving the multicast transport problem.

- **Scalable Naming and Announcement Protocol (SNAP):** We argue that pure transport level sequence numbers do not appropriately reflect the structure of application level data, making it impossible for resource-constrained receivers to selectively request retransmissions within a data stream. We show how to overcome the end host and network heterogeneity problem in multicast sessions using scalable data naming, and an efficient name dissemination protocol. These techniques allow receivers to tailor their reception to suit their local requirements.

- **Asymptotic Timer Analysis:** At the core of our transport protocol framework is a loss recovery scheme based on randomized receiver feedback. We carry out a detailed analysis and simulation study of the asymptotic behavior of this scheme. We show that the effectiveness of the scheme relies heavily on the nature of the underlying topology of the group.

- **libsstp:** We have implemented the above schemes as a generic, reusable user-level transport protocol framework called *libsstp*. Libsstp has a well-defined API that supports selective reliability, out-of-order delivery, and application-specific data naming. We have validated our protocol framework and implementation by developing a range of applications using it. libsstp has been used with significant success in building applications such as a real-time information dissemination service used for timely data such as weather and stock quotes; a collaborative shared whiteboard application called MediaBoard; a light-weight control protocol for software-based parallelized special effects video processing; a reliable multicast proxy service, and a distributed archival system.

- **Image Transport for the WWW:** We have applied the principle of out-of-order data delivery to JPEG image transport over the World Wide Web and demonstrate the benefits of this approach over HTTP/TCP. We have developed a specialized unicast-only transport protocol called ITP that is tailored for image transport on the WWW. We customize ITP even further for JPEG transport.

1.5 Overview of Dissertation

The remainder of this dissertation is organized as follows.

In the next Chapter, we describe the background and related work. Chapters 3, 4, 6, 7 address the core components of our transport framework.

In Chapter 3, we present the soft state model for multicast transport and analyze its performance under a variety of network conditions using the consistency metric. Soft state uses loose synchronization of protocol state between the end points making it ideally suited to large-scale multicast sessions with dynamic membership. We present the basic announce/listen model and show how its performance can be improved using special scheduling techniques at the sender that distinguishes data items based on *age,* as well as by adding rate-controlled receiver feedback. The announce/listen protocol presented in Chapter 3 uses a straightforward data model comprising a linear table of {*key, value*} pairs.

In Chapter 4, we address the issue of heterogeneity in large-scale multicast sessions. The combination of data naming and an efficient name dissemination scheme allowing receivers to tailor the semantics of reliability by selecting which data items need be reliably recovered. We also propose a hierarchical data model for the protocol to scale to large data stores.

An important component of our transport protocol framework is the randomized timer-based receiver-driven loss recovery scheme commonly termed "slotting and damping." In Chapter 5, we present a detailed simulation and analysis of this randomized timer scheme under various network conditions, especially in the case of large group sizes.

Chapter 6 demonstrates the benefits of out-of-order data delivery in the context of JPEG image transport on the World Wide Web. This work shows how partially received JPEG images can be processed out of order and further refinements applied to enhance the quality of the rendered image while the download progresses. This improves the interactivity of the download, as measured by peak signal-to-noise ratio (measure of likeness) with respect to the transmitted image.

In Chapter 7, we tie together the concepts presented in Chapters 3, 4, 5, 6 into *libsstp*, a reusable software implementation for use in interactive multicast data applications.

Finally, we conclude and present areas for future research in Chapter 8.

Chapter 2

Background and Related Work

"Progress, far from consisting in change, depends on retentiveness. Those who cannot remember the past are condemned to repeat it."

— George Santayana

"Change is inevitable, except from a vending machine."

— Bumper Sticker

In this Chapter, we survey background research and work related to our soft state-based transport framework. Section 2.1 presents network layer routing protocols for intra-domain as well as inter-domain multicast. We also briefly present the current status of deployment of IP multicast within the Internet and describe the multicast backbone (MBone). We then present the current state of the art in multicast transport. In Section 2.3, we survey open-loop "announce/listen" protocols. This is followed by a discussion of several end-to-end as well as router-assisted reliable multicast transport protocols in Section 2.4. We then discuss the semantics of data delivery in transport protocols in the context of unicast and multicast.

2.1 Overview of IP Multicast

The core IP service model comprising "best effort" unicast IP was extended in 1989 to include IP multicast for efficient wide-area network-layer multi-point data delivery [30]. IP multicast leads to efficient bandwidth utilization for one-to-many and many-to-many communication. Data delivery occurs by forwarding a copy of a packet along each link of a distribution tree. Besides providing efficient multi-point delivery, network layer multicast also provides a "group" abstraction in which a sender of data can refer to a group of receivers, without listing them explicitly.

IP multicast also reduces the load on the sender because a transmission is performed once per group. Subsequent duplication occurs at branch points along the distribution tree in the underlying multicast routing topology, as shown Figure 2.1. Senders simply send their packets to an abstract "group address" and receivers express their interest in receiving these packets by joining the corresponding multicast group address through a group membership protocol [36]. The abstract group address is selected from the special class D range (`224.*.*.*` through `239.*.*.*`) of IPv4 addresses, and serves as a handle or key to the entire multicast group, thus obviating the need for higher-level applications to maintain explicit membership lists.

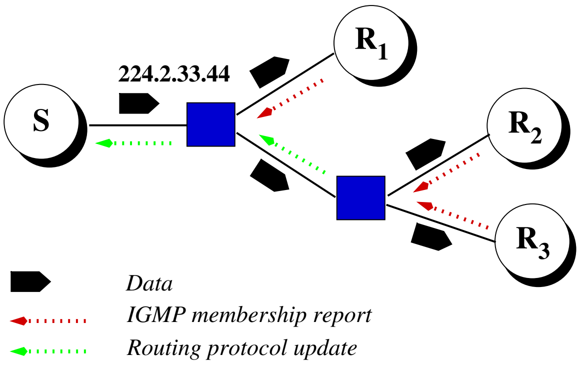

Figure 2.1: Receivers R_1, R_2, and R_3 register interest in multicast group 224.2.33.44. Multicast routers replicate and forward the data packet from the source S to all three receivers along a source-rooted tree constructed using DVMRP.

Much past work addresses the problem of multicast routing. We begin by discussing different designs for multicast routing protocols.

2.1.1 Flood and Prune Protocols

One class of routing protocols (DVMRP [30] and dense mode PIM [31]) involves "broadcast and prune," where traffic is flooded from the source, using reverse path forwarding [26]. Additionally, when a router R receives a multicast message from a source S with destination address G, and R has no neighbors that wish to receive traffic for (S, G), it sends a message in response, indicating that the neighbor should "prune (S, G)", i.e., should stop sending traffic for (S, G) to this router. This class of protocols does not support large numbers of groups with topologically distant members because of two drawbacks:

Too much flooded data.
To reach all potential receivers, flood and prune protocols must periodically flood data to reach all parts of the Internet. However, in practice, for a given receiver, only a very small portion of the groups would be of interest.

Too much prune state.
Each router R must remember all the (S, G) pairs it received from each neighbor (representing all the (S, G) pairs the neighbor is *not* interested in receiving), in addition to all the (S, G) pairs R has sent prunes for. In other words, the prune state in routers grows proportionally with the number of sources s and number of groups g that a router is *not* interested in!

2.1.2 Explicit Tree Protocols

The other class of protocols, (CBT [8], sparse mode PIM [29], and BGMP [71]) explicitly builds a shared tree based on a root C, so that only routers on the distribution path of a multicast group need to keep state about the group. CBT and BGMP create a bi-directional tree, whereas sparse mode PIM creates a uni-directional tree. The shared tree approach is more scalable since the router state does not grow as rapidly as in the dense mode protocols, since routing state is no longer maintained for groups in which there is no interest!

In addition, sparse mode PIM allows for switching between a state-efficient shared tree and a latency-optimal source-rooted tree. Each router decides if it is receiving a "sufficiently high volume" of traffic from a particular source, and if so, joins a tree rooted at that source, pruning itself from the shared tree. This dynamic switching between uni-directional shared tree and per-source trees is complex and has stability problems as per-source trees time out. Also, the root of the uni-directional shared tree becomes a bottleneck.

All the shared tree approaches require the use of periodic announcement messages to locate the rendezvous point, or RP, for a group, i.e., learn the mapping from the group address G to its RP. In sparse mode PIM, a bootstrap mechanism within a domain advertises candidate RPs and a hash function maps G to one of the set of candidate RPs. This mechanism does not scale beyond a domain because it is too expensive to do Internet-wide advertisements of the list of candidate RPs. In addition, this mechanism creates highly suboptimal trees if the candidate RP is selected using a hash function from among Internet-wide candidates, rather than being co-located with high bandwidth senders to optimize data paths.

There have been several recent proposals that specifically address the wide-area IP multicast routing problem. The inter-domain multicast routing protocol BGMP [71] proposes using a shared bi-directional distribution tree among domains such that any intra-domain protocol (i.e., DVMRP or PIM) can be run within each domain. Routing between BGMP domains requires that multicast address allocation reflect the underlying unicast network topology, or at least provide core location information. This alignment with the unicast routing hierarchy also makes BGMP routing entries aggregatable resulting in state savings in inter-domain routers. There is less consensus on how such address allocation is to be done in a scalable and deployable manner. Some approaches that have been suggested are:

- Multicast Address Set Claim (MASC) [71], a scheme proposed in conjunction with BGMP for dynamically assigning blocks of multicast addresses to each domain, and using inter-domain unicast routing, e.g., BGP [78] to distribute reachability information. Once R is localized in this manner to a domain, a mechanism such as PIM bootstrap is used to map G to the RP within that domain. While we feel that the shared bi-directional inter-domain tree architecture in BGMP is a scalable distribution mechanism, we are less convinced that the MASC architecture is sufficiently dynamic and free from allocation conflicts, especially in the face of network partitions. If multicast addresses need to be allocated in blocks to domains, either statically or dynamically, multicast addresses will become a scarce resource.
- GLOP addressing [87] is a static assignment of multicast addresses based on unicast domains, in which each domain is assigned 256 multicast addresses. 256 addresses per domain are not sufficient to support anything but a very restricted set of applications (perhaps a few streams broadcast by an ISP). Another scheme [94] assigns class D addresses based on 24 bits of the unicast address space. This scheme cannot be used along with "routing realms" connected to the rest of the Internet via network address translators (NATs) [77] that do not have any globally assigned unique unicast addresses, and therefore would not have any multicast addresses.

Another proposal to overcome the wide-area rendezvous problem is MSDP [35], the Multicast Source Discovery Protocol. MSDP is a scheme in which tunnels are configured between candidate RPs in various domains. When a source S transmits on group G, knowledge that (S, G) is an active (source, group) pair is flooded throughout all domains. This scheme suffers from severe scaling problems if many sources and groups are active simultaneously.

Recent research has revisited the basic abstraction of a group, since perhaps it was too ambitious and generalized. This has lead to the development of at least two independent proposals [57, 105] that argue in favor of a modified abstraction for a group that is less general, but affords a more scalable wide-area routing protocol. One such scheme is the EXPRESS multicast model, which explicitly names the source of data in the group address. Hence, the group is identified by the 8-byte quantity (S, G), where G is a group identifier with respect to S. Another scheme, the "Simple Multicast" routing protocol (SM) proposes extending the multicast address architecture by making end hosts aware of the core router of the multicast distribution tree. SM overcomes the core location problem in the wide-area by explicitly distributing the core address at the application layer along with the 4-byte group address. In this scheme, the group "address" (G) is extended with the unicast address (C) of the core or RP. Hence, the new extended group identifier is (C, G). The additional address bytes may be carried in an IP option or "next header," following the IP header.

Both EXPRESS and SM mitigate the difficult problem of globally coordinated multicast address allocation by localizing address management to a single node. Since G is unique with respect to the root R of the distribution tree (i.e., the source S in EXPRESS, or the core C in SM), there is no need for a separate address allocation infrastructure. The key difference between EXPRESS and SM is their sender model. At the network layer, EXPRESS supports data delivery from only one source per group, whereas SM preserves the source model of the existing IP multicast by providing support for multiple senders. The designers of the EXPRESS protocol make the strong assumption that the only important application requiring efficient wide-area multicast on a large-scale is IP television. Other existing applications using transports such as RTP [125] and SRM [40] require network layer support for many-to-many communication. For example, scalable timers in RTP as well as SRM using *slotting and damping* require the use of a multicast back channel from every receiver. Such cases can be supported by using an application-level agent that performs session management to provide the multi-sender abstraction over the underlying one-to-many model. For example, such all non-root senders may transmit their data to this session management agent which in turn re-multicasts this data along the distribution tree.

2.2 Multicast Deployment

One of the concerns in extending IP is a deployment path in the current internetwork. The MBone was intended as the deployment vehicle for IP multicast. The MBone is a virtual "overlay" network that interconnects islands of networks with native multicast. Each of these tunnels runs the *mrouted* multicast routing daemon which connects to other routers running *mrouted* using IP-in-IP tunnels. Configuration of the tunnels was manual and "peering" on the MBone was largely through "friendly" interactions via e-mail, and phone. The MBone was proposed as a transition vehicle while the research community worked on a longer-term wide-area multicast solution. As a result, it had little monitoring and debugging support, which made it hard to administer.

As evidenced in the discussion in Section 2.1, scalable wide-area routing continues to be an open area of research and experimentation and there is little consensus in the research community.

In fact, the GLOP address partitioning scheme was designed to facilitate wide-area experimentation with different proposals. As a result, even though the design and deployment of IP multicast protocols started almost a decade ago, we do not yet have a ubiquitous multicast infrastructure. Some researchers have argued that such extensions to the IP service model to provide a ubiquitous layer 3 multicast delivery service are fraught with technical difficulties and have suggested instead that new services be layered on top of the existing IP infrastructure as an overlay content distribution and content adaptation network. Our work in this dissertation assumes that a multi-point delivery service is available, but it does not as such mandate a specific structure for it. Such a service may be available either through native layer 3 IP multicast or through a content distribution service, or more likely through a combination of the two. Even though we have targeted our transport level schemes to native IP multicast, the specific delivery service is not likely to impact most aspects of our design. Hence, our schemes may be used to build robust data transport over content distribution networks.

In the following sections, we discuss previous research in the area of multicast transport. We start with the simplest protocol — announce-listen, which is a "quasi-reliable" transport protocol. We then discuss various reliable multicast transports and present a brief overview of delivery semantics in transport protocols.

2.3 Announce/Listen-based Transport Protocols

The announce-listen communication model has been the basic building block of MBone application design. Here, a sender periodically announces its data set to a group of receivers who listen to these transmissions and build their local store of the sender's data. The announce-listen model is best effort and does not provide any delivery guarantees as such. However, it is simple to engineer and has built into it robustness against hosts and network failures. For example, since transmissions are periodic, lost data can be recovered during a subsequent successful transmission. In our work on soft state, we extend the basic announce/listen model and show how it can be tuned to provide a probabilistic notion of reliability.

The Session Announcement Protocol (SAP) [49] is an announce-listen protocol used on the MBone to advertise conference information. The advertisements themselves are specified using the Session Description Protocol (SDP) [52]. A conference advertisement contains the name of the session, a description and timing information along with transport protocol address (in this case, the multicast address and port number pairs for the various applications and media types in the session. Additional information such as contact and encryption information, bandwidth specifications or application-specific attributes are also allowed in SDP. The popular MBone "session directory" [53] uses SAP and SDP. sdr receivers construct a local repository of all announced sessions by listening on a well-known multicast communication channel over which the announcements are transmitted.

The Real-time Transport Control Protocol (RTCP) [124] is an example of an announce-listen protocol. Here, RTP sources periodically transmit *sender reports* that contain information on synchronization, transmission statistics, source identification, and application termination. Receivers in turn send periodic *receiver reports* that contain information on reception statistics (e.g., received sequence numbers, inter-packet jitter).

Other applications of the announce-listen communication model include the multicast user directory service [122]. In this work, Schooler uses announce/listen to perform resource discovery in the MBone. A user locates the desired resource — user directories — by listening for directory announcements. By limiting the reception scope, the user can control the locality of the resource that he/she is attempting to locate.

2.4 Reliable Multicast Transport Protocols

There are two main classes of reliable multicast transport protocols based on the type of loss recovery mechanism. The first class uses negative-acknowledgements from the receiver to trigger retransmissions from the source and the other relies on positive ACKs. We survey specific instances of these below.

2.4.1 Scalable Reliable Multicast (SRM)

SRM is a NACK-based, fully-decentralized reliable multicast protocol originally described by Floyd, *et al.,* in [40]. The SRM framework builds on Clark and Tennenhouse's principle of Application Level Framing (ALF) [23], which provides an elegant solution to the problem of reliable-multicast API design because its flexibility offers applications the opportunity to actively participate in the loss-recovery procedure.

To avoid ACK-implosion, SRM uses NACKs. Receivers detect losses from discontinuities in sequence numbers (or by other means with a generic data naming scheme [112]) and transmit NACKs as a request for retransmission of the lost data[1]. A randomized algorithm determines when a receiver transmits a NACK. These NACKs are multicast to the entire group so that any receiver, in particular the closest receiver with the requested data, may generate a repair in response to a NACK. The repair messages are also multicast to the entire group, so that all receivers that missed that packet can be repaired by a single response. The repair message traffic likewise makes use of the randomized timer algorithm.

To avoid NACK implosion, receivers that observe a NACK for data that they too have not received do not send their own NACK[2] and await the repair data. The goal of the randomized NACK transmission algorithm is to minimize the number of duplicate NACK messages sent. To accomplish this, each receiver delays the transmission of a NACK by an amount of time given by the expression

$$\textit{backoff} \quad = \quad D \cdot (C_1 + C_2 r)$$

where *backoff* is the amount of delay, D is an estimate of the one-way delay from the receiver to the source that generated the lost data packet, C_1, C_2 are non-negative protocol constants, and r is a uniformly distributed random number in $[0, 1]$. This random delay provides receivers with the opportunity to *suppress* the transmission of similar pending NACKs; that is, delaying the transmission of NACKs by a random amount increases the likelihood that a NACK from one receiver is delivered

[1] To be true to the original intentions of the SRM designers, we must admit that our use of the term "NACK" is somewhat inaccurate since it implies that the underlying protocol generates NACKs to guarantee that all data is eventually received by all receivers. In fact, SRM is receiver-reliable and does not require that all receivers obtain all data. Instead, receivers issue "repair requests" to repair only those data wanted. For this paper, we use the terms "NACK" and "repair request" interchangeably.

[2] More precisely, they scale their transmission timer awaiting a response. All receivers, if they have not received the repair data, will eventually transmit a NACK.

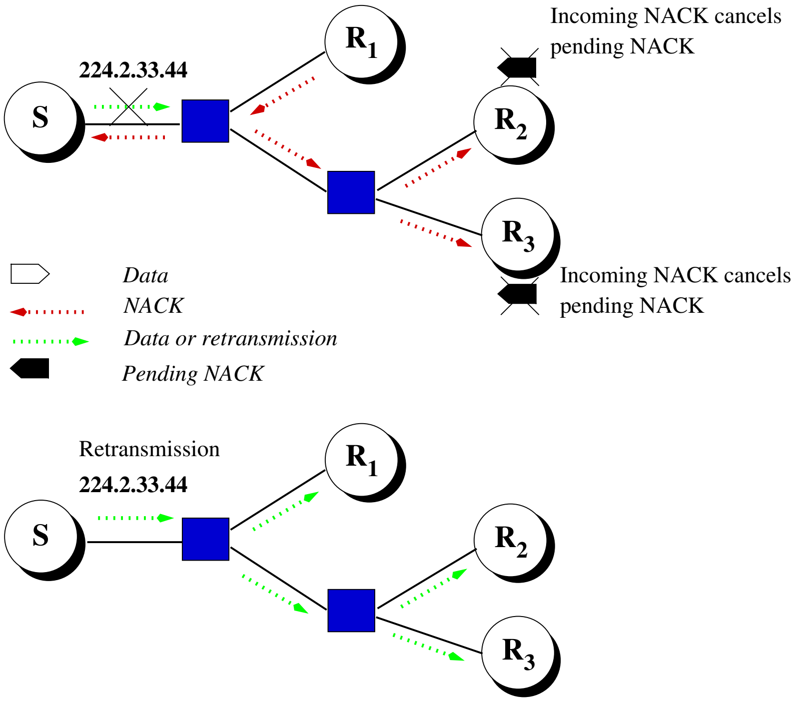

Figure 2.2: NACK suppression in Scalable Reliable Multicast.

to another receiver before that receiver sends its own NACK, and thus, reduces the total number of NACKs. Figure 2.2 illustrates the suppression mechanism in SRM.

As in [40], we call $C_1 D$ the *deterministic* delay and $C_2 Dr$ the *random* delay. The deterministic-delay component induces suppression effects across receivers situated at varying distances from the point of loss (e.g., a chain topology), while the random-delay component induces suppression effects across receivers situated at equal distances from the point of loss (e.g., a star topology). We say that a receiver's timer *fires* if no suppressing NACK has been received when its backoff period has expired.

Since NACKs are multicast to the group, any receiver that has the data can respond, not just the original source. However, we again have the potential for a control-traffic storm if all hosts respond simultaneously. Thus, to avoid repair-packet storms, SRM reuses its NACK suppression machinery to limit the number of redundant repair packets. Because both NACKs and repairs are sent to the entire multicast group, we call this the SRM *global recovery* mechanism.

Other NACK-based schemes include the Stanford log-based receiver reliable multicast protocol (LBRM) [58] that relies on the availability of a logging server with persistent storage. The source transmits its updates to the logging server reliably using a positive ACK protocol, similar to TCP. Subsequent repairs are fetched by receivers from the logging server. The authors also propose a distributed logging scheme with secondary servers at client domains. Each secondary server reliably receives a copy of updates from the primary logging server and uses it to respond to NACKs or repair requests from the receivers from within its domain. LBRM reduces the amount of NACK traffic handled by each logging server by constructing a two-level hierarchy. The main drawback of this scheme is the administrative overheads of installing and maintaining large storage servers and specifying and enforcing policies for their use by different applications.

In Chapter 5, we analyze the asymptotic scaling properties of the randomized NACK scheme and show how it depends on the topology of the multicast group.

2.4.2 Tree-based Reliable Multicast Protocols

Some schemes including Tree-based Reliable Multicast (TRAM) [68] and Tree-based Multicast Transport Protocol (TMTP) [148] take a different approach to loss recovery. Here, the multicast session members are organized into a transport/session-level hierarchy and attempt to reduce the scale of feedback traffic by localizing it to smaller scopes.

TMTP combines both sender and receiver initiated techniques. Participants are organized into domains with a single domain manager responsible for error recovery and local retransmission in each domain. Error control at the sender utilizes periodic unicast ACKs from domain managers, time-outs and retransmissions. Domain managers send ACKs upon receipt of a multicast packet. To signal a missing packet, domain members multicast a NACK to the domain manager in combination with NACK suppression. Localized error control is supported by limited scope multicasting via the IP time-to-live (TTL) field.

RMTP and TRAM are intended for bulk data transport. RMTP was the earliest tree-based protocol and used statically established hierarchy. The target application for RMTP was large scale data shipping for billing applications within AT&T. Here, ACKs are fused at each level of the hierarchy. Sending periodic ACKs also allows for performing congestion control at the sender. The semantics of congestion control in this case are "slowest rate" semantics, where the entire session operates at the bandwidth of the slowest network path.

TRAM uses dynamic trees to implement local error recovery using ACKs and to scale to a large number of receivers without seriously impacting the sender. Here too, as in LBRM, the administrative overheads in establishing and maintaining the transport-level hierarchy. TMTP uses only end system nodes to construct this hierarchy and attempts to automate the process using monitored loss rates between receivers. However, the stability of such a dynamic system is unstudied under realistic network conditions.

2.4.3 Router-assisted Schemes

Pragmatic General Multicast (PGM) [127] is a combined network and transport layer solution to the problem of scalable wide-area multicast. End hosts generate NACKs in response to missing packets. Routers are maintain transport-level state that assists in suppressing duplicate NACKs from downstream members.

Light-weight Multicast Service (LMS) is a similar extension to the network-layer multicast service to provide NACK suppression. Here, intermediate routers maintain state that affects forwarding decisions and guarantees that no more than one NACK message "escapes" a subtree of a source-rooted multicast distribution tree. This scheme aims to optimize the amount the feedback traffic generated using loss recovery and route it efficiently.

Both PGM and LMS assume source-based distribution trees and do not perform optimally with shared trees. Additionally, requiring transport protocol state to be maintained along the distribution path in routers violates layering principles.

2.4.4 Reliable Multicast Framework (RMF)

There has been some recent and ongoing work in reliable multicast *frameworks.* The Reliable Multicast Framework (RMF) [28] does not uniquely define a single reliable multicast protocol. Instead, it provides sufficiently rich set of data structures and mechanisms within a single framework so that various reliable multicast protocols can be implemented and may even interoperate. The key features of RMF are (i) self-identifying packets, which permit fine-grained per-packet reliability semantics and also allow a sender to induce desired receiver behavior on packet reception, and (ii) a universal multicast receiver that interoperates with any sender through the use of self-identifying packets. Universal receivers allow several protocols to be defined using a set of common packet formats. The notion of a single RMF session supporting multiple protocols, for example, a hybrid session combining ACK- and NACK-based schemes, is not clearly understood, especially since many loss recovery schemes require significant cooperation among session members.

Another recent Internet draft by Crowcroft and others [25] on a Reliable Multicast Framing Protocol proposes a *two-layered* data stream with embedded objects and sequence offsets within objects. Their work assumes properties of the data stream such as bounded liveness of data, i.e., the application performs a *close* signaling the end of the object. This is a restriction for applications such as *wb*, where the user may revisit an old page, and add to its existing contents. In addition their approach uses a special `BIND` message that binds application level names to object IDs.

2.4.5 Forward Error Correction-based Schemes

Forward error correction uses redundancy in the data stream to allow the receiver to reconstruct lost packets. Nonnenmacher and Biersack explore the application of FEC schemes to uncorrelated packet loss in a multicast session. They devise a window or block-based scheme in which receivers experiencing uncorrelated losses, i.e., losses of distinct packets by different members can be repaired by a single encoded packet multicast to all the receivers. For example, consider a sender that uses a window size of 7 and applies a 7/10 Reed Solomon erasure correcting code — packets $0, 1, \ldots 6$ carry original data and packets $7, 8, 9$ are redundancy packets computed using the original data packets in this window. A receiver is capable of reconstructing packets $0, 1, \ldots, 6$ using any 7 unique packets in $[0..9]$.

If each of receivers R_1, R_2, and R_3 loses packet 1, 2, and 3 respectively, the sender can repair all three uncorrelated losses using a single transmission of packet 7. The larger the window size, the more effective unshared loss recovery is. However, this scheme requires the entire window of data for to apply the encoding before any transmission occurs and is generally not applicable to low-latency applications such as shared whiteboards where the data stream consists of a stream of small updates that must be delivered in a timely fashion to the receivers.

2.5 Delivery Semantics in Transport Protocols

The so-called CATOCS debate on ordering semantics in the context of multicast protocols drew much attention a few years ago [19, 11, 20]. Cheriton and Skeen argued that ordering semantics are better handled by the application and that enforcing an arbitrarily chosen ordering rule results in performance problems [19]. In our work, we reinforce this approach to protocol design and refrain from imposing a particular ordering semantics across all applications.

RDP [138, 99] is a reliable datagram protocol intended for efficient bulk transfer of data for remote debugging-style applications. RDP does not enforce ordered delivery unless specified by the application. It implements sender-driven reliability and does not support receiver-tailored nor application-controlled reliability. NETBLT [22] is a receiver-based reliable transport protocol that uses in-order data delivery and performs rate-based congestion control.

There has been much recent work on Web data transport for in-order delivery, most of which address the problems posed to congestion control by short transaction sizes and concurrent streams. Persistent-connection HTTP [97], part of HTTP/1.1 [37], attempts to solve this using a single TCP connection, but this causes an undesirable coupling between logically different streams because it serializes concurrent data delivery. The MEMUX protocol (derived from Web MUX [43] proposes to deliver multiplexed bidirectional reliable ordered message streams over a bidirectional reliable ordered byte stream protocol such as TCP [146]. We note that the problem of shared congestion control disappears when congestion state is shared across TCP connections [5, 96, 132] or more generally, across all protocols using the CM.

The WebTP protocol argues that TCP is inappropriate for Web data and aims to replace HTTP and TCP with a single customizable receiver-driven transport protocol [47]. WebTP handles only client-server transactions and not other forms of interactive Web transactions such as "push" applications. It is not a true transport layer (like TCP) that can be used by different session (or application) protocols like HTTP or FTP, since it integrates the session and transport functionality together. In addition, WebTP advocates maintaining the congestion window at the receiver transport layer, which makes it hard to share with other transport protocols and applications.

In contrast, our work on image transport is motivated by the philosophy that one transport/session protocol does not fit all applications, and that the only function that *all* transport protocols *must* perform is congestion management. The Congestion Manager (CM) extracts this commonality into a trusted kernel module [6], permitting great heterogeneity in transport and application protocols customized to different data types (e.g., it is appropriate to continue using TCP for applications that need reliable in-order delivery and RTP/RTCP over UDP for real-time streams, etc.). The CM API allows these protocols to share bandwidth, learn from each other about network conditions, and dynamically partition available bandwidth amongst concurrent flows according to user preferences.

We now briefly present an overview of transport protocols tuned for spatially structured data types such as images. While much work has been done on video transmission, image transport has received little attention despite constituting a large fraction of Internet traffic. Turner and Peterson describe an end-to-end scheme for image encoding, compression, and transmission, tuned especially for links with large delay [137]. As a result, they develop a retransmission-free strategy based on forward error correction. Han and Messerschmitt propose a progressively reliable transport protocol (PRTP) for joint source-channel coding over a noisy, bandwidth constrained channel. This protocol delivers multiple versions of a packet with statistically increasing reliability and provides reliable, ordered delivery of images over bursty wireless channels [48]. The Flexible Image Transport System (FITS) is a standard format endorsed by the International Astronomical Union for the storage and transport of astronomical data [38]. It specifies various file header formats, but not a protocol for transmission over a loss-prone network.

The Fast and Lossy Internet Image Transmission protocol (FLIIT) [27] improves the perceived delay of a download by eliminating retransmissions. Instead, the FLIIT sender strategically shields "important" portions of the image data, for example, by applying FEC to the high order bits of the DC channels of the image. FLIIT assumes a bit budget and allocates this between the original image data and the amount of redundancy based on the observed loss rate in the channel. Our work in this dissertation also aims to improve interactivity. However, rather than design new compression schemes for image transmission, we focus on the transport protocol and application interface issues such that many different image formats can be supported.

Heybey [56] considers the problem of video coding and develops an application-level framing architecture for it. However, much emphasis is placed on developing framing strategies that translate into an optimized hardware implementation. In our work, we focus on the protocol aspects and show how a generic protocol may be used effectively when customized for specific image formats.

Richards and others [118] have also considered using ALF to build high performance transport protocols. However, they attempt to extend existing TCP implementations to achieve this and present their evaluation of the overheads involved in this approach.

2.6 Summary of Related Work

In this Chapter, we surveyed the IP multicast service model and deployment paths for wide-area multicast via the MBone. The MBone is primarily a research network and more recently, there has been commercial activity in the area of content distribution networks. We then moved up the protocol stack and described the different approaches to multicast transport — unreliable as well as reliable. We presented the annouce/listen protocol which forms the basis of our light-weight soft state-based transport framework solution, and discussed examples of a announce/listen-based transport protocols in use on the MBone today. We then discussed several reliable multicast protocols: end-to-end NACK-based approaches like SRM, tree-based schemes like TMTP and TRAM, router-assisted schemes, FEC-based schemes, as well as generic framework approaches.

All of these protocols and frameworks treat the transport protocol in isolation, without considering benefits that arise from the interaction between the application and transport protocol. In the remainder of this dissertation, we explore this topic further and develop a generic light-weight transport protocol whose behavior is further customizable based on the application. Next, we formalize the notion of "soft state" and demonstrate its robustness and performance properties. We show how soft state-based protocols can be tuned to provide probabilistic measures of reliability.

Chapter 3

Soft State-based Transport

If at first you don't succeed, try, try again.

— William Edward Hickson

This Chapter describes a model and analysis for soft state-based communication. We motivate our choice of soft state as a protocol building block, especially in the context of multicast transport in Section 3.1. In Section 3.2, we present the data and communication models for soft state. We then analyze the simple open-loop announce/listen protocol in Section 3.3. Based on our analysis, we propose two techniques to improve the performance of the traditional announce/listen framework: (1) a simple, two-level differentiated transmission scheme, described in Section 3.4, and (2) a novel application of feedback to guide the link scheduling decisions at the source, described in Section 3.5. Section 3.6 develops SSTP, a practical protocol framework for realizing and reusing the soft state communication primitive across multiple applications. In Section 3.7, we review past and ongoing work related to soft state and in Section 3.8, we present our concluding remarks on the soft state model.

3.1 Motivation

"Soft state" is an often cited yet vague concept in network protocol design in which two or more network entities intercommunicate in a loosely coupled, often anonymous fashion. Researchers often define this concept operationally (if at all) rather than analytically: a source of *soft state* transmits periodic "refresh messages" over a (lossy) communication channel to one or more receivers that maintain a copy of that state, which in turn "expires" if the periodic updates cease. Though a number of crucial Internet protocol building blocks are rooted in soft state-based designs — e.g., RSVP refresh messages, PIM membership updates, various routing protocol updates, RTCP control messages, directory services like SAP, and so forth — controversy is building as to whether the performance overhead of soft state refresh messages justify their qualitative benefit of enhanced system "robustness". We believe that this controversy has risen not from fundamental performance tradeoffs but rather from our lack of a comprehensive understanding of soft state. To better understand these tradeoffs, we propose herein a formal model for soft state communication based on a probabilistic delivery model with relaxed reliability. Using this model, we conduct queueing analysis and simulation to characterize the data consistency and performance tradeoffs under a range of workloads and network loss rates. We then extend our model with feedback and show, through simulation, that adding feedback dramatically improves data consistency (by up to 55%) without increasing network resource consumption. Our model not only provides a foundation for understanding soft state, but also induces a new fundamental transport protocol based on probabilistic delivery. Toward this end, we sketch our design of the "Soft State Transport Protocol" (SSTP), which enjoys the robustness of soft state while retaining the performance benefit of hard state protocols like TCP through its judicious use of feedback.

Given the attractive properties of soft state and the proliferation of the announce/listen primitive in so many Internet protocols over the past decade, one would expect that a great deal of research would exist that not only clearly articulates what soft state is but characterizes the fundamental performance tradeoffs of soft state designs. Yet such work is scant. Not only is there a dearth of analysis and refinement of soft state, but there is no well-defined communication framework, no common protocol architecture, and no application API that is based on the soft state model. We believe this is a great misfortune because such work could help guide protocol designers and engineers to decide when and where the performance tradeoffs of soft state are worth the benefit and a common implementation and framework would provide reusable protocol building blocks for application designers. In this Chapter, we address this void with a formal model for soft state communication based on a probabilistic delivery model with relaxed reliability. Our contributions are as follows:

- We present a novel model for soft state protocols and probabilistically define an associated *consistency metric*.
- We theoretically analyze our model for the simple open-loop announce/listen protocol.
- We systematically characterize data consistency and performance tradeoffs of our soft state model under a range of workloads and network loss rates for the simple open-loop case and its variants,
- We extend the open-loop variant of announce/listen by adding receiver feedback to enhance data consistency and performance without increasing network resource consumption.
- Based on our model, as well as the observation that several protocols have inherently "soft" or periodically changing data, e.g., route advertisements [86, 55, 78], DNS updates [88], MBone session directories [53], stock quote or general information dissemination services [107], we propose a soft state-based transport protocol (SSTP) framework. The SSTP framework provides a parameterized spectrum of reliability semantics all derived from one framework — from simple announce/listen communication to feedback-based reliable transport. SSTP also optimally allocates bandwidth based on packet loss rates and application workload to maximize consistency. The result is a parameterized framework that can be tuned to provide one of a continuum of "reliability levels". We also incorporate ideas from application-level framing (ALF) [23] to provide an interface that allows it to be tailored to the specific application.

3.2 The Data Model

To evaluate the tradeoffs and performance of soft state communication, we must first carefully define a framework and model that firmly grounds our analysis and discussion.

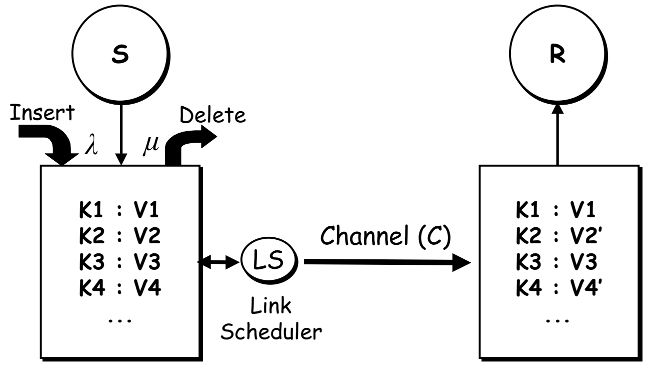

Figure 3.1: Soft state data model comprises an evolving table of $\{key, value\}$ pairs.

Our model for "soft" data is a table of $\{key, value\}$ pairs at the sender, or publisher. The publisher may add, delete, or update a record at any given time. The scheduler at the source periodically announces a record chosen from its table by the scheduler on to a (lossy) channel with capacity C, according to some specified scheduling algorithm. One or more subscribers tune into the channel to receive updates from the publisher. On receiving an announcement, each subscriber that has joined the channel updates its local copy of the table. An expiration time is associated with each data item stored at the receiver. If an update is not received before the timer expires, the entry is deleted (and in practice an external notification event is generated).

The set of all data items in the sender's table at any given instant t, is termed the *live data set*, $L(t)$. An update process at the publisher adds records to its table. Each record is also associated with a lifetime after which the publisher ceases to announce it and the record is eliminated from both the sender's and receivers' tables. Figure 3.1 illustrates this model.

3.2.1 Consistency

Unlike ARQ, where receipt of an acknowledgement explicitly indicates state synchronization between sender and receiver, a soft state protocol generally provides no feedback to the sender as to what the receiver has successfully received. Instead, the end systems simply participate in the announce/listen process and the assumption is that the receiver's data store converges to a consistent state over time. Many protocols based on this premise have been described and some characterize this property as *eventual consistency* [40, 50], but a formal definition for this has not yet been proposed.

To measure the effectiveness of soft state protocols using our model, we introduce the *consistency metric*, $c(k, t)$, defined per live $\{key, value\}$ pair $\{k, val(k)\}$ for processes P and Q communicating over a loss-prone network as the probability that both processes have the same value for a given key. This is denoted as,

$$c(k,t) \quad = \quad Pr.[P.val(k) = Q.val(k)], \quad 0 \leq c(k,t) \leq 1$$

where $P.val(k)$ is P's value for key k.

The *instantaneous system consistency*, $c(t)$ at a given instant, t, is defined as the average consistency measured across all live data items at that instant.

$$c(t) \quad = \quad \frac{\sum_{k \in L(t)} c(k,t)}{|L(t)|}$$

The *average system consistency* is the time average of the instantaneous system consistency over the entire lifetime of a system.

$$E[c(t)] \quad = \quad \lim_{T \to \infty} \frac{\int_{t=0}^{t=T} c(t)dt}{T}$$

The definition of $E[c(t)]$ above also provides us with a method to empirically compute the consistency metric of a system — as the time average of $c(t)$ over long durations.

A protocol is said to be *eventually consistent* if this probability approaches 1 in the long run, after the item is introduced into the system, i.e.,

$$\frac{\sum_{k \in K} \lim_{t \to \infty} c(k)}{|K|} \quad \approx \quad 1$$

Another important metric of protocol performance is the average latency from the instant a new or updated $\{key, value\}$ pair is introduced into the system to the first time it is received correctly at the receiver. We call this the receive latency T_{rec}.

In the remainder of this Chapter, we present several analytical and simulation results showing the dependence of the consistency metric on packet loss rates, available bandwidth, and announcement workloads. We term this dependence a *consistency profile*.

Many protocols based on soft state rely on nothing more than the announce/listen mechanism for maintaining consistency in the face of packet loss. This simple open-loop repetitive announcement process transmits state updates in a quasi-reliable manner from an announcer to a listener over a loss-prone network. For a static input at the source, announce/listen provides a simple form of reliability since eventually the receiver's state will match the sender's once all the records have been successfully transmitted.

The simple open-loop periodic retransmission scheme provides an extremely simple substrate for "quasi reliable" systems. It is an attractive alternative to ARQ-based reliable transport protocols, especially in the case of multicast, since managing receiver feedback scalably in large groups continues to remain a formidable challenge. For example, it has been successfully used in the the multicast-based session directory tools [61, 53] to disseminate MBone conference information to large groups. However, pure open-loop periodic retransmissions are redundant and do not use bandwidth efficiently. The challenge for the so called "soft state" transports, including the announce/listen protocol, is therefore to (i) maximize system consistency and minimize user-perceived latency in receiving data items, and (ii) minimize redundant transmissions. In the following sections, we evaluate several soft state-based transports and show how to optimize them for given network conditions using adaptive scheduling techniques.

3.3 "Open-Loop" Announce/Listen Protocol

To evaluate the performance of soft state systems that use open-loop announce/listen for data transport, we develop an analytic model based on class-based queueing networks [9].

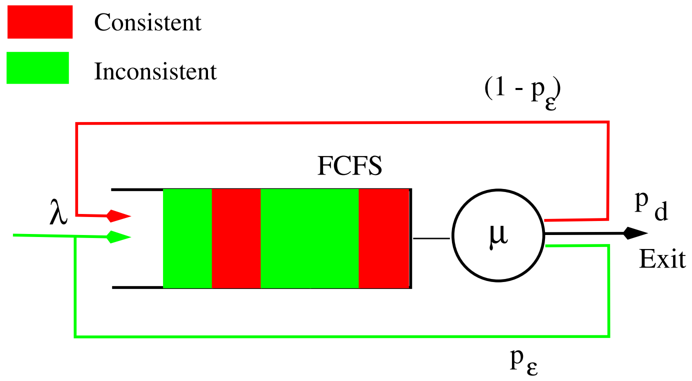

Figure 3.2: Queueing model for announce/listen-based transport protocol.

The parameters for our model are: p_ϵ, the probability that an announcement transmitted on the channel is lost by one or more subscribers, or the average per-transmission channel loss rate. Since the consistency metric treats all successful transmissions identically, regardless of their relative position in the transmitted stream, it is sufficient to specify the average packet loss rate. The metric is insensitive to the exact pattern of losses, but is only affected by the mean of the packet loss process. In contrast to other application- and media-specific metrics such as rate-distortion model [24] for multimedia signals that are sensitive to loss patterns in addition to the average drop rate, our metric is more general and applies to a wider class of systems.

In addition, we also assume that these soft state transports are ALF-based, in that individual transmissions are independent application data units (ADUs) [23]. In addition, the transport protocol does not enforce in-order delivery of packets at the receiver, which allows us to ignore the effects of packet reordering in our model, even though, in reality, though receivers suffer extra latency when individual fragments of a large ADU are reordered.

λ is the average rate of update of the publisher's table, and μ_{ch}, the available session bandwidth for this source's announcements. We model the network as a single server queue with multiple job classes or states. Each record goes through the following stages.

- Records enter the system in the "inconsistent" state, since the receiver has no knowledge of them. New records are inserted at the end of the transmission queue and the sender is assumed to have sufficient buffer space to hold all arriving announcements.
- The transmission channel acts as a server with service rate μ_{ch} and uses FIFO scheduling.
- The record changes state to "consistent" when an announcement containing it successfully reaches the receivers (with probability $(1 - p_\epsilon)$).
- Each record has a bounded lifetime after which it is expired from the sender and receivers. For example, in session directories, announcements expire when the associated conference

	I/Exit	C/Exit	Death/Exit
I/Enter	$p_\epsilon(1-p_d)$	$(1-p_\epsilon)(1-p_d)$	p_d
C/Enter	0	$(1-p_d)$	p_d

Table 3.1: State transition probabilities for a data item in the soft state model.

session ends. After obtaining service, an announcement exits the system with probability p_d, its death probability. The data expiration process is an inherent characteristic of the workload, and governs this death probability. In our model, we approximate the expiration process using a fixed and independent death probability per packet event though this does not take into account the possibility that an older record is more likely to expire than a new one.

Table 3.1 lists the probability of state change between consistent and inconsistent as a record leaves the server.

If $n_C(t)$ and $n_I(t)$ denote the number of consistent and inconsistent records in the system at any instant, the consistency metric for this system is the time average of the fraction $\frac{n_C(t)}{n_C(t)+n_I(t)}$. Computing the net flow $\hat{\lambda}_I$ and $\hat{\lambda}_C$ into the queue for each class, we get:

$$\begin{aligned}\hat{\lambda}_I &= \lambda + p_\epsilon(1-p_d)\hat{\lambda}_I \\ \hat{\lambda}_C &= (1-p_\epsilon)(1-p_d)\hat{\lambda}_I + (1-p_d)\hat{\lambda}_C\end{aligned}$$

Solving the above system of equations yields,

$$\begin{aligned}\hat{\lambda}_I &= \frac{\lambda}{1-p_\epsilon(1-p_d)} \\ \hat{\lambda}_C &= \frac{(1-p_\epsilon)(1-p_d)}{p_d}\hat{\lambda}_I \\ &= \frac{(1-p_\epsilon)(1-p_d)\lambda}{p_d(1-p_\epsilon(1-p_d))}\end{aligned}$$

Now,

$$\begin{aligned}\hat{\lambda} &= \hat{\lambda}_I + \hat{\lambda}_C \\ &= \frac{\lambda}{p_d}\end{aligned}$$

We first use Jackson's theorem [9] in the following steps for the single queue system with multiple job classes to compute the joint probability distributions of the number of consistent and inconsistent jobs.

$$p(n_I, n_C) = \frac{(n_I+n_C)!}{n_I!n_C!} \times \frac{\hat{\lambda}_I^{n_I}\hat{\lambda}_C^{n_C}}{\hat{\lambda}^{n_I+n_C}} \times (1-\rho)\rho^{n_I+n_C}$$

where, $\rho = \frac{\hat{\lambda}}{\mu_{ch}}$. The solution is valid only when $\rho < 1 \Rightarrow p_d > \frac{\lambda}{\mu_{ch}}$.

The *average system consistency* $E[c(t)]$ is then given by:

$$
\begin{aligned}
E[c(t)] &= \sum_{n_I+n_C>0} \frac{n_C}{n_I+n_C} \times p(n_I, n_C) \\
&= \sum_{k\geq 0} \frac{\hat{\lambda}_C}{\hat{\lambda}_I+\hat{\lambda}_C}(1-\rho)\rho^{k+1} \\
&= \frac{\hat{\lambda}_C}{\mu_{ch}} \\
&= \frac{(1-p_\epsilon)(1-p_d)}{1-p_\epsilon(1-p_d)} \times \frac{\lambda}{p_d \mu_{ch}}
\end{aligned}
$$

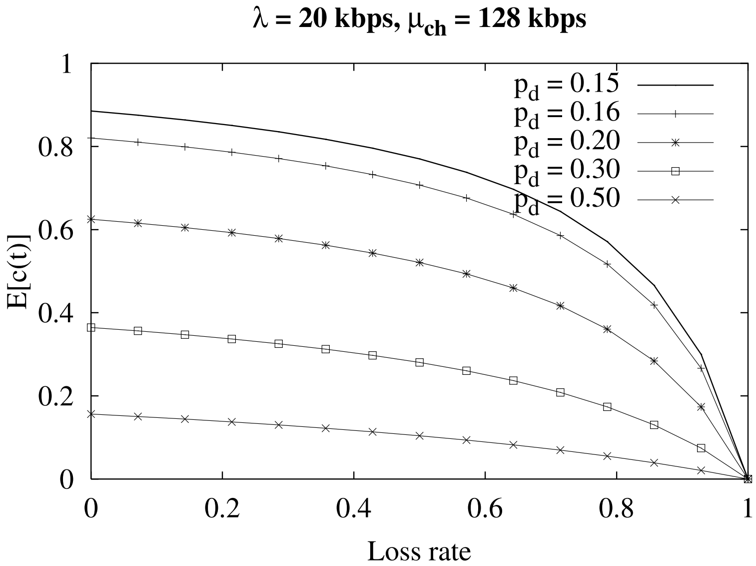

Figure 3.3: Consistency degrades with increasing packet loss rate and announcement death rate. A workload with a 15% death rate is 95% consistent for error rates in 1-10%.

Figure 3.3 shows $E[c(t)]$ graphically for different loss rates and announcement death rates. For a given death rate, as expected, we find that the system consistency goes down as the channel loss rate increases. We also observe that consistency falls sharply when the announcement death rate increases since data items are too short-lived to be propagated successfully to receivers. As seen in Figure 3.3 the system consistency lies between 85% and 95% for loss rates in the 1-10% range and an announcement death rate of 15%.

The open-loop announce/listen protocol analyzed above treats all data items — old and new — alike, retransmitting data items that may have already been received by the members of the group. From our model, we find that the fraction of bandwidth consumed by redundant transmissions is given by:

$$
\begin{aligned}
w &= \frac{\hat{\lambda}_C}{\hat{\lambda}} \\
&= \frac{(1-p_\epsilon)(1-p_d)}{1-p_\epsilon(1-p_d)}
\end{aligned}
$$

Figure 3.4 shows this result graphically. At loss rates of up to 50% and a death rate of 10%, over 90% of the total bandwidth is wasted on redundant retransmissions.

In reality, periodic source-based retransmissions are not entirely wasteful and benefit late joiners in an ongoing multicast session by reducing the delay such receivers experience in "catching" up with the rest of the session. Even in the case of unicast transmission, periodic source announcements allow the receiver to reconstruct the data store following a crash. Several techniques can be applied to the basic open-loop protocol to improve its consistency. In Sections 3.4 and 3.5, we show that maintaining multiple transmission queues at the sender and adding receiver feedback in a controlled manner allow for better bandwidth management at the sender.

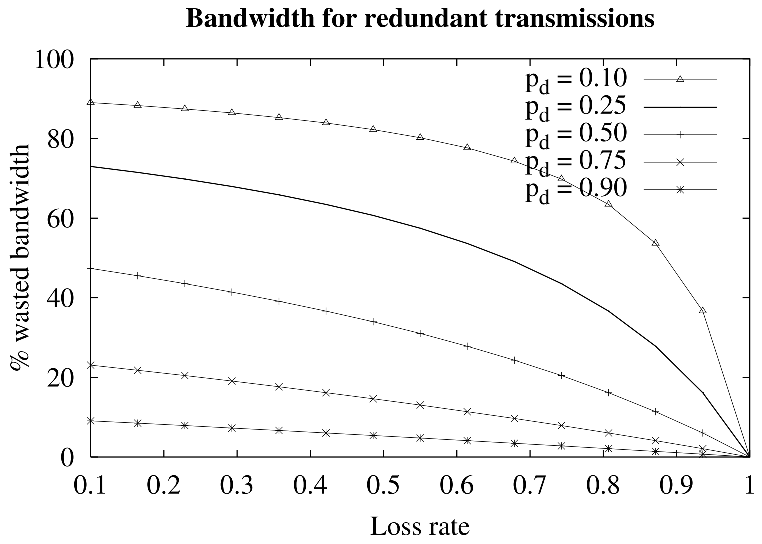

Figure 3.4: At loss rates between 0-20% and an announcement death rate of 10%, about 90% of the total available bandwidth is wasted.

3.4 Multiple Transmission Queues

Since redundant transmissions of previously consistent data items do not contribute to system consistency, one way to improve the performance of the basic open-loop announce/listen protocol is to reduce the fraction of bandwidth for repeated retransmissions. We do this by differentiating new and old data items. Several policies exist for aging data items, but we consider one simple aging scheme with two transmission queues for our analysis. We refer to the two transmission queues as the "hot" (or foreground) queue for new data items, and the "cold" (or background) queue for data items that have been transmitted at least once from the sender. The available data bandwidth is shared between the two queues proportionally (e.g., using a randomized lottery scheduler [139], weighted fair queueing [32] or stride scheduling [140]). Proportional sharing is preferred over strict priority scheduling since it prevents starvation of cold data items in the background transmission queue. Bandwidth allocated to foreground transmissions directly increases the likelihood that a new data item is successfully delivered, and hence contributes to system consistency. Unused excess hot bandwidth is consumed by transmissions from the cold queue.

We evaluate the consistency of this scheme using simulations [1]. Our simulations comprise a single sender and single receiver with a lossy communication channel. Having two transmission queues raises the important issue of allocating the total data bandwidth μ_{data} for the hot (μ_{hot}) and cold (μ_{cold}) queues and our simulations quantify the impact of this bandwidth allocation policy on system consistency.

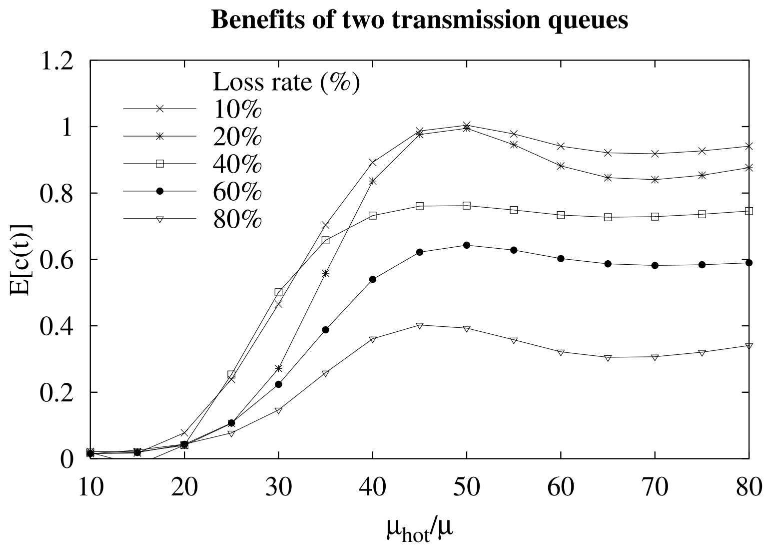

Figure 3.5: Two-level scheduling improves consistency by 10% to 40%. $\mu_{data} = 45kbps$, $\lambda = 15kbps$. Consistency is maximum when $\mu_{hot} > \lambda$.

Figure 3.5 shows the impact of increasing μ_{hot}, the bandwidth allocated to foreground transmissions, when μ_{data}, the total data bandwidth is held fixed. The results show that increasing μ_{hot} has a positive effect on the average system consistency, but only while $\mu_{hot} > \lambda$ (up to about 40%, in this experiment). The sender must allocate sufficient bandwidth to new data arriving at rate λ, to prevent the hot queue from growing indefinitely. The optimal consistency level is reached for $\mu_{hot} \geq \lambda$. However, as we see from Figure 3.5, consistency does not significantly change as we increase the bandwidth for the hot queue beyond λ.

[1] Unfortunately, this extended model with two-level scheduling is not analytically tractable using Jackson's theorem.

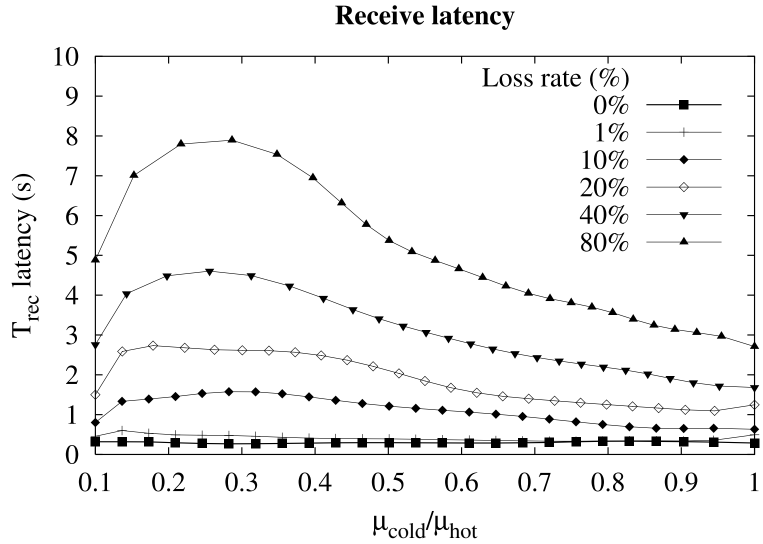

Figure 3.6: Increasing the cold bandwidth reduces queueing delay. When $\mu_{cold} >> \mu_{hot}$, no data item is retransmitted resulting in low latencies for successful transmissions.

The benefits of cold retransmissions are in the form of reduced average receive latency. We study the effect of increasing μ_{cold} (and hence mu_{data}), while maintaining μ_{hot} at its optimal level, just higher than the arrival rate. From Figure 3.6, we find that the receive latency T_{rec} initially increases, but drops as more bandwidth is added for background transmissions. This is due to two competing effects:

(i) At low values of μ_{cold}, all successful transmissions experience very small latency. When $\mu_{cold} \approx 0$, data items are never retransmitted and all successfully delivered items thus experience low delay. Since the average T_{rec} is measured only over all successful transmissions, our measurement excludes data items that take indefinitely long to reach the receiver, and hence the apparently low latency. The 300 ms latency for $\mu_{cold}/\mu_{data} \leq 1\%$ is explained by approximating the system as a single-server single-queue system with bandwidth $\mu_{hot} \approx \mu_{data}$. With exponential interarrivals and service times, the average latency is $E[w] = \frac{\rho}{\mu_{data}(1-\rho)}$. However, without retransmissions ($\mu_{cold} \approx 0$), and in the face of high loss rates, a significant fraction of data is never successfully transmitted, resulting in a low average consistency. Hence, turning off background transmissions is detrimental to system performance especially in the face of high loss rates.

(ii) Increasing the cold bandwidth increases the likelihood of a successful retransmission and, therefore, reduces T_{rec}, as shown in Figure 3.6.

3.5 Impact of Receiver Feedback

The inefficiency of the open-loop protocols discussed in Section 3.4 stems from the source's incomplete knowledge of receiver state. Adding receiver feedback attempts to remedy this by communicating receiver status back in order to improve bandwidth management at the sender. In this section, we discuss our simulation results quantifying the impact of adding receiver feedback in the form of negative acknowledgments (NACKs) to the original announce/listen framework. Once again, our simulations have one sender and one receiver. We find that adding feedback can improve consistency by 10% to 50% for loss rates between 5% and 40%.

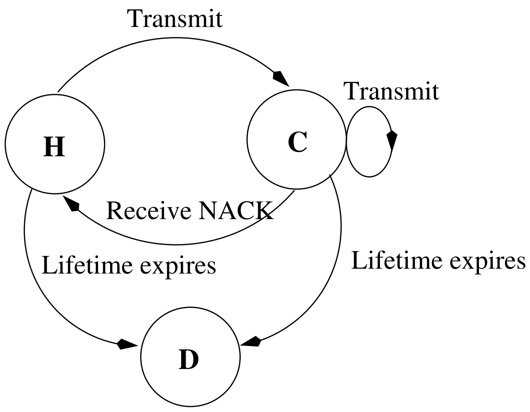

Figure 3.7: State transitions between "hot" (H), "cold" (C), and "dead" or invalid (D) states.

The sender maintains two transmission queues — (i) a hot (or foreground) queue that contains data that is thought to be inconsistent, and (ii) a cold (or background) queue for repeated retransmissions. As in the previous cases, *late joiners* who need to catch up with the current state of an ongoing session benefit from repeated retransmissions. Data items get scheduled for transmission as follows: a new data item is transmitted through the foreground queue, and subsequently moved to the background queue, as shown in Figure 3.7. The two queues share the available data bandwidth proportionally, and we control this allocation in our experiments. The receiver generates a NACK upon detecting a loss. In response to the NACK, the sender schedules a retransmission of the requested data item, by moving it from the cold queue to the tail of the hot queue. Hence, hot bandwidth is allocated to new data items and retransmissions requested by the receiver, while cold bandwidth is used for background retransmissions of previously transmitted data.

- **Data vs. feedback.** We simulate the effect of increasing the fraction of total bandwidth allocated for feedback and find that adding feedback improves system consistency from 60% to almost 98% at a loss rate of 40%. Figure 3.8 shows these results. Allocating a small fraction of the total available bandwidth for feedback messages significantly improves system consistency. Consistency is maximum (at close to 100%, in this example) when sufficient bandwidth is available to transmit NACKs generated in response to data loss. Beyond this threshold level, consistency drops rapidly as the feedback bandwidth grows at the expense of the sender's data bandwidth. For example, in Figure 3.8, when feedback receives 70% of the total bandwidth, the system consistency collapses rapidly.

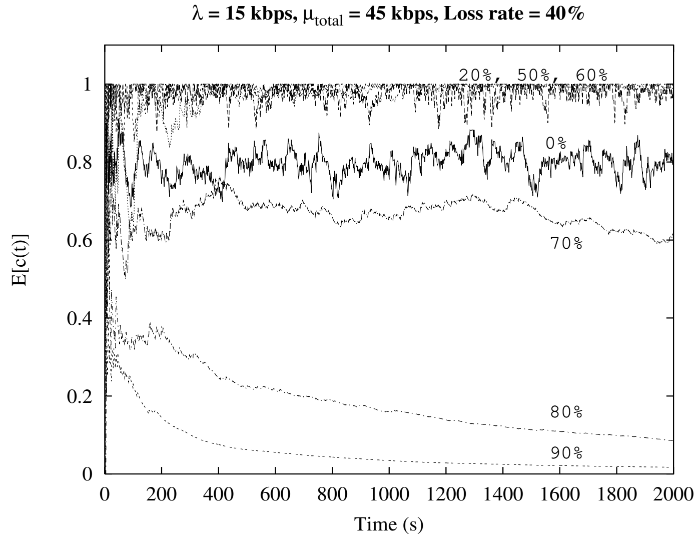

Figure 3.8: In open-loop ($\mu_{fb}/\mu_{tot} = 0$), consistency is about 80%. When $\mu_{fb}/\mu_{tot} = 20-60\%$, consistency reaches 99%. At higher values, when insufficient bandwidth is available for data, consistency collapses.

We also study the impact of adding feedback bandwidth, while maintaining μ_{data} fixed and find that the average system consistency increases by about 10% when the loss rate is about 10% and by 50% for even higher loss rates ($\geq 50\%$). This is shown in Figure 3.9. Consistency reaches a maximum between 90% and 100% depending on the loss rate, indicating that increasing the feedback bandwidth beyond this threshold level does not significantly affect consistency.

Since the packet loss rate also affects the optimal data vs. feedback allocation, the protocol must monitor loss rates via receiver reports and use this information to adaptively reallocate bandwidth to maintain this "optimal" consistency level.

- **Hot vs. cold bandwidth.** To manage the available data bandwidth at the sender, we study the impact of allocating bandwidth to hot and cold data queues. In Figure 3.10, we find that the consistency metric remains close to 5% as long as the arrival rate exceeds μ_{hot}. When μ_{hot} is increased beyond λ, the consistency sharply rises to almost 100%. Increasing μ_{hot} beyond λ does not have a significant impact on the consistency metric. Hence, $\lambda \leq \mu_{hot}$ is the optimal region beyond which the marginal benefit from additional bandwidth to the "hot" queue is limited and below which system consistency shows marked degradation. If the system's consistency metric is to be maximized, the application must adhere to its allowed maximum level. Later, in Section 3.6, we show how our transport framework uses this to notify the application to refrain from injecting new records if system consistency is to be maximized.

- **Effect of loss rate.** Since the channel loss rate indirectly affects the number of NACKs and

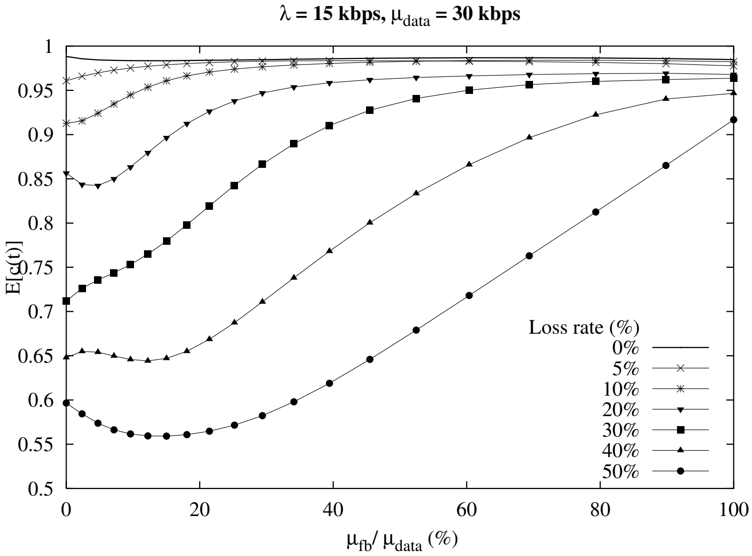

Figure 3.9: Consistency is improved by allocating sufficient bandwidth for feedback. At loss rates over 50%, allocating additional feedback bandwidth feedback reduces consistency.

hence the number of retransmissions, we study the impact of loss rate on the consistency metric, varying the sender's bandwidth allocation between its two transmission queues. From Figure 3.11, we see that the loss rate limits the maximum consistency that can be attained with a given amount of total bandwidth, regardless of how it is scheduled between the hot and cold transmissions. However, the relative proportion of hot vs. cold bandwidth does not significantly affect consistency, once sufficient bandwidth is available to absorb new arrivals.

The consistency profiles discussed here influences bandwidth management. In Section 3.6 we elaborate on a profile-driven allocation scheme that aims to utilize the available bandwidth optimally.

3.6 A Soft State Transport Framework

Conventional reliable transport protocols like TCP are built on "hard" protocol state at the end points and export a single restrictive interface to the application — that of a sequenced, in-order, byte-stream. While some extensions to relax the constraints of TCP have been proposed, the underlying abstraction provided by TCP is rigid and does not lend itself to arbitrary application customization. For example, extensions to TCP byte sequence numbers to support application-defined records is not straightforward. (See [34] for an interesting discussion on this.) Motivated by this, we propose a new framework for reliable transport protocols whose behavior, e.g., degree of reliability, message ordering semantics, bandwidth allocation policies, can be customized by the application. Using our formalism of soft state, we propose the *soft state transport protocol* framework (SSTP) for reliable data transport. To the best of our knowledge, this is the first soft

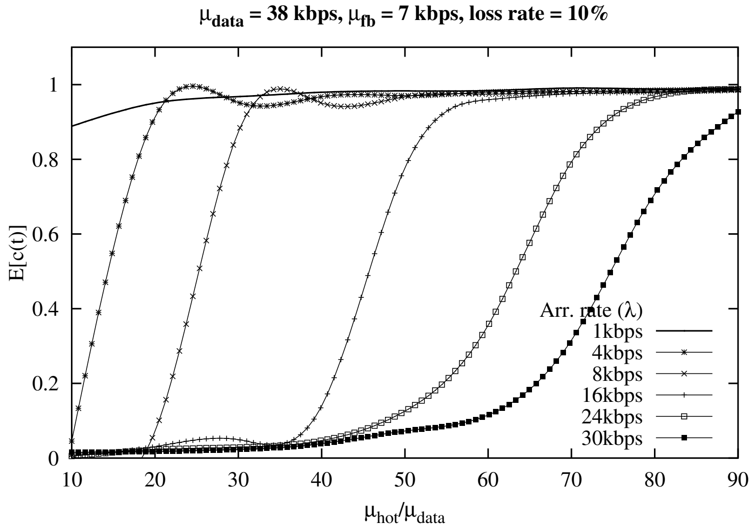

Figure 3.10: $\lambda \leq \mu_{hot}$ is the optimal region beyond which the marginal benefit from additional bandwidth to the "hot" queue is limited and below which system consistency shows marked degradation.

state transport protocol whose properties can be predicted using a model.

In contrast to the conventional approach to transport design, the SSTP architecture is guided by ALF and exposes a powerful, yet simple programming interface allowing it to be tailored to the needs of the user application. SSTP aims to provide the necessary interface and mechanisms for an application to control the degree of reliability and message delivery semantics. An SSTP sender transmits original application data as well as periodic soft state announcements summarizing all previously transmitted data. SSTP receivers use NACKs to report lost data items to the sender, which in response performs the appropriate retransmissions. SSTP may be applied to multicast as well as unicast transport. In the case of multicast, we use the slotting and damping [21, 40] method for managing feedback traffic in a scalable manner.

The following two salient features of SSTP are described in this section: (i) application-controlled bandwidth management, and (ii) a hierarchical data model to efficiently support large data stores.

3.6.1 Application-controlled Bandwidth Allocation

SSTP provides a parameterized framework to schedule available bandwidth between data and feedback messages appropriately to achieve consistency levels desired by the application. Based on the amount of bandwidth allocated to data and announcements (or, "cold" data), a continuum of consistency levels is provided. SSTP uses measured packet loss rates using RTCP-style receiver reports and empirically derived *consistency profiles* to carefully control bandwidth allocation.

SSTP does not attempt to perform congestion control nor determine the total available data rate to the session member, but rather, relies on a congestion management module, such as

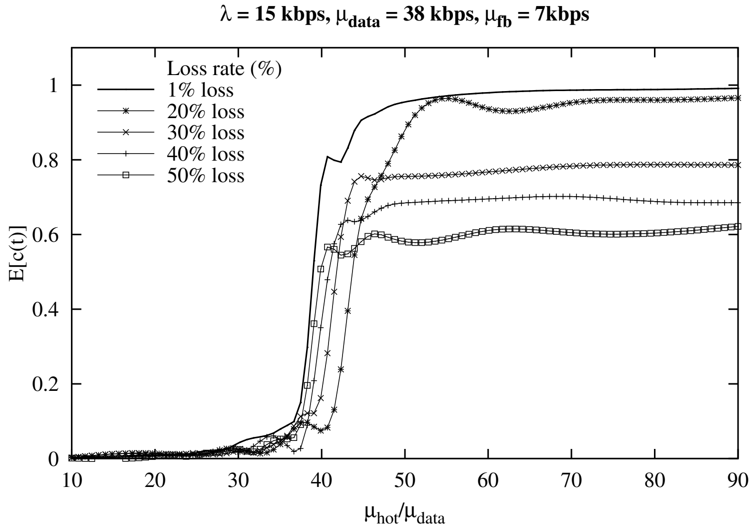

Figure 3.11: The system consistency shows a “knee”, beyond which the marginal benefit from additional bandwidth to the “hot” queue is limited and below which system consistency shows marked degradation.

the CM [6], to obtain this information. SSTP merely decides how this available bandwidth is to be used by the application and transport protocol. While most existing research addresses the issue of detecting network congestion and reacting to it by lowering the transmission rate (or reception rate, as in layered multicast) [62, 66, 65, 15, 136, 14, 80, 117], the issue of how best to utilize available bandwidth in reliable transport has received far less attention. Even though this decision is generally application-specific, we can use the consistency metric for a large class of applications that fit the data model described previously in Section 3.2.

Rather than treat all data as equal, SSTP allows the application to reflect its priorities into the data transport protocol. Using a hierarchical scheduler (e.g., CBQ [39] or H-FSC [129]), the application flexibly controls the amount of bandwidth allocated to its different data classes. Figure 3.12 shows an example of such an allocation hierarchy. An application can experience the maximum possible consistency under given network loss rates by scheduling its available session bandwidth based on consistency profiles derived from our model. Consistency profiles predict system consistency for given network loss conditions and announcement characteristics.

Using stored consistency profiles similar to Figure 3.9, the bandwidth allocator outputs values $\{\mu_{data}, \mu_{feedback}\}$. The share of bandwidth for the different transmission queues is obtained from the T_{rec} profile, similar to Figure 3.6. The allocator also notifies the application if it detects that rate of arrival of new data from the application exceeds the bandwidth available for it, i.e., μ_{hot}. This dictates the maximum rate at which the application can send to maintain the requested level of consistency. This notification from SSTP gives the application an opportunity to adapt to the rate constraint in the best possible application-specific manner.

SSTP uses the following information in making bandwidth allocation decisions:

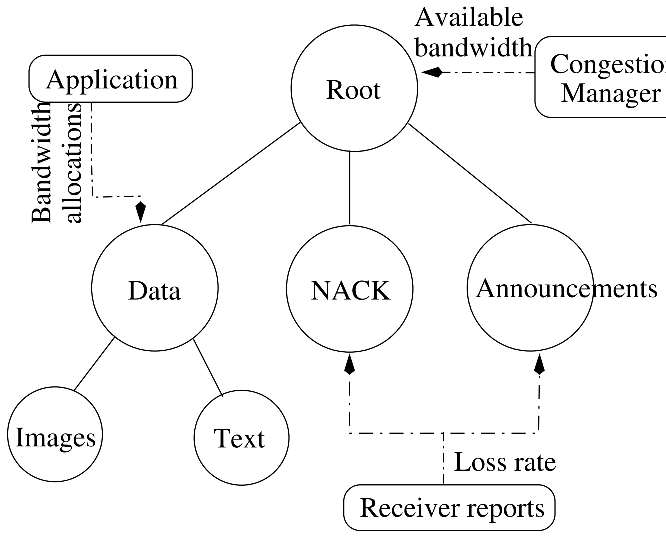

Figure 3.12: Profile-driven scheduler for application adaptation.

- The average packet loss rate, periodically obtained from RTCP-like receiver reports;
- The application's consistency target (e.g., 90% consistency), and optionally a "soft" delay requirement[2] for individual data items;
- The total available session bandwidth, either configured manually as in most non-TCP applications today (e.g., the MBone video conferencing applications [85, 64, 63] and the Real Audio$^{\mathrm{TM}}$ player [116]) or available from a congestion control algorithm.

3.6.2 Hierarchical Data Model

Our simple data model presented in Section 3.2 fits a number of existing systems such as routing updates and the current session directory protocol. However, if such soft state systems are to scale to extremely large systems, the "table of key-value pairs" model needs to be refined. In order to scale announcement-driven loss recovery to applications with large data sets, SSTP supports hierarchical namespaces. The SSTP hierarchy provides a good summarization structure for soft state announcements. Such a hierarchy maps to logically independent objects within an application and allows such objects to be treated independently during loss recovery. Since the structure of application data is exposed to SSTP, this eliminates the undesirable coupling induced by a TCP-like in-order byte-stream abstraction.

In Chapter 4, we develop a naming and announcement protocol that uses hierarchical namespaces to support large data stores more efficiently. Our scalable naming and announcement protocol uses namespace summaries to reduce the number of messages in detecting and recovering from losses. A sender transmits new data upon arrival from the application. In addition, the sender

[2]While SSTP does not guarantee end-to-end delay, it uses delay information as hints to determine the operating region in the T_{rec} profile.

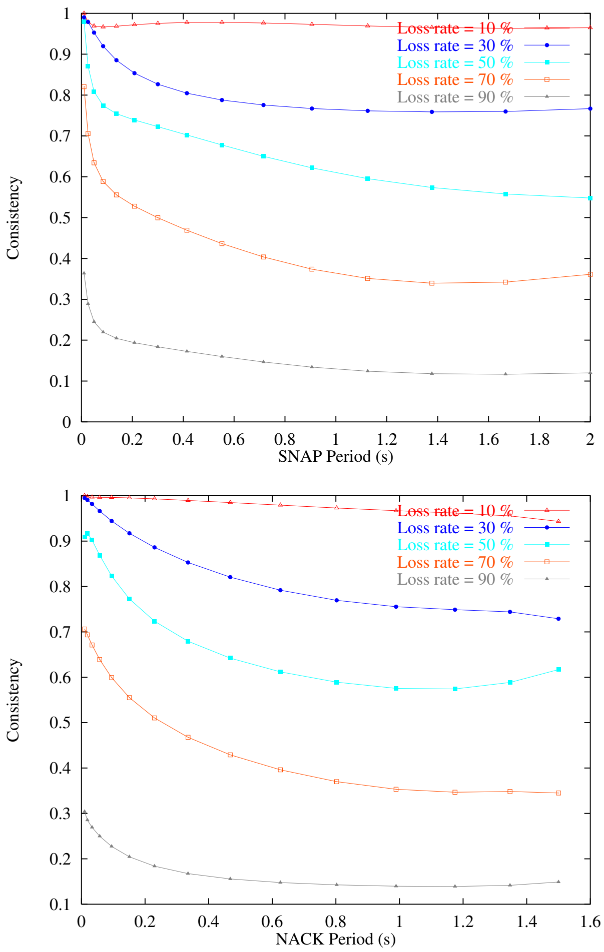

Figure 3.13: When session announcements are more frequent, the receivers detect and correct more “tail losses” resulting in higher consistency. NACK bandwidth has a similar effect on consistency.

also carries out “cold” transmissions of the root summary. Upon receiving a summary announcement, if a receiver detects a mismatch at the root namespace node, a feedback message requesting further namespace repair is scheduled for transmission. In response to such a feedback query from the receiver, the sender (or any participant in a multicast session), responds with a set of next level signatures. In this manner, loss recovery proceeds recursively down the namespace hierarchy. We show here varying the control bandwidth can affect the protocol consistency.

We evaluate the performance of the SNAP protocol when the control bandwidth is varied in the multiple receiver case using simulations.

Figure 3.6.2 shows how consistency improves when the announcement frequency and NACK bandwidth allocated for receiver feedback messages are increased. In Figure 3.6.2 shows the dependence of system consistency on the input rate of new data for different levels of network losses. We find that this dependence is quite similar to the unicast case presented earlier. In Figure 3.6.2,

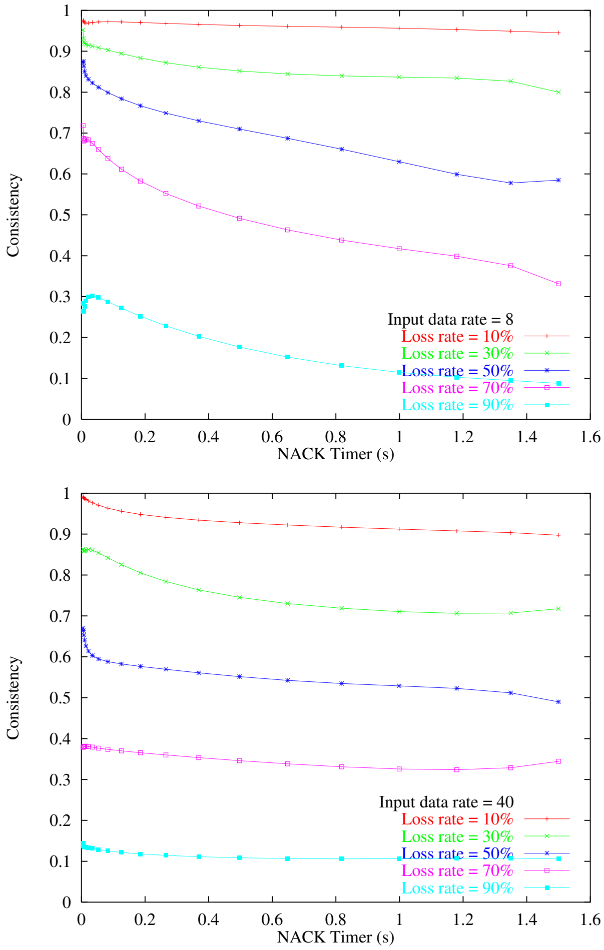

Figure 3.14: Higher the input rate, lower the consistency metric with a fixed announcement bandwidth. The loss rate also adversely impacts protocol performance.

we find that the granularity of objects within the hierarchy determines how effective a given amount of bandwidth is in achieving consistency. We find that tail losses are easily detected with a small amount of announcement bandwidth when the objects are coarse-grained. However, coarse-grained objects make selective reliability less effective since a large object is likely to span relevant as well as irrelevant data items.

An additional advantage of the recursive descent procedure is that a receiver may refrain from requesting further repair along a branch if there is no application-level "interest" for data items belonging to it. For example, a PDA browser may not wish to repair high resolution image data types. The sender communicates such hints to the receivers using application-level meta-data tags associated with the namespace nodes. Receiver-driven reliability using such application-level data names is described in detail in [113]. Our hierarchical data model for SSTP simultaneously solves the namespace scaling problem and provides a rich naming structure that is amenable to ALF.

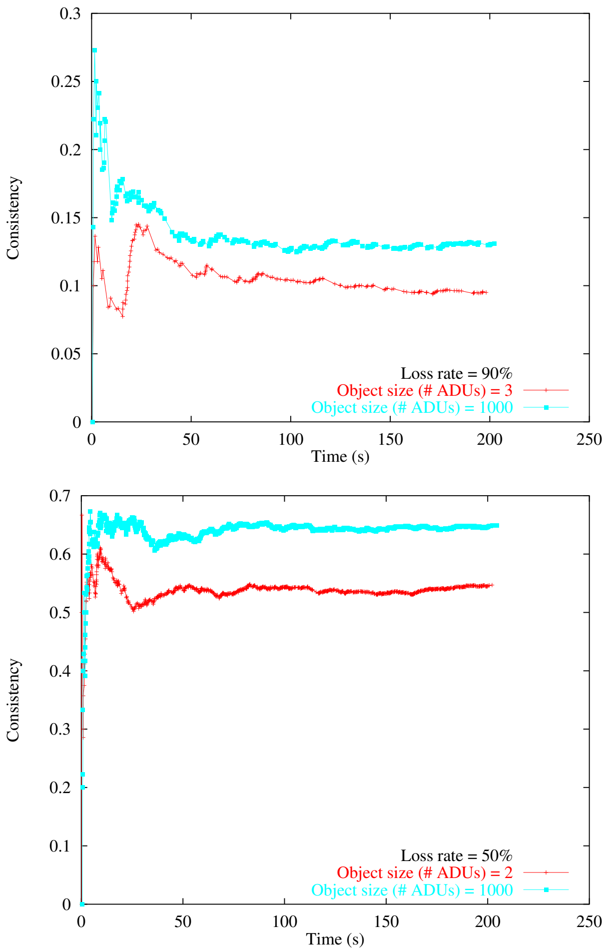

Figure 3.15: Larger the size of each object (in number of ADUs), smaller the number of announcements and hence higher the resulting consistency given a fixed amount of announcement bandwidth.

By controlling the bandwidth allocated to original data transmissions and summary announcements, we can control the level of consistency and latency to recover lost items.

3.7 Related Work

In this Section, we survey some important related work and compare it with our formalism of soft state. Chandy et al. [17] formally define soft state probabilistically and use it as a primitive for exchanging state information for distributed resource management. However, their soft state model is restricted to continuous state variables and their main innovation is in the application of estimation techniques to infer state values between state updates. Their work does not relate the model to existing soft state-based protocols such as announce/listen.

Sharma et al. [126] study the general problem of scalable timers in soft state protocols and present an adaptive algorithm for (i) dynamically adjusting the sender's refresh rate, and (ii) estimating the sender's transmission rate at the receiver to determine timeouts for aging out state.

In [40], Floyd et al. describe the Scalable Reliable Multicast protocol as being eventually consistent. The authors propose an SRM framework, in which data is expired using application hints, analogous to our death process. Handley et al. [51] list eventual consistency as one of the goals of the shared state in the network text editor, NTE. However, neither paper provides an evaluation of system consistency. In [50], Handley demonstrates that adding feedback in the form of "address clash reports" to detect and correct address clashes in an announce/listen-based address allocation protocol can greatly increase its scalability to larger groups. Even though this is a specific case, it motivates us to study the more general case of adding feedback for reliability. In our work, we study the impact of adding receiver feedback in the more general context of soft state transport protocols.

The notion of "probabilistic reliability" was also proposed by Birman et al. [12] in their work on *bimodal multicast*, in which receivers recover a lost stream of items in reverse order. This scheduling choice makes the protocol more stable in large groups, and provides bimodal delivery guarantees, i.e., almost all or very few members receive each transmission (a probabilistic version of the "all or none" atomic broadcast [13]). Our work differs from this in that it is not restricted to multicast transport. In our framework, the delivery of a given piece of data is probabilistic — there is a predictable and tunable likelihood of reception.

Amir et al. [3] present SCUBA, a consensus-based bandwidth allocation strategy for multicast video, where sources gather receiver votes in a scalable fashion to adjust transmission rates. Our work also addresses the issue of receiver-based bandwidth allocation, however, we focus on reliable multicast transports with hierarchically structured data stores. In [4] Amir et al. also introduce the notion of soft state gateways and multiple transmission queues for the scalable exchange of RTCP-like control traffic between islands of high network high bandwidth bridged by low bandwidth links. However, their work does not analyze the performance nor investigate the tradeoffs between different allocation policies. This scheme is a specific instantiation of our more general parameterized SSTP framework.

3.8 Concluding Remarks

In this Chapter, we have presented a model based on Jackson queueing networks that formalizes the notion of soft state. Based on this model, we have introduced a new consistency metric, a probabilistic measure of the effectiveness of different protocol variants, from "open-loop" announce/listen-style communication to feedback-based reliable transport. We show that consistency improves by 10-40% by appropriately aging data items and allocating progressively lower bandwidths for older data. This technique of distinguishing new from old data in combination with receiver feedback in the form of negative acknowledgments improves consistency by 12-50%. In each of these cases, we have shown the optimal bandwidth allocation for which the available bandwidth is best utilized in terms of the consistency metric. Our results presented here appear in [114].

Using the consistency metric as our basis, we apply these results to the design of an adaptive framework for soft state transport protocols (SSTP). SSTP provides a continuum of reliability "levels" that can be customized by the application. It also includes a profile-driven allocation algorithm that uses measurements of network loss rates to adapt to the optimal bandwidth allocation for the required consistency. While SSTP does not solve the problem of determining the available bandwidth, it uses consistency profiles derived from our soft state model to best utilize this bandwidth. In addition, SSTP incorporates application-level framing principles to provide a flexible and powerful primitive for applications to reflect their performance preferences into the protocol machinery.

Chapter 4

Scalable Data Naming

A signature always reveals a man's character — and sometimes even his name.
— Evan Esar

In this Chapter, we enhance the basic soft state building block in two important ways. (1) We extend the simple data model in Chapter 3 to a a hierarchical namespace structure to support applications with large data repositories. (2) We propose a new data naming scheme that exposes the structure of application data to the transport layer, thereby enhancing the expressibility of the application's reliability semantics. We apply our results from Chapter 3 to control the bandwidth allocation between protocol control messages used for namespace announcements and application data messages.

The remainder of this chapter describes the manifestation of our hierarchical naming concept in a concrete protocol that we call the Scalable Naming and Announcement Protocol (SNAP). SNAP provides an application-aware data naming scheme for receiver-tailored reliability and generalizes the announce/listen protocol concept in order to handle large namespaces. SNAP organizes data using hierarchical namespaces, and uses a periodic source-initiated refresh mechanism to announce namespace updates. The hierarchical organization of data coupled with a scalable namespace announcement protocol allows the SSTP framework to scale to long running applications that have a large data footprint. It also allows receiver applications to tune the semantics of reliability on a fine-grained basis. To demonstrate the efficacy of our scheme, we have designed and implemented our scalable naming and announcement protocol (SNAP) in the multimedia application toolkit MASH [82] as a reusable protocol module.

The rest of the Chapter is organized as follows. Section 4.1 motivates the work described in this Chapter and answers some key questions. Why do we need application-aware data naming? How can we generalize the basic soft state model in Chapter 3 to develop a more comprehensive multicast transport protocol that gracefully handles real-world applications with large namespaces and also supports selective reliability? In Section 4.2 we describe the hierarchical data naming scheme of SNAP. An integral portion on the data naming problem is the source naming problem and Section 4.3 describes our approach for global and unique identification of sources that is independent of network layer addresses. The actual meta-data dissemination protocol is described in detail in Section 4.4, and we report on its resulting performance in Section 4.5. In Section 4.6, we describe our current prototype API to SNAP. Finally, in Section 4.7, we summarize the contributions of the work presented herein, and directions for future research.

4.1 Motivation

A fundamental challenge in the design of a reliable multicast protocols is the so-called *implosion problem* [106]. If the receivers in a multicast group all react to a packet loss simultaneously by transmitting a control message back to the source, a traffic impulse implodes upon the source, and for very large groups, this implosion effect not only overwhelms the processing capability of that source but severely congests the network. One way to overcome this implosion problem is to simply omit feedback mechanisms altogether, an approach that has been quite successfully adopted in the multicast session directory service that is implemented by the MBone session directory tool *sdr* [53]. In this approach, the sending application disseminates a dynamic data store as a set of key/value pairs represented as a table. The sender protocol periodically multicasts each entry in the table to some agreed upon multicast group, and receivers interested in the data simply tune in to the multicast group in question and listen to the key/value bindings. Over time, each receiver builds up a copy of the data store and the table is eventually received reliably in its entirety. This style of open-loop communication is often called an *announce/listen* protocol [122] because senders periodically *announce* their data while receivers simply *listen* to the announcements to build up the data store.

Although the announce/listen framework is robust, simple to implement, and easy to understand, its performance is suboptimal both in terms of bandwidth (because data is redundantly transmitted in a continuous fashion) as well as delay (because the sender schedules data transmissions at fixed intervals ignorant of packet losses or receiver interest). Moreover, as the data store becomes large and/or receivers become interested in only small subsets of the overall available data, the approach becomes ever more inefficient. To overcome these limitations, the Scalable Reliable Multicast protocol (SRM) [40] adds a level of indirection to the announce/listen framework. In SRM, a source enlists the announce/listen framework to disseminate "meta-data" that summarizes all of the available data without actually sending it. In turn, receivers use scalable feedback mechanisms (based on "multicast damping", see [40]) triggered by the meta-data to request the delivery or retransmission of any data that is desired. (As an optimization, a source might multicast new data once upon creation to avoid the delay incurred by first announcing its existence to trigger the receiver request.) This approach is often called a "receiver reliable" protocol because the receiver rather than the source implements the reliability requirements. As a result, different receivers can implement selective reliability and tune their reliability requirements to the local user's disposition or application environment.

By allowing a receiver to selectively repair portions of the data stream, we can effectively account for an application's semantics in the design of its network protocol. This approach to protocol design, which derives from the Application Level Framing (ALF) protocol architecture [23], is a boon to protocol performance because the application is optimized for the network and vice versa. For example, in a shared whiteboard application, a receiver might issue retransmission requests for lost data on the current page and ignore missing data for pages that are not in view (or perhaps repair this data at low rate in the background). If we instead used a protocol that was ignorant to application semantics, an application might have to retrieve many megabytes worth of undisplayed data (and incur an unreasonably lengthy delay) before obtaining perhaps the small amount of data that represents the page in view.

While the premise of a receiver-reliable protocol is conceptually straightforward, a problem arises when we attempt to realize the protocol with traditional primitives. If, for instance, we name protocol data units with sequence numbers as is traditionally done in many reliable transport protocols like TCP, we would hide the structure of the underlying application data — e.g., how would a whiteboard know that the data referenced as sequence number 8792 is associated with page 12? In other words, sequence numbers map all application data onto a linear namespace and thereby discard its semantic structure. Yet it is precisely this structure that is crucial to optimizing protocol performance for the application in accordance with ALF. To this end, instead of sequence numbers, we might use a two-dimensional structure, consisting of page numbers and drawing operation identifiers within a page. The announce/listen component of SRM would disseminate the page structure as meta-data, which in turn, would allow a receiver to associate packetized data items for meaningful application structures.

In addition to requiring a rich naming structure, receiver-based reliability mechanisms must scale to very large groups and very large data sets. Not only might the data store itself become large, but the meta-data that describes that data store might also become large. Hence, we must ensure that the meta-data dissemination protocol scales gracefully with the amount of data in the session.

In this Chapter, we propose a novel naming scheme that simultaneously solves the meta-data scaling problem and provides a rich naming structure that is amenable to ALF. Our naming scheme exploits hierarchy to effectively add a level-of-indirection to the meta-data dissemination protocol. In our approach, hierarchy provides a summarization structure that we utilize to reduce the amount of overhead in the announce/listen component of the meta-data dissemination protocol. Here, senders announce meta-data summaries to the multicast session, which in turn, trigger namespace recovery mechanisms at each receiver in a receiver-directed fashion. In short, senders announce "meta-meta-data" that describes the namespace and receivers use the SRM recovery mechanism to reliably retrieve an arbitrarily large namespace. An elegant consequence of our hierarchical representation is that we can control which pieces of the namespace are disseminated at what rates using announce/listen and which pieces are repaired using receiver-directed recovery. This provides a tunable tradeoff between the consistency among data stores at each receiver and the bandwidth consumed by the protocol's control traffic. In addition to this tunable tradeoff, the namespace hierarchy allows an application to easily impose its own data structure over that hierarchy and thereby infer application-level meaning from the transport-level name, which is not generally possible with a flat sequence number space.

4.2 SNAP: Hierarchical Data Naming

The SNAP naming system uses a hierarchical structure to represent and name application data. Hierarchical naming allows data transmission from different objects in the application data store to proceed independently. The mapping from application-level structures to a SNAP namespace is flexible and under application control and provides a sufficiently rich structure for applications such as shared whiteboards, webcast applications, etc. The new naming scheme also provides a *common syntax* between the the application and transport that enables receiver reliability.

We start with an overview of the main components of the SNAP naming hierarchy — application data units, fragments, nodes, namespaces, and name maps, illustrated in Figure 4.1.

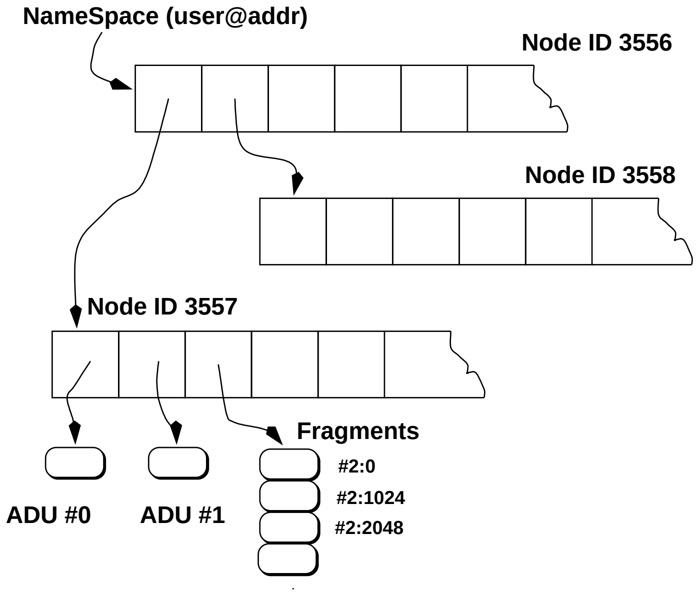

Figure 4.1: Example illustrating namespace, nodes, ADUs, and fragments.

An *Application Data Unit (ADU)* is the smallest unit of data that can be processed independently by an application. Applications hand ADUs to the transport protocol "atomically" to be delivered to the session. Examples of an ADU include a single scan of a JPEG image, or a whiteboard drawing operation [63].

While ADUs can be of arbitrary sizes, a *packet*, which is the unit of network transmission, is limited by the characteristics of the path between sender and receiver(s). The maximum transmission unit, or MTU, of a network determines the amount of data that the transport protocol can transmit at a time without having to fragment it en route to the receiver. While the path MTU in a unicast session is clearly defined as the smallest MTU of any link in the path from a datagram's source to its destination [70], the same is not true in a multicast session because it involves multiple receivers and hence multiple data paths from the sender to receivers. If a link has a smaller MTU than the sender's packet size, the packet must be fragmented and reassembled at the edges of the low MTU segment. Relying on IP to perform link-level fragmentation and reassembly is undesirable because fragment loss can result in a (multicast) retransmission of the entire packet. To limit the inefficiency resulting from packet retransmissions, it is important for the transport protocol not to choose a large packet size. Since there is no clear method to determine the optimal transmission size in a multicast session, we use 1024 bytes, a common transmission size for most link technologies[1].

If an ADU handed to the SNAP framework exceeds the packet size, it is fragmented into multiple pieces each of which fits into a packet. A fragment loss does not trigger the retransmission

[1]For links that have smaller transmission sizes than 1024 bytes, the protocol can limit the bandwidth consumed by retransmissions using scoped or localized repair to avoid retransmission floods affecting all group members.

of the whole ADU, but just the lost fragment, and that too only if the application cares for it. Since an application is incapable of processing a fragment without receiving the entire ADU, ADU fragments are reassembled before propagating to the receiving application. While the pure ALF model recommends that the application be "network aware" and only transmit ADUs that don't exceed the MTU, handling fragmentation within the transport protocol shields the details of the network for indifferent applications, and applications that genuinely have large ADUs. Fragments are identified using a starting byte offset and length in bytes. Byte offsets are required to account for the fact that different members may fragment an ADU differently depending on their interface MTU.

A *namespace* is a hierarchical structure onto which an application maps its data. We only allow each data source to create a distinct namespace in the session, and do not currently permit sources to share namespaces. Allowing multiple data sources to modify a shared namespace introduces problems of inconsistency when concurrent and conflicting operations are performed on the same namespace. For this reason, we defer the design of consistency algorithms for our framework as future work, and instead, only allow source-specific namespaces.

Nodes (or "containers") in the namespace hierarchy are called *nodes*. Nodes refer to ADUs or other nodes or both. A node is also the unit of selective reliability, i.e., by selecting specific nodes, a receiver can choose to repair data belonging to the corresponding nodes in the namespace. This, as we have argued, is essential for customizing the receiver's data set. Each node is identified by a *node ID* (NID), that is assigned to it by the source sequentially in the order of creation. The key advantage of hierarchical namespaces is that the application can generate data at any time into a node. In other words, there is no notion of "closing" a node once it has been opened and written into.

Certain reliable multicast applications such as shared whiteboards use a persistent data model, where data lingers in the session for the entire lifetime of the session. In general, this is a convenient model for applications that need to support late joiners[2]. However, data persistence causes difficulties in the NID assignment scheme. Consider a scenario where a source, after generating some data in nodes, say 1 through k, crashes and re-joins a session. Assigning node descriptors sequentially starting from 0 causes collisions between the first k new nodes generated by the source and data already generated during its earlier incarnation. By randomizing node descriptors, we can greatly reduce the likelihood of such collisions. The following analysis of collision probabilities shows that picking a random initial node descriptor (IND) and subsequent sequential assignment results in a smaller probability of collision than picking each node descriptor randomly. Figure 4.2 illustrates these two cases. n is the average number of nodes generated during a single incarnation. b is the number of bits used for node descriptors (We use $b = 32$). Then, $N = 2^b$ is the total number of available nodes. Assume that b is sufficiently large so that $n << N$. We evaluate the probability of collisions among node descriptors in different incarnations. The probability that no collisions occur with the randomized IND is given by $(1-\frac{2n-1}{N})$. With the scheme that picks a random descriptor for each node, the probability that there are no collisions is $\prod_{i=0}^{n}(1-\frac{n+i}{N}) \leq (1-\frac{n}{N})^n \leq (1-\frac{2n-1}{N})$. Hence, the randomized IND scheme with subsequent contiguous allocation is better. Another advantage is that selecting contiguous node descriptors allows receivers to detect lost nodes in a data-driven manner from gaps in the NID sequence.

[2]This does not require that all the data be stored in main memory. The application may spool old data to secondary storage.

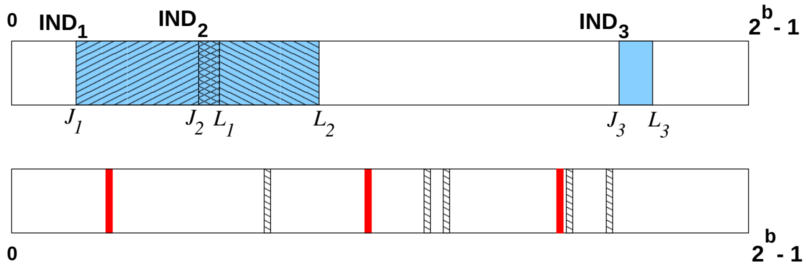

Figure 4.2: Two randomized descriptor allocation schemes. J_i (Join) and L_i (Leave) represent a source's IND and final descriptor in incarnation i. In the first case, where node IDs are allocated sequentially, collision occurs when two ranges have even one ID in common. In the second allocation scheme, a collision occurs when an ID is picked that coincides with any one of the n previously allocated node IDs. In this case, we assume that collisions within the same incarnation can be detected and avoided.

As an optimization to efficiently handle applications such as the LBL shared whiteboard tool, *wb* [63], that generates a large number of (small) ADUs within each page or node, we allow applications to use sequence numbers at the lowest level. ADUs are numbered sequentially within a node. The advantage of this optimization is that data-driven, loss detection is possible in a number of cases. Another advantage is that the extra *ADU sequence number* reduces the state of the index data structure that maps application-level ADU names to nodes. An ADU is therefore identified within the transport protocol by the tuple $(NID, ADU seqno)$.

The binding between application names and node IDs is called the *name map*. This is an optional component of the namespace system that is used by the application maps its data onto the SNAP namespace by assigning a name to each node. For example, names could be URL strings in the webcast application. However, it is conceivable that a sophisticated application may use other types of names, perhaps with a different profile specification just as different application-specific RTP [124] profiles are possible. Our framework allows a different namemap module to be used with the core naming and announcement protocol.

4.3 Source Identifiers

Data names are unique with respect to an origin source, and SNAP source identifiers are not derived from network-layer addresses, an important design choice that allows us to interoperate in an evolving network architecture that comprises network address translators and other proxying agents within the core of the network that do not preserve the IP header as the packet traverses through the network.

A namespace is source-specific and reflects the structure of data generated by that source. In typical reliable multicast applications such as whiteboard, each receiver maintains a copy of every active source's namespace. We need a source identification mechanism that meets the following requirements.

- **Uniqueness.** Two distinct sources must have distinct identifiers within a session to enable receivers to determine the owners of data and maintain source-specific state. A source ID collision occurs when two distinct sources use the same identifier while transmitting data;

this is an event that must ideally be avoided, or at the very least, its occurrence must be minimized.

RTP [124] uses randomly generated 32-bit source identifiers for multicast audio/video transmission and detects collisions and resolves them, but does not perform any repair when one is detected. The only ill-effects of a collision in this case are degraded quality for a brief duration of time, until the collision is resolved. For reliable multicast applications, collisions could potentially cause irrecoverable damage to application data.

- **Time invariance.** Unlike RTP applications like audio and video where application state is ephemeral, reliable multicast applications require that a source identifier for a given source has to be the same at all points in time within the session. In particular, when a source leaves and rejoins the session at a later point in time, it must be possible to re-use the same source ID as in its previous incarnation. The same applies to source crashes and subsequent recovery. Existing schemes like RTP's randomized identifiers violate this requirement.

- **Terminal independence.** In several reliable multicast applications, a source must have the choice of retaining the same identity independent of the terminal or host used to participate in the session. A source (user) must also be able to retain ownership of data created by it in an earlier incarnation in the same session. Furthermore, a single reliable multicast session should be amenable to multiple sources from the same end host or user. These requirements preclude the use of user- and machine-specific identifiers based on a combination of host IP address and user name.

To support time invariance, our source identification scheme allows sources to use arbitrary application-level strings. For example, a user in a whiteboard application could identify herself using a user name and host address, as shown in Figure 4.1. However, since string names have variable length and incur a high per-packet overhead especially for small ADUs, we hash the name into a fixed-length 64-bit integer derived from the MD-5 function applied on the string. Because the original identifier is an arbitrary user-supplied string that can be retained when moving from one terminal to another, terminal independence is easily achieved as well. Our solution minimizes the probability of collisions and resolves any collisions that might occur using the technique described in the RTP specification [124]. In addition, we ensure that the two important requirements of time invariance and terminal independence are met. The probability of source ID collision in this scheme is the same as in RTP, but that we have achieved the additional goals of time invariance and terminal independence.

4.4 Discovering the SNAP Namespace

SNAP receivers need to discover each source's namespace so that they can issue retransmission requests. To disseminate the namespace structure reliably, we use the basic SRM mechanism to disseminate the namespace tree and session messages to trigger namespace repairs. A special index node `/map` is allocated to contain the namemap bindings. The /map node always appears as the left-most child of the root node in a namespace. Figure 4.3 illustrates this.

Receivers maintain a snapshot of the namespace generated by each source in the session. When a source creates a new namespace node, a message is transmitted from the `/map` node. Trans-

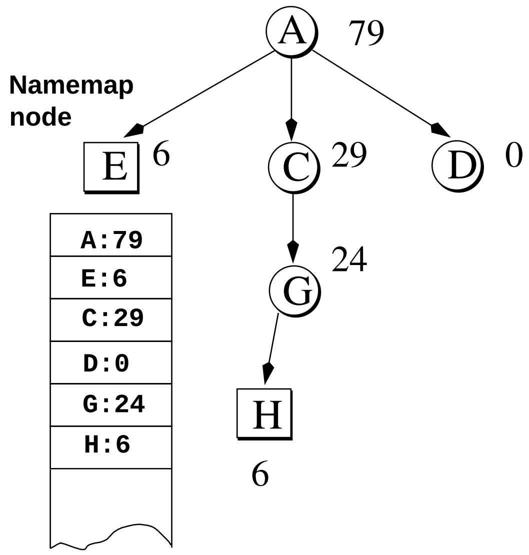

Figure 4.3: Namemap bindings are treated as regular data by the SRM recovery algorithms, permitting re-use of protocol machinery. It also allows receivers to detect lost nodes in a data-driven manner. For simplicity, we have ignored fragmentation in this example.

mitting new data from this node indicates to the receivers that a new node in the namespace has been created. In response, each receiver updates its snapshot of the namespace tree for the corresponding source. Loss detection and recovery occurs in two modes: (i) data-driven loss recovery, when the arrival of a data segment signals a loss to the receiving application, and (ii) control-driven loss recovery, where the arrival of a control or announcement message enables the receiver to detect and trigger recovery steps.

4.4.1 Data-driven Loss Detection

We first describe the data-driven loss recovery mechanisms that enable receivers to learn about undiscovered pieces of the namespace tree. Since data is identified by NID and ADU seqno, losses within the same node can be identified by gaps in the ADU sequence number field. A gap in the NID sequence number indicates a missing node. Since ADUs are allowed to span multiple packets, we also need a mechanism to detect missing fragments. Fragments carry byte boundaries which can be used to detect missing fragments. Each fragment also carries a “more” bit that is used to signal to the receiver(s) that more fragments are to be expected within an ADU. The last fragment of an ADU has the “more” bit set to zero. On receipt of a fragment with “more” set to 0, the receiver checks to see if an entire ADU has been received before propagating it up to the receiving application. When a receiver receives a new ADU before completely receiving the previous ADU, it schedules a repair request for the trailing fragment(s) of the ADU. For example, assume that a receiver has seen the first n bytes of ADU a when ADU $a + 1$ starts to arrive. A request is scheduled for $a : n + 1, EOA$, where EOA is a special value that indicates the end of an ADU. In response, the rest of the ADU is retransmitted either by the original source, or by any peer member in the session that successfully received the data. The transport protocol does not act independently of the application in repairing losses. Lost data is repaired only if the application so desires. After performing the necessary name translation from node ID to name, SRM queries the application with the node name to determine whether a repair request needs to be scheduled[3]. If so, a request is scheduled according to a randomized timer algorithm similar to the basic SRM loss recovery scheme [40]; if not, the loss is ignored. In response to a repair request, any peer member in the session is allowed to respond with data. Data is buffered at the application at both the source and receivers, and the transport protocol only maintains the structure of the namespace. Upon receipt of a repair request, the transport protocol queries the application for the requested data.

4.4.2 Announcement-driven Loss Detection

While data-driven loss detection mechanisms handle several cases of losses, they do not detect "tail losses" within a node, or tail losses of missing nodes. Late joiners to the session also need a special mechanism to discover interesting portions of the namespace hierarchy. In *wb*, a new receiver queries the source using for a list of all its pages by transmitting a "page vector request". The source in response replies with a list of all its pages. This scheme, where the source transmits a vector of available pages does not scale well to long-lived sessions with a long history. Imposing an application-defined hierarchy on the namespace permits a receiver to navigate the namespace and selectively fetch only branches of the tree it is interested in. In order to transmit a concise representation of the namespace hierarchy, we summarize the state of its namespace using the signature function.

Source-based announcements fall in the general category of announce-listen protocols where sources periodically announce their data and receivers listen to these announcements to reconstruct the data. Announce-listen protocols are conceptually simple and do not suffer from feedback implosion problems when used over IP multicast. The session announcement protocol (SAP) [49] is an "open loop" reliable protocol that multicasts data periodically. A receiver simply waits for source announcements to receive all data. In the basic SRM algorithm with linear sequence numbers, the source periodically transmits "meta-data" indicating the last ADU sequence number transmitted so far. SRM receivers use this information to NACK data. In SNAP, each source announces its signature which is "meta-meta-data". Receivers NACK meta-data, to repair pieces of the namespace, and data, just as in SRM. We use the SRM slotting and damping algorithms [40] to reduce the amount of redundant SNAP traffic.

Signatures

The exact structure of the namespace tree can be conveyed completely and without any loss of information by providing an in-order and pre-order traversal of the hierarchy. For long-lived sessions with large and persistent namespaces, this is too large and expensive to disseminate to the entire multicast group. In order to limit the bandwidth consumed by state announcement messages, we need a more compact representation, or "fingerprint", of the namespace hierarchy. We call this compact representation of the namespace hierarchy its signature. The signature serves to inform all receivers about the current structure of a node and all its descendants, and we define it as follows.

[3] Alternatively, an application declares its preferences in a profile at startup time.

The signature, s, is a function from a node into the set of 64-bit integers. The signature of a node is defined recursively as the last ADU sequence number contained in it. If a node C is a leaf-level node that contains only ADUs, its signature is simply the right edge of the node. This is represented as the highest sequence number in its ADU sequence, and the highest byte of the last ADU. The highest ADU sequence number helps receivers detect trailing losses within a node. The highest byte of the last ADU helps receivers determine missing fragments.

For an internal node that points to other nodes, the signature is computed as a hash function over its own right edge and the signatures of its children nodes $C_1, C_2, \ldots, C_n$. An example of such a hash function h is MD-5 [119]. MD-5 has the special property that no known computationally feasible technique exists to produce two distinct messages having the same hash value. Such a hash function gives us signatures that are unique with high probability. The signature of a source's namespace can be viewed as a unique (with high probability) fixed length representation of all its data. However, we note that the signature is a one-way function and cannot be reversed to reconstruct the tree that generated it.

$$s(C) = \begin{cases} \texttt{right\_edge}, \text{ if } D \text{ is a leaf-level node;} \\ h(\texttt{right\_edge}, s(C_1), \ldots, s(C_k)), \text{ otherwise} \end{cases}$$

where `right_edge` of a node C is defined as the tuple (max ADU seqno, last byte offset in last ADU) within C.

Note that it is necessary to transmit the highest byte offset, in addition to the highest ADU sequence number, for the following reason. A source announcement may be transmitted after a source has begun transmission of an ADU, but has not completely transmitted it. This may occur because fragments of the ADU are streamed through a token bucket that has a maximum burst size. If we allow source announcements to be sent only on complete ADU boundaries, we can use the highest ADU sequence number alone to represent the right edge.

Source Announcements

As explained earlier, each source announces "meta-meta-data" and "meta-data" to the session. Periodically, each source multicasts a session announcement message containing the signature of the root and the signatures of at most k nodes. The period between announcements and the amount of data to send in each announcement are scaled to constrain control traffic and the exact rate is determined by external policy. For example, if we constrain sources to allocate no more than 5% of their session data bandwidth on announcements we restrict the size and/or rate of announcement messages. From the announcement rate and the network MTU, we can compute a maximum announcement packet size, which in turn determines k. A higher frequency of session announcements results in lower loss detection latency at the receivers. We now outline a heuristic that picks the k "best" nodes whose meta-data is transmitted along with the root signature. Since we cannot transmit meta information about all the nodes in a source's namespace, we give priority to nodes that have not been announced for the longest period of time. We use a randomized scheme similar to lottery scheduling [139]. The analysis of this algorithm is akin to the coupon collectors problem [45]. The expected time of the algorithm is proportional to $(n \log n + n \log^2 n + |C|)$, where n is the target number of nodes to be selected. The total number of nodes is C. The details of the algorithm and its analysis are described in [111].

Receiver Processing

We now proceed to discuss the receivers' processing upon receiving an announcement packet from source S. Each receiver uses the signature to determine if it has a correct snapshot of S's namespace. If the local copy of S's signature does not match the one carried in the announcement, the receiver transmits a request to repair the root node of the namespace. All requests are multicast to the session and use a randomized backoff scheme to reduce the number of duplicates when losses are correlated.

If the receiver is interested in the entire namespace, it performs a systematic *recursive descent* procedure to explore the namespace, detect mismatched nodes, and repair them. However, the recursive descent algorithm is the last resort and is not invoked in the common case, where applying the meta-data updates transmitted with each signature is sufficient to bring the receiver up-to-date.

Below, we describe the receiver's processing for the different loss cases.

1. **No Loss.** On receipt of an announcement message from source S, a receiver R first compares the root level signature sent by S with the corresponding local version. If the two signatures match, R assumes that it has a current snapshot of S's namespace. Note that there is a vanishingly small probability that two different trees will generate the same MD-5 root signature. However, the inconsistency is only transient as a new signature is periodically transmitted by the source as more data is transmitted and the namespace tree develops.

2. **Missing ADUs.** If the local signature computed on S's namespace does not match the one carried in the announcement, the receiver invokes loss recovery for the namespace meta-data by issuing a a *namespace repair request*. The purpose of the namespace repair request is to query the source (or any other eligible member in the SRM session), to repair the state of a mismatched node. In response, the source, or any member with an up-to-date copy of the source's tree, multicasts a namespace repair packet for a requested node with a list of all the children and their signatures. The receiver descends the namespace tree recursively in this manner to locate missing branches. When a leaf-level node with only ADUs is reached, the node's signature gives the last ADU generated, and the receiver is now in a position to schedule a repair request for the missing ADUs. This is illustrated in Figure 4.4.

3. **Missing namespace nodes.** A receiver may lose entire namespace nodes and the corresponding meta-data information from the /map node, perhaps due to persistent congestion. Missing namespace nodes can be detected by gaps in the sequence of map entries and recovered from. Tail losses from the /map node are detected when a receiver receives a session announcement containing the map node's signature. This is illustrated in Figure 4.5. If a receiver receives data in a node whose name mapping it does not yet know, it generates a repair request for the missing mapping. Rather than discard ADUs whose names are unknown at the receiver, we buffer the data at the SRM layer until the corresponding name of the ADU is available to propagate up to the application[4]. Mapping information appears as regular data to the SRM protocol and we leverage the SRM request/repair machinery to recover lost portions of the namespace.

[4] Alternatively, unnamed data may be passed up to the application and the application could be notified when the name is discovered.

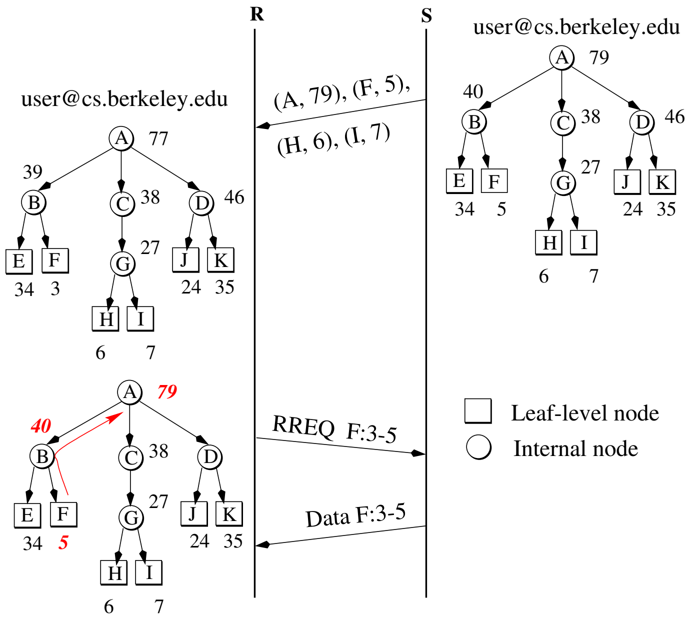

Figure 4.4: Recovering from tail losses using SNAP announcements. A-K are NIDs and the number beside each node is its signature. For leaf-level nodes, the signature is also the right edge of the node, and provides sufficient information to schedule repair requests. This figure shows a receiver recovering from a tail loss in node F.

4. **Selective reliability.** Because SNAP uses hierarchical naming, receivers can perform selective reliability by deciding when and when not to generate repair requests for nodes in the namespace tree. This is especially beneficial when a new user joins a long-lived *wb* session to review a specific page, say the agenda page from the a weekly meeting in January 1997. By organizing its data hierarchically, the source allows receivers to tailor its requests to receive only the data it requires. Figure 4.6 shows an example where R does not care for data below node C. Nodes whose contents are uninteresting to a receiver are termed *don't care* nodes. In order to prevent spurious signature mismatches at higher levels of the namespace tree, the receiver maintains the most up-to-date signature for the root of the *don't care* subtree.

4.5 SNAP: Performance Evaluation

To evaluate the performance of SNAP within the SRM framework, we conducted a simulation study using the network simulator *ns* [81]. We used a multicast group with one source and up to 55 receivers[5]. Background traffic in the simulations was generated using TCP connections, which induced packet losses. The topology used was a tree of degree 4, with the source at the root. A constant bit-rate data source was used with a randomly generated namespace. We looked

[5]Because of the prolific memory requirements of *ns* multicast simulations, we were unable to experiment with larger groups.

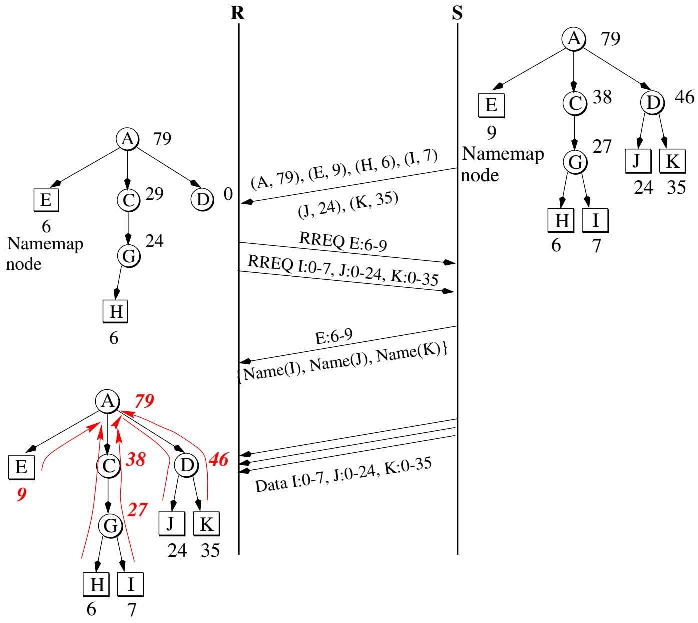

Figure 4.5: Receiver R recovers from missing nodes that went undetected because of tail losses in the namemap node `/map`. Once again, the notation is the same as in Figure 4.4. First, R recovers missing bindings from the map. Missing data from the nodes I, J, and K is then recovered.

at two metrics: (i) convergence time, or the latency to recover from a tail loss, and (ii) session size scalability which is measured by the bandwidth used by SNAP control traffic as the session size is increased. In order to measure the worst case convergence time for a late joiner, we used the recursive descent scheme, without any heuristics or selective reliability.

We define the convergence time as the elapsed time from the instant at which a packet is dropped in the network (as a result of overflow at a router queue) to the instant that a receiver receives the data packet. We measure the worst case convergence time in our simulations, i.e., time for the last receiver to recover from the loss. The convergence time has three components, the average waiting time for the first update from the source, the time taken to discover the location of the loss using SNAP, and the time to recover from a loss. Figure 4.7 shows the convergence behavior of the state update protocol with varying periodicity of updates. As expected, the convergence time improves with more frequent SNAP announcements. However, there is a tradeoff between the amount of bandwidth consumed by SNAP control messages, and the recovery latency of the protocol in discovering and reacting to losses. This convergence time is significantly reduced when applications exercise selective reliability to selectively repair portions of the namespace. Our simulations also used nodes with exactly one ADU to stress the control-driven repair mechanisms. Therefore, the observed latencies represent the worst-case scenario. When used in applications that continuously generate data in each node, data-driven recovery is likely to repair most losses with lower latency.

The chief concern with multicasting control messages such as repair requests, state announcements and namespace repair requests/replies is the amount of bandwidth consumed in very large session sizes. To evaluate the effectiveness of suppression resulting from the “slotting and

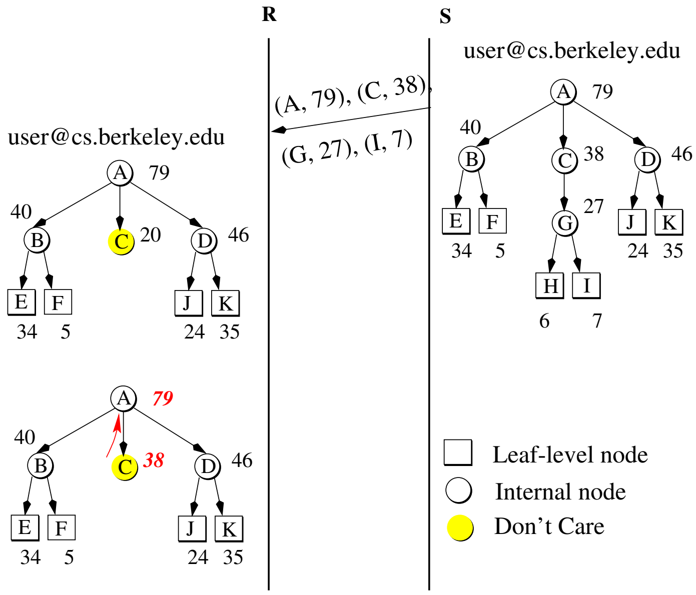

Figure 4.6: Selective recovery using SNAP. If the receiving application at R does not care for data belonging to a certain subtree, it simply updates the signature of the node at the root of such a subtree, and refrains from scheduling a repair request for portions of the namespace below those branches. The correct signature at the root of the *don't care* subtree allow the R to compute the correct signature at the next higher levels and eventually the signature at the namespace root. This prevents the receiver from generating spurious namespace repair requests.

damping" mechanism in SNAP, we measured the number of copies of each control message multicast to the group. Figure 4.8 shows this behavior as the group size is scaled up to 50 nodes. For SNAP updates, the average number of copies per message was about 3, and remained approximately constant with increasing session sizes. On the average, about 2 copies of a request message are transmitted to the session. This remains roughly constant with large group sizes. Schemes such as local recovery that are generally applicable to SRM can also be applied to this case to improve scaling behavior and eliminate the extra duplicates.

4.6 Implementation

We have implemented a prototype of the naming scheme and SNAP in the MASH [82] toolkit. The MASH platform is a scripting-based programming environment for networked multimedia applications. It provides composable basic building blocks such as network objects, codecs, widgets, and an event-driven programming model. We have implemented SNAP as a library of C++/OTcl classes in the toolkit. SNAP runs at user-level in the same address space as the application. Since the application and protocol share an address space, upcalls [23] by the transport protocol to query the application are implemented as function calls. The namemap that implements the mapping from names to transport level identifiers is a separate module as shown in Figure 4.9.

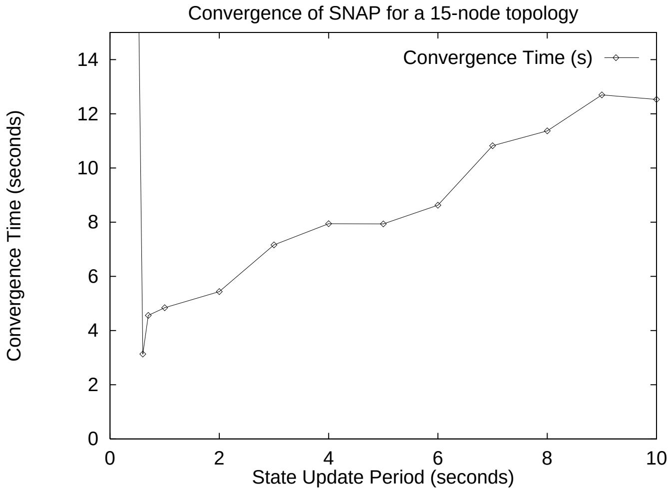

Figure 4.7: Convergence times of SNAP with decreasing frequency of updates.

The software architecture of SNAP is shown in Figure 4.9. The protocol framework has two interfaces: one with an event system that allows the library to register events and invokes the specified handlers when these occur. In our implementation, the Tcl [95] event loop provides these functions, but it is possible to install an alternate event handling system provided it supports an equivalent API.

The other important interface to SNAP is the application-transport interface. Our basic framework has the following API. The direction (upcall/downcall) of each function call in this API is also indicated.

↓ `snap_alloc_node parent`
`snap_alloc_node` allocates a node as a child contained within `parent`, which is the ID of the parent node. Each node ID is 32 bits wide, allowing at most 4 billion nodes per source within each application. This function returns the node ID of the new allocated node to the caller.

↓ `snap_start`
`snap_start` signals to the SNAP layer that there is data ready for transmission. Since the transmission of data is clocked by the underlying rate regulating mechanism, the application merely registers its interest in transmitting data. The application is immediately called back with a request for data. (See `snap_get_data` below.) SNAP sets a timer in its rate control leaky bucket to accumulate enough tokens for the ADU just transmitted. Subsequent ADUs that arrive during this period are maintained in an application level ADU queue.

↑ `snap_recv_data nid seqno data`
`snap_recv_data` is a notification delivered to the application by the transport protocol when a complete ADU has been received either resulting from the original transmission or from a retransmission. Retransmissions are indistinguishable from original data and are subjected to the same rate regulations. If an ADU has to be fragmented at the sender, it is

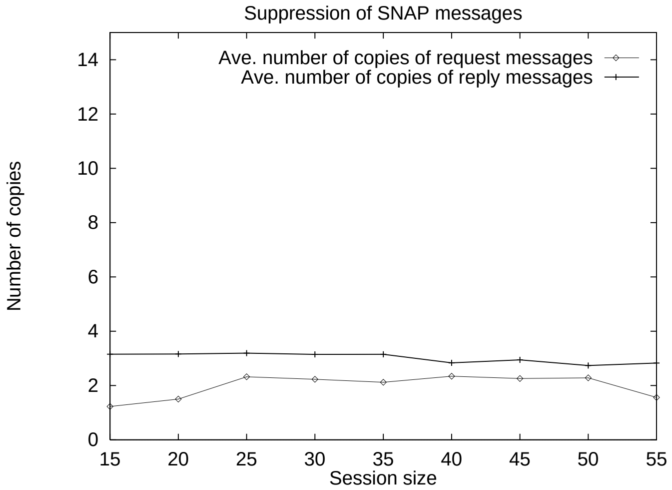

Figure 4.8: Effectiveness of SRM-style suppression in SNAP. On the average, about 2 identical copies of a request and 3 identical copies of a response are multicast.

reassembled before delivery up to the application.

↑ `snap_get_data nid seqno`
On receiving a repair request, the transport protocol requests the application for the ADU corresponding to node `NID`, sequence number `seqno`. This function is invoked when the token bucket internal to the SNAP library has accumulated enough tokens to transmit an ADU that was registered using a prior `snap_request_to_send`. This is also used by the SNAP library when responding to a repair request generated by a session member. This notification to the application is necessary since buffering is the responsibility of the application.

↑ `snap_recover nid seqno`
Receiver-tailored retransmission is achieved via this function call. `snap_schedule_request` explicitly queries the application to check if a repair request for the lost node needs to be scheduled by SNAP on behalf of the application.

A key ingredient to the success of the general framework approach to reliable multicast is experience drawn from design and implementation of a variety of applications. We have implemented a number of existing applications including the MASH MediaBoard [133], and mashcast [145] to effectively demonstrate the power of application-level framing.

4.7 Concluding Remarks

In this Chapter, we have presented a generalized framework for reliable multicast that supports diverse reliability semantics. The two key components of the framework are (i) a hierarchical naming system, and (ii) a scalable session announcement protocol. The result is a receiver-driven reliability protocol where receivers tailor their reliability requirements. Based on simulation studies

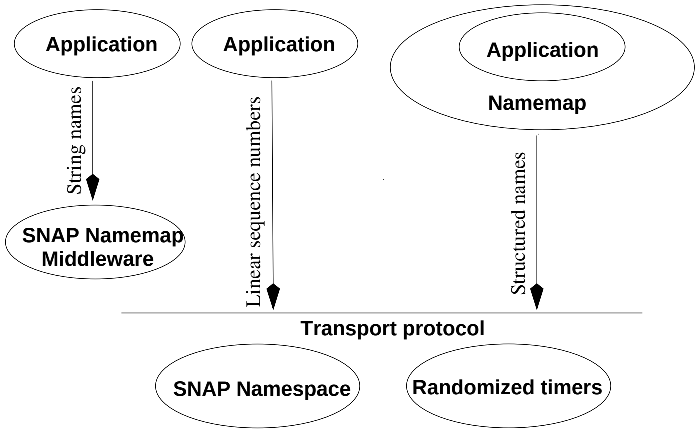

Figure 4.9: Namemap is a middleware component that implements the mapping from structured names used by the application to transport level identifiers. SRM timer mechanisms and SNAP are core transport layer functions. The transport layer exposes a low-level API to the application or middleware component that enables selective reliability.

of convergence time and message complexity of the protocol, we demonstrate that the protocol's bandwidth consumption scales well to large group sizes. We have designed and implemented this Scalable Naming and Announcement Protocol (SNAP) as part of the MASH [82] toolkit. We discuss some avenues for future work in this area.

Global recovery in SRM results in request and reply floods transmitted to the entire group, even for losses that are localized. Several solutions have been proposed for local recovery in the literature [74, 40, 98, 127]. Integrating SNAP with a local recovery scheme could provide greater scalability. Finally, the deployment of IP multicast in the Internet has been impeded to some extent by the absence of a multicast congestion control algorithm. This problem has recently received significant attention in the research community. We hope to use our framework as a vehicle to design, test, and deploy different congestion control schemes on the Internet.

Our work on scalable data naming and soft state namespace updates in this chapter provide the basis for a reliable multicast transport framework that can be used in a wide range of applications. In Chapter 7, we describe an implementation of these techniques in libsstp, a soft state-based transport protocol framework and describe how applications tune reliability semantics in different ways. We also present a scaling study of the underlying timer mechanisms used during loss recovery of data and meta-data messages in Chapter 5. In Chapter 6, we compare in-order and out-of-order data delivery at the receivers and propose that the transport protocol leave ordering issues to the higher layer application so that interactivity is not compromised. The use soft state and randomized timers for loss recovery allow us to scale SSTP to large session sizes, while SNAP and out-of-order delivery improve interactivity.

Chapter 5

Asymptotic Scaling of Randomized Timers

I waited and waited, and when no message came, I knew it must have been from you.

— Ashleigh Brilliant

The SNAP protocol presented in Chapter 4, performs loss recovery of data as well as meta-data items in a multicast session using the "slotting and damping" method with randomized timers. As discussed earlier in Chapters 1 and 3, a key issue in designing a robust multicast feedback scheme is to avoid tight synchronization between group members. The previously known technique of slotting and damping provides a way to gather feedback from receivers, but uses randomization to break the synchronization between multiple receivers attempting to signal the same loss event to the sender and other members of the session and forms the basis of our loosely synchronized receiver feedback algorithms. In this Chapter, we present a detailed evaluation of the slotting and damping algorithm. The Chapter proceeds as follows. We present a brief overview of the randomized timer algorithm under study in Section 5.1. In Section 5.3, we describe our evaluation methodology. We discuss the effects of varying the protocol parameters for the various topologies in Sections 5.4, 5.5, and 5.6, and conclude in section 5.7.

5.1 Overview of Randomized Timers

A fundamental problem in the design of a reliable multicast protocol is the well-known *message implosion* [40, 106] problem. Reliable transport protocols rely on some form of feedback between or among communicating end-points to confirm the successful delivery of data. While some protocols rely on positive acknowledgments or ACKs (signalling the successful receipt of data), others rely on negative acknowledgments or NACKs (signalling the failure to receive expected or desired data). Positive acknowledgment-based schemes are successful for reliable unicast transport but scale poorly in the multicast case when there are many receivers. In this case, each delivered packet causes a flood of positive acknowledgments sent from the receivers back to the source, overwhelming either the source or the intervening routers, if not both.

A number of solutions to the ACK implosion problem have been proposed. Log-based reliable multicast [58] uses logging servers to constrain recovery traffic to localized groups of receivers. TMTP [148] and Lorax [73] construct a hierarchy in the form of a tree, in which multiple identical ACKs are fused together before they are propagated up the tree toward the root. RMTP [74] uses a similar approach based on trees that are (statically or dynamically) configured into the network rather than constructed by the application. XTP [21] takes a markedly different approach, however, and instead *multicasts* control traffic to all end-points. To limit the proliferation of this control traffic, XTP employs a "slotting and damping" algorithm: a receiver waits for a random amount of time before generating control traffic and cancels that message if some other hosts multicasts the same information first. This slotting and damping mechanism also forms the basis of our soft state-based transport protocol framework. The algorithms in SRM [40] elaborate this simple yet powerful primitive with adaptive timers that improve performance across wide-area, heterogeneous networks.

While TMTP, Lorax, and RMTP limit recovery traffic using unicast transmission over an artificially constructed hierarchy, XTP and SRM limit recovery traffic using multicast transmission and explicit suppression. Although this latter approach is potentially more robust because it does not require an elaborate protocol for tree construction, maintenance, and reconfiguration, it also entails potentially more overhead because recovery traffic is multicast to the entire group and not just to those members impacted by the packet loss. To address this problem, [40] proposes that SRM be cast as two complementary pieces: a *global recovery* component that ensures the delivery of all desired data across the entire multicast session, and a *local recovery* component that constrains the reach of recovery traffic to the multicast neighborhoods where packet loss occurs. Although [40] focuses primarily on global recovery, the SRM authors argue that local recovery is an important and necessary optimization to scale their protocol to large, heterogeneous sessions. Since then, several promising approaches to local recovery have been proposed [69, 75] and the problem remains a focal point of ongoing research. In SSTP too, we use the slotting and damping algorithm and perform a combination of local and global loss recovery.

Even though a viable local recovery strategy is critical to SSTP's scalability, in certain configurations (e.g., where packet loss occurs near the root of the distribution tree), the degree to which local recovery enhances performance may be limited and the protocol's overall performance may strongly depend on that of the global recovery scheme. Hence, we claim that a thorough understanding of global recovery in SSTP is not only important in and of itself, but will also be useful in predicting the performance of SSTP even when coupled with local recovery.

In this Chapter, we use analysis and simulation to investigate the scaling behavior of global loss recovery in SSTP. We study the growth control traffic (measured by NACK counts) as a function of group size for various topologies and protocol parameters, on a set of simple, representative topologies — the cone, the linear chain, and the binary tree. We find that the number of NACKs, as a function of group size, for the cone is always linear, for the linear chain is between constant and logarithmic, and for the tree is between constant and linear. We show, by studying various cases, that the randomized timer equation used in slotting and damping, $t = (C_1 + C_2 r)d$, allows us sufficient flexibility in selecting the level of latency that can be tolerated in return for a reduced number of redundant control messages.

A number of performance metrics have been used to characterize recovery schemes for reliable multicast, but two widely used metrics are:[1] (1) the degree of duplicate control traffic, and (2) the recovery latency. The first metric can be summarized as the average number of NACKs sent for each dropped packet, which clearly depends on the size of the group experiencing the loss. We denote this number by $N(G)$, where G is the number of members experiencing the loss. The larger this metric, the less effective the randomized timer algorithm is at suppressing duplicate NACKs and avoiding NACK implosion. $N(G)$ is a non-decreasing function of G, so the suppression performance for large group sizes is a critical factor in SSTP's performance.

We define the second metric, the loss-recovery latency, as the time delay between the instant a packet drop is detected to the time at which the first NACK is sent (from the perspective of a particular session member). Recovery latencies for these randomized algorithms typically decrease as group sizes increase, so the sensitivity of latency on group size is not of primary importance in the scaling behavior of SSTP.

In this Chapter, we focus on the performance of SSTP with *large* group sizes; that is, roughly speaking, the asymptotic scaling limit. Thus, we focus on the number of duplicate messages and do not address latency performance. Since the timer mechanisms for NACKs and repair messages are similar, we restrict our attention to NACKs. Therefore, our Chapter addresses the following question: how does the number of duplicate NACK messages increase as the group size grows? In short, what is the scaling behavior of $N(G)$ in SSTP?

The scaling behavior of SSTP depends both on the topology of the underlying network as well as the details of the timer algorithm. To explore the relationship between topology and scaling behavior, we experimented with three simple network topologies: the cone (a variant of a clique), line, and tree, shown in Figures 5.1 and 5.2 While these topologies are instructive because they explore the behavior of SSTP under extreme topologies, they are by no means exhaustive.

The scaling behavior also depends on several aspects of the timer algorithms. We focus on two such factors. First, we look at the dependence of the scaling behavior on the constants C_1 and C_2. There are several applications, such as large-scale multi-player games that are highly interactive, for which *low-latency* loss recovery is important, and the choices of C_1 and C_2 critically impact this. In general, the expected latency to transmit the first NACK upon detecting a loss is bounded above by $(C_1 + C_2/f)D$, where f is a function of the network topology and is always at least 2. Thus, there is a trade-off between recovery latency and the choices of C_1 and C_2. In particular, smaller values of these constants lead to better latency, but also to increased $N(G)$. The need for low latency by many applications motivates our work on investigating the (C_1, C_2) parameter space, and in particular, our consideration of $0 \leq C_1 \leq 1$ (little or no deterministic suppression).

We also briefly consider the case where C_1 and C_2 are a function of the location in the topology; this aspect of our work was inspired by the results on adaptive timers in [76]. There, the timer constants were set in response to the number of duplicates observed and the latency of the responses, and this naturally led to the parameters being different for different members — *e.g.*, members located at different depths in a tree would have different settings. We do not directly address the dynamic nature of these timer adjustments, but merely study how location dependence in C_1 and C_2 changes performance.

We then investigate how the scaling behavior depends on the accuracy of the delay D. In SSTP, the i^{th} group member estimates $D_{ij}, j = 1, 2, \ldots n, j \neq i$, the delay from itself to each of the

[1]The metrics we describe here ignore topological heterogeneity, where not all receivers are identical. More detailed performance metrics would measure the latencies on a per receiver basis.

other members of the group. Delay estimates are calculated from round-trip time (RTT) information which is derived from timestamps in *session messages* of the SSTP protocol. Since the protocol's control bandwidth is limited to a constant fraction of the total available session bandwidth, the estimated RTT does not readily track changes in actual delay for large session sizes[2]. We study how RTT estimation might affect asymptotic scaling behavior in the different topologies by comparing performance in two extreme cases: one with exact RTT estimations and one where all members have the same hardwired RTT estimate.

5.2 Previous Work

In this section, we summarize some important prior work related to the analysis of SSTP. The seminal work of Floyd *et al.* [40] simulated group sizes of up to a few hundred nodes ranging across a set of simple topologies. They showed that it was often possible to choose values of C_1 and C_2 that resulted in $N(G)$ scaling as a constant independent of G. In particular, picking $C_1 = C_2 = 2$ achieved this for the chain topology, and picking $C_2 = \sqrt{G}$ resulted in constant scaling for the star topology (a special case of the cone topology in our work). Using simulations they demonstrated that $N(G) \leq 4$ for random trees with bounded degree for session sizes of up to 100. They also proposed an adaptive algorithm to dynamically adjust C_1 and C_2 based on past information for better performance.

Our work extends their important findings in two ways. First, we investigate performance for session sizes of up to two orders of magnitude larger than in [40], thus improving our collective understanding of SSTP's asymptotic behavior. Reassuringly, our results agree with [40] where the experiments overlap. More generally, we have assessed in detail the behavior of $N(G)$ as a function of C_1 and C_2. Not only do these results help us predict the performance of SSTP, but they could influence the design of related sub-components of SSTP, e.g., the choice of bounding values of C_1 and C_2 in the proposed adaptive algorithm. A more recent paper [76] studied scaling behavior for group sizes up to 200 members, with $C_1 = 0$ and C_2 set adaptively.

In addition, Nonnenmacher and Biersack [93] looked at the effect of timer distribution on scaling behavior and showed that exponentially distributed timers yield better scaling properties. They found that having this distribution depend on the group size could result in improved scaling. We do not address the effects of different timer distributions at any great length in this Chapter.

This Chapter is primarily concerned with global recovery in SSTP with constant C_1 and C_2. Variants of SSTP have been proposed that use local recovery, in which NACKs and repairs are not sent to the entire group. [40], [75] look at two methods to limit the range of these methods: hop-scoping, and local recovery groups. [76] considers methods for adaptively setting the values for C_1 and C_2. We do not consider any of the local recovery methods, nor adaptive timer setting. Thus, our work should not be seen as a statement about how SSTP-like protocols should function in the future, when they may well incorporate such features, but rather as an attempt to study the current deployed version of SSTP with its use of global recovery. Our hope is that understanding this basic version of the protocol may inform future design efforts to improve it.

[2]Even in the case of a single TCP connection, where RTT estimates are gathered on every ACK, the sender's RTT-estimator is known to often be inaccurate [128].

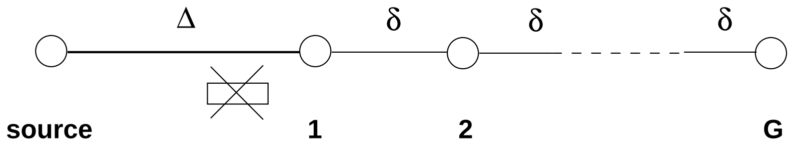

Figure 5.1: "Linear chain" topology used in our evaluation of randomized timers. The X-ed packet marks the location of packet loss.

5.3 Simulation Methodology

In our simulations, we studied three classes of network topologies: *cone*, *linear chain*, and *binary tree*, each with a single source. The cone is a topology where each member has the same delay δ to every other member, and a distance Δ from the source. Similarly, for the linear chain and the binary tree, δ represents the link delay between adjacent members, and Δ is the link delay from the source to the closest member(s). Figures 5.1 and 5.2 show Δ and δ for the three topologies.

We are only modeling the behavior of NACKs, so we need only consider the receivers that suffer losses. Thus, we only consider the case where the loss occurs on the link adjacent to the source [3]. This causes little loss of generality, since if the loss occurs elsewhere we need only model the topology beneath the loss point. Note, however, that the size of the group we are considering, G, is the size of the *loss group* – the number of members experiencing a particular packet loss – and not always the size of the entire group. Session messages in SSTP give members knowledge about the size of the entire group, but not about the size of the loss group. If members knew the size of the loss group they might also be able to employ various forms of local recovery (hop-scoped recovery, or local recovery groups) that would more directly address the NACK traffic problem (not just limiting the number of NACKs, but also the portion of the group they are sent to). Thus, we do not consider varying the timer constants with group size, as in [93], as this does not seem like a realistic possibility.

Furthermore, we assume that losses are detected immediately when the next packet arrives. Since a packet is delivered to different receivers at different absolute times, losses are detected at different times. This typically allows the receivers closer to the source to suppress the NACKs from receivers further away. One of the key points in our investigation is how the setting of the timer constants affects this behavior.

We used the VINT network simulator *ns* [83] for our work. In its original form, ns turned out to have prolific memory usage with heavy-weight nodes, links, and multicast routing infrastructure, and could not support more than a few hundred nodes on an ordinary workstation. However, we took advantage of ns's extensible object-oriented architecture and made several modifications and extensions to it. Using the basic ns framework for event handling, we extended the simulator to support regular topologies with static routing without explicit routing table state. These modifications and extensions to ns enabled large-scale simulations of up to 50,000 nodes.

Losses occur on the link closest to the source, and are thus shared by all receivers in the group. We measure the average number of NACKs generated in response to a loss. The variation between different measurements is induced by the randomness in the recovery algorithm we are

[3] Measurements reported in [147] show that most correlated losses occur close to the source.

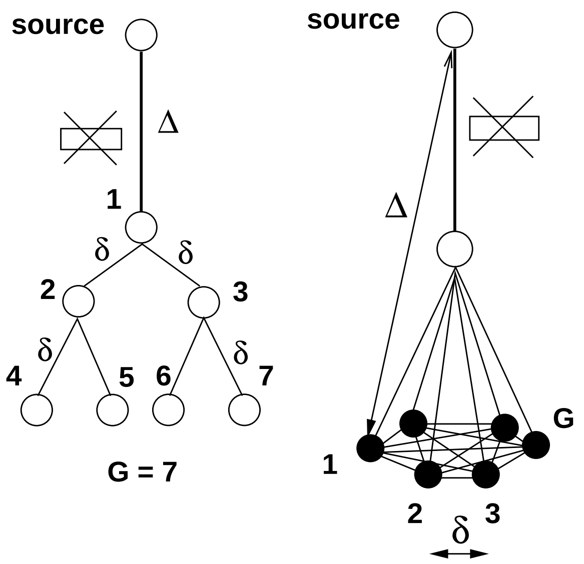

Figure 5.2: Binary tree and cone topologies used in our randomized timer analysis: the X-ed packet marks the location of packet loss.

studying. We ran between 30 and 50 simulations of each case to compute the average value of the metrics, depending on the variance of the measured samples. Table 5.1 summarizes notation used in the rest of this Chapter.

In the following sections, we present our analytical and simulation results for the three topologies under study.

5.4 Scaling in the Cone Topology

The cone topology can be used to model the case of a broadcast LAN. If the source is on the LAN then $\Delta = \delta$ but when the source is off the LAN, the delay from the LAN to the source is much greater than the LAN propagation time, yielding $\Delta >> \delta$. In general, the cone topology can be used to model cases where all receivers have similar round-trip time estimates to the source. In practice, accurate RTT estimation is a hard problem and RTT estimators tend to be coarse-grained, resulting in in broad classes of receivers, each with multiple receivers having similar RTT values.

We use the following probabilistic analysis to compute the expectation of the number of duplicate feedback messages, $N(G)$. Because all the receivers are at the same distance from the loss in a cone, the deterministic backoff component has no impact on the number of duplicates (all timers have the same constant offset). The average delay in transmitting the first NACK depends on the expected value of the minimum timer and is given by $\Delta(C_1 + \frac{C_2}{G+1})$. This result follows directly from noting that the expectation of the minimum of G uniformly distributed random variables in $[0, 1]$ is $\frac{1}{G+1}$. The number of duplicates is equal to the expected number of timers that fire within $[t_{min}, t_{min} + \delta]$, where t_{min} is the value of the smallest timer. Since backoffs are uniformly distributed in the interval $[C_1\Delta, (C_1 + C_2)\Delta]$, we can easily compute this expectation. Defining

Symbol	Description
Δ	Delay from source to the closest receiver
δ	Delay of link connecting receivers
R	Δ/δ
G	Group size
N	Average number of copies of a single NACK
L	Average NACK latency caused by backoff
D_i	Estimate of one-way delay from node i to the source node
backoff$_i$, at host i	$D_i \times (C_1 + C_2 \times r_i)$ where, r_i are uniformly distributed random variables in $[0, 1]$
t_i	Absolute time at which receiver i's timer *fires*

Table 5.1: Summary of notation used in our randomized timer analysis.

$\alpha = \frac{\delta}{C_2 \Delta}$ we have:

$$E[N] = \begin{cases} 1 + G\alpha - \alpha^G, & \alpha < 1 \\ G & \alpha \geq 1 \end{cases} \tag{5.1}$$

Thus from Equation 5.1, the number of duplicates is roughly linear in the group size. [40] reports a similar result for the *star* topology, which is a *cone* with $\Delta = \delta$. Observe that this linear dependence applies regardless of whether the delay estimates are accurate or not. If the estimated value of the delay (assuming all members achieve the same estimate) is larger than the true estimate, then the number of duplicates is smaller, but the dependence on G is still linear. Our simulations, shown in Figure 5.3, confirm this result.

As we have just seen, $N(G)$ grows roughly linearly for any fixed timer distribution. However, as shown by Nonnenmacher and Biersack [93], if one makes the distribution dependent on the size of the loss group then one can change this linear scaling. For instance, if one takes a bimodal distribution such that with a probability $p = \frac{a}{G}$ a receiver sends a NACK immediately upon detecting a loss, and with probability $1 - p$ sends a NACK after a delay δ, then as G diverges $N(G)$ is given by $a(1 - e^{-a}) + Ge^{-a}$. By tuning a one can lower the slope of the linear dependence, and if one sets $a = \ln G$ the growth is logarithmic, not linear. One can remove the linear term entirely by considering the scheme where each receiver picks a number k from an exponential distribution with average $\frac{a}{G}$ and sets the backoff to $k\delta$. This is essentially a discrete version of the exponential distribution considered by Nonnenmacher and Biersack [93]. Here, the average number of NACKs is $E(N) = a$ and the average latency is $E(L) = \frac{e^a \delta}{1 - e^{-a}}$. One can show that this achieves the lowest latency for a given number of NACKs (or equivalently, the smallest number of NACKs for a given latency) in the asymptotic limit. However, as we argued earlier, schemes that have the timer distribution depending on G are perhaps of little interest since the parameter G must be the size of the loss group, and once one has this information it might be better used in some local recovery approach rather than using it merely to tune the timer parameters.

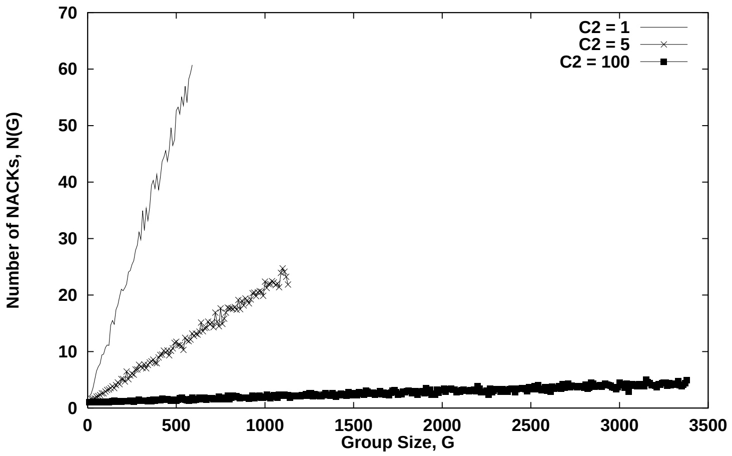

Figure 5.3: In the cone topology, the number of duplicates N(G) grows linearly in G. The slope of the line in each case is $\alpha = \frac{\delta}{C_2 \Delta}$. This result holds for the range of timers given by $C_1 \geq 0, C_2 > 0$.

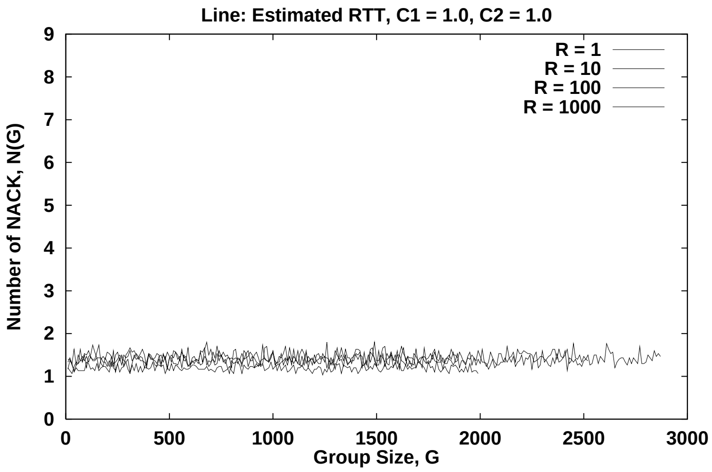

Figure 5.4: In the linear chain topology, N(G) is a constant for $\Delta/\delta = 1, 10, 100, 1000$, with exact RTT estimation and $C_1 = C_2 = 1$. Similar results hold for other C_1 and C_2 as long as $C_1 > 0$.

5.5 Scaling in the Linear Chain

For the linear chain topology, we first consider the case where the RTT estimation maintained by the receivers is exact. We study the behavior in several timer ranges, determined by the parameters C_1 and C_2.

5.5.1 Exact RTT Estimation

- $C_1 > 0$ **and** $C_2 \geq 0$

When $C_1 > 0$ and $C_2 \geq 0$, the data in Figure 5.4 suggests that $N(G)$ is constant in G. In the discussion that follows, we present an important result for the linear chain topology. We show that in this specific range for the timer parameters $C_1 > 0$ and $C_2 \geq 0$, there is a integer bound k on the maximal number of NACKs sent when a loss is detected by all receivers.

The receiver located at position i in the regular linear chain is at a distance D_i from the source, given by the expression in picks a backoff that satisfies the condition in Expression 5.3. This receiver picks a backoff that satisfies the condition in Expression 5.4 below.

$$D_i = \Delta + (i-1)\delta \tag{5.2}$$
$$C_1 D_i \leq backoff_i \leq C_1 D_i + C_2 D_i \tag{5.3}$$
$$\Rightarrow C_1(\Delta + (i-1)\delta) \leq backoff_i \leq (C_1 + C_2)(\Delta + (i-1)\delta) \tag{5.4}$$

Now, consider some message sent at time $t = 0$, and assume that losses are detected immediately. This assumption approximates our analysis, since losses are detected when the subsequent packet in the sender's transmission sequence arrives at the receiver. However, we find that our approximate analysis corroborates the simulation results and hence the assumption does not impact the qualitative nature of the result. Receiver i detects the loss at time $(\Delta + (i-1)\delta)$ and sends its NACK, if not suppressed, no later than a time given by $(\Delta + (i-1)\delta) + (C_1 + C_2)(\Delta + (i-1)\delta)$ and no sooner than a time given by $(\Delta + (i-1)\delta) + C_1(\Delta + (i-1)\delta)$. Under what condition will one of i or j be suppressed? We calculate this by computing the overlap condition that must be satisfied, taking into account each receiver's backoff time as well the propagation time between them. Assuming $j > i$, receiver i and receiver j cannot both send NACKs if i suppresses j. This will happen is the following condition is true:

$$(C_1 + C_2)(\Delta + (i-1)\delta) + (j-i)\delta < (j-i)\delta + \delta + C_1(\Delta + (j-1)\delta) \tag{5.5}$$

Equation 5.5 follows by recalling that it takes time $(j-i)\delta$ for i's NACK to propagate from i to j. Thus, the first member on the line suppresses all but the next k members, where k is given by $k = \lfloor \frac{C_2 \Delta}{C_1 \delta} \rfloor$. Thus, $N(G)$ is bounded from above by the integer $k+1$. Our simulation results suggest that the *average number* $N(G)$ is much less than this upper bound, and in particular, is insensitive to R.

For $C_1 > 0$, the value of $N(G)$ appears, as shown in Figure 5.5, to be roughly independent of C_2. The dependence on C_1 is also shown in Figure 5.6, where, for a fixed G, N decreases with increasing C_1 as expected.

- $C_1 = 0$
When $C_1 = 0$, there is no deterministic delay and the preceding argument fails. In fact,

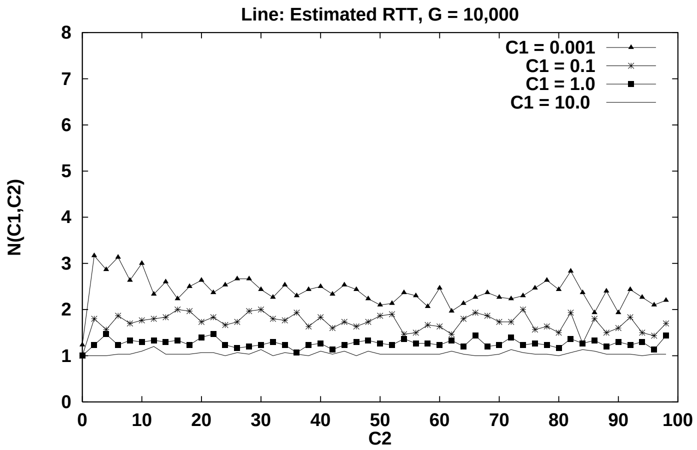

Figure 5.5: The effect of constants C_1 and C_2 on scaling in the linear chain topology. N as a function of C_1 and C_2.

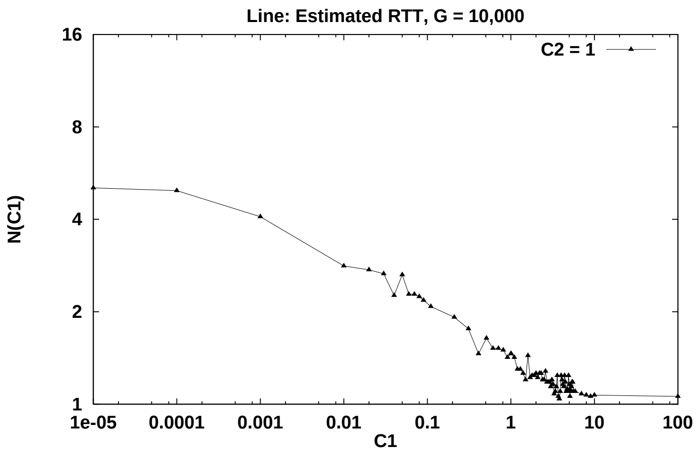

Figure 5.6: The effect of the deterministic constant C_1 on scaling in the linear chain topology. N as a function of C_1.

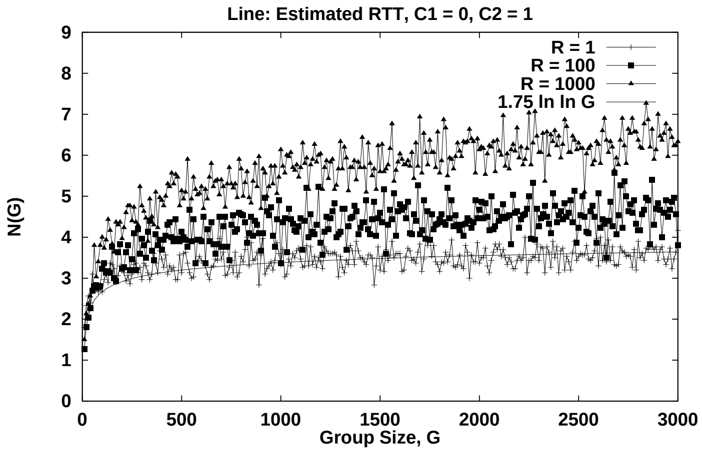

Figure 5.7: N(G) diverges as $\ln \ln G$ for $\Delta/\delta = 1, 10, 100, 1000$, with RTT estimation, $C_1 = 0$, $C_2 = 1$.

it appears that $N(G)$ diverges slowly with the group size G, as shown in Figure 5.7. We can argue that $N(G)$ does not grow faster than a certain expression derived below (but are not able to provide a lower bound). The probability that node i is not suppressed is upper-bounded by the probability that it is not suppressed by the members ahead of it in line, i.e., nodes $1, 2, \ldots, i-1$. This occurs if and only if (ignoring ties) the backoff timer $t_i = min\{t_1, \ldots t_{i-1}\}$. Considering the special case of $\Delta = \delta$ for convenience and using the notation $z_+ = \max[0, z]$, we have

$$Pr[t_i = min\{t_1, \ldots t_{i-1}\} | t_i = x] = \prod_{j=1}^{j=i-1} Pr[t_j \geq x] \quad (5.6)$$

$$= \prod_{j=1}^{j=i-1} (1 - x/j\delta)_+ \quad (5.7)$$

Transforming equation 5.7 by changing variables,

$$N(G) \leq \sum_{i=1}^{G} \int_0^1 \prod_{j=1}^{j=i-1} (1 - \frac{y}{j}) \frac{dy}{i} \quad (5.8)$$

Approximating $\prod_{j=1}^{j=i-1}(1 - \frac{y}{j})$ as $e^{-y \sum_{j=1}^{i-1} \frac{1}{j}}$ and then noting that $e^{-y \sum_{j=1}^{i-1} \frac{1}{j}} \approx e^{-y \ln i}$ and substituting into the integral, we see that this expression diverges as $\ln \ln G$. The results of our simulations for this case are shown in Figure 5.7.

5.5.2 Without RTT Estimation

We now consider the case where there is no adaptive RTT estimation, and all receivers use the same hardwired delay estimate D.

- $C_2 = 0$
  Note that since deterministic delay is useless when round-trip times are not used (all members have the same deterministic delay), $C_2 = 0$ results in no suppression at all, and $N(G) = G$. This is true independent of topology; if there is no RTT estimation, then one needs $C_2 > 0$ or else $N(G) = G$, and $N(G)$ is independent of C_1.

- $C_1 = 0$ **and** $C_2 = 1$
  Figure 5.8 shows $N(G)$ for the case $C_1 = 0$ and $C_2 = 1$ and fixed RTT. The growth, for all values of $R = \frac{\Delta}{\delta}$ appears to be logarithmic. Similar logarithmic-like behavior is observed in simulations with different values for C_2 and D.

  The following probabilistic analysis suggests why, for $C_1 = 0$ and $C_2 = 1$, $N(G)$ grows as a logarithmic function of the group size. Given the values for the timer parameters, the backoffs at the receiver are picked in the range $[0, D]$. We first compute the probability that the NACK at node i is not suppressed. The following condition (5.9) must hold, for i's timer to fire:

$$D_j + r_j\delta + D_{ji} \quad \geq \quad D_i + r_i\delta,\ \forall j \neq i \tag{5.9}$$

  where D_j is the one-way delay to receiver j from the source and D_{ij} is the one-way delay from receiver i to receiver j. r_i, r_j are uniformly distributed random numbers picked in $[0, 1]$ by the random timer mechanism. We must then have the conditions in (5.10 through 5.13) below.

$$\begin{aligned} r_i &< r_j,\ \forall j < i & (5.10)\\ r_j\delta + 2d_{ij} &> r_i\delta,\ \forall j > i,\ and & (5.11)\\ d_{ij} &\geq \delta, \forall d_{ij} & (5.12)\\ \Rightarrow r_i\delta &< 2\delta + r_j\delta,\ \forall j > i & (5.13) \end{aligned}$$

  From equations 5.10 through 5.13 above, we can conclude that a NACK at node i cannot be suppressed by a NACK at a later node. The condition for suppression at node i is therefore $r_i \leq min\{r_1, r_2, r_3, \ldots, r_{i-1}\}$. Since the probability that receiver i fires is $P[i\ fires] = \frac{1}{i}$, we get the number of redundant messages to be logarithmic in G, given by the expression in (5.14).

$$E[N] = \sum_{i=1}^{i=G} P[i\ fires] \approx \ln G + 0.577 \tag{5.14}$$

  Similar logarithmic growth is seen empirically for larger C_2 also.

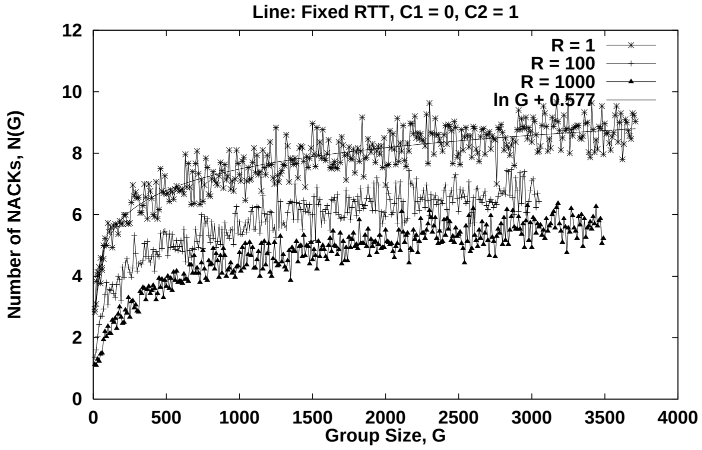

Figure 5.8: N(G) grows as a logarithmic function of G for $\Delta/\delta = 1, 10, 100, 1000$, fixed delay (no RTT estimation), $C_1 = 0, C_2 > 0$. $N(G) = \ln G + 0.577$, when $C_1 = 0, C_2 = 1$.

- **Super-linear dependence on D_i**
  With $C_1 = 0$, $N(G)$ *diverges* as $\ln \ln G$ for the linear chain topology. In order to reduce this growth in N(G) to a constant, while still retaining $C_1 = 0$ for the sake of low repair latency, we can make C_2 a function of the delay from the source. This follows the work Liu *et al.* who propose, in [76], using a new adaptive timer algorithm. Analysis similar to the previous case (equations (5.10 – 5.13)) shows that the number of duplicates is bounded by a constant when we use $C_2 = D^\epsilon$ for any $\epsilon > 0$. This follows from 5.15 through 5.16 below.

$$N(G) \leq \sum_{i=1}^{G} \int_0^1 \prod_{j=1}^{j=i-1} (1 - \frac{y}{j^{1+\epsilon}}) \frac{dy}{i^{1+\epsilon}} \tag{5.15}$$

$$\leq \sum_{i=1}^{G} \frac{1}{i^{1+\epsilon}} \leq \frac{1}{\epsilon} \tag{5.16}$$

The graph in Figure 5.9 shows through simulations that $N(G)$ converges to a constant for $\epsilon = 0.5$. We should note that because we do not have a lower bound for the case of C_2 fixed ($\epsilon = 0$). Our simulation results show that $N(G)$ diverges for $\epsilon = 0$, but our analytical proof is only for $\epsilon > 0$.

The behavior of $N(G)$ for the line case is summarized in Table 5.5.2.

5.6 Scaling in the Binary Tree

In the binary tree topology (Figure 5.2), N(G) grows linearly with G when RTT is not estimated, as shown in Figures 5.10 and 5.11. The slope of this linear growth depends on C_2 and

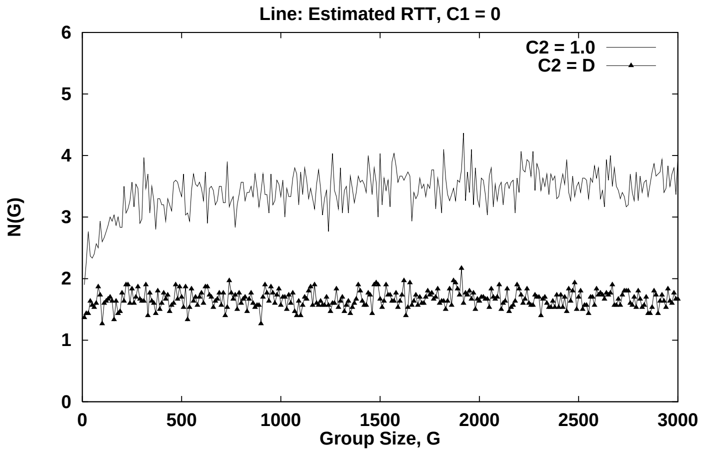

Figure 5.9: $N(G)$ converges to a constant when $C_2 = \sqrt{D}$ for the linear chain.

Δ/δ	RTT	C_1	C_2	N(G)	Figure
$1, 10, 100, 1000$	Fixed	$C_1 \geq 0$	$C_2 > 0$	Logarithmic ($\ln G + \gamma$, when $C_1 = 0, C_2 = 1$)	5.8
$1, 10, 100, 1000$	Fixed	$C_1 > 0$	$C_2 = 0$	Linear ($N(G) = G$)	
$1, 10, 100, 1000$	Estimated	$C_1 > 0$	$C_2 \geq 0$	Constant (≤ 4)	5.4
$1, 10, 100, 1000$	Estimated	$C_1 = 0$	$C_2 > 0$	Diverges	5.11

Table 5.2: Summary of asymptotic scaling in the linear chain topology

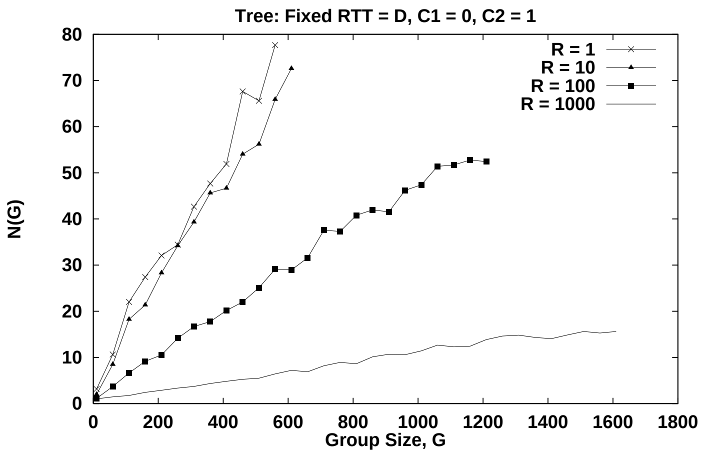

Figure 5.10: With $C_1 = 0$, $C_2 > 0$ and without RTT estimation, N(G) scales linearly with G for different values of $R = \Delta/\delta$.

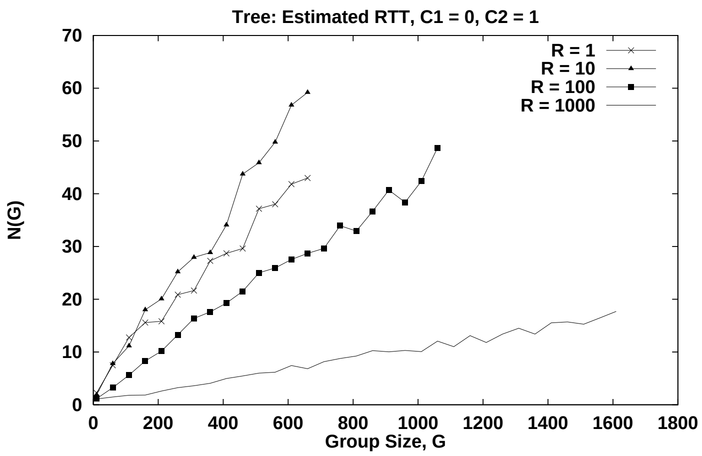

Figure 5.11: With $C_1 = 0$, $C_2 > 0$ and with accurate RTT estimation, N(G) scales linearly with G for different values of $R = \Delta/\delta$.

D (the fixed RTT). This linear behavior is in contrast with the logarithmic behavior observed in the line topology, but similar to the behavior in the cone topology. When RTT is known exactly, we still have linear behavior for $C_1 = 0$, as shown in Figure 5.11. The slope of this linear growth depends on both $\frac{\Delta}{\delta}$ and C_2.

However, as soon as we have $C_1 > 0$, $D(G)$ appears to asymptotically reach a constant. Figure 5.12 shows the function $N(G)$ for different values of $0 \leq C_1 \leq 1$. The growth law for intermediate G is linear, and then the slope decreases as G increases. For all cases where we have been able to reach sufficiently large G, the slope continues to decrease until $N(G)$ goes to a constant.

When $C_1 > 0$, we see that the asymptotic scaling behavior depends on whether deter-

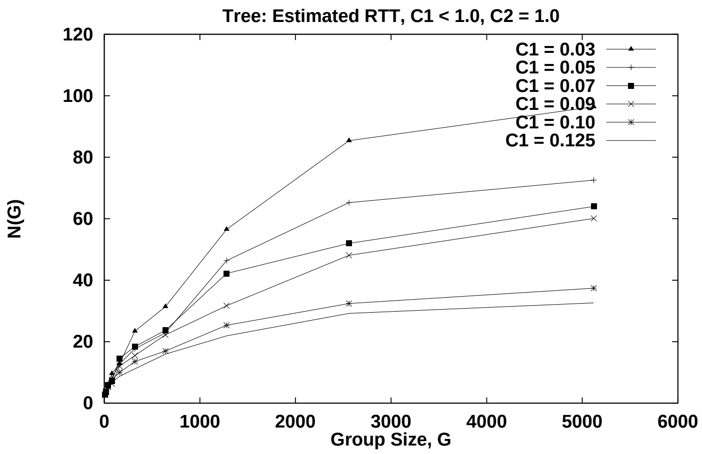

Figure 5.12: $N(G)$ in the binary tree for $R = \Delta/\delta = 1$, accurately estimated RTT and $0 < C_1 \leq 1$, $C_2 = 1$.

ministic suppression or randomized suppression is dominant in reducing the number of NACKs. In cases where deterministic suppression is dominant, the asymptotic scaling is constant. Scaling is linear when suppression depends on the randomized suppression. In Figure 5.15, these two important effects are evident: as Δ/δ increases, deterministic suppression becomes weaker and randomized suppression is more effective. For large values of $\Delta/\delta > 100$, backoff timer ranges are large enough and the average separation between timers grows.

We now try to illustrate this behavior in a different form. The function $\frac{G}{N}$ plotted against G is shown in Figure 5.13. This ratio appears to be a linear functions of G, with the slope depending on C_1. If we label the slope of this line by m and the intercept by f, we have, for small C_1 and large G, the following form for N:

$$N \quad = \quad \frac{G}{mG + f}$$

The fit parameters m and f are functions of C_1 and C_2. This linear fit applies over a wide range of C_1, C_2 values. This functional form for $N(G)$ is consistent with our observation of a linear increase for small values of G, followed by this slope decreasing and the curve flattening to a constant. In particular, note that $\lim_{G\to\infty} N \to \frac{1}{m}$, a constant for a given value of C_1 and C_2. Thus, the slope of this functional fit in Figure 5.13 yields the asymptotic value for $N(G)$. Figure 5.14 shows this dependence on a log scale. $\frac{1}{m}$ decreases with increasing C_1 as expected.

If we hold G fixed and vary R (the ratio of Δ to δ) we find that the dependence is not monotonic. Figure 5.16 shows this unimodal behavior. This behavior may be explained by the following reasoning. There are two kinds of suppression, deterministic and random, so-called depending on whether the possible firing times overlap or not. Deterministic suppression decreases with R, but random suppression increases with R. Thus, as R is increased we first see an increase as the deterministic suppression becomes less effective, and then see a decrease as random suppression becomes dominant and deterministic suppression is no longer much of a factor (and so cannot decrease significantly further).

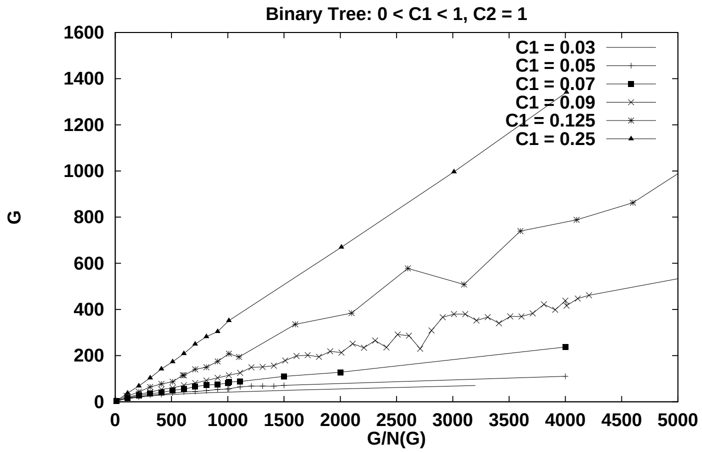

Figure 5.13: G/N vs. G in the binary tree for $R = \Delta/\delta = 1$, RTT estimated, $0 < C_1 \leq 1$, $C_2 = 1$.

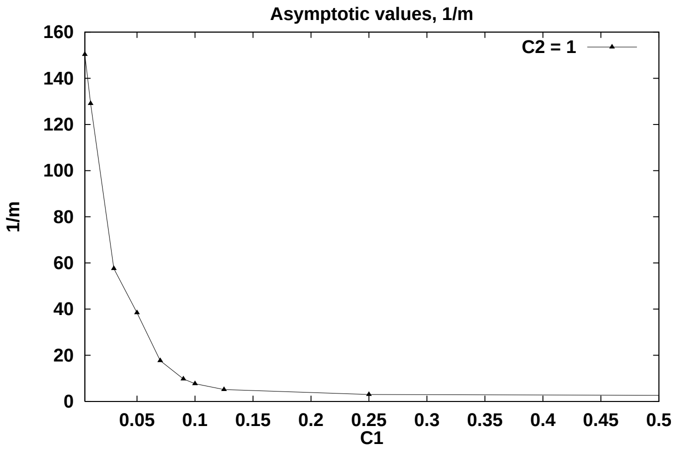

Figure 5.14: $C_2 = 1$, C_1 is varied.

Δ/δ	RTT	C_1	C_2	N(G)	Figure
$1, 10, 100, 1000$	Fixed	$C_1 \geq 0$	$C_2 > 0$	Linear	5.11
$1, 10, 100, 1000$	Fixed	$C_1 > 0$	$C_2 = 0$	$N(G) = G$	
$1, 10, 100, 1000$	Estimated	$C_1 = 0$	$C_2 > 0$	Linear	5.11
$1, 10$	Estimated	$0 \leq C_1 \leq 1$	$C_2 \geq 0$	$G/(mG + f)$ $\lim_{G\to\infty} G/(mG + f) =$ constant	5.12
$100, 1000$	Estimated	$0 \leq C_1 \leq 1$	$C_2 \geq 0$	Linear	5.15

Table 5.3: Summary of asymptotic scaling in the tree topology

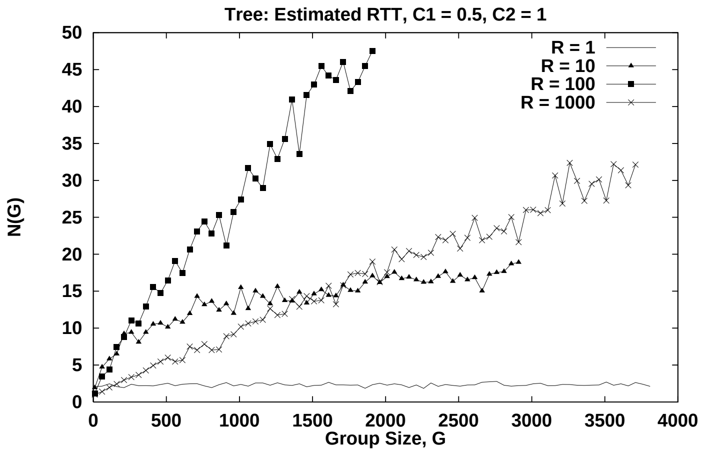

Figure 5.15: $N(G)$ in the binary tree with $\Delta/\delta = 1, 10, 100, 1000$, RTT estimated, $C_1 = 0.5$, $C_2 = 1$.

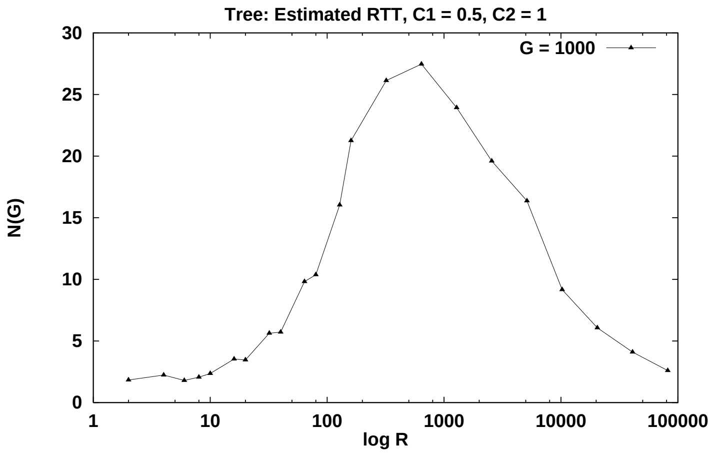

Figure 5.16: For small values of R, the round-trip times from the source to the receivers are distinguishable, and deterministic suppression effectively keeps the NACK count low. When Δ/δ increase, randomized suppression is the dominant cause for suppression. The "turning point" value of Δ/δ depends on the topology.

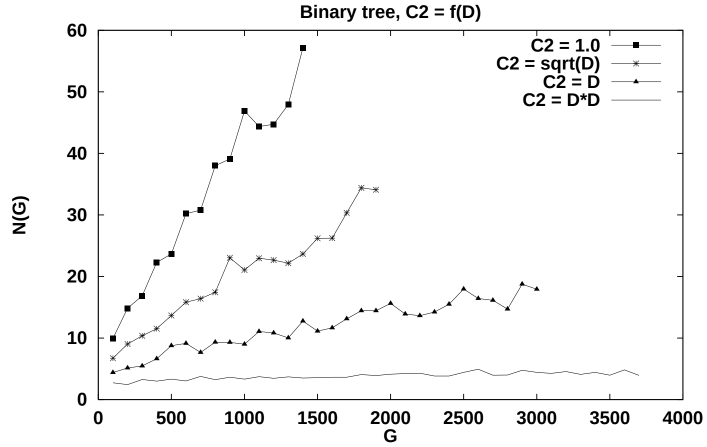

Figure 5.17: When $C_2 = D^{0.5}$ in the binary tree, N(G) has improved scaling.

With $C_1 = 0$, and $C_2 > 0$, $N(G)$ grows linearly with G. In order to reduce this growth in $N(G)$ to a constant, while still retaining $C_1 = 0$, as we did for the linear chain topology, we make C_2 a function of the delay from the source. The adaptation algorithm described in [76] results in C_2 values that increase roughly linearly in D, the distance of a receiver from the source.

Here, we do not model the dynamics of the adaptation, but instead merely insert the dependence on D directly. We consider several variants, with C_2 increasing as D, D^2, and $\sqrt{D}$. Figure 5.17 shows the results of these simulations. We find that C_2 needs to be "super-linear" in D to make scaling constant.

5.7 Concluding Remarks

In this Chapter, we used analysis and simulation to study the scaling behavior of global loss recovery in SSTP. The SSTP loss recovery protocol is NACK-based and uses a randomized, timer-based decentralized algorithm to reduce NACK implosion. We use the number of NACKs $N(G)$ generated in response to a loss, as a metric for scalability. The two protocol parameters, C_1 and C_2, govern the deterministic and random delays in the firing of a NACK from a receiver. There is a trade-off between low-latency loss recovery and the number of NACKs – in general, making these parameters small leads to lower latency, but usually at the expense of poorer asymptotic scaling. We study $N(G)$ as a function of group size, G, for various protocol parameters, on a set of simple, representative topologies — the cone, the linear chain, and the binary tree.

In the cone topology, we find that random backoff is the dominant reason for suppression and scaling is linear. This linear scaling can be reduced by using a distribution that is dependent on the group size. The cone models topologies in which receivers have similar round-trip time estimates to the source. For the linear chain $N(G)$ is between constant (when $C_1 > 0, C_2 > 0$, and RTT estimation is perfect), and logarithmic, when RTT is not estimated. In the tree, scaling is between constant (when $C_1 > 0, C_2 > 0$, and RTT estimation is perfect), and linear, when RTT is not estimated. For the linear chain we show that $C_2 = D^{\epsilon}$ results in constant scaling even when

$C_1 = 0$, where D is the one-way delay to the source. Similarly, for the binary tree, $C_2 = D^2$ results in constant scaling.

We find that in topologies where deterministic suppression is effective in reducing the number of duplicate NACKs, asymptotic scaling tends to a constant. For topologies in which randomized suppression is mainly responsible for eliminating duplicates, asymptotic scaling is not constant, e.g., in the cone topology and in the binary tree with $\Delta >> \delta$, $N(G)$ grows linearly. We have shown, by studying the different, that the randomized timer equation used in slotting and damping, provides us sufficient flexibility in selecting the level of latency that can be tolerated in return for a reduced number of redundant control messages.

In conclusion, we have shown that there is a rich parameter space in the SSTP protocol and that the best asymptotic scaling performance is sensitive to the choice of these parameters. We expect our results to be useful in obtaining a better understanding of the reasons for SSTP's scaling properties in different situations, and in aiding the design and analysis of future modifications to SSTP and similar protocols that use multicast transmission and suppression. Our results appear in [115].

Chapter 6

ITP: An Image Transport Protocol

response time *n. An unbounded random variable T_r associated with a given TIMESHARING system and representing the putative time which elapses between T_s, the time of sending a message, and T_e, the time when the resulting error diagnostic is received.*

— S. Kelly-Bootle
The Devil's DP Dictionary

In this Chapter, we turn our attention to semantics of data delivery at the receiver — in-order vs. out-of-order delivery and the impact it has on application performance. Because of its dominance among today's applications, we study JPEG image downloads using HTTP. Even though we focus on the unicast case here, other multicast-based applications such as the MediaBoard [134] also benefit from out-of-order data delivery. In general, understanding delivery abstractions is critical to designing an effective multicast transport protocols because TCP is not a feasible alternative for transporting multicast data.

The remainder of this Chapter is organized as follows. Section 6.1 provides motivation for choosing the specific case of images and the problems with using TCP as an image transport protocol. In Section 6.2, we discuss our design goals for ITP and present empirical evidence in favor of our approach and list the design goals for ITP. Section 6.3 describes various aspects of the ITP protocol — out-of-order delivery, receiver-reliability, and congestion management. This is followed by a discussion on applying ITP to JPEG transport in Section 6.4. In Section 6.5, we present a performance evaluation that demonstrates the advantages of ITP over the traditional TCP approach under a variety of conditions. Finally, we conclude this Chapter in Section 6.6.

6.1 Motivation

Images constitute a significant fraction of traffic on the World Wide Web, e.g., according to a recent study, JPEGs accounted for 31% of bytes transferred and 16% of documents downloaded in a client trace [46]. The ability to transfer and render images on screen in a timely fashion is an important consideration for content providers and server operators because users surfing the Web care about interactive latency. At the same time, download latency must be minimized without compromising end-to-end congestion control, since congestion control is vital to maintaining the long-term stability of the Internet infrastructure. In addition, appropriate reaction to network congestion also allows image applications to adapt well to available network conditions.

The HyperText Transport Protocol (HTTP) [37] uses the Transmission Control Protocol (TCP) [109] to transmit images on the Web. While the use of TCP achieves both reliable data delivery and good congestion control, these come at a cost—interactive latency is often significantly large and leads to images being rendered in "fits and starts" rather than in a smooth way. The culprit is TCP, which is ill-suited to transporting latency-sensitive images over loss-prone networks where losses occur because of congestion or packet corruption. When one or more segments in a window of transmitted data are lost in TCP, later segments often arrive out-of-order at the receiver. In general, these segments correspond to portions of an image that may be handled by the application upon arrival, but the in-order delivery abstraction imposed by TCP holds up the delivery of these out-of-order segments to the *application* until the earlier lost segments are recovered. As a result, the image decoder at the receiver cannot process information even though it is available at the lower transport layer. The image is therefore rendered in bursts interspersed with long delays rather than smoothly.

The TCP-like in-order delivery abstraction is appropriate for image encodings in which incoming data at the receiver can only be handled in the order it was transmitted by the sender. Some compression formats are indeed constrained in this manner, e.g., the Graphical Interchange Format, GIF [44] which uses lossless LZW compression [72, 142] on the entire image. However, while some compression formats require fully reliable and in-order delivery, several others do not. Notable examples of formats that encourage out-of-order receiver processing include JPEG [141, 104] and the emerging JPEG2000 standard [67]. In these cases, a transport protocol that facilitates out-of-order data delivery allows the application to process and render portions of an image as they arrive, improving the interactivity and perceived responsiveness of image downloads. An application-aware transport protocol enables the image decoder at the receiver to implement effective error concealment algorithms on partially received portions of an image, further improving perceived quality. In fact, just as there are benefits to tailoring the network protocol to suit image formats, image compression formats too *should* be tailored for the underlying communication medium. This is the basis of joint source/channel coding schemes. It is for such compression formats that we design an application-aware transport protocol.

One commonly suggested approach to tackling this problem of in-order delivery is to extend existing TCP implementations and its application programming interface so that received data can be consumed out-of-order by the application. However, merely tweaking an in-order byte-stream protocol like TCP without any additional machinery to achieve the desired effect is not adequate because out of order TCP segments received by the application in this manner do not correspond in any meaningful way to processible data units at the application level.

We propose the Image Transport Protocol (ITP), a transport protocol in which application data unit (ADU) boundaries are exposed to the transport module, making it possible to perform out-of-order delivery. Because the transport is aware of application framing boundaries, our approach expands on the application-level framing (ALF) philosophy, which proposes a one-to-one mapping from an ADU to a network packet or protocol data unit (PDU) [23].

In contrast to [23], ITP deviates from the TCP-like notion of reliable delivery and instead incorporates selective reliability, where the receiver is in control of deciding what is transmitted from the sender at any instant. This form of reliability is appropriate for heterogeneous network environments that will include a wide variety of clients with a large diversity in processing power, and allows the client, depending on its computational power and available suite of image decoding algorithms, to request application data that would benefit it the most. Furthermore, other image standards such as JPEG2000 support region-of-interest (ROI) coding that allows receivers to select portions of an image to be coded and rendered with higher fidelity. Receiver-driven selective reliability is an important for applications to benefit from this feature.

Despite the disadvantages of in-order delivery as far as interactivity is concerned, using TCP has significant advantages from the viewpoint of congestion control. Any deployable transport protocol must perform congestion control for the Internet to remain stable, which suggests that a significant amount of additional complexity would have to be designed and implemented in ITP. Fortunately, we are able to use the recently proposed Congestion Manager (CM) architecture [6, 7] to perform stable, end-to-end congestion control, and invoke its API to schedule data transmissions.

In this Chapter, we describe the motivation, design, implementation, and evaluation of ITP, an ALF-based image transport protocol. Our key contributions are as follows.

- We present the design of ITP, a transport protocol that runs over UDP, incorporating out-of-order data delivery and receiver-controlled selective reliability. ITP can be used by any application-level protocol, such as HTTP [10, 37] or FTP [110].

- We show how to tailor ITP for JPEG image transport, by introducing a framing strategy and tailoring the reliability protocol by scheduling request retransmissions.

- We describe a receiver optimization enabled by ITP to interpolate a missing portions of an image using a simple error concealment algorithm.

- We present the results of performance experiments across a range of network conditions conducted using a user-level implementation of ITP. They demonstrate that the rate of increase in PSNR with time is significantly higher for ITP compared to TCP-like delivery of JPEG images.

6.2 Design Considerations

In this section, we discuss the key considerations that directed the design of ITP.

1. *Support out-of-order delivery of ADUs to the application, while efficiently accommodating ADUs larger than a PDU.*
   Our first requirement is that the protocol accommodate out-of-order delivery, but does so in a way that allows the receiver application to make sense of the mis-ordered data units it receives. In the pure ALF model [23], each ADU is matched to the size of a protocol data unit (PDU) used by the transport protocol. This implies that there is no "coupling" between two packets and that they can be processed in any order. Unfortunately, it is difficult to ensure that an ADU is always well matched to a PDU because the former depends on the convenience of the application designer and what is meaningful to the application, while the latter should not be too much larger (if at all) than the largest datagram that can be sent unfragmented,

in order to minimize retransmission overhead in the event of a packet loss. This means that there are times when an ADU is larger than a PDU, requiring an ADU to be fragmented by the transport protocol for efficiency.

2. *Support receiver-controlled selective reliability.*
Our next design consideration addresses reliability. When packets are lost, there are two possible ways of handling retransmissions. The conventional approach is for the sender to detect losses and retransmit them in the order in which they were detected. While this works well for protocols like TCP that simply deliver all the data sequentially to a receiver, interactive image transfers are better served by a protocol that allows the receiving application and user to have a say in which losses are retransmitted from the sender, and in what order. For example, a user should be able to express interest in a particular region of an image, causing the transport protocol to prioritize the corresponding data over others. In general, the receiver knows best what data it needs, is any, and therefore allowing it to control requests for retransmission is best-suited to improving user-perceived quality.

3. *Support easy customization for different image formats.*
Our third design consideration is motivated by the observation that there are many different image formats that can benefit from out-of-order processing, each of which may embed format-specific information in the protocol. For example, the JPEG format uses an optional special delimiter called a *restart marker*, which signifies the start of an independently processible unit to the decoder. Such format- or application-specific information should be made available to the receiver in a suitable way, without sacrificing generality in the basic protocol.

   The customizability of ITP borrows from lessons learned from the design of other application-level transport protocols such as the Real-time Transport Protocol (RTP) [125]. In ITP, as in RTP, a base header can be customized by individual application protocols, with profile-specific extension headers incorporating additional information.

4. *Application and higher-layer protocol independence.*
While this work is motivated by interactive image downloads on the Web, we do not want to restrict our solution to just HTTP. In particular, we do not want to change the HTTP specification in any way and the goal is to replace HTTP/TCP with HTTP/ITP for image data.

5. *Sound congestion control.*
Finally, congestion-controlled transmissions are important for any deployable transport protocol on the Internet. But rather than reinvent complex machinery for congestion management (a look at many of the subtle bugs in TCP congestion control implementations that researchers have discovered over the years shows that this is a hard task [102]), we leverage the recently developed Congestion Manager (CM) architecture [6]. The CM abstracts away all congestion control into a trusted kernel module independent of transport protocol, and provides a general API for applications to learn about and adapt to changing network conditions [7]. Our design uses the CM to perform congestion control, with packet transmissions occurring only when permitted by the CM via its API.

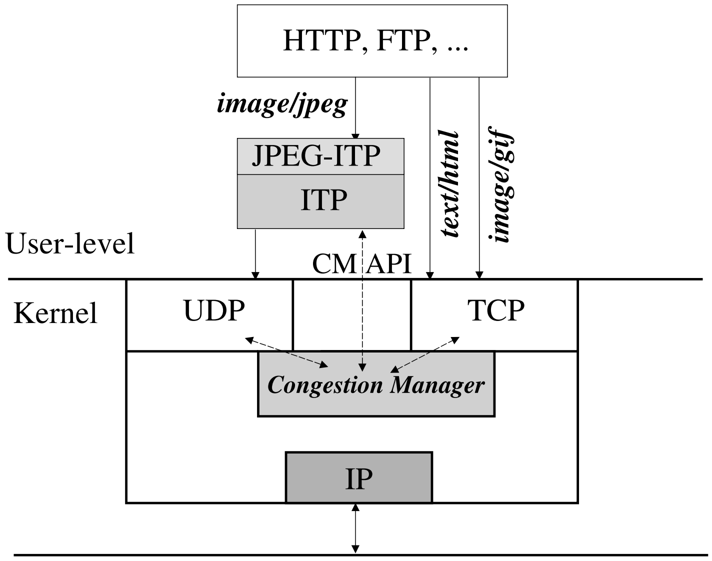

Figure 6.1: The system architecture showing ITP, its customization for JPEG, and how HTTP uses it instead of TCP for MIME type "image/jpeg" while using a conventional TCP transport for other data types. All HTTP protocol messages are sent over ITP, not just the actual image data, which means that ITP replaces TCP as the transport protocol for this data type.

6.3 ITP Design

In this section, we describe the design of ITP and the techniques used to meet the aforementioned design goals. ITP is designed as a modular user-level library that is linked by the sender and receiver application. The overall system architecture is shown in Figure 6.1, which includes an example of an application protocol such as HTTP or FTP using ITP for data with MIME type "image/jpeg" and TCP for other data. It is important to note that ITP "slides in" to replace TCP in a way that requires no change to the specification of a higher-layer protocol like HTTP or FTP.

6.3.1 Out-of-order Delivery

One of the main departures of ITP from traditional transport protocols is its out-of-order delivery abstraction. Providing such an abstraction at the granularity of a byte, however, would make it hard for the application to infer what application data units a random incoming sequence of bytes corresponds to. The application handles data in granularities of an ADU, so ITP provides an API by which an application can send or receive a complete ADU. We now describe the mechanics of data transfer through the sending and receiving ITP hosts.

The sending application invokes `itp_send()` to send an ADU to the receiver. Before shipping the ADU, ITP incorporates a header, shown in Figure 6.2 that includes an incrementing ADU sequence number and ADU length. The sequence number and length of an ADU are used by

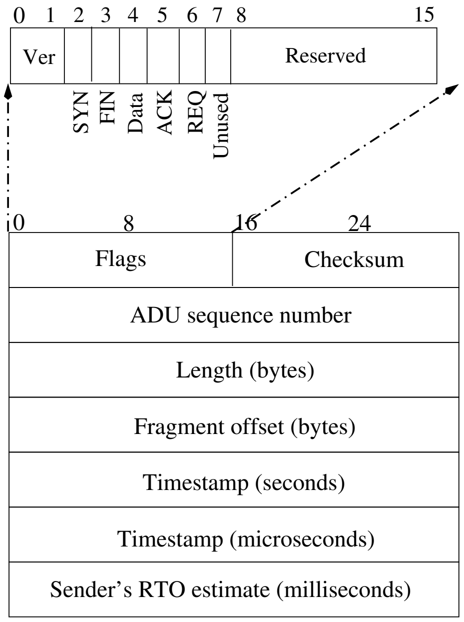

Figure 6.2: The 28-byte generic ITP transport header contains meta-data pertaining to each fragment, as well as the ADU that the fragment belongs to, such as the ADU sequence number and length, the fragment offset within the ADU, a sender timestamp, and the sender's estimate of the retransmission timeout.

the receiver to detect losses, perform reassembly within an ADU, and verify that the complete ADU has arrived.

When a complete ADU arrives at the receiver, the ITP receiver invokes a well-known callback function implemented by the application, called `itp_app_notify()`. In response, the application calls an ITP library function `itp_read()` to read the incoming ADU into its own buffers, and returns control to ITP. This interaction is shown in Figure 6.3. The important point to note is that this sequence of steps occurs when a complete ADU arrives at the receiver, *independent* of the order in which it was transmitted from the sender.

Unfortunately, not all ADUs are small enough to fit in one PDU which is the maximum unfragmented datagram on the path to the receiver. This requires that any ADU larger than a PDU be fragmented into PDU-sized units before transmission. Using arbitrarily-sized ADUs as the granularity of loss recovery is inefficient. Consider for example an ADU transmitted by the transport protocol that was fragmented by a lower layer for transmission, and exactly one of the fragments was lost in transit. The receiver must ask for the entire ADU to be retransmitted if the unit of naming and transmission by ITP is an ADU, thereby degrading protocol goodput. Rather than suffer performance due to redundant retransmissions, ITP bridges the mismatch between network-supported packet sizes and application-defined data units by breaking up an ADU into *fragments* no bigger than the maximum transmission unit of the path and identifying each fragment by its byte-offset

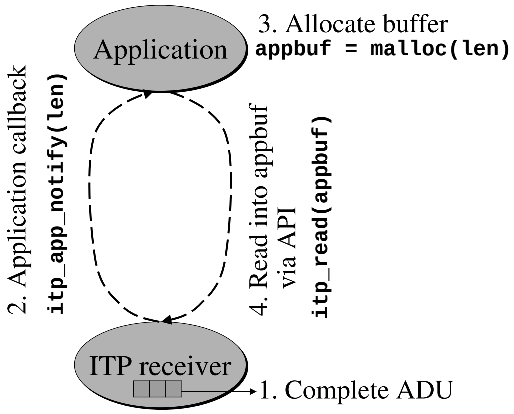

Figure 6.3: The sequence of operations when a complete ADU arrives at the ITP receiver.

and length within an ADU as well as the ADU sequence number.[1] We emphasize that this is done to avoid inefficiencies in retransmission, but is not exposed to the receiving application. As a result, applications are not forced to limit their framing to network packet sizes, and incomplete ADU data are not visible to them.

6.3.2 Reliability

One of the design goals in ITP is to put the receiver in control of loss recovery. This suggests a protocol based on *retransmission request* messages sent from the receiver. In addition to packet loss, ITP must reliably handle connection establishment and termination, as well as host failures and subsequent recovery without compromising the integrity of delivered data. We incorporate TCP-like connection establishment and termination mechanisms for this.

Connection management

Although an important application of ITP is downloading images on the Web using HTTP, we do not want to restrict all higher-layers to HTTP-like protocols where the client initiates the connection. For example, when used by FTP, the server performs the active open rather than the client.

We chose to emulate the three-way connection establishment procedure of TCP [128]. The initial sequence number chosen by both sides determines the ADU sequence space for each transfer direction. We do not view the three-way handshake as a performance problem, despite the extra round-trip that it entails; indeed, should this be a concern, it can be modified to allow data to

[1] Path MTU discovery [89] can be used to determine this value.

be piggybacked along with the establishment message.[2]

We also choose to mimic the FIN-ACK mechanism of TCP, transitioning into exactly the same states as a terminating TCP (CLOSE_WAIT for a passive CLOSE; FIN_WAIT_1, optionally followed by CLOSING or FIN_WAIT_2, and then a TIME_WAIT for an active one). As in TCP, the active closer transitions from the TIME_WAIT state to CLOSED after the 2MSL timeout. The sender signals the last ADU in a transmission sequence by setting the FIN bit in the flags of the ITP header. The receiver uses this to detect when all transmitted data items (of interest to the receiver) have arrived and to terminate the connection.

We believe that the design choice of preserving the TCP connection establishment and termination procedures are the right ones for ITP, given the combination of applications we would like to support (all combinations of servers and clients performing active/passive opens and closes), as well as the difficulties in designing robust connection establishment and termination procedures. This decision allows us to be fairly certain of the correctness of the resulting design.

We do, however, address the significant problem of connections in the TIME_WAIT state at a busy server. The problem is that in most HTTP implementations, the server does the active close rather than the client, which causes the server to expend resources and maintain the TIME_WAIT connections. This design is largely forced by many socket API implementations, which do not allow applications to easily express a half-close.[3] One recently proposed solutions to this in the context of TCP is to use a "time-wait negotiation" between TCP peers at the start of a connection [33].

However, we solve this problem by providing a "half-close" call to the ITP API that allows the client use it. When one side (e.g., an HTTP client, soon after sending a GET message) decides that it has no more data to send, but wants to receive data, it calls `itp_halfclose()` which sends a FIN to the peer. Of course, retransmission requests and data ACKs continue to be sent. In the context of HTTP, the TIME_WAIT state maintenance is therefore shifted to the client, freeing up server resources.

Loss recovery

All retransmissions in ITP occur only upon receipt of a retransmission request from the receiver, which names a requested fragment using its ADU sequence number, fragment offset, and fragment length. While many losses can be detected at the receiver using a data-driven mechanism that observes gaps in the received sequence of ADUs and fragments, not all losses can be detected in this manner. In particular, when the last fragment or "tail" of a burst of fragments transmitted by a sender is lost, a retransmission timer is required. Losses of previous retransmissions similarly require timer-based recovery.

One possible design is for the receiver to perform all data-driven loss recovery, and for the sender to perform all timer-based retransmissions. However, this runs contrary to our goal of receiver-controlled reliability because the sender has no knowledge of the fragments most useful to the receiver. Unless we incorporate additional complex machinery by which a receiver can convey

[2] We do not recommend this mode as it tends to make defense against denial-of-service attacks like SYN-floods hard to handle (e.g., using SYN-cookies, which ITP can incorporate with little difficulty).

[3] The `shutdown(socket_fd, how)` call, with `how` set to 1 is supposed to cause a half-close, telling the peer that no more data will be originated on the connection, but not all TCP implementations handle this correctly. Furthermore, the Hosts Requirements RFC 1122 lists the half-close as a "MAY implement" option.

to the sender what fragments it is interested in, the sender ends up retransmitting old, uninteresting data on a timeout.

Our solution to this problem is to move timer handling to the receiver. If the receiver detects no activity for a timeout duration, a retransmission request is sent. If no gaps are detected in the received ADU stream, a retransmission request is sent for the next expected ADU, i.e., 1 + last ADU sequence number received, thereby initiating recovery from a tail loss, if there was one. Since the retransmission timer is always active until a FIN, this message is repeated periodically until the receiver is ready to terminate.[4]

It is rather difficult for accurate round-trip time estimation to be performed at the receiver when data flows from sender to receiver. Hence, we allow the sender calculate the retransmission timeout (RTO) as in TCP with the timestamp option [60], and pass this RTO to the receiver in the ITP header (Figure 6.2).

ITP also incorporates data-driven retransmission requests. To do this, the receiver maintains a maintains a list of incomplete and missing ADUs. When a fragment is received, missing fragments or ADUs are detected by looking up the data structure. The receiver now has three tasks:

- Decide whether it is time to ask for the fragment.
- Decide how many fragments to ask for.
- If at least one fragment can be requested at this time, decide which fragments to request.

Two considerations dictate whether it is time to ask for a fragment. First, if a request has already been made for the fragment, it should not be made again unless an RTO has elapsed since the first request. The receiver logs the time of last request and ensures that a subsequent request is sent only if the elapsed time is longer than an RTO.

Second, packets may get reordered on the Internet [101], and the receiver must guard against asking for a reordered (but not lost) fragment. The approach in TCP is to wait for a threshold number (three) of duplicate ACKs and retransmit the first unacknowledged segment. Unfortunately, this does not work well when windows are small or when ADUs are small in size (as is often the case for ITP applications). Our solution to this problem is motivated by the observation by Paxson that a small delay before sending an ACK in TCP often catches reordered segments [103]. ITP modifies this approach by adapting it to the transmission rate r (in fragments/sec) from the sender, which it monitors using an exponentially-weighted moving average filter. The receiver waits for a duration equal to $3/r$ seconds before sending a request, during which reordered fragments may arrive and cancel a pending retransmission request.

In our initial design, the receiver requested exactly one missing fragment on detecting losses, even if more losses were detected. Our experiments after implementing this strategy revealed a subtle interaction with selective reliability, which does not occur in TCP.

Consider the case when a timeout occurs and the congestion window at the sender is set to 1, as shown in Figure 6.4. A retransmission request from the receiver causes the sender to send one request fragment. When this fragment is ACKed, congestion control causes the sender's window to grow to 2. The sender may have other old data that the receiver has not yet received, but because all reliability is receiver-controlled, the sender cannot unilaterally retransmit old data. The sender

[4] Note that this approach does not imply that an HTTP server ends up periodically probing the client asking if there is any data after a GET of a URL. Once a half-close is received from the client, the server disables the timer.

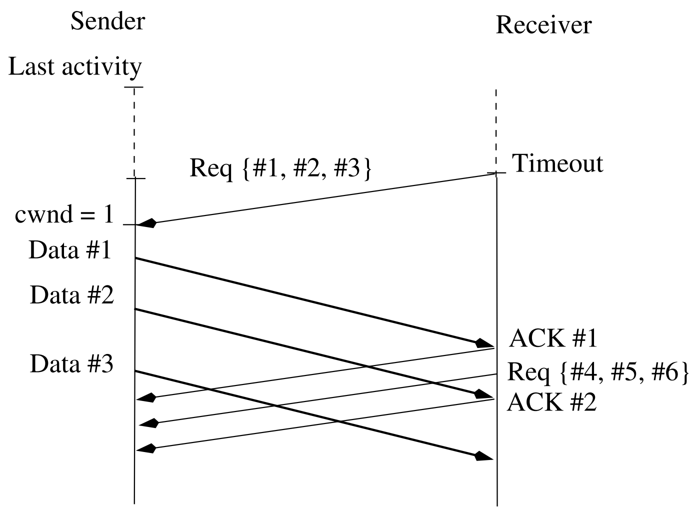

Figure 6.4: Retransmissions #1, #2, and #3 are transmitted before the next request is received by the sender. Sending three requests in each request message keeps the "pipe" full.

therefore decides to send new fragments and use up its newly opened congestion window (slow start), making timely loss recovery of other lost fragments difficult. The receiver therefore loses the ability to order and prioritize a particular set of retransmissions before any other new data is received.

This problem is solved if the receiver sends at least three retransmission requests each time a loss is detected, assuming that many losses have occurred and the receiver is interested in recovering them. This allows the sender to build up an ordered list of pending retransmissions and use up a newly opened congestion window for retransmissions requested by the receiver rather than new data. Every time a loss is detected, up to three fragments are potentially capable of being transmitted from the sender before the next retransmission request reaches it. The number three is a consequence of TCP-style slow start implemented in the CM.

The most difficult part in loss recovery is to decide which fragment to request at any time among the missing ones. This is difficult because of the tension between application-specificity and generality. We would like to put the application in control of what to request, but save each application the trouble of writing the complex loss detection code. Furthermore, we would like to provide a reasonable default behavior to handle applications that do not care to customize their reliability schedules.

ITP provides a simple default scheduling algorithm for retransmission requests that simply requests fragments from all the missing ADUs from the most recent one to the least recent, subject to the above conditions of not requesting them too soon. However, it also allows application-specific customization of reliability by extending the base header, as described in Section 6.4.2 for JPEG. This allows a JPEG receiver to request only fragments from ADUs that it is currently interested in, based on the decoding algorithm it implements.

ITP's receiver-controlled selective reliability differs in significant ways from WebTP, which does share similar reliability goals. For example, WebTP uses a fully-qualified URL to

```
PROCESSRXMITREQ(fragment)
    Send requested fragment via cm_send();
    InformCM();

INFORMCM()
    now ← current_time;
    if (now − last_activity > timeout_duration)
        cm_update(. . . , CM_PERSISTENT, . . . );
    else
        cm_update(. . . , CM_TRANSIENT, . . . );
```

Figure 6.5: How the ITP sender handles a retransmission request.

identify an ADU similar to the work reported in [113], while ITP uses a simpler fixed-length ADU sequence number but disseminates a mapping at the beginning of a connection that enables customization. ITP uses the simpler strategy of sending the RTO in the packet header to the receiver compared to WebTP, which uses estimates the mean inter-arrival packet time[5] and sending a retransmission request if no packet arrived in some deviation from this. ITP incorporates ideas that can be used by a general selectively reliable protocol, but our primary contributions are its customization and evaluation in the context of image transport. The scheduling algorithm presented in Section 6.4 for JPEG-ITP retransmission requests shows how a receiver can customize the retransmission schedule.

6.3.3 Using the Congestion Manager

ITP relies on the CM for congestion control, using the CM API to adapt to network conditions and to inform the CM about the status of transmissions and losses [7]. Since ITP reliability is receiver-based, there is no need for positive ACKs from the receiver to the sender for reliability. ACKs from the receiver are solely for congestion control and estimating round-trip times. The CM requires the cooperation of the application in determining the state of the network. By informing the ITP sender about the status of transmissions, an ITP ACK allows the sender to update CM state. When the ITP sender receives an ACK, it calculates how many bytes have cleared the "pipe" and calls `cm_update()` to inform the CM of this.

When a retransmission request arrives at the sender, the sender infers that packet losses have occurred, attributes them to congestion (as in TCP), and invokes `cm_update()` with the `lossmode` parameter set to CM_TRANSIENT, signifying transient congestion. In a CM-based transport protocol where timeouts occur at the sender, the expected behavior is to use cm_update() with the `lossmode` parameter set to CM_PERSISTENT, signifying persistent congestion. In ITP, the sender never times out, only the receiver does. The sender only sees a request for retransmission arriving after a timeout at the receiver, so when a retransmission request arrives, it needs to determine if that occurred after a timeout or because of out-of-sequence data. We solve this problem by calculating the elapsed time since the last time there was any activity on the connection from

[5]We believe this is less well-understood than our approach, and note that the congestive collapse episodes of the mid-1980s were largely because of bad retransmission strategies.

the peer, and if this time is greater than the retransmission timeout value, then the CM is informed about persistent congestion. Figure 6.5 shows what the ITP sender does when it receives a request for retransmission.

6.3.4 Design Summary

In summary, ITP provides out-of-order delivery with selective reliability. It handles all combinations of active/passive opens and closes by server and client applications by borrowing TCP's connection management techniques. Application-level protocols like HTTP do not have to change their specifications to use ITP.

ITP differs from TCP in the following key aspects. It does not force a reliable in-order byte stream delivery and puts the receiver in control of deciding when and what to request from the sender. It uses a callback-based API to deliver out-of-order ADUs to the application. ITP includes a "half-close" method that moves the TIME_WAIT maintenance to the client in the case of HTTP. In TCP the sender detects re-ordered segments only after three duplicate ACKs are received, while in ITP, receivers detect re-ordering based on a measurement of the sending rate. We emphasize that ITP has a modular architecture and relies on CM for congestion control. ACKs in ITP are used solely as feedback messages for congestion control and round-trip time calculation, and *not* for reliability.

6.4 JPEG Transport using ITP

In this section, we discuss how to tailor ITP for transmitting JPEG images. JPEG was developed in the early 1990s by a committee within the International Telecommunications Union, and has found widespread acceptance for use on the Web. The compression algorithm uses block-wise discrete cosine transform (DCT) operations, quantization, and entropy coding [104]. JPEG-ITP is the customization of ITP by introducing a JPEG-specific framing strategy based on restart markers and tailoring the retransmission protocol by scheduling request retransmissions.

6.4.1 Framing

The current model for JPEG image transmission on the Internet is to segment it multiple packets. However, JPEG uses entropy coding, and the resulting compressed bitstream consists of a sequence of variable-length code words, and packet losses often result in catastrophic loss if pieces of the bitstream are missing at the decoder. Arbitrarily breaking an image bitstream into fixed-size ADUs does not work because of dependencies between them.

However, JPEG uses *restart markers* to allow decoders to resynchronize when confronted with an ambiguous or corrupted JPEG bitstream, which can result from partial loss of an entropy coded segment of the bitstream. The introduction of restart markers helps localize the effects of the packet loss or error to a specific sub-portion of the rendered image. This segmentation of the bitstream into independent restart intervals also facilitates out-of-order processing by the application layer. The approach used by JPEG to achieve loss resilience provides a natural solution to our framing problem.

When an image is segmented into restart intervals, each restart interval is independently processible by the application and naturally maps to an ADU. The image decoder is able to decode

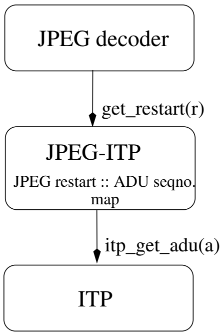

Figure 6.6: JPEG-ITP maintains a mapping of restart intervals to ADU sequence numbers.

and render those parts of the image for which it receives information without waiting for packets to be delivered in order. The base ITP header is extended with a JPEG-specific header shown in that carries framing information, which includes the spatial position of a 2-byte restart interval number.

Our implementation of JPEG-ITP uses 8-bit gray-scale images in the baseline sequential mode of JPEG. We require that the image server store JPEG images with periodic restart markers. This requirement is easy to meet, since a server can easily transcode offline any JPEG image (using the `jpegtran` utility) to obtain a version with markers. When these markers occur at the end of every row of blocks, each restart interval corresponds to a "stripe" of the image. These marker-equipped bistreams produce exactly the same rendered images as the original ones when there are no losses. Since JPEG uses a blocksize of 8x8 pixels, each restart interval represents 8 pixel rows of an image. We use the sequence of bits between two restart markers to define an ADU, since any two of these intervals can be independently decoded. Our placement of restart markers achieves the effect of rendering an image in horizontal rows.

6.4.2 Scheduling

Figure 6.6 shows the key interfaces between ITP and JPEG-ITP, and between JPEG-ITP and the decoder. ITP handles all fragments and makes only complete ADUs visible to JPEG-ITP. To preserve its generality, we do not expose application-specific ADU names to ITP. Thus, when a missing ADU needs to be recovered by the decoder, JPEG-ITP needs to map the restart interval number to an ITP ADU sequence number. To do this, the JPEG-ITP sender reliably transmits this mapping as the first ADU of the connection, before transmitting the image ADUs. This name map is used to schedule ITP retransmission requests.

ITP maintains a priority list of the retransmission schedule by exporting an asynchronous API function `itp_get_adu()` that customized protocols like JPEG-ITP and applications can use to inform ITP of the desired ADU. ITP uses this priority information to schedule requests for missing fragments from these ADUs ahead of others. In addition, JPEG-ITP exports an API function to the decoder that allows the latter to specify restart intervals that must be prioritized during recovery, e.g., if the decoder uses error concealment as in Section 6.4.3, this is used to preferentially request ADUs that have not been interpolated from the existing partial image.

6.4.3 Error Concealment

Out-of-order delivery allows the JPEG decoder to refine a partial image using error concealment based on interpolation techniques. Portions of the image corresponding to the received ADUs are decoded and rendered. Before rendering, a post-processing step is applied to the image to conceal lost stripes. Error concealment exploits spatial redundancy in images and aims to increase the perceptual quality of the rendered image.

Each missing pixel value is the result of a linear interpolation, or average, of its neighbors. This step is applied to all missing restart intervals at the receiver. Therefore, in 2-D, the missing pixel $x_{i,j}$ is given by:

$$x_{i,j} = \frac{x_{i-1,j} + x_{i+1,j} + x_{i,j-1} + x_{i,j+1}}{4} \tag{6.1}$$

The boundary conditions are determined by the pixel values of neighboring blocks. Using the lexicographic ordering of pixels in a block, $\mathbf{x} = \{x_{0,0}, x_{0,1}, ...x_{0,B-1}, x_{1,0}..., x_{B-1,B-2}, x_{B-1,B-1}\}$, the estimate of the missing block may be computed as

$$\hat{\mathbf{x}} = A^{-1}\mathbf{c} \tag{6.2}$$

where A is a block tri-diagonal matrix given by

$$A = \begin{bmatrix} L & I & O & \cdots & \\ I & L & I & O & \cdots \\ O & I & L & I & O \\ \cdots & O & I & L & I \\ & \cdots & O & I & L \end{bmatrix} \tag{6.3}$$

and L is a 8x8 tri-diagonal matrix formed from $\{1, -4, 1\}$.

$\mathbf{c}$ is a vector that represents the boundary conditions imposed by the pixels $above(u)$, $below(d)$, to the $left(l)$ and to the $right(r)$ of the current block.

$$\begin{aligned} c(0,0) &= l(0) + u(0) \\ c(0,B-1) &= r(0) + u(B-1) \\ c(B-1,0) &= l(B-1) + d(0) \\ c(B-1,B-1) &= r(B-1) + d(B-1) \end{aligned}$$

Other sophisticated error concealment techniques have been proposed in the literature, especially for video. For example, in [120], the authors propose the use of a Markov Random Field image model and optimally interpolate the missing pixels. The emphasis of our scheme, however, is on simplicity and on maximizing interactivity, rather than precision, for which we find empirically that our simple averaging strategy seems to work well.

6.4.4 Other Formats

We have described a simple framing strategy and further refinement using error concealment scheme for JPEG over ITP. The same techniques also extend to progressive JPEG images. In progressive JPEG, the quantized DCT coefficients corresponding to each block are divided into a series of scans. These scans may either represent different frequencies (low to high), or different bit-planes of the quantized coefficients (most significant to least significant bits). A coarse representation of the image is rendered with the receipt of the first scan, which is successively refined as subsequent scans arrive. Each scan can be segmented into restart intervals, which results in the ability to process and render out-of-order within a scan, leading to quicker response times and interactivity. Error-concealment can be carried out in a multi-resolution manner by performing concealment within one scan at a time.

Similar techniques are also possible for transmission of JPEG2000, which is a recent proposal for wavelet-based image coding scheme that results in higher compression ratios and better fidelity. The standard supports several features such as layered coding and "region of interest" (ROI) coding. Designing transport support for ROI coding requires customized scheduling of retransmission requests at the receiver, which is provided by ITP.

6.5 Performance Evaluation

In this section, we evaluate our implementation of ITP under a variety of network loss rates. Our implementation of ITP performs out-of-order data delivery at the receiver and uses the averaging method to interpolate missing packets at the receiver. We have customized ITP for JPEG transport where the images contain restart intervals. We have not implemented nor evaluated other formats We first discuss the performance metrics we use and present the results of our evaluation.

6.5.1 Peak Signal-to-Noise Ratio (PSNR)

Image quality is often measured using a metric known as the PSNR, defined as follows. Consider an image whose pixel values are denoted by $x(i,j)$ and a compressed version of the same image whose pixel values are $\hat{x}(i,j)$. The PSNR quality of the compressed image (in dB) is:

$$\mathrm{PSNR} = 10 \times \log_{10} \frac{255^2}{E||x(i,j) - \hat{x}(i,j)||^2} \tag{6.4}$$

In our experiments, we use PSNR with respect to the transmitted image as the metric to measure the quality of the image at the receiver. Note that PSNR is inversely proportional to the mean-square distortion between the images, which is given by the expression in the denominator of Equation 6.4. When the two images being compared are identical, e.g., at the end of the transfer when all blocks from the transmitted image have been received, the mean-square distortion is 0 and the PSNR becomes ∞. We recognize that PSNR does not always accurately model perceptual quality, but use it because it is a commonly used metric in the signal processing literature.

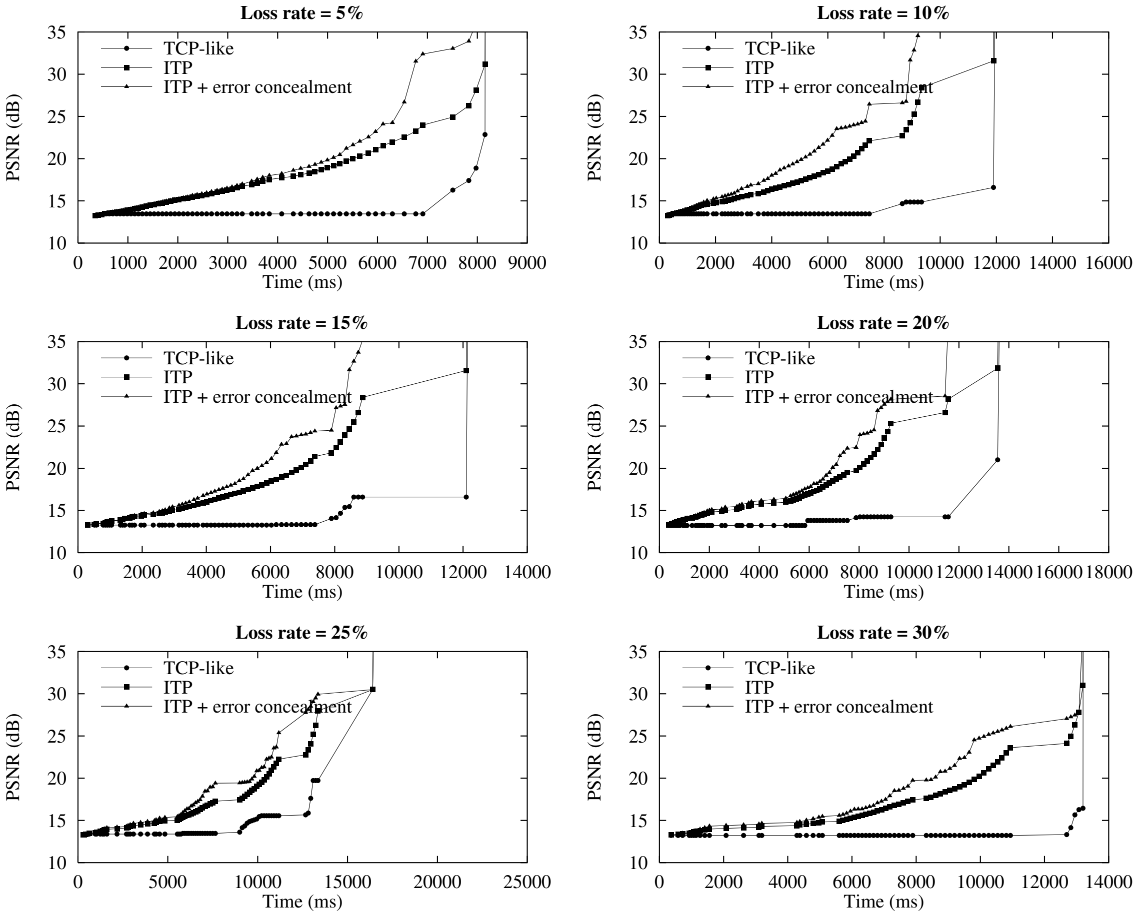

Figure 6.7: PSNR vs. Time for ITP and TCP-like transports. The quality of the image (as measured by PSNR) is identical in all three scenarios at the start and at the end of the transfer. However, the sample paths differ — the best performance is seen with ITP optimized with error concealment, while TCP shows the poorest performance. ITP shows a steady improvement in quality, and is therefore perceptually superior for interactive applications such as the Web.

6.5.2 Experimental Results

We measure the evolution of instantaneous PSNR as the JPEG image download progresses. When JPEG-ITP receives a complete restart interval from ITP, it is passed to the decoder. The decoder output is processed to fill in missing intervals using the error concealment step explained earlier and the image is updated. We measure PSNR with respect to the original JPEG image transmitted under three scenarios: (i) when TCP-like in-order delivery is enforced, (ii) when out-of-order delivery is allowed, and (iii) when error concealment is performed on the mis-ordered data units.

Figure 6.7 shows the results of this experiment under a variety of loss rates. We use a simple Bernoulli loss model where each packet is dropped at the receiver with an independent probability given by the average loss rate.

We find that across a range of loss rates between 5% and 30%, TCP-like delivery causes the quality of the rendered image to remain low for extended intervals of time. In comparison, ITP with out-of-order delivery shows a smoother evolution of PSNR during the transfer. In addition, the PSNR of the ITP-delivered image is superior to that delivered by TCP while the transfer is in progress, becoming identical only at the end of the transfer, as expected. This smooth evolution of quality makes ITP better suited for interactive image downloads. When error concealment is applied as an added optimization on the partial image, we find that the benefits are between 2–8 dB. In combination, the two techniques outperform TCP by 10–15 dB.

Figure 6.8 shows the progression of displayed images for the three different scenarios and Figure 6.9 shows the corresponding PSNR values. Starting with almost identical image snapshots at $2s$, the ITP-delivered images (with and without error concealment) show steady improvement in quality relative to the TCP-delivered snapshot. At $10s$, the ITP image is 3.3 dB and a further improvement of 1.3 dB is achieved through interpolation on the partial image. As we can see from the image, the benefits of interpolation are greater when more of the image is available, which further strengthens the case for out-of-order delivery in ITP. The ITP images continue to improve and at $12s$, they are 12 dB (without error concealment) and 20 dB (with error concealment) better than the corresponding TCP-delivered images. We also conduct a transfer across a 1.5 Mbps link to study the effect of receiver scheduling. Here, the receiver prioritizes requests for data items that cannot be concealed using interpolation.

In summary, we find that the rate of increase in PSNR with time is significantly higher for ITP compared to TCP-like delivery.

6.6 Concluding Remarks

In this Chapter, we observe that the reliable, in-order byte stream abstraction provided by TCP is overly restrictive for richer data types such as image data. Several image encodings such as sequential and progressive JPEG and JPEG 2000 are designed to handle sub-image level granularities and decode partially received image data. To improve perceptual quality of the image during a download, we proposed a novel Image Transport Protocol (ITP). ITP uses an application data unit (ADU) as the unit of processing and delivery to the application by exposing application framing boundaries to the transport protocol. This enables the receiver to process ADUs out of order. ITP can be used as a transport protocol for HTTP and is designed to be independent of the higher-layer application or session protocol. ITP relies on the Congestion Manager (CM) to perform safe and stable congestion control, making it a viable transport protocol for use on the Internet today.

We have shown how ITP is customized for specific image formats, such as JPEG. Out of order processing facilitates effective error concealment at the receiver that further improve the download quality of an image. We have implemented ITP as a user-level library that invokes the CM API for congestion control. We have also presented a performance evaluation demonstrating the benefits of ITP and error concealment over the traditional TCP approach, as measured by the peak signal-to-noise ratio (PSNR) of the received image.

In summary, ITP provides the basis for a general purpose selectively reliable unicast transport protocol that can be applied to diverse data types. Our design and implementation provide a generic substrate for congestion-controlled transports that can be tailored for specific data types.

$t_1 = 2s$ $t_2 = 10s$ $t_3 = 16s$

Figure 6.8: Snapshots of the displayed image with a TCP-like transport (first row), with ITP (second row), and with ITP enhanced with error concealment (last row) at 10% loss rate. The entire transfer of the 184 KB image takes $16.57s$ to complete.

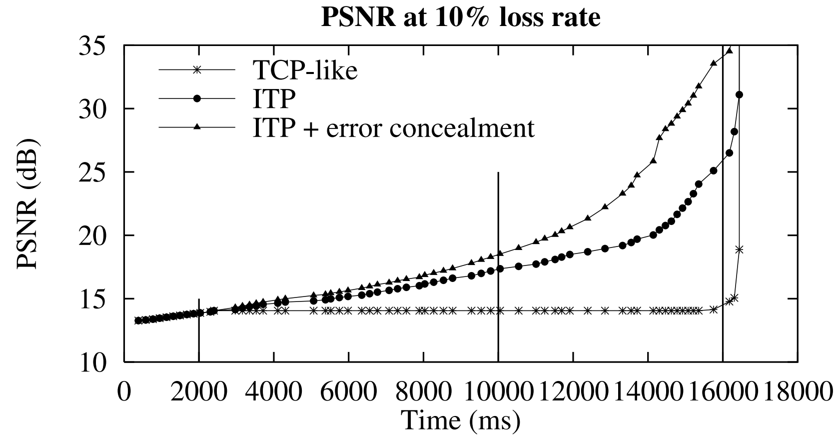

Figure 6.9: PSNR corresponding to the snapshots shown in Figure 6.8. Starting at almost identical image snapshots at $2s$, the ITP image (with and without error concealment) progress steadily in quality, while the TCP-delivered image only catches up close to completion time.

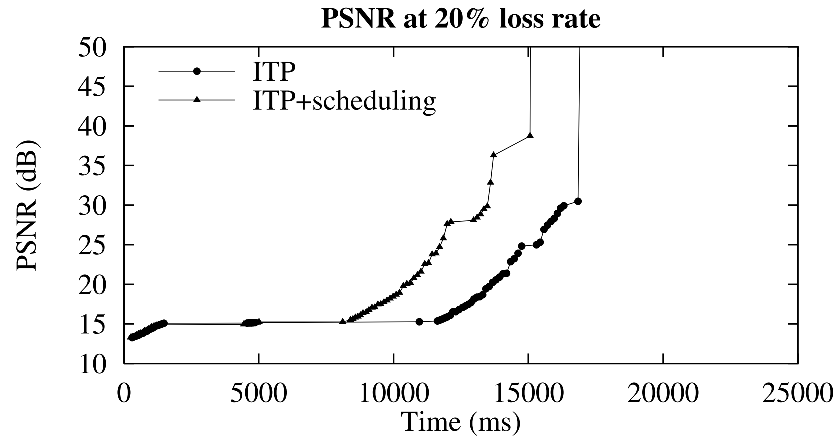

Figure 6.10: When receiver request scheduling takes into consideration those “stripes” that cannot be interpolated, the quality of the rendered image can be improved by 5–15 dB.

Even though we have studied the specific case of unicast image transfer, the lessons learned here are especially relevant and applicable to multicast transport as well.

Chapter 7

libsstp: A User-level Transport Protocol for Interactive Multicast Applications

Those parts of the system that you can hit with a hammer (not advised) are called hardware; those program instructions that you can only curse at are called software.
— Anonymous

In this Chapter we describe our implementation of *libsstp*, which is user-level library that implements our transport protocol framework for multicast applications. Libsstp is intended as a vehicle for research and experimentation on issues in multicast transport such as loss recovery, data naming, data consistency, and congestion control. This Chapter proceeds as follows. Section 7.1 describes the software architecture and implementation of libsstp and its simple yet powerful programming interface. We discuss the different applications in Section 7.2, ranging from an information dissemination tool to a controller for a special effects video processing system running on a network of workstations. Finally, we conclude this Chapter in Section 7.3.

7.1 Libsstp Software Architecture

The software architecture of libsstp is shown in Figure 7.1 and consists of the following main components. We now describe the two main pieces: the core SSTP protocol framework and the event subsystem.

The SSTP protocol piece is responsible for data naming and name announcement which are done using the SNAP protocol as discussed in Chapter 4. libsstp runs over UDP and incorporates the basic slotting and damping algorithm for limiting receiver feedback. It provides "local recovery" to limit the scope of request and repair packets in an attempt to reduce the amount of network traffic sent to the global scope in a large session. Libsstp also provides the necessary mechanisms to traffic shape the local data sources in a session by rate limiting them. The limiting rate is controlled by the application through hooks provided in the interface. It is conceivable that a congestion control algorithm that determines "bottleneck" network bandwidth could automatically tune the transmission rate to avoid excessive packet loss. Libsstp uses application callbacks for significant network events occur to facilitate selective retrieval of specific data items by the application.

The libsstp implementation is a user-level library and is composed of about 10,000 lines of C and C++ code. Libsstp provides a C programming interface, as well as a tcl command interface. It is implemented as an event-based library and plugs into any event system (e.g., the Tcl/Tk toolkit [95]) via a generic event API that allows handlers for timer and input/output events to be registered. We use events to implement application callbacks that provide the appropriate hooks for the application to tailor its behavior. The event API is shown in Figure 7.1.5 and allows an application to register and de-register handlers for network and I/O events.

In the remainder of this section, we describe the modules within libsstp and the relevant application interfaces.

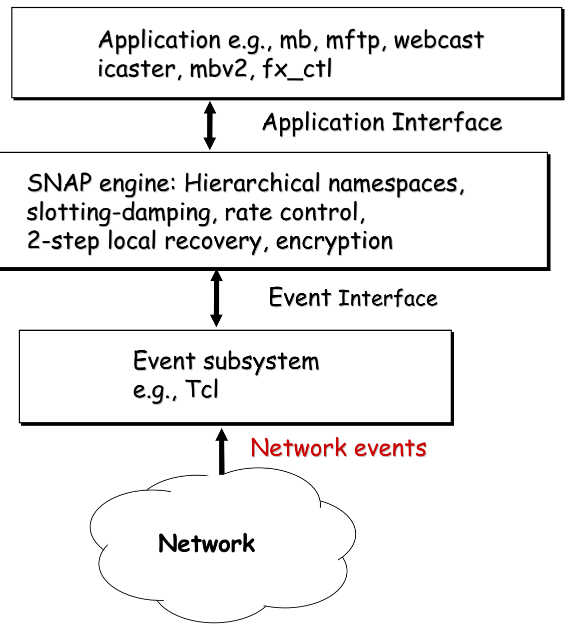

Figure 7.1: Software architecture of libsstp, our user-level library that implements the soft state-based transport protocol.

The core protocol functions are divided into five main categories: source, session functions that allow the application to manipulate the SSTP session and source objects; data and callback functions, that affect the data path and assist the receiver in performing selective reliability; as well as debugging functions.

7.1.1 Session Object

The session object represents an SSTP session specified using a multicast group address, from the range `224.*.*.*` — `239.*.*.*`, the send and receive UDP port numbers, and a TTL to limit the scope of the session. An application process may simultaneously create and use multiple SSTP sessions, and each session is uniquely identified within the application process by a

32-bit session ID of type `sstp_session_t`. The following code shows the syntax of the functions used to create, reset and delete a session object, and to manipulate the amount of session bandwidth allocated to local sources.

```
sstp_session_t sstp_create_session(u_int32_t addr, u_int32_t sport,
                                    u_int32_t rport, u_int32_t ttl);

void sstp_destroy_session(sstp_session_t sess);

int  sstp_reset_session(sstp_session_t sess, u_int32_t addr,
                        u_int32_t port, u_int32_t ttl);

void sstp_set_session_bandwidth(sstp_session_t sess, int bps);

int  sstp_get_session_bandwidth(sstp_session_t sess);
```

`sstp_create_session` creates a new session on the multicast channel given by `addr`. `sport` and `rport` are the send and receive addresses respectively. The default `ttl` is 15. The function returns an `sstp_session_t` handle for this session, or returns `NULL` if an error occurred in creating a session with the specified `addr`, `sport`, `rport`, and `ttl`. `sstp_destroy_session` destroys an existing session `sess`. `sstp_reset_session` resets an existing SSTP session to use a new multicast group. `sess` is a token for the session that was returned by a call to `sstp_create_session`.

The `session_bandwidth` functions control the session bandwidth parameter in libsstp. This parameter is used within libsstp to traffic shape messages in this session using the leaky bucket algorithm. The default bandwidth is set to 128 kbps.

If the application hands a large ADU to libsstp, it is appropriately chunked into smaller transmission units of at most MTU-sized PDUs to avoid the redundant retransmission problem that arises when we rely on IP fragmentation and reassembly. We use a MTU of 1408 bytes, which is sufficiently small to cover most commonly used link layer technologies, since automatic path MTU discovery is ill-defined for multicast destinations, and there are no known mechanisms for performing this in a scalable manner. However, many applications including the MediaBoard generate a large number of small data packets. For example, each segment of a multi-segment scrawl is transmitted as a separate ADU. An application may be allowed to buffer and delay transmission until a MTU-sized packet-worth of data is available for transmission, even when the required bandwidth is available through the leaky bucket. Libsstp allows the application to control the mode of buffering and transmission to achieve this using the `delay_until_full_packet` flag, as shown below. Using the buffered mode may increase the application's perceived latency at the receiver.

```
int sstp_delay_until_full_packet(sstp_session_t sess, int flag);
int sstp_get_delay_until_full_packet(sstp_session_t sess);
```

An application uses these functions to request that ADUs be buffered until a full packet's worth of ADUs are available for transmission. This function returns the original state of the `delay_until_full_packet` flag before the call to the function. When the flag is reset to zero, any previously buffered packets are immediately sent out. Typically, an application will set the flag, invoke sstp_send a number of times, and reset the flag to its original value:

```
int save = sstp_delay_until_full_packet(sess, 1);
sstp_send(...);
sstp_send(...);
/* ... */
sstp_delay_until_full_packet(sess, save);
```

sstp_get_delay_until_full_packet retrieves the current value of the flag without changing any internal state.

7.1.2 Source Object

The SSTP session object is responsible for all transmissions and network-related functionality. Each session contains one or more SSTP source objects that represent data sources. For example, the MediaBoard application may have one source object that represents the local user's drawing operations and one source object each for every other sender in the session. The namespace is structured as a hierarchy of *nodes*, each of which has a path name as well as a unique descriptor returned to the application when it is created using sstp_calloc.

```
sstp_source_t sstp_create_source(sstp_session_t session,
                                 const char *label);

u_int32_t sstp_calloc(sstp_source_t source, unsigned int parent,
                      const unsigned char *node_name,
                      int name_len);
```

sstp_create_source creates a new source within session. label is an alphanumeric string supplied by the application from which a unique 64-bit internal source identifier is derived using MD5 [119] as a one-way hash function. The function returns an sstp_source_t handle for the source within the session. A session may have multiple sources, each with a different label. The internal identifier of each source is unique, time invariant and location independent. It depends only on the unique label provided by the application. The function returns NULL if a local source with the same label already exists, or if allocation has otherwise failed.

sstp_calloc allocates a SNAP node within the hierarchical namespace of source source, and returns a descriptor to the node within source's namespace. parent is the descriptor of the parent node of the newly created node, and node_name is a buffer containing the application-defined name or description of the newly created node. For example, in webcast, documents may be described by their respective URLs. Alternatively, applications may use the {attribute, value} convention to describe an ADU. name_len is the length in bytes of the description in the node_name buffer.

7.1.3 Data Path

An application uses the function sstp_send, shown in in the code below to send data. The data is named using the node descriptor returned from an earlier call to sstp_calloc. The send function also takes application-specific information, including an application-level timestamp and an ADU type.

```
unsigned int sstp_send(sstp_source_t source, unsigned int nid,
                       unsigned char *data, int len,
                       const sstp_adu_info *info);

typedef struct sstp_adu_info {
        unsigned char type;
        ntp64 timestamp;
};

typedef struct ntp64 {
        unsigned int upper;    /* more significant 32 bits */
        unsigned int lower;    /* less significant 32 bits */
} ntp64;
```

`sstp_send` attempts to send `len` bytes from the data buffer from node `nid` within the namespace of `source`. If info is not `NULL`, it points to an `sstp_adu_info` structure that is used to fill the appropriate fields of the SSTP ADU header. If this parameter is NULL, the library uses default zero values. If the `delay_until_packet_full` flag is set, the data is merely copied into a packet buffer, but not immediately transmitted. Transmission occurs when the current packet buffer has been filled to its maximum capacity (determined by the MTU). When the packet buffer is filled, it joins the tail of the SSTP transmission queue. All SSTP transmissions are rate-controlled using a leaky bucket with a configured rate. This rate is presently manually configured using `sstp_set_session_bandwidth`, but may also be used in conjunction with a multicast congestion control algorithm. `sstp_send` returns immediately the sequence number of the ADU within `nid`, and the data buffer must be freed by the caller. Application-specific ADU type information as well as a 64-bit NTP timestamp are also provided to SSTP and transported in the ADU headers.

7.1.4 Application Callbacks

Libsstp provides two important callbacks that notify the application of significant protocol events. `sstp_recv` notifies the application when data is available to the application and `sstp_should_recover` asks the application if the current loss needs to be repaired. When the transport "layer" receives a request for retransmission, the application is in turn requested for the relevant data through the callback `sstp_read_adu`. These functions are explained below.

```
void sstp_recv(sstp_source_t source, unsigned int nid,
               unsigned int seqno, const unsigned char *data,
               int len, const sstp_adu_info *info);


int sstp_should_recover(sstp_source_t source, unsigned int nid,
```

```
                          unsigned int ss, unsigned int es);

void sstp_read_adu(sstp_source_t source, unsigned int nid,
                   unsigned int seqno, unsigned char **data_ptr,
                   unsigned int *len_ptr, sstp_adu_info *info,
                   sstp_free_proc *free_proc_ptr);
```

`sstp_recv` is an application-defined handler invoked when a complete ADU is received by the `session.` `source` is the local incarnation of the original source of this data, nid, the ID of the node to which this ADU belongs, `seqno,` sequence number of this ADU within `nid.` The ADU occupies the first `len` bytes of the buffer data. Buffering transmitted ADUs is the responsibility of the application and ADUs are evicted from the libsstp buffers once they are handed to the application. Further access to data is obtained through the application using `sstp_read_adu`. The application may decide to discard or spool to disk some application data from its memory buffers depending on how much space it has available for buffering. If spooled to disk, `sstp_read_adu` provides an easy way to access it.

`sstp_read_adu` is an application-defined handler invoked when a repair request is received and data must be transmitted in response. source refers to source object corresponding to the *original source* of this data and nid, the ID of the node in the repair request. seqno is the sequence number of the requested ADU within nid. On return, the buffer pointed to by `data_ptr` contains the data and the length of the data is in `len_ptr.` The application also returns a pointer to a free function via `free_proc_ptr` to release the data buffers once the retransmission has occurred. In addition, the application can fill in the `sstp_adu_info` structure pointed to by info with appropriate values for the ADU header.

`sstp_should_recover` is an application-defined handler invoked when a loss is detected. `ss` and `es` define the range of lost ADUs within `nid` in the transmission from `source.` The function returns 0 if the loss is to be ignored, or 1 if recovery is required.

7.1.5 Event Handling

Libsstp is an event-based system with a single execution stream. Here, the application and protocol register interest in events — for example, the protocol may register interest by specifying a handler function for the arrival of packet on the control port. The event loop waits for events and when the specified event occurs, the corresponding event handler is invoked. Since event handlers are not preempted in this system, it is best suited for handlers that are relatively short-lived. The main drawback of using events in place of threads is that it does not achieve *true processor concurrency.* However, if end host performance is not a bottleneck, events provide a convenient alternative to threads as a means of structuring network applications.

7.2 Libsstp Applications

In this section, we describe some applications that have been developed using libsstp. The libsstp interface and internals have vastly improved based on the experience gained from developing real-world applications.

```
#include <sstp-event.h>
int  sstp_create_timer_handler(milliseconds, proc, clientData);
void sstp_delete_timer_handler(token);
```

`int milliseconds (in)`
How many milliseconds to wait before invoking proc.

`TimerProc *proc (in)`
Procedure to invoke after milliseconds have elapsed.

`void* clientData (in)`
Arbitrary one-word value to pass to proc.

`int token (in)`
Token for previously-created timer handler (the return value from some previous call to `create_timer_handler`).

```
void sstp_create_file_handler(fd, mask, proc, clientData);
void sstp_delete_file_handler(fd);
```

`int fd (in)`
Unix file descriptor for an open file, network socket or device.

`int mask (in)`
Conditions under which proc should be called: OR-ed combination of `READABLE`, `WRITABLE`, and `EXCEPTION`. May be set to 0 to temporarily disable a handler.

`FileProc *proc (in)`
Procedure to invoke whenever the file or device indicated by file meets the conditions specified by mask.

`void* clientData (in)`
Arbitrary one-word value to pass to `proc`.

Figure 7.2: The libsstp API to register and de-register events.

MediaBoard

MediaBoard is a distributed shared drawing tool intended for use in online collaborative settings in conjunction with audio and video applications [133]. It was inspired by the LBL whiteboard tool *wb* [63, 79], which was the original context in which the SRM timer algorithms were designed. While MediaBoard is similar to wb in its basic functionality, it improves it in two key aspects. First, in wb, the application and transport protocol were commingled in an inseparable manner. Second, the wb data representations were based on an obsolete graphics package that used the PostScript language, making it unsuitable for further extension and experimentation. The MediaBoard design separates the application functionality from the underlying transport protocol invoked using the well-defined libsstp API. MediaBoard was the first SSTP application and its design proved to be an invaluable research vehicle to test the concepts of application-level framing and selective reliability in a real context and provide critical feedback during the design of libsstp as well as its API. MediaBoard internally uses both bitmap and structured representations of data and relies on the Tk toolkit for rendering and manipulating graphics, In addition, because of its persistent data model, in which all drawing operations are stored (either in memory or on disk), the MediaBoard also allows the user to "time travel" by rewinding and playing back drawing operations. This allows the end user to view different snapshots during the evolution of the canvas through time. MediaBoard also attempts to enhance the feeling of "tele-presence", i.e, the feeling that all distributed users are present in the same room, by providing tool tips as well as the ability to follow an active user.

MediaBoard uses a 2-level namespace hierarchy in which the first level is used to represent each page in the drawing board and the subsequent level represents the drawing operation within the page. A later version of MediaBoard (version 2.0) takes further advantage of libsstp and its selective retransmission features. Here, pages currently in view are reliably recovered by the receiver to provide the user with enhanced interactivity. This tool is available for download along with the rest of the main MASH distribution at `http://www-mash.cs.berkeley.edu/mash/software/download.html`.

Infocaster

Libsstp has been used to develop a periodic information dissemination tool called *infocaster* [143]. The infocaster is used to disseminate stock quotes information by periodically transmitting to the infocaster channel. The stock server schedules different update periods for different stock quotes, depending on the level of trading activity. Infocaster clients only care for the most current quote information which they receive and display. They request a retransmission of lost data only if it is expected to reach the receiver *before* the subsequent update from the sender. The loss recovery algorithm in infocaster is receiver-driven and the clients uses measured application-level statistics such as the average information update period as well as network loss conditions to decide whether to schedule a retransmission request for the lost data. This allows infocaster clients to conserve bandwidth that would otherwise be used to retrieve a "stale" version of the data. Infocaster uses libsstp's multicast distribution mechanism and loss recovery machinery for efficient information dissemination. It achieves tunable receiver behavior through libsstp's application callback interface.

Reliable Multicast Proxies

One alternative to end-to-end congestion control and application adaptation for reliable multicast applications in highly heterogeneous environments is to use application-level gateways or proxies [18]. A Reliable Multicast Proxy (RMX) may be deployed when there is a large discontinuity in network conditions, for example, when handheld PDA devices are connected to the rest of the high speed Internet via low-bandwidth lossy wireless links. In this case, the wireless RMX participates in a global session and on behalf of the end hosts, but performs data format conversion to produce lower fidelity data in the appropriate format that the handheld clients are able to handle. SNAP is used in this context to reliably recover only portions of the namespace that the population of handheld clients is able to handle.

Distributed Archival

Another application of libsstp is for developing distributed control protocols used in the production of high quality archives of MBone conferences [123]. Here, a distributed archival system is deployed to record and store live MBone content. Depending on the location of the recording agents and the network conditions at the time of recording, each recording agent may only be able to capture a suboptimal version of each source's transmission. However, the archival quality is further improved by applying post-processing algorithms on the individual recordings to reconstruct a high quality version for the entire session. The distributed MBone archival system proposed in [123] uses libsstp to selectively repair missing portions during the reconstruction phase.

Light-weight Control Protocols

Libsstp has been used as a basis for developing a light-weight control protocol for the "Parallel Software-only Video Processing" system (PSVP) [100] that is used for titling and compositing (e.g., picture-in-picture) using compressed Internet video sources. PSVP exploits the temporal, spatial and functional parallelism inherent in video special effects processing to achieve real-time performance on a network of workstations (NOW) connected by a high-speed network. In order to orchestrate effects processing tasks among the host processors in such a parallel environment, PSVP uses an SSTP as the basis for a light-weight control protocol. Since PSVP processors are not pre-allocated statically, the control protocol must have the ability to address groups of processors without requiring prior knowledge of individual hosts within a group. libsstp provides this abstraction because it uses IP multicast. In addition, libsstp supports receiver-driven recovery on a per-message granularity, i.e., a receiver may choose to recover some messages and not others within a single session.

The PSVP dynamically parallelizes a video special effect into a hierarchy of tasks and maps the resulting parallel subtasks onto the available system resources. PSVP exploits temporal, spatial and functional parallelism inherent in the specific effect. The system comprises implementation agents, that carry out the parallel subtasks as well as control agents that are responsible for coordinating tasks at lower levels of the hierarchy, e.g., demultiplexing input streams and multiplexing the output streams once processing agents have signaled completion. The control protocol disseminates parameter information which is required to control the special effect to all the PSVP processors. Since agents at different levels of the hierarchy are interested in different portions of the control parameter namespace, SNAP is used to reliably retrieve parameters of interest.

7.3 Concluding Remarks

We described the libsstp toolkit, which is a user-level implementation of the SSTP protocol. Libsstp exposes a simple, yet powerful programming interface that allows applications to customize the semantics of reliable delivery on a per-message basis. Libsstp is event-based and its API includes generic event handling. We discussed the *session* and *source* objects that are two key abstractions within libsstp. We also discussed the data path through the toolkit and the application callback support which provide the necessary hooks for the application to intervene when significant network-level events occur.

Libsstp has been used to develop a wide range of applications — MediaBoard, which is a shared drawing tool; an information dissemination tool, called infocaster; to design control protocols for a parallelized software-based video special effects processing system; in reliable multicast proxies; and in a distributed archival system. Each of these applications has proved invaluable in refining the design of interfaces as well as the internal architecture of libsstp.

Chapter 8

Conclusions and Future Work

Though no one can go back and make a brand new start, anyone can start from now and make a brand new ending.

— Anonymous

We conclude this dissertation in this Chapter by suggesting some key directions for future research and presenting our conclusions. We also indicate where our protocol framework implementation and the applications developed using it are available online.

8.1 Future Directions

The work in this dissertation motivates some interesting and potentially fruitful areas for future work. Some of these are direct extensions derived from our work in this dissertation and are closely related to our soft state-based transport protocol framework. Other ideas focus on interesting new areas or novel applications of our framework and motivate research in significantly new directions.

8.1.1 Soft State Model for RSVP

In this work, we have developed a basic model for soft state-based communication and used it to model a number of end-to-end transport protocols. An interesting extension of this model is to RSVP [149, 16] reservation refresh messages between routers and the design of a refresh protocol that adapts to the observed loss rate. A recent proposal for "summary refresh" aims to reduce the amount of refresh messaging by organizing reservation state hierarchically [130]. A number of interesting questions emerge: how must bandwidth be allocated to the different levels of the hierarchy so as to maximize the consistency of reservation state? Is there an optimal hierarchy that would result in consuming the least refresh bandwidth based on the rates at which reservations change, or in other words, based on flow lifetimes. Perhaps each flow can be assigned to its "optimal" position in the hierarchy based on its lifetime characteristics.

8.1.2 Compact Namespace Representations

The SNAP protocol represents each node with a distinct identifier in the recursive descent procedure. For a large namespace tree, representing a node and its signature individually in the namespace update messages can be an overhead in terms of the number of messages. One possibility to overcome this overhead is to represent the "current" snapshot of the namespace in a compact form. An important requirement of this compact format is that is must reveal which subtrees at a given level are not synchronized between the sender and receiver. This problem may be posed as one of mapping an arbitrary tree (with arbitrary branch factor) created by the application onto a binary tree (a tree with branch factor of 2) of arbitrary height. Once this is achieved, a compact representation is possible by run-length encoding the resulting binary tree represented using in-order and pre-order traversals. The main advantage representing namespaces more compactly in this manner, is that less bandwidth needs to be allocated to namespace refreshes in a multicast transport session.

8.1.3 Content Peering

Content distribution networks (CDNs) have grown in popularity as a method to enhance WWW performance. A content distribution network is a network of service nodes, deployed throughout the Internet, that Web publishers can use to distribute their content on a subscription basis. A CDN is essentially an overlay network that relies on the underlying IP network and has geographically distributed service nodes that enable rapid, reliable retrieval from any end-user location. CDNs attempt to "push" content to the edges of the network, closer to end users, thereby reducing document download times and improving bandwidth usage between service provider networks. Global load balancing ensures that users are transparently routed to the "best" content source.

CDNs facilitate "content peering" agreements that allow different service nodes in a distribution network, perhaps managed by different parties, to exchange content, and providing better availability. Extensive content peering arrangements, especially between different CDN providers, call for a protocol that allows different providers to exchange information on what content is locally available. One way to achieve this is by using periodic announcements much like a network routing protocol that periodically exchanges host reachability information. A SNAP-like protocol for directory exchange can be applied to these scenarios to optimize the number of messages exchanged, and only propagate portions of the directory that have changed. This technique may be used to build an optimized protocol that reduces bandwidth overhead of the routing protocol. The degree of change dictates how rapidly content reachability information must be refreshed and how much bandwidth is required for maintaining consistency between the different routing databases.

Similar work has been performed in the context of service discovery in highly dynamic ad hoc wireless networks where services are mobile. In such environments, referring to the service using a host address is ineffective. The Intentional Naming System (INS) [1] proposes an integrated naming and resolution architecture and an application-level name-based routing protocol to locate mobile services.

8.1.4 Napster Overlay Networks

A related technology to CDNs is the recent Napster protocol for file sharing [91]. Napster allows users to share content via TCP using a simple publish-subscribe model involving the end hosts, but without requiring a central content server. Occasionally, a centralized napster server searches and constructs an index of all the available content. Other work in this area attempts to anonymize file sharing and indexing, so that indexing agents do not discover the identity of the content being published. Only the clients that send and receive the data are aware of the identity of the content exchanged. One recent scheme uses a well-known hash function to produce a message digest or signature of the content name and uses the hash information to construct the index. Subscribers use the signature of the content name to locate content. The drawback of this scheme is that one-way hash functions are irreversible and partial matches are not possible when matching the signatures rather than the original strings. However, building an intelligent indexing agent that takes partial matches into account is a significant challenge. SNAP and soft state-based transport can play an important role in efficient index construction and updates.

Napsters have recently shown the need to perform data type or application-specific bandwidth allocation and metering within the network. Recent analyses of traffic from a campus network has shown that napster traffic can be a bandwidth hog [90], starving out lower bandwidth traffic such as electronic mail and WWW connections. For example, a network administrator may enforce a class-based policy that limits napster traffic to 10% of the outgoing link capacity of an organization connected via a 45 Mbit/s DS3 link to the rest of the Internet, leaving the remaining 60% for other types of shorter flows.

Another critical component in the napster model for "grassroots" multicast is a more robust and efficient topology formation scheme that will also allow real-time content delivery, besides just allowing client-to-client file transfers. This suggests that napster networks are a special case of content delivery overlays, and similar techniques in robust topology construction can be applied here too.

Besides being a new application for many-to-many communication without requiring network-layer multicast, napster has generated much controversy in the public press for making it easy for users to distribute content without appropriate legal authorization and difficult for law enforcement agencies to detect such misuse.

8.1.5 Hierarchical Session Directory

The session directory tool *sdr* uses a linear table of entries and announces each one periodically. Since there is no structure to its announcement database, the time to receive a given announcement grows linearly with the number of entries in the database. Previous research [131] proposes a split architecture for the session announcement protocol (SAP) to improve its performance and simultaneously support announcements for layered media sessions. In this architecture, multiple protocol proxy agents are used — global SAP agents operate at a low frequency, and announce the availability of sessions to a larger scope. Local caching proxy agents receive global announcements and re-broadcast them at a higher frequency, and consume more bandwidth but within a restricted scope. Scoping is configured administratively in border multicast routers by means of forwarding and blocking rules for specific ranges of multicast addresses. This scheme can be enhanced using a SNAP-like protocol that categorizes announcements hierarchically and uses an iterative protocol to transmit updates. The iterative protocol allows the receiver to fetch announcements from only those categories that are relevant to it, thereby dedicating more bandwidth to announcements of interest.

8.1.6 Multicast-based Software Updates

A pressing problem today in software engineering is software maintenance — the problems associated with version control and software updates. Some researchers have proposed using multicast-based distribution mechanisms to deliver software updates to large groups of users. Such "self updating" software must perform version management automatically, without human intervention and in order to do so, must first solve the problem of naming objects and modules and representing their inter-dependencies. Once such a naming and identification scheme is available, a SNAP-like protocol can be employed to deliver differential updates of updated modules. Both server and client authentication are required to prevent intrusions and protect the client systems from malicious software as well as ensure that only the eligible clients are allowed to participate, for example, if it is accompanied by a subscription and payment for the software.

8.2 Availability

All of the software and protocol implementations developed in this dissertation are available on-line in source and binary code form.

Our stand-alone implementation of the soft state transport protocol *libsstp* is available from:

`http://www-mash.cs.berkeley.edu/mash/software/srm2.0/`

Our extensions to *ns-2* to perform large-scale simulations to study the asymptotic scaling behavior of timer-based recovery is available from the *ns-2* [83] distribution, available from:

`http://www-mash.cs.berkeley.edu/ns/`

All the applications written have been written within the context of the MASH programming environment and are available from the MASH web page:

`http://www-mash.cs.berkeley.edu/mash/`

8.3 Summary

In this dissertation, we proposed a new framework for soft state-based multicast transport for interactive applications. We have presented a formal model for "soft state" as an end-to-end construct that enables loose state synchronization between sender and receivers. We have analyzed this model to study the performance of our protocol and its variants. Our soft state-based transport treats protocol control state at the end points as "soft" by not requiring that they be perfectly consistent at all instants. This allows us to avoid tight sender-receiver synchronization, as in TCP-like instantaneous receiver acknowledgements. Our soft state-based transport protocol provides a *relaxed reliability*, instead of TCP-like deterministic reliability.

To accommodate heterogeneity among receivers and network paths, we allow receivers to tailor the semantics of reliability. Hence, a receiver incapable of or uninterested in processing portions of the data stream may refrain from receiving it reliably. We do not rely on the transport-level sequence space, but rather, use application-specific namespaces to express receiver preferences while requesting retransmissions. This application-level namespace is exposed to the transport protocol and is used by the receiver to selectively retrieve specific data items. The use of such a common "vocabulary" to describe data puts the application in control of loss recovery. Even though the future of ubiquitous wide-area multicast routing extensions is uncertain at this time, our schemes only require a multi-point distribution service, and are orthogonal to the exact details of the service.

Finally, since many new data types including certain image formats can be processed and rendered out of order at the receiver, we do not enforce a TCP-like delivery order on the data stream. Instead, we provide out-of-order delivery to the receiving application and demonstrate its benefits for image delivery. This specific technique is also applicable to unicast transmission and we design and implement a JPEG image transmission protocol for use with HTTP.

Our transport protocol is layered on top of UDP [108] in the protocol stack, and we have implemented it as a user-level library called *libsstp*, a library for soft state-based reliable transport. We also present probabilistic analyses of the performance of our protocol in terms of the performance of the basic algorithms for loss recovery, using "slotting and damping," as well as the tradeoffs involving consistency and bandwidth consumption.

Bibliography

[1] William Adjie-Winoto, Elliot Schwatz, Hari Balakrishnan, and Jeremy Lilley. The design and implementation of an intentional naming system . In *Proceedings of the Seventeenth Symposium on Operating Systems Principles*, December 1999.

[2] K. Almeroth and M. Ammar. Multicast Group Behavior in the Internet's Multicast Backbone (MBone). *IEEE Communications Magazine*, June 1997.

[3] Elan Amir, Steven McCanne, and Randy Katz. Receiver-driven Bandwidth Adaptation for Light-weight Sessions. In *Proceedings of ACM Multimedia '97*. ACM, November 1997.

[4] Elan Amir, Steven McCanne, and Randy Katz. An Active Service Framework and its Application to Real-time Multimedia Transcoding. In *Proceedings of SIGCOMM 1998*, Vancouver, Canada, Sep 1998. ACM.

[5] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, and R.H. Katz. TCP Behavior of a Busy Web Server: Analysis and Improvements. In *Proc. IEEE INFOCOM*, March 1998.

[6] H. Balakrishnan, H. S. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proceedings of SIGCOMM 1999*, Cambridge, MA, Sep 1999. ACM.

[7] H. Balakrishnan and S. Seshan. *The Congestion Manager*. Internet Engineering Task Force, Nov 1999. Internet Draft draft-balakrishnan-cm-01.txt (`http://www.ietf.org/internet-drafts/draft-balakrishnan-cm-01.txt`). Work in progress, expires April 2000.

[8] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing. In *Proceedings of SIGCOMM '93*, pages 85–95, San Francisco, CA, September 1993. ACM.

[9] Forest Baskett, Mani Chandy, Richard Muntz, and Fernando Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *Journal of the Association for Computing Machinery*, 22(2):248–260, 1975.

[10] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol–HTTP/1.0*. Internet Engineering Task Force, May 1996. RFC 1945.

[11] K. Birman. A Response to Cheriton and Skeen's Criticism of Causal and Totally Ordered Communication. In *Operating System Review*, volume 28, pages 11–21, January 1994.

[12] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal Multicast. Technical Report TR98-1683, Cornell University, Ithaca, NY, May 1998.

[13] Kenneth Birman, Andre Chiper, and Pat Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Transactions on Computer Systems*, 9(3):272–314, August 1991.

[14] Jean-Chrysostome Bolot and Thierry Turletti. A Rate Control Mechanism for Packet Video in the Internet. In *Proceedings IEEE Infocom '94*, Toronto, Canada, June 1994. ACM.

[15] Jean-Chrysostome Bolot, Thierry Turletti, and Ian Wakeman. Scalable Feedback Control for Multicast Video Distribution in the Internet. In *Proceedings of SIGCOMM '94*, University College London, London, U.K., September 1994. ACM.

[16] R. Braden, L. Zhang, D. Estrin, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP) – version 1 function specification, July 1995. Internet Draft expires 1/96.

[17] K. Mani Chandy, Adam Rifkin, and Eve Schooler. Using Announce-Listen with Global Events to Develop Distributed Control Systems. *Concurrency: Practice and Experience*, pages 1021–1027, 1998.

[18] Yatin Chawathe, Steven McCanne, and Eric Brewer. RMX: Reliable Multicast in Heterogeneous Networks. In *Proc. IEEE INFOCOM*, March 2000.

[19] D. Cheriton and D. Skeen. Understanding the Limitations of Causally and Totally Ordered Communication Systems. *Proc. 14th ACM Symposium on Operating Systems Principles*, pages 44–57, Dec 1993.

[20] D. Cheriton and D. Skeen. Comments on the Responses by Birman, van Renesse and Cooper. *Operating Systems Review*, page 32, January 1994.

[21] G. Chesson. XTP/Protocol Engine Design. In *Proceedings of the IFIP WG6.1/6.4 Workshop*, Rüschlikon, May 1989.

[22] David D. Clark. The Design Philosophy of the DARPA Internet Protocols. In *Proceedings of SIGCOMM '88*, Stanford, CA, August 1988. ACM.

[23] David D. Clark and David L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. In *Proceedings of SIGCOMM '90*, Philadelphia, PA, September 1990. ACM.

[24] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley and Sons, Inc., 1991.

[25] J. Crowcroft, Z. Wang, A. Ghosh, and C. Diot. RMFP: A Reliable Multicast Framing Protocol, March 1997. Internet Draft (RFC pending).

[26] Yogen Dalal and Robert Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, December 1978.

[27] John Danskin, Geoffrey Davis, and Xiyong Song. Fast Lossy Internet Image Transmission. In *Proceedings of ACM Multimedia '95*. ACM, November 1995.

[28] Brian DeCleene et al. RMF: A Transport Protocol Framework for Reliable Multicast Applications, November 1999. Draft specification.

[29] Stephen Deering, Deborah Estrin, Dino Farinacci, and Van Jacobson. An Architecture for Wide-Area Multicast Routing. In *Proceedings of SIGCOMM '94*, University College London, London, U.K., September 1994. ACM.

[30] Stephen E. Deering. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, December 1991.

[31] Steven Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ahmed Helmy, David Meyer, and Liming Wei. Protocol Independent Multicast version 2 Dense Mode Specification, August 1997. Internet Draft.

[32] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Proceedings of SIGCOMM '89*. ACM, September 1989.

[33] T. Faber, J. Touch, and W. Yue. The TIME-WAIT state in TCP and its Effect on Busy Servers. In *Proc. INFOCOM '99*, 1999.

[34] Aaron Falk and Vern Paxson. Minutes of the "RUTS" IETF BOF, December 1998. ftp://ftp.ee.lbl.gov/ietf/ruts-98-minutes.

[35] Dino Farinacci, Yakov Rekhter, Peter Lothberg, Hank Kilmer, and Jeremy Hall. Multicast Source Discovery Protocol (MSDP), June 1998. Internet Draft.

[36] W. Fenner. *Internet Group Management Protocol, Version 2*, Nov 1997. RFC-2236.

[37] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, Jan 1997. RFC-2068.

[38] Definition of the Flexible Image Transport System (FITS). `http://fits.gsfc.nasa.gov/documents/nost_1.2/fits_standard.html`, 1998.

[39] Sally Floyd and Van Jacobson. Link-Sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[40] Sally Floyd, Van Jacobson, Steven McCanne, Ching-Gung Liu, and Lixia Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *Proceedings of SIGCOMM '95*, Boston, MA, September 1995. ACM.

[41] Ron Frederick. *Network Video (nv)*. Xerox Palo Alto Research Center. ftp://ftp.parc.xerox.com/net-research.

[42] Jim Gemmell, Eve Schooler, and Jim Gray. Fcast: Scalable Multicast File Distribution: Caching and Parameters Optimizations. Technical Report MSR-TR-99-14, Microsoft Bay Area Research Center, San Francisco, CA, June 1999.

[43] J. Gettys. MUX protocol specification, WD-MUX-961023. `http://www.w3.org/pub/WWW/Protocols/MUX/WD-mux-961023.html`, 1996.

[44] Graphics Interchange Format (SM), Version 89a. `ftp://ftp.ncsa.uiuc.edu/misc/file.formats/graphics.formats/gif89a.doc`, 1990.

[45] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, second edition, 1994.

[46] S. Gribble and E. Brewer. System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace. In *Proc. 1997 Usenix Symposium on Internet Technologies and Systems*, December 1997.

[47] Rajarshi Gupta, Mike Chen, Steven McCanne, and Jean Walrand. A Receiver-Driven Transport Protocol for the Web. In *Proc. INFORMS 2000 Telecommunications Conference*, March 2000.

[48] R. Han and D. G. Messerschmitt. Asymptotically Reliable Transport of Multimedia/Graphics Over Wireless Channels. In *Proc. SPIE Multimedia Computing and Networking*, January 1996.

[49] Mark Handley. *SAP: Session Announcement Protocol*. Internet Draft, Nov 19, 1996.

[50] Mark Handley. Session Directories and Internet Multicast Address Allocation. In *Proceedings of SIGCOMM 1998*, Vancouver, Canada, Sep 1998. ACM.

[51] Mark Handley and Jon Crowcroft. Network Text Editor (NTE): A Scalable Shared Text Editor for the MBone. In *Proceedings of SIGCOMM 1997*, Cannes, France, Sep 1997. ACM.

[52] Mark Handley and Van Jacobson. *SDP: Session Directory Protocol*. Internet Draft, Mar 26, 1997.

[53] Mark Handley and Van Jacobson. *sdr — A Multicast Session Directory*. University College London.

[54] Vicky Hardman, Peter Kirstein, et al. *Robust Audio Tool*. University College London. http://www-mice.cs.ucl.ac.uk/multimedia/software/.

[55] C. Hedrick. *Routing Information Protocol*. Rutgers University, June 1988. RFC-1058.

[56] Andrew T. Heybey. Video Coding and the Application Level Framing Protocol Architecture. Technical Report TR 542, MIT LCS, Cambridge, MA, June 1992.

[57] Hugh Holbrook and David Cheriton. IP Multicast Channels: EXPRESS Support for Large-scale Single-source Applications. In *Proceedings of SIGCOMM '99*, Cambridge, MA, September 1999. ACM.

[58] Hugh Holbrook, Sandeep Singhal, and David Cheriton. Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation. In *Proceedings of SIGCOMM '95*, Boston, MA, September 1995. ACM.

[59] IANA-assigned MIME Types. `ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/media-types`.

[60] V. Jacobson, R. Braden, and D. Borman. *TCP Extensions for High Performance*. Internet Engineering Task Force, May 1992. RFC 1323.

[61] Van Jacobson. *Session Directory*. Lawrence Berkeley Laboratory. ftp://ftp.ee.lbl.gov/conferencing/sd.

[62] Van Jacobson. Congestion Avoidance and Control. In *Proceedings of SIGCOMM '88*, Stanford, CA, August 1988.

[63] Van Jacobson and Steven McCanne. *LBL Whiteboard*. Lawrence Berkeley Laboratory ftp://ftp.ee.lbl.gov/conferencing/wb.

[64] Van Jacobson and Steven McCanne. *Visual Audio Tool*. Lawrence Berkeley Laboratory. ftp://ftp.ee.lbl.gov/conferencing/vat/.

[65] Raj Jain. Congestion Control in Computer Networks: Issues and Trends. *IEEE Network Magazine*, pages 24–30, May 1990.

[66] Raj Jain, K.K. Ramakrishnan, and Dah-Ming Chiu. Congestion Avoidance in Computer Networks With a Connectionless Network Layer. Technical Report DEC-TR-506, Digital Equipment Corporation, August 1987.

[67] JPEG2000 Links. `http://www.jpeg.org/JPEG2000.htm`.

[68] Miriam Kadansky and Dah-Ming Chiu. Tree-based reliable multicast (tram), January 2000. Internet Draft expires 7/2000.

[69] S. K. Kasera, J. F. Kurose, and D. F. Towsley. Scalable Reliable Multicast Using Multiple Multicast Groups. In *Proceedings of ACM SIGMETRICS Conference on Measurement & Modeling of Computer Systems*, June 1997.

[70] J. C. Kent and J. C. Mogul. Fragmentation considered harmful. In *Proc. ACM SIGCOMM*, October 1987.

[71] Satish Kumar, Pavlin Radoslavav, David Thaler, Cengiz Alaettinoglu, Deborah Estrin, and Mark Handley. The MASC/BGMP Architecture for Inter-domain Multicast Routing. In *Proceedings of SIGCOMM 1998*, Vancouver, Canada, Sep 1998. ACM.

[72] A. Lempel and J. Ziv. A universal algorithm for sequential data compression. *IEEE Trans. on Inf. Theory*, 23(3):337 – 343, 1977.

[73] Brian Neal Levine, David B. Lavo, and J.J. Garcia-Luna-Aceves. The Case For Reliable Concurrent Multicasting Using Shared Ack Trees. In *Proceedings of ACM Multimedia '96*, Boston, MA, November 1996. ACM.

[74] John C. Lin and Sanjoy Paul. RMTP: A Reliable Multicast Transport Protocol. In *Proceedings IEEE Infocom '96*, pages 1414–1424, San Francisco, CA, March 1996.

[75] Ching-Gung Liu, Deborah Estrin, Scott Shenker, and Lixia Zhang. Local Recovery in SRM. *Submitted to IEEE Transactions on Networking*, 1998.

[76] Ching-Gung Liu, Deborah Estrin, Scott Shenker, and Lixia Zhang. Recovery Timer Adaptation in SRM. *Submitted to IEEE Transactions on Networking*, 1998.

[77] Jeffrey Lo and K. Taniguchi. IP Network Address (and Port) Translation, June 1998. Internet Draft expires 6/99.

[78] K. Lougheed and Y. Rekhter. *A Border Gateway Protocol (BGP)*. Cisco Systems and T. J. Watson Research Center, IBM Corp., June 1989. RFC-1105.

[79] Steven McCanne. A Distributed Whiteboard for Network Conferencing. Unpublished manuscript, May 1992.

[80] Steven McCanne. *Receiver-driven Layered Multicast*. PhD thesis, University of California, Berkeley, December 1996.

[81] Steven McCanne et al. UCB/LBNL/VINT Network Simulator - ns (version 2). http://www-mash.cs.berkeley.edu/ns/.

[82] Steven McCanne et al. Towards a Common Infrastructure for Multimedia-Networking Middleware. In *Proceedings of the Seventh International Workshop on Network and OS Support for Digital Audio and Video*, St. Louis, CA, May 1997. ACM.

[83] Steven McCanne and Sally Floyd. *The LBNL Network Simulator*. University of California, Berkeley. http://www-mash.cs.berkeley.edu/ns/.

[84] Steven McCanne and Van Jacobson. *vic: video conference*. Lawrence Berkeley Laboratory and University of California, Berkeley. ftp://ftp.ee.lbl.gov/conferencing/vic.

[85] Steven McCanne and Van Jacobson. *vic:* A Flexible Framework for Packet Video. In *Proceedings of ACM Multimedia '95*. ACM, November 1995.

[86] J. McQuillan et al. A New Routing Algorithm for the ARPANET. *IEEE Transcations on Networking*, May 1980.

[87] David Meyer. *Glop Bit Usage*. Cisco Systems, 1999. draft-ietf-mboned-glop-bits-00.txt.

[88] P. Mockapetris. *Domain Names – Implementation and Specification*. SRI International, Menlo Park, CA, November 1987. RFC-1035.

[89] J. C. Mogul and S. E. Deering. *Path MTU Discovery*. SRI International, Menlo Park, CA, April 1990. RFC-1191.

[90] Napster Statistics from the University of Wisconsin. `http://wwwstats.net.wisc.edu/`.

[91] Napster.com. `http://www.napster.com/`.

[92] Netsizer Internet Growth Reports. `http://www.netsizer.com/`.

[93] J. Nonnenmacher and E. W. Biersack. Optimal Multicast Feedback. *IEEE Infocom*, 1998.

[94] Masataka Ohta and Jon Crowcroft. Static Multicast, June 1999. Internet Draft.

[95] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.

[96] V. Padmanabhan. *Addressing the Challenges of Web Data Transport*. PhD thesis, Univ. of California, Berkeley, Sep 1998.

[97] V. N. Padmanabhan and J. C. Mogul. Improving HTTP Latency. In *Proc. Second International WWW Conference*, October 1994.

[98] C. Papadopoulos, G. Parulkar, and G. Varghese. An Error Control Scheme for Large-Scale Multicast Applications. In *Proceedings IEEE Infocom '98*, San Francisco, CA, 1998.

[99] C. Partridge and R. M. Hinden. *Version 2 of the Reliable Data Protocol (RDP)*. Internet Engineering Task Force, Apr 1990. RFC 1151.

[100] Ketan Patel and Lawrence A. Rowe. A Multicast Control Scheme For Parallel Software-only Video Effects Processing. In *Proceedings of ACM Multimedia '99*. ACM, August 1999.

[101] V. Paxson. End-to-End Routing Behavior in the Internet. In *Proc. ACM SIGCOMM '96*, August 1996.

[102] V. Paxson. Automated Packet Trace Analysis of TCP Implementations. In *Proc. ACM SIGCOMM '97*, September 1997.

[103] V. Paxson. End-to-End Internet Packet Dynamics. In *Proc. ACM SIGCOMM '97*, September 1997.

[104] William B. Pennebaker and Joan L. Mitchell. *JPEG Still Image Data Compression Standard*. Van Nostrand Reinhold, 1993.

[105] Radia Perlman, Jon Crowcroft, Tony Ballardie, and Cheng-Yin Lee. A Design for Simple Low Overhead Multicast, December 1998. Internet Draft (work in progress).

[106] Sridhar Pingali, Don Towsley, and James F. Kurose. A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols. In *Proceedings of SIGMETRICS '94*, Santa Clara, CA, May 1994.

[107] PointCast Inc. *PointCast Home Page*. http://www.pointcast.com.

[108] J. B. Postel. *User Datagram Protocol*. International Sciences Institue, CA, August 1980. RFC-768.

[109] J. B. Postel. *Transmission Control Protocol*. SRI International, Menlo Park, CA, August 1989. RFC-793.

[110] J. B. Postel and J. Reynolds. *File Transfer Protocol (FTP)*. Internet Engineering Task Force, Oct 1985. RFC 959.

[111] Suchitra Raman. Design and Analysis of a Framework for Reliable Multicast. UCB CS Masters Thesis, May 1998.

[112] Suchitra Raman and Steven McCanne. Generalized Data Naming and Scalable State Announcements for Reliable Multicast. Technical report, University of California, Berkeley, CA, June 1997.

[113] Suchitra Raman and Steven McCanne. Scalable Data Naming for Application Level Framing in Reliable Multicast. In *Proceedings of ACM Multimedia '98*, Bristol, UK, September 1998. ACM.

[114] Suchitra Raman and Steven McCanne. A Model, Analysis and Protocol Framework for Soft State-based Communication. In *Proceedings of ACM SIGCOMM '99*, Cambridge, MA, September 1999. ACM.

[115] Suchitra Raman, Steven McCanne, and Scott Shenker. Asymptotic Behavior of Global Recovery in SRM. In *Proceedings of ACM SIGMETRICS '98*, Madison, WI, June 1998. ACM.

[116] RealNetworks, Inc. *RealPlayer*. http://www.real.com/.

[117] Reza Rejaie, Mark Handley, and Deborah Estrin. RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. *IEEE Infocom*, 1999.

[118] Antony Richards et al. The Application of ITP/ALF to Configurable Protocols. In *Proc. First International Workshop on High Performance Protocol Architectures (HIPPARCH '94)*, December 1994.

[119] R. Rivest. *The MD5 Message-Digest Algorithm*. MIT Laboratory for Computer Science and RSA Data Security, Inc., 1992. RFC-1321.

[120] P. Salama, N. B. Shroff, and E. J. Delp. *Error Concealment in Encoded Video Streams*. Kluwer Academic Publishers, 1998. Book Chapter in "Signal Recovery Techniques for Image and Video Compression and Transmission", edited by N. P. Galatsanos and A. K. Katsaggelos.

[121] Khalid Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 1996.

[122] Eve M. Schooler. A multicast user directory service for synchronous rendezvous. Computer science department, California Institute of Technology, September 1996.

[123] Angela Schuett, Randy Katz, and Steven McCanne. A Distributed Recording System for High Quality MBone Archives. In *Proc. First International Workshop on Networked Group Communication,*, November 1999.

[124] Henning Schulzrinne, Steve Casner, Ron Frederick, and Van Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. Internet Engineering Task Force, Audio-Video Transport Working Group, November 1991. Internet Draft expires 3/1/96.

[125] Henning Schulzrinne, Steve Casner, Ron Frederick, and Van Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. Internet Engineering Task Force, Audio-Video Transport Working Group, January 1996. RFC-1889.

[126] Puneet Sharma, Deborah Estrin, Sally Floyd, and Van Jacobson. Scalable Timers for Soft State Protocols. In *Proceedings IEEE Infocom '97*, Kobe, Japan, 1997.

[127] Tony Speakman et al. Pragmatic Good Multicast (PGM) Transport Protocol Specification, June 1999. Internet Draft (RFC pending).

[128] W. Richard Stevens. *TCP/IP Illustrated, Volume 1 – The Protocols*. Addison-Wesley, first edition, December 1994.

[129] I. Stoica, H. Zhang, and T. S. E. Ng. A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service. In *Proceedings of SIGCOMM 1997*, Cannes, France, Sep 1997. ACM.

[130] George Swallow. RSVP Hierarchical Summary Refresh, October 1999. Internet Draft.

[131] Andrew Swan, Steven McCanne, and Larry Rowe. Layered Transmission and Caching for the Multicast Session Directory Service. In *Proceedings of ACM Multimedia '98*, Bristol, UK, September 1998. ACM.

[132] J. Touch. *TCP Control Block Interdependence*. Internet Engineering Task Force, April 1997. RFC 2140.

[133] Teck-Lee Tung. MediaBoard: A Shared Whiteboard Application for the MBone. UCB CS Masters Thesis, February 1998.

[134] Teck-Lee Tung and Suchitra Raman. A Distributed MediaBoard Using the Scalable, Reliable Multicast Toolkit. UCB CS 262 Project Report, December 1996.

[135] Thierry Turletti. *INRIA Video Conferencing System (ivs)*. Institut National de Recherche en Informatique et an Automatique. http://www.inria.fr/rodeo/ivs.html.

[136] Thierry Turletti and Jean-Chrysostome Bolot. Issues with Multicast Video Distribution in Heterogeneous Packet Networks. In *Proceedings of the Sixth International Workshop on Packet Video*, Portland, OR, September 1994.

[137] C. J. Turner and L. L. Peterson. Image transfer: an end-to-end design. In *Proc. ACM SIGCOMM*, August 1992.

[138] D. Velten, R. Hinden, and J. Sax. *Reliable Data Protocol*. Internet Engineering Task Force, July 1984. RFC 908.

[139] Carl A. Waldspurger and William E. Weihl. Lottery Scheduling: Flexible Proportional-Share Resource Management. In *First Symposium on Operating Systems Design and Implementation (OSDI)*, pages 1–11. USENIX Association, 1995.

[140] Carl A. Waldspurger and William E. Weihl. Stride Scheduling: Deterministic Proportional-Share Resource. Technical Report MIT/LCS/TM-528, MIT Laboratory for Computer Science, Cambridge, MA, June 1995.

[141] Gregory K. Wallace. The JPEG sill picture compression standard. *Communications of the ACM*, 34(4):31–44, April 1991.

[142] Terrence A. Welch. A Technique for High Performance Data Compression. *IEEE Computer*, 17(6):8–19, 1984.

[143] Tina Wong, Thomas Henderson, Suchitra Raman, Adam Costello, and Randy Katz. Policy-Based Tunable Reliable Multicast for Periodic Information Dissemination. In *Proceedings of Workshop on Satellite Based Information Services*, Dallas, TX, October 1998.

[144] World Wide Web Consortium. `http://www.w3.org/`.

[145] Kristin Wright. MASHCast: Applying SRM Middleware to Webcast. Presentation at the Winter '98 MASH Retreat.

[146] Message Multiplexing (memux) Charter. http://www.w3.org/Protocols/HTTP-NG/1999/02/mux-Charter-222.html, 1999.

[147] Maya Yajnik, Jim Kurose, and Don Towsley. Packet Loss Correlation in the MBone Multicast Network. *IEEE Global Internet Conference*, 1996.

[148] R. Yavatkar, J. Griffioen, and M. Sudan. A Reliable Dissemination Protocol for Interactive Collaborative Applications. In *Proceedings of ACM Multimedia '95*, San Francisco, CA, November 1995. ACM.

[149] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network Magazine*, pages 8–18, September 1993.