

An Active Protocol Architecture for Collaborative Media Distribution

by
Dimitrios Christos Vyzovitis

MSc, Advanced Computing
Imperial College of Science, Technology, and Medicine
University of London, UK, 2000

Diploma, Electrical and Computer Engineering
Aristotle University of Thessaloniki, Greece, 1999

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of
Master of Science in Media Arts and Sciences
at the
Massachusetts Institute of Technology
June 2002

© 2002 Massachusetts Institute of Technology. All Rights Reserved

Signature of Author

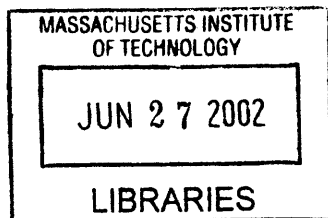
Program in Media Arts and Sciences
May 20, 2002

Certified by

Dr. Andrew Lippman
Senior Research Scientist
Program in Media Arts and Sciences
Thesis Advisor

Accepted by

Dr. Andrew Lippman
Chairperson
Departmental Committee in Graduate Studies
Program in Media Arts and Sciences



ROTCH



An Active Protocol Architecture for Collaborative Media Distribution

by
Dimitrios Christos Vyzovitis

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on May 20, 2002
in partial fulfillment of the requirements for the degree of
Master of Science in Media Arts and Sciences
at the
Massachusetts Institute of Technology

Abstract

This thesis embarks on distributing the distribution for real-time media, by developing a decentralised programmable protocol architecture.

The core of the architecture is an adaptive application-level protocol which allows collaborative multicasting of real-time streams. The protocol provides transparent semantics for loosely coupled multipoint interactions. It allows aggregation and interleaving of data fetched simultaneously from diverse machines and supports the location and coordination of named data among peer nodes without additional knowledge of network topology. The dynamic stream aggregation scheme employed by the protocol solves the problem of network asymmetry that plagues residential broadband networks. In addition, the stateless nature of the protocol allows for fast fail-over and adaptation to departure of source nodes from the network, mitigating the reliability problems of end-user machines.

We present and evaluate the algorithms employed by our protocol architecture and propose an economic model that can be used in real-world applications of peer-to-peer media distribution. With the combination of an adaptive collaborative protocol core and a reasonable economic model, we deliver an architecture that enables flexible and scalable real-time media distribution in a completely decentralised, serverless fashion.

Thesis Advisor: Dr. Andrew Lippman

Title: Senior Research Scientist, Program in Media Arts and Sciences

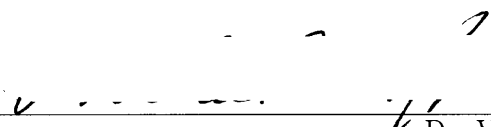
This work was supported by the MIT Media Laboratory Digital Life Consortium and Intel.

An Active Protocol Architecture for Collaborative Media Distribution

by
Dimitrios Christos Vyzovitis

The following people served as readers to this thesis:

Thesis Reader


Dr. V. Michael Bove Jr.
Principal Research Scientist
MIT Media Laboratory

Thesis Reader

Dr. Hari Balakrishnan
Assistant Professor of Electrical Engineering and Computer Science
MIT Laboratory for Computer Science

Acknowledgements

First and foremost, I would like to thank Andy Lippman, my advisor, for his patience and guidance. His perserverance in identifying the essence of problems and how they relate to real life, and his patience with my infinite last minute experimentation disposition, have been instrumental in the development of this thesis. Similarly, I would like to thank my readers, Mike Bove and Hari Balakrishnan, for taking the time to read and comment on the thesis.

Jeremy Silber has helped in the implementation of very early versions of the DRMTP protocol and contributed in the discussion of economics for collaborative media distribution.

Ali Rahimi has inadvertently played a role in this thesis, as many of my ideas about a distributed protocol architecture have evolved from our post-midnight and 6 am breakfast discussions.

Finally, I would like to thank Tom Gardos and David Reed for the discussions we have had over time. These discussions, on distributed protocols, peer-to-peer systems and other themes, have certainly contributed in the ideas presented in this thesis.

Contents

1	Introduction	19
1.1	Technical Hurdles	20
1.2	Socioeconomic Hurdles	21
1.3	Contribution of this Work	22
1.4	Thesis Overview	24
2	Towards Collaborative Media Distribution	25
2.1	A System Model	26
2.2	Peer-to-Peer Systems	30
2.2.1	Napster	30
2.2.2	Gnutella	34
2.2.3	Advanced Peer-to-Peer Lookup Algorithms	35
2.3	The Promise of Multicast	37
2.3.1	IP Multicast	37
2.3.2	Multicast Transport	38
2.3.3	Scalable Reliable Multicast	40
2.3.4	Multicast Congestion Control	41
2.4	Programmable Networks	45
2.4.1	Session Discovery and Configuration	45
2.4.2	Active Networks and Active Service Frameworks	46
2.5	Concluding Remarks	47
3	A Protocol Architecture for Collaborative Media Distribution	49
3.1	Building an Active Protocol Architecture	49

3.1.1	Protocol Design Considerations	49
3.1.2	Architecture Outline	51
3.2	Session Structure	52
3.3	Implosion Control	55
3.4	DRMTP: The Distributed Real-time Transport Protocol	56
3.4.1	Stream Control	56
3.4.2	Stream Establishment	59
3.4.3	Detection of Error Conditions	63
3.4.4	Error Correction	67
3.4.5	Congestion Control	68
3.4.6	Source and Primary Controller Failure Recovery	71
3.5	SDCP: The Session Discovery and Configuration Protocol	72
3.5.1	File Discovery	72
3.5.2	Address Discovery and Allocation, and Source Configuration	73
3.6	Concluding Remarks	74
4	Analysis and Evaluation	75
4.1	Scalability and Latency Bounds	75
4.1.1	Number of Feedback Messages	76
4.1.2	Feedback Latency	77
4.1.3	Choosing the Parameters of the Distribution	78
4.2	Experimental Evaluation in a Network Testbed	81
4.2.1	The Network Testbed	81
4.2.2	Traffic Localisation	83
4.2.3	Source Failure Recovery	83
4.2.4	Scalability in the Presence of Network Effects	87
4.3	Protocol Behavior in Boundary Conditions	89
4.3.1	Source Upstream Congestion Control	89
4.3.2	Protocol Fairness	90
4.4	Concluding Remarks	90

5	An Economic Model for Collaborative Distribution of Media	95
5.1	Licensing	96
5.2	Affinity Points	97
5.3	Payment	99
5.4	Concluding Remarks	100
6	Conclusion	103
A	Protocol Specification	117
A.1	Message Format	117
A.1.1	Message Envelope	117
A.1.2	DRMTP Messages	118
A.1.3	SDCP Messages	118
A.2	Node Application Programming Interface	120

List of Tables

A.1	Message Envelope Fields	118
A.2	DRMTP messages	119
A.3	Compound types in DRMTP messages	119
A.4	SDCP messages	120

List of Figures

2.1	An illustration of the system model.	26
2.2	Simplified Napster system model. (a) A node registers and advertises local files. (b) The node submits a query. (c) The server returns a recommendation. (d) The node selects a source and transfers the file.	31
2.3	Napster as an overlay network	33
2.4	Query propagation in Gnutella. (a) Node N_1 broadcasts the query to neighbouring nodes in the overlay. (b) Second hop propagation. (c) The query reaches N_2 , where it is matched, and N_2 contacts N_1 with the match.	36
2.5	Congestion in multicast transport protocols, in a flow from the source S to sinks R_1, R_2, R_3, R_4 . (a) Global congestion caused by a congested link near the source. All sinks lose packets of the flow. (b) Local congestion for a receiver subset. Only sinks downstream the congested path (R_1, R_2, R_3) lose packets.	42
3.1	Local patching for late joins. (a) Initial configuration, with N_1 providing S_1 (red) to N_2, N_3 , and N_4 . (b) N_5 joins S_1 and N_3 locally patches with S_2 (blue).	54
3.2	Real-time operation with aggregation of two half-rate streams.	54
3.3	Stream error conditions.	67
4.1	Expected number of feedback messages from an exponentially distributed timer, as a function of the number of feedback nodes, for $\lambda = 10$	76
4.2	Expected feedback latency from an exponentially distributed timer, as a function of the number of feedback nodes, for $\lambda = 10$	78
4.3	Growth of $W_{max,d}$ and N_d as a function of scope depth. (a) $W_{max,d}$ (b) N_d	79

4.4	Expected number of messages and feedback latency as a function of d for our choice of $W_{max,d}$ and N_d , and effects of poor estimates (a) Expected number of feedback messages (b) Expected feedback latency	80
4.5	Network testbed	82
4.6	Stream establishment traffic in the first experiment. (a) 18.85.9.x subnet. (b) 18.85.45.x subnet.	84
4.7	Data traffic in the first experiment. (a) 18.85.9.x subnet. (b) 18.85.45.x subnet.	85
4.8	Control traffic in the source failure experiment. (a) 18.85.9.x subnet. (b) 18.85.45.x subnet.	86
4.9	Data traffic in the source failure experiment. Trace taken from the 18.85.9.x subnet.	87
4.10	Data trace for the network effect experiment. (a) 18.85.9.x subnet. (b) 18.85.45.x subnet.	88
4.11	Evolution of session population and the number of streams in each subnet.	88
4.12	Source upstream congestion experiment scenario. R is the primary controller of the stream, and S_1, S_2 are two sources with limited upstream capacity.	89
4.13	ns trace from the source upstream congestion experiment. (a) Throughput stream $S_1 \rightarrow R$ (b) Throughput for stream $S_2 \rightarrow R$ (c) Aggregate throughput.	91
4.14	Source upstream congestion experiment scenario. R is the primary controller of the stream, and S_1, S_2 are two sources. Nodes A and B establish a TCP flow, competing for the bottleneck link.	92
4.15	ns trace for the protocol fairness experiment. (a) Throughput for stream $S_1 \rightarrow R$ (b) Throughput for stream $S_2 \rightarrow R$ (c) TCP flow throughput (d) Aggregate throughput for R	93
5.1	Affinity point distribution with a pyramid scheme. Nodes n_1, n_2, n_3 are the first order referral set of n , receiving $\frac{1}{3}\alpha_{f,t}$ affinity points. Nodes n_5, n_6 are the second order referral set, as first-order referrals to node n_1 , receiving $\frac{1}{6}\alpha_{f,t}^2$ affinity points.	99
A.1	Node Application Programming Interface	121

List of Algorithms

1	Stream establishment by a DRMTP receiver	59
2	Request supression for DRMTP streams	60
3	Bid supression for new DRMTP streams	62
4	Error correction for a DRMTP receiver	69
5	Caching and supression of SDCP <i>match</i> messages.	73

Chapter 1

Introduction

The Internet has grown from a medium for computer communications to one that is becoming the backbone for communications in general, including telephony, entertainment and sensor networks. In this context, a primary problem is how we distribute and share information throughout networked communities. In particular, there are issues of scale, flexibility, and dynamism that have not been addressed by existing systems. This is an imperative for fundamental cultural reasons that are suggested by Napster and Instant Messaging: after having used the network as a static library, increasing numbers of people are using it to access each other rather than servers. For example, since the introduction of ICQ in the mid-90's, instant messaging has grown to one of the most widely used services in the network [47, 14]. Similarly, the Napster network grew to millions of members within a year of its inception [6].

This cultural imperative is supported by the technology. With time, as raw computing power, connectivity, and bandwidth increase, the difference between computers diminishes and they become a network of peers – even if some are idiosyncratically labeled as servers. In such an environment the traditional model of broadcast distribution of information loses its meaning. Every node on the network can act as a distributor, every user can be a broadcaster. And when it comes to digital media, this environment presents an opportunity for information to spread efficiently and effectively by following user interactions. The premise follows from the *group forming* nature of the Internet. As David Reed puts it in [62], in networks that support affiliations among their members, the value of connectivity scales exponentially. In this context, shared responsibility for information distribution has economic as well as technical

implications for the efficacy and efficiency of delivery. Hence we coin the term *Collaborative Media Distribution*.

1.1 Technical Hurdles

The main issue is network scaling. Central networks, built around the client-server model, have a limit to scalability. For example, a server-based system requires significant resources just to service an HTTP connection. In 1996, the Netscape server system was delivering more bits than are stored in the US Library of Congress each day. This was an arduous task in and of itself, causing delays and overloading the network connection of Netscape. The same problem is faced by all major content sites in the present day, mandating the development distributed networks of servers – such as those built by Akamai [82] – simply to carry the load.

One response to this was a renewed interest in multicast. Multicast was introduced to the Internet in the early '90s [15], mapping broadcast techniques onto a fundamentally point-to-point system. However, broadcasting did not easily translate from the airwaves or printing press to the Internet. A host of issues interfered, generally derived from the fact the the Internet is an inherently heterogeneous environment and multicast packet delivery is unreliable. The first attempts to introduce reliability resulted in poor scalability, as the more receivers the greater the error rate and its diversity among receivers. The ensuing avalanche of error reports and corrections caused multicast to begin to drown, a phenomenon known as *implosion*.

An important limiting factor of early multicast transport protocols was their server-centric approach. Recognizing the fact that any node receiving data in a multicast group can act as a sender, later protocols addressed issues of scalability with distributed error correction and local recovery schemes. This experience illustrates again the basic principle that makes the Internet a distinct communication network: *Every client can be a server* [45]. Recent work in the Media Lab [40] explored this approach for the distribution of real-time data. The simple theory was that if every client became a server, then the network could combine local repair with widespread reception of real-time data, and it could thus simulate an infinite bandwidth network.

While these systems and protocols hinted on the efficiency of collaborative media distribu-

tion, it was not until the advent of peer-to-peer systems that its potential efficacy was realised. Peer-to-peer systems exposed the idea to the general public by providing simple mechanisms for locating media available at end-user machines. They serve as an elegant example of a group forming network by generating unprecedented momentum: The Napster network had over 20 million users on its heyday.

Nevertheless, the first generation of peer-to-peer systems employed inefficient algorithms for location and strictly point-to-point distribution, limiting their scalability for real-time transmission of information and high bandwidth types of media like movies. Recent work in lookup algorithms [69, 79, 65, 61] has significantly improved data location capabilities, but peer-to-peer systems still today do not take advantage of the progress made with multicast transport protocols. They also make clear another problem that may constrain the applicability of the paradigm: *The Network is asymmetric*. End-users with broadband connections may have access to significant downstream capacity, but their upstream capacity is proportionally limited. In addition, end-user nodes are inherently unreliable; nodes come and go at unpredictable times adding an additional hurdle to protocol reliability.

1.2 Socioeconomic Hurdles

The flipside lies on the socioeconomic obstacles of building and sustaining a collaborative media distribution scheme. While the cultural imperative of group forming networks and the technical advances suggest unrestricted collaborative distribution as a natural solution, it is a destabilising issue for entities that rely on a centralised architecture for revenue.

The tractability of centralised revenue models, backed by support from current day legal systems, provides the backbone of entertainment and publishing industries and creates a structure for royalty flow towards content creators. The lack of a viable revenue model for distributed schemes of content distribution poses both an economic and social problem. Copyright owners and related industries are fighting to preserve their revenue and power, despite the fact that scarcity of resource no longer justifies a centralised model of distribution. Simultaneously, content creators are reluctant to support a scheme that can potentially commoditise their work without providing economic gains. The result so far is a push for legislation like

the HRA¹ and DMCA² acts and the proposed SSSCA/CBDTPA bill³ [83], which criminalises common behavior and rendered the first generation of peer-to-peer systems commercially unsustainable. These developments are nothing more than manifestations of the battle between established economic forces and rising cultural and technical phenomena. As Lessig argues in [44], the balance is thin and there is a clear danger of falling into a new era of dark ages.

However, there is reason to believe that a combination of cultural imperative and technical support can override legal systems. The 55 MPH speed limit is an example. Lacking a technological limit on automobile speed, it was widely disregarded [56, 48, 26]. In this case, the cultural imperative is the increasing amount of media sharing that is in progress [9, 42, 81]. The technological support is the raw ability for every computer to become a server. Note how Sun's image of the networked computer as a diskless engine permanently attached to a server missed the point. Given this combination, there is a structural threat to industries that rely on the legally or technically supported central model.

1.3 Contribution of this Work

This work provides a solution to reconciling the technical and cultural impetus for collaborative media distribution with the economic forces behind it. It is in the context of *distributing the distribution* that we embark on the work of this thesis. We carry it to an extreme and show how a distributed programmable protocol can do all of what we want.

We develop a high level active protocol architecture, which adheres to the end-to-end design principle [66] of the Internet and provides the primitives for flexible and efficient location and distribution of information. On the same time we enable security and privacy, accountability, and embed cost and loyalty distribution model. These primitives are used for developing a programmable protocol substrate, demonstrated with a prototype implementation and backed by an economic model for real-time collaborative media distribution.

The cornerstone of the architecture is DRMTP (*Distributed Real-time Multicast Transport Protocol*), an adaptive application-level [13] protocol core which allows collaborative multi-

¹Home Recording Act

²Digital Millenium Copyright Act

³Security Systems Standards and Certification Act/Consumer Broadband and Digital Television Promotion Act

casting of real-time streams. The protocol provides transparent semantics for loosely coupled multipoint interactions. It allows aggregation and interleaving of data fetched simultaneously from diverse machines and supports the location and coordination of named data among peer nodes, such as a record album or television program, without additional knowledge of network topology. The dynamic stream aggregation scheme employed by the protocol solves the problem of network asymmetry that plagues residential broadband networks. In addition, the stateless nature of the protocol allows for fast fail-over and adaptation to departure of source nodes from the network, mitigating the reliability problems of end-user machines. Coupled with well established techniques, like traffic localization [54], stream patching [40, 67, 37], and TCP-friendly congestion control [25, 77], we deliver a protocol that enables scalable real-time media distribution in a completely decentralised, serverless fashion.

DRMTP is supported by a dynamic content and source discovery protocol, which determines the properties of the network and availability of information based on high-level content description. This way, users are able to locate and access media without ever knowing about the existence of potential sources in the network and without noticing intermittent failures in the act of the distribution.

Along those lines, we have also developed a novel dynamic, mostly functional language named MAST (*Meta Abstract Syntax Trees*). The language has full support for mobile code and distributed computation and can be embedded in the payload of the content discovery protocol or even DRMTP itself. However, the presentation of the language is beyond the scope of the thesis, as we limit to the basic components of the protocol architecture.

Finally, we develop a micro-payment scheme for cost distribution and loyalty payment, which explicitly allows redistribution of content by end users. Our scheme includes an affinity point computation algorithm, which rewards end-users for redistribution, thus providing economic incentives for the sharing of media. The ramification of this approach is that end-users are encouraged to provide access to their media store, thus maximizing the efficiency of the distribution with DRMTP. Simultaneously, the cost of distribution for copyright owners and content providers is drastically reduced, and availability of information is automatically determined by popularity, transcending the lifetime of the original host.

1.4 Thesis Overview

We review the fundamental techniques of peer to-peer-systems, multicast and programmable networks in Chapter 2. They will be the basis for the solution we present. We also note how this can become a realtime medium suitable for dynamic shared access to information as well as archived (newspaper-like) delivery. We stress communal access to stored material in this thesis. Rather than redesign a point-to-point telephone network, this work focuses on information such as entertainment, news and data that changes slowly relative to the amount of access.

Chapter 3 describes the basics of our protocol architecture: the DRMTP protocol and the session discovery and configuration protocol. We present the basic algorithms, deferring some of the details for Appendix A

We analyse the properties and performance of our protocol architecture in Chapter 4. There we discuss scalability bounds through simple mathematical analysis, illustrate fail-over capabilities and scalability in the presences of network effects with a prototype implementation, and congestion control through simulation.

We close with the description of our economic model for collaborative media distribution in Chapter 5. There, we embed in the distribution a payment protocol modeled on a combination of the Amway model with the existing distribution structure for music.

Chapter 2

Towards Collaborative Media Distribution

The popularity of peer-to-peer systems, such as Napster [87], Gnutella [84], Morpheus [86], and Kazaa [85] to name a few, has clearly demonstrated the paradigm of collaborative media distribution. Any digitally encoded media file can be available throughout the network, in a large number of otherwise unrelated end-user nodes. Furthermore, nodes have large enough storage capacity and capability to hold a copy of any local media file for as long as users of the node desire. Likewise, with broadband connectivity, the bandwidth available at end users increases quickly. For end-users interacting with the network, *it doesn't matter where the bits come from*. Users are interested in locating files matching their interests and transferring them to their local node, preferably at a rate that allows real-time playback.

Within this framework we are trying to answer the question of designing a scalable system for efficiently locating and transferring a media file. The file may be available to many different nodes and a suitable subset of them should be used for the transfer. Similarly, transfer should happen in a way that minimises network overhead and if possible aggregates traffic towards multiple nodes concurrently accessing the same file. Transfer should support online real time playback if network conditions allow, and it should dynamically adapt to contingencies. Such contingencies include congestion in the network and unexpected node failures.

In this chapter we set the stage for the work in this thesis. We develop a system model for collaborative media distribution and discuss the current generation of peer-to-peer systems.

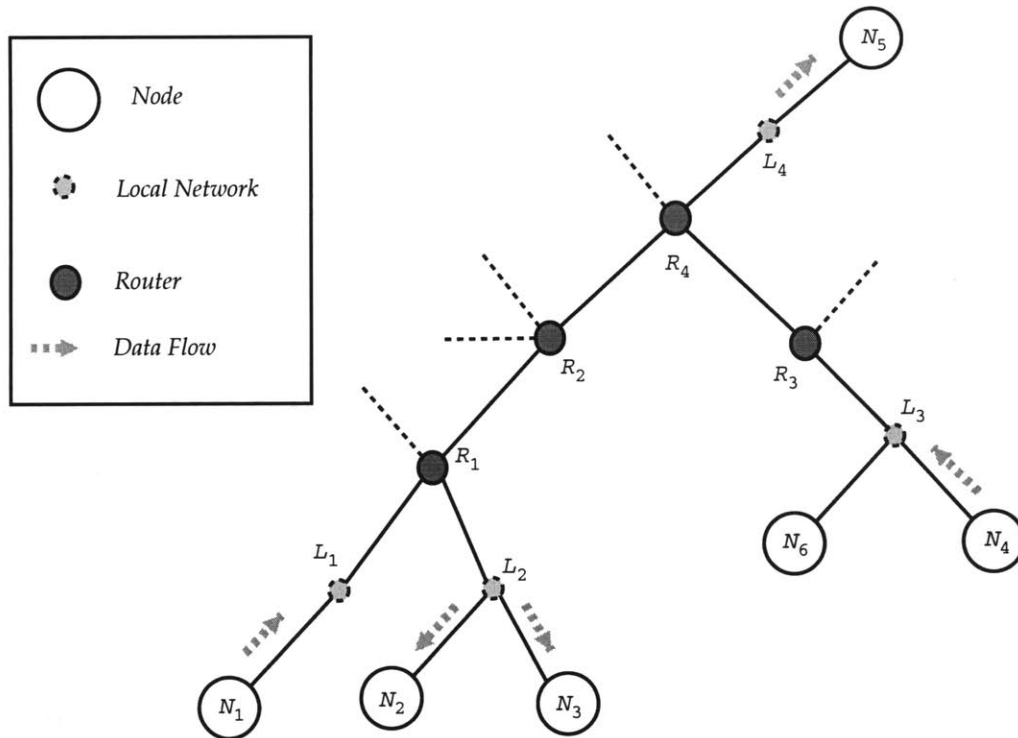


Figure 2.1: An illustration of the system model.

We then argue for the use of multicast as the fundamental communication paradigm and the benefits of a programmable multicast protocol substrate, on our way to outlining the protocol architecture developed in this thesis.

2.1 A System Model

The operating environment is a connectionless global area internetwork, the Internet. Any host in the Internet can be a node for the system. Figure 2.1 illustrates the system model for a portion of the network, depicting routers and the local organisation of hosts.

Each node maintains a local store of media files, which can be locally accessible by the owner or transmitted in part or whole to other nodes. The set of content files that can be part of the system is infinite countable: each file is distinct and new files can be added at any time. Of course not all hosts are active in the network at all times, as new hosts can be added at any time, and existing hosts may depart unexpectedly. Similarly, not all content files

are available in the system on the same time, and new files can be added or existing files be removed from any node. Thus, if $nodes$ is the set of all nodes and $files$ is the set of all files in the system, $act(nodes) \subseteq nodes$ denotes the set of active nodes and $act(files) \subseteq files$ denote the set of files available in the network. In Figure 2.1, $act(N) \equiv \{N_1, N_2, N_3, N_4, N_5, N_6\}$.

Each file $f \in files$ has a unique immutable identifier¹ $ident(f)$, a well defined length² $length(f)$ in bytes, and for the purposes of distribution a well defined framing F_f^{file} . The framing is a subdivision of the file in smaller parts, each with a well-defined length, and allows us to define a real-time operation for the distribution. Let

$$F_f^{file} = \{f_1 \cdots f_n\} \quad (2.1)$$

be the framing of a file f . A *schedule* for the framing is a sequence of relative times at which frames become available:

$$schedule(F_f^{file}) = t_1, \dots, t_n \quad (2.2)$$

Real-time access to a file is defined by an immutable schedule R_f^{file} , intrinsic to the file:

$$R_f^{file} = r_1, \dots, r_n \quad (2.3)$$

A schedule $schedule(F_f^{file})$ is *real-time*, if

$$\forall t_i \in schedule(F_f^{file}), r_i \in R_f^{file}, t_i < r_i \quad (2.4)$$

Individual frames can be locally stored at a node, with local store including storage devices of the node and temporary memory. A node n has a complete file f in store if and only if it has all the frames of the file:

$$store(n, f) \leftrightarrow \forall f_i \in F_f^{file}, store(n, f_i) \quad (2.5)$$

A file is available in the network if and only if all frames are available in the network, possibly

¹Such an identifier can be constructed with a digest function like MD5 [63]

²Live content does not have a well defined length. However, we can define a file that includes a prediction of the frames that will be produced by the generation process, and insert frames to the network as they become available.

in different nodes:

$$f \in act(files) \leftrightarrow \forall f_i \in F_f^{file}, \exists n \in act(nodes), store(n, f_i) \quad (2.6)$$

In addition to $ident(f)$, files have a high-level, human-readable description. The description however depends on the node where the file is stored; the user-owner of the node can attach the description to files, and modify it at will. We denote by $desc(n, f)$ the description of file f in node n . The description captures the meta-data of the file. Meta-data include information about the type of the file, a type-specific structured description, and so on. For example, a music file encoded in MP3 format may have meta-data like artist, track and album names, year of publication, label of publication (and copyright owner), etc. Similarly, a user could add a description or assessment of the file in his own terms. Users locate files in the system by a means of queries on descriptions. A query is a user-defined boolean function which matches a description of a file at a node. Hence, a user may be interested in accessing a file f , if a query $query$ matches at some node n , that is if $\exists f, n, query(desc(f, n))$.

Files are transferred in the system in *sessions*. A session is specific to a file, and includes all participating nodes. Some nodes participate as sources, and other nodes participate as sinks. A source in a session provides one or more frames of the file to one or more sinks. By this definitions, nodes may participate both as sources and sinks in the session. Since the file identifier is unique and immutable and a session is associated with a specific file, the sessions are also unique and immutable. If $sessions$ is the set of all sessions, with a one to one correspondence to the $files$ set, then $act(sessions) \subseteq sessions$ is the set of sessions which are active in the system.

With these definitions in mind, we can provide a high-level description of system operation:

1. A user, interacting with a node n constructs a query $query$ which describes the file of interest.
2. n evaluates the query to one or more other nodes in the system, until at least one node m is found such that $query(desc(m, f))$ evaluates to true.
3. n joins the session for file f , as a sink initially, and transfers the file from some of the sources in the session. By receiving some of the frames in the session, n immediately

becomes a potential source in the session.

The problem has many facets. A mechanism is necessary for the query to be evaluated to a subset of $act(nodes)$ so that a match is found. Next, the session must be initiated: sources that can provide the file in parts or whole must be located in the network and prepare to serve data streams. And finally, frames must be transferred by the sources of the session to the node.

In general, a node is aware only of its local store. It does not have any knowledge about the existence of other nodes in the network or the files or frames available in their local store. Nodes communicate with each other by message passing. Messages, which may contain operation instructions and frames, are transported by the network in the form of packets. The overall efficiency of system operation can be expressed in terms of number of messages that are exchanged, and packets that are transported by the network. In addition, the system does not operate in isolation; rather, since the nodes are Internet hosts, there is background traffic carried by the network, mainly in the form of TCP flows. Hence, operation of the system can be evaluated in terms of:

- The number of messages E_m exchanged in order to locate and transfer a file, between any number of nodes.
- The number of packets E_p generated in the network from these messages
- The effect of system traffic on competing traffic in the network.
- The time it takes to complete location and transfer.

For example, consider again Figure 2.1, and let f be a file of interest for nodes N_2 , N_3 , and N_5 , available at nodes N_1 and N_4 . In order to locate the file, the *sinks* must reach the sources by sending query messages. A great deal of complexity arises by the discovery process itself, as the sinks require a way to become aware of the presence of sources and prepare them for the file transfer. We discuss how this could be done in the next few sections, so for the moment let us assume that the sinks are aware of the sources and have perfect knowledge of the topology, which allows them to select the closest source. If sinks operate without coordination between each other, based solely on *point-to-point* messages, then we require $E_m = 3$ messages for establishing the data transfer: $N_2 \dashrightarrow N_1 : get$, $N_3 \dashrightarrow N_1 : get$, and $N_5 \dashrightarrow N_4 : get$. This

also requires $E_p = 7$ packets to be routed in the network, as packets are forwarded by routers. Messages from L_2 to L_1 require two packets, since the two local networks are one hop away from each other. Similarly, messages from L_3 to L_4 require 3 packets. For the file transfer, we require a minimum of $|F_f^{file}|$ messages for each, assuming a perfect network that does not lose messages and that a single packet is used for each frame. This translates to a minimum of $7 \cdot |F_f^{file}|$ packets in the network. The minimum time necessary for each transfer would be roughly equal to the time required to cross the network links between the source and the sink, if the source was able to perfectly pipeline the packets for arrival and no other traffic was crossing the routers. Of course, even with no competing traffic in the network, the transfers $N1 \rightarrow N_2$ and $N1 \rightarrow N3$ will cross the same network links and require some queuing in the R_1 router.

This example illustrates some of the complications that arise: sources must be located by the exchange of messages, files must be transported by more messages, and the overall interaction results in many more packets in the network. However, things are even more complex: packets may be lost and sources may unexpectedly fail in the midst of file transfer. To deal with packet loss contingencies we must use protocols for reliable transfer. Furthermore, the protocols in use should generate the minimum traffic possible and compete fairly with background traffic when it appears in order to avoid congestion, which causes the packet loss [23]. And in order to deal with failure contingencies, our protocols should use a *loosely coupled* model, that does not rely on a source surviving an entire file transfer. We discuss approaches to the problem in the remaining of the chapter.

2.2 Peer-to-Peer Systems

2.2.1 Napster

The primary example of peer-to-peer system is Napster. Napster is a very simple system, which attempts to solve the discovery problem with the use of a centralised registry or server (Figure 2.2).

The system works as following:

1. A node joining the network registers with the napster server, and provides a list of files that is sharing. This is accomplished with an *advertise* message, which contains

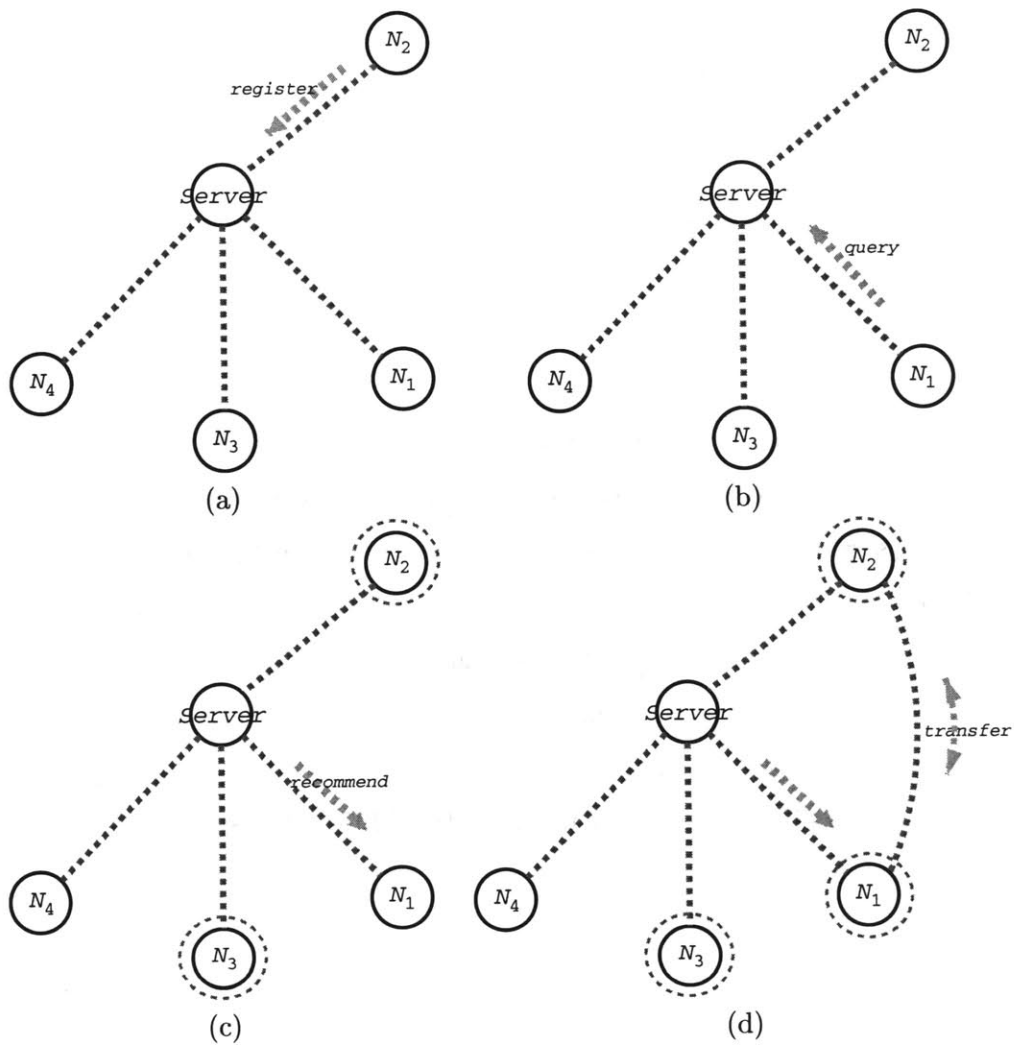


Figure 2.2: Simplified Napster system model. (a) A node registers and advertises local files. (b) The node submits a query. (c) The server returns a recommendation. (d) The node selects a source and transfers the file.

descriptions of all the files locally available. The description is limited to a file name, which can be considered a limited form of meta-data.

2. A *query* message is submitted to the napster server, which matches the query against all active advertisements. The server returns a recommendation of matching nodes to the querier. The query is limited to wildcard matches for file names.
3. The node selects one of the sources and establishes a point-to-point connection to the source for transferring the file.

In a first approximation, the system requires just two messages for resolving the query (*query* and *recommend*), plus the number of messages required for transferring the file. The actual network utilisation is not simple to estimate though – a great deal of problems are hidden under the hood.

The problems have to do with the scaling, efficiency, and fault-tolerance characteristics of the system. First, the centralised design requires that all nodes advertise their entire local state to the registry. Apart from the state maintenance problems that this involves, given the capacity of local stores of modern personal computers, the state transfer may require significant bandwidth. Furthermore, the server must be powerful enough to handle the state of all the nodes and efficiently process queries on it. Hence, the server becomes the bottleneck of operation, limits the scalability of the system, and reduces the efficiency by generating significant network traffic for state maintenance. In addition, the system's fate is bound to the server. If the server fails, then the system goes down with it – ironically, this is was the fate of the original Napster network.

In addition, the system does not offer any mechanism for managing network traffic, placing an additional burden to scalability. To appreciate the problem, let us overlay the transfer part of Figure 2.2 to an example model of the underlying network infrastructure. Figure 2.3 illustrates the situation. There are two things to notice: First, the N_1 , which is the query node, is unaware of the actual network topology of the network. Therefore, the selection of the source can be completely incorrect – in the example the selected source is N_2 . By transferring the file from N_2 instead of N_3 which is closer in the network, parts of the network are unnecessarily loaded. In addition, given that TCP is used for the transfer, the actual throughput of the transfer is *reduced*. This is not immediate without an understanding of how TCP works:

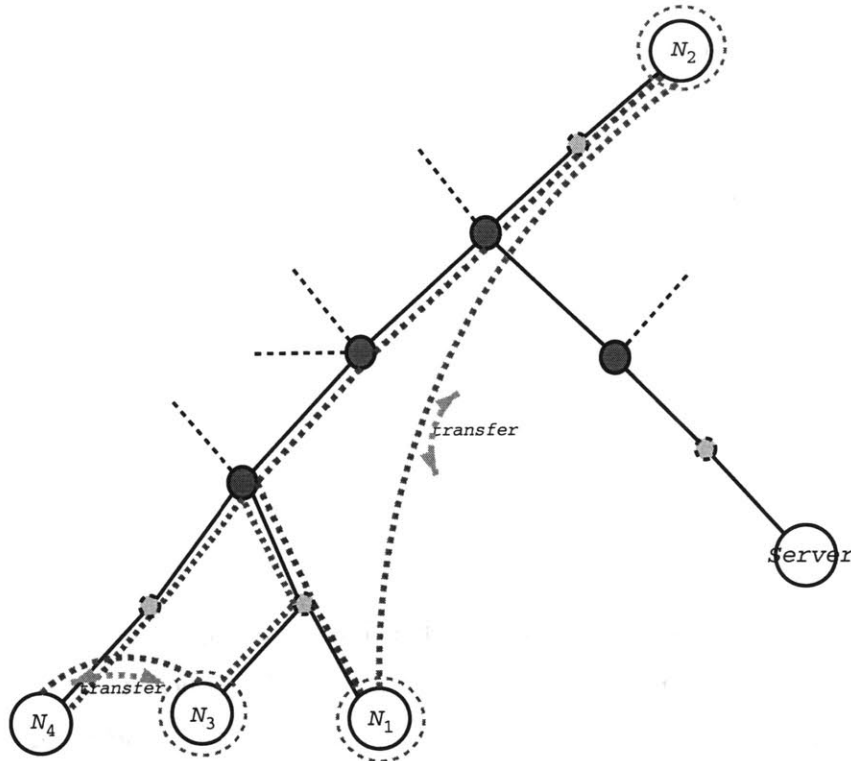


Figure 2.3: Napster as an overlay network

The throughput is inversely proportional to the round-trip time of the network path [55]. Hence, longer connections peak at lower transfer rates and, as a result, the unnecessary load of the network is extended for a longer time³. Of course, there are ways to select an optimal path to different nodes, as for example with Resilient Overlay Networks [5]. Unfortunately, these techniques require significant communication for maintaining information about network topology, practically limiting scalability to a few hundred nodes.

The other detail to notice is that N_1 is completely unaware of a concurrent transfer of the same file, that takes place between N_3 and N_4 . As a result, the actual network traffic is more than double the traffic that would be necessary if N_3 was simultaneously transmitting to both N_1 and N_4 . Although the probability of concurrent transfer may seem remote, network effects of media distribution should be taken into account. During popularity surges, popular media

³Traffic localisation is one of the premises of Content Distribution Networks like Akamai [82]. However, in such systems, replication of the content is required in each edge server, while the store of end users is not leveraged at all.

files will tend to be accessed by many nodes in a short time. To quantify this, consider a high-quality encoded MP3. Typically, such a file will have a size of $10MBytes$ for a 5 minute track. With a $1Mbps$ downstream connection, even if the full bandwidth is completely utilised by a single node⁴, the transfer will require 78 seconds. Therefore, at most 1000 users can access the file in a day without any temporal overlap of the file transfer. If we extrapolate this to a 2-hour movie encoded in MPEG format, the number of completely non-overlapping users in a day drops to 10 or 15. These examples serve to illustrate the effects of unmanaged traffic for media distribution can be on the network, and hint on the use of multicast as the underlying transport technology.

2.2.2 Gnutella

Gnutella takes a different approach to file location. Instead of relying on a centralised registry, with all the fault-tolerance and scalability problems that this implies, it constructs an overlay network. In the overlay, each node maintains connections to some other nodes it is aware of. There is no state exchange between the nodes; rather queries are propagated along the overlay. Each node locally evaluates the query; if the query is successful it directly replies to the source of the query, otherwise it forwards the query to all known nodes with which it maintains connections. The propagation of query messages is controlled by a time-to-live (TTL) parameter, which is decremented with every hop on the overlay. When the TTL reaches zero, the message is not further propagated.

Figure 2.4 illustrates an example of query propagation in the gnutella network. We take the same setting as the previous example, with the target file available in N_2 and N_3 , but do not show query propagation further than 3 hops. If the query TTL was higher than that, node N_3 will also receive the query and reply to N_1 with a match. Once again, after the search completes, N_1 selects a matching node and transfers the file from it, with a similar strategy as Napster. Hence, the transfer process suffers from the same deficiencies as content transfer in Napster. The difference is that scalability and fault-tolerance is not restricted by the existence of a centralised registry. But a different problem creeps in: query propagation is very expensive. For example, a query with an initial TTL of s in an overlay with an

⁴Broadband connectivity is based on sharing the downstream bandwidth between local nodes.

average degree⁵ d will result in $O(d^s)$ messages. Furthermore, the actual links in the overlay network bear no relationship to the actual network topology, resulting in multiple packets in the network per message. Finally, an additional hidden cost in gnutella includes network maintenance by periodically exchanging messages between adjacent nodes in the overlay.

2.2.3 Advanced Peer-to-Peer Lookup Algorithms

As we have seen, scalability problems of the first generation of peer-to-peer systems⁶ affect both content discovery and transport. The content discovery scalability problem has different manifestations in the two approaches we discussed. In the centralised registry approach, the system is limited by registry scalability. In decentralised approaches based on Gnutella, scalability is limited by the number of messages and network traffic generated by the search algorithm. The problem of transport is similar in both approaches.

These scalability problems have spurred a wave of research in peer-to-peer systems and algorithms. So far, the result is new generation of lookup algorithms, including Chord [69], Tapestry [79], Pastry [65], and CAN [61]. These algorithms are used so far for cooperative storage and file systems – for instance Chord is used in CFS [18] and Tapestry is used in OceanStore [16], but they can be used as a more general overlay network routing mechanism.

The approach taken by these algorithms is to construct a *distributed hash-table*: Given a key k describing a content file, locate a node n where k is mapped in a purely decentralised manner. All four algorithms operate with local knowledge, require each node to maintain a few links to other known nodes in the network, and can return a match in $O(\log n)$ messages. Such algorithms are excellent for session discovery, as the key can be the file identifier. However, content discovery with high level queries remains unresolved. Similarly, although some of the algorithms can support proximity routing for locating the closest (in network sense) node matching the key, the transport problem remains largely unresolved. Finally, these algorithms have a hidden maintenance cost, the effect of which in scalability for highly dynamic is not clearly understood yet.

⁵The degree is the number of connections of a node to other nodes.

⁶From newer first generation systems, Kazaa is based on proprietary protocols, with no publically available specification. Morpheus and Limewire, two other popular peer-to-peer clients are based on Gnutella.

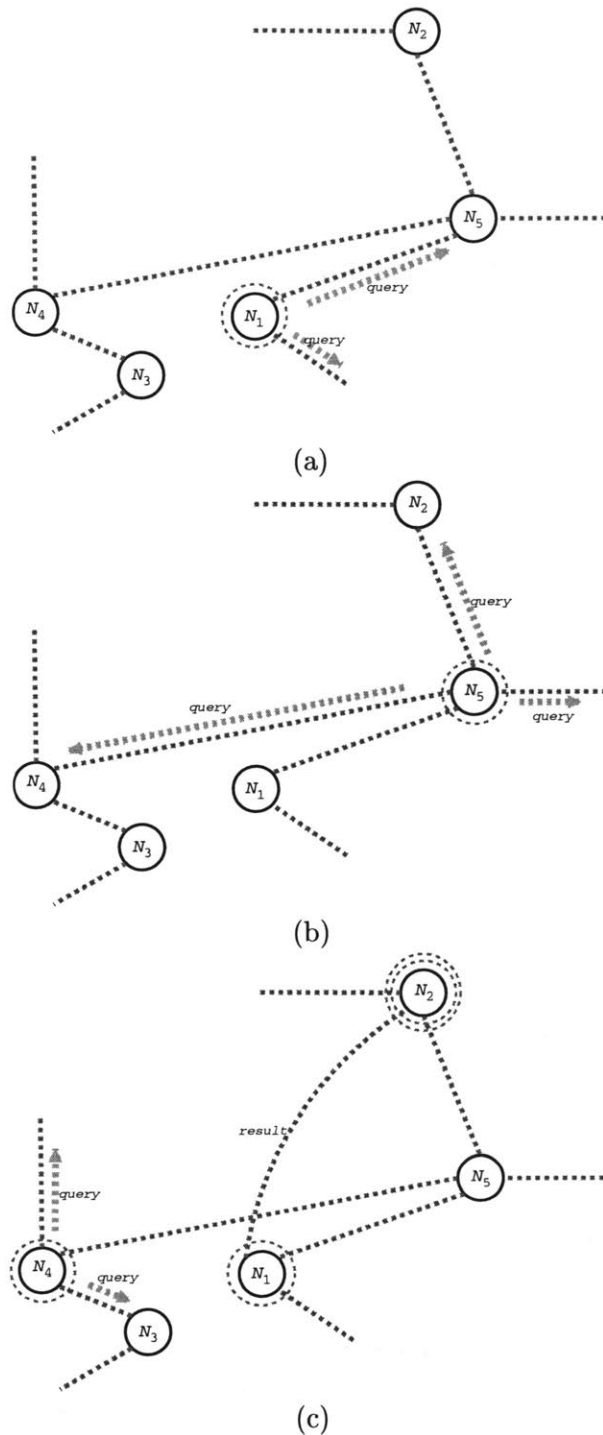


Figure 2.4: Query propagation in Gnutella. (a) Node N_1 broadcasts the query to neighbouring nodes in the overlay. (b) Second hop propagation. (c) The query reaches N_2 , where it is matched, and N_2 contacts N_1 with the match.

2.3 The Promise of Multicast

Collaborative media distribution is an inherently multipoint communication activity. Even if a node looks ahead during playback, the rest of the file must still be received. Multiple nodes may be requesting the same file concurrently, and similarly multiple nodes may be providing the same file, in part or in whole. Thus a mechanism for true multipoint communication can vastly improve system scalability, both by simplifying the problem of content and session discovery, and by providing the means for aggregating data transfer between hosts. For example, consider Figure 2.3 once again. With a suitable multipoint communication mechanism, both file transfers could be provided by N_1 . Similarly, the transfers $N_1 \rightarrow N_2$ and $N_1 \rightarrow N_3$ would be generating half the network packets than two point-to-point connections, as they are sharing the same network path.

2.3.1 IP Multicast

The suitable low-level technology for multipoint communications already exists in the Internet: IP multicast [15]. Although it has not yet been fully deployed, large portions of the Internet are already multicast-enabled and it is an inherent part of IPv6, the next generation IP protocol [34]. The model of IP-multicast is *best effort unreliable delivery* of datagrams to logical addresses, known as multicast groups. Multicast addresses are part of the reserved class-D addresses, offering an 28-bit address space.

Multicast groups are open; that is, any host on the Internet can receive packets destined for multicast addresses by simply *joining* the group. Similarly, a member of a group has no means for finding other members of the group. Information about membership in various parts in the network is exchanged between hosts and routers and propagated among routers using the Internet Group Management Protocol (IGMP) [22]. On receiving information about the state of membership in the network, routers can construct distribution trees for each group. The distribution trees include the routers towards which multicast packets should be sent; their construction is part of the multicast routing protocol [73, 15, 21, 17, 8].

A sender to the group can control the *scope* of a packet by setting the TTL field. As with any IP packet, routers will decrement the TTL packet before forwarding and drop packets for which the TTL is zero. Therefore, a node can limit multicast packets to a particular distance

in the network with a suitable TTL value; for example, a TTL of 1 will only deliver the packet to the local network. A second mechanism for restricting the scope of multicast packets is *administrative scoping* [51], where scoping boundaries can be assigned to specific multicast addresses. The two approaches can co-exist, as administrative scoping is a static address-based scoping mechanism for enforcing routing boundaries, while decision while TTL-scoping is a dynamic per-packet based mechanism for controlling the scope of the packet within the distribution tree of an address.

2.3.2 Multicast Transport

Multicast transport presents significant challenges in itself. Most of the problems arise from the difference between multipoint and point-to-point communications. While the latter is an understood topic, with TCP [59] being the prevalent protocol in use in the Internet today, the semantics of multipoint communication are fundamentally different. The main problem arises from heterogeneity among the members of a group. Since members are scattered in the different parts of the network, they suffer different packet losses and in general are able to sustain different transmission rates. How can the sender of a packet ensure that the packet is received, if reliability is necessary? When should packet transmission in a stream progress? How can the members of the group synchronise their decisions of packet transmissions? These are just some of the fundamental problems that multicast transport protocol designers have to face.

An influential idea that pervaded the design of early protocols, was the reliable broadcast algorithm by Chang and Maxemchuck [12]. With this algorithm, progress and permission to transmit was controlled by the circulation of a token among members. The use of a token offered a control mechanism for senders in the protocol, and was used by the Multicast Transport Protocol (MTP) [7] for providing a protocol with sender-controlled streams. Sender-control mechanisms accommodated for a protocol design that is close to the semantics of point-to-point communication.

Protocols which are based in explicit positive acknowledgement of packet receipt, such as MTP, are called sender-initiated. Unfortunately, sender-initiated protocols suffer from a very serious deficit: the protocol cannot scale. As the number of receivers in the group increases, so does the traffic for providing acknowledgements to the sender, defeating the purpose of

multicast. The solution to this problem is to use a receiver-initiated approach [71]. With receiver-initiated approaches, receivers do not report positive acknowledgements. Instead, when a packet loss is detected by a receiver, the receiver can report the error with a negative acknowledgement and request a retransmission. However, this solution is not scalable either. When a packet is lost by a large number of receivers – for example when the loss occurs near the source – the scheme will result to an avalanche of negative acknowledgement, a problem known as *implosion*.

The solution to defeating implosion is to synchronise error reporting among receivers. An approach to synchronisation is *polling*: the sender periodically polls receivers for errors, simultaneously limiting the set of receivers that can report. Once a report is received, the sender multicasts the missing packet, which can then be received by any member of the group. This approach is used in [2], but the result is high latency between error occurrence and correction. Furthermore, not all receivers may have suffered the same loss, making transmission redundant for some of them. If some of the receivers are experiencing heavy congestion, correction will involve many more retransmissions, worsening of the actual congestion.

A different approach was taken by Reliable Multicast Transport Protocol (RMTP) [58]: In RMTP, some receivers are statically assigned the role of a *designated receiver*. Designated receivers are responsible for providing feedback to the sender, which is also statically assigned. Non-designated receivers locally report errors to the designated receivers with unicast messages, which are responsible for requesting retransmissions by the sender. While this approach scales better and reduces the implosion impact, it comes with a serious drawback: the sender and designated receivers are statically assigned. Furthermore, selection of the designated receivers requires knowledge of the network topology, making the protocol unsuitable for dynamic operation. The eXpress Transport Protocol (XTP) [20, 36] proposed a different approach: multicast of control traffic with a randomised slotting and damping of receiver feedback. The scheme allowed receivers to synchronise with a randomised algorithm, providing the first scalable solution for the implosion problem. This scheme was further extended by Scalable Reliable Multicast (SRM), as we describe in the sequel.

2.3.3 Scalable Reliable Multicast

By large, the problem of implosion and dynamic protocol state was solved by Scalable Reliable Multicast (SRM) [24]. SRM solves the receiver synchronisation problem by employing a randomised algorithm for error reporting. When a receiver perceives a packet loss, instead of immediately reporting an error it establishes a random timer. When the timer expires, and if error reporting has not been suppressed as we describe shortly, the receiver multicasts the error report to the entire group, and sets a new timer. If no correction is received before the timer expires, for example because the error report message or the correction packet was lost, the error reporting process restarts. This process is followed by all the receivers in the group; and since error reports are multicast, they will be received by all the members in the group. Thus, a receiver waiting to report the error can suppress the report on receiving an error report message from another receiver or the correction itself. Naturally, the effectiveness of the scheme depends on the selection of timer intervals [52, 53, 60]; an optimal timer interval can be selected according to the size of the group, the distance from the sender or by dynamically adapting to observed control traffic levels.

This simple and elegant algorithm provides with a mechanism for controlling suppression. Furthermore, since error reports are themselves multicast, any member of the group can provide correction packets allowing for distributed error correction among members of the group. Hence, sessions in SRM are *lightweight*: the protocol places no restrictions on membership or the roles of participating nodes. Multiple senders can easily be accommodated in the SRM framework, and any receiver can act as a sender. Similarly, members can join or leave the session at any time, without altering protocol operation.

There are a few variations in the basic SRM algorithm that provide more fine-grained control of the correction process [46, 39]. These variations try to localise the error correction process and avoid sending the correction packets further than it is necessary [54]. The first variation in localising error correction traffic is to restrict the scope of correction packets, so that it covers only the receivers suffering the loss. The second variation includes the use of multiple multicast groups, with receivers suffering the same losses joining a new multicast group where corrections can be multicasted. The combination of these two techniques leads to the concept of local groups [57, 35], where receivers locally organise for creating new multicast groups for retransmissions to be directed.

The framework of SRM and the local correction techniques that have been developed, is a good starting point for building a collaborative media distribution scheme. Kermodé [40] used these concepts for developing the Hierarchically Partitioning Real Time Protocol (HPRTP), used by a smart caching system for media distribution, paving the way towards collaborative media distribution.

2.3.4 Multicast Congestion Control

SRM set the stage for scalable reliable multicast, but did not attempt to provide a congestion control scheme in the system. Throughout the design, the assumption was that in general there is enough bandwidth provide for the basic requirements for transmission of protocol data. Unfortunately, this is not always the case in the Internet, where congestion can appear in paths of the network. The congestion problem becomes even more evident in a large scale system for collaborative media distribution, where a very large population of nodes may distribute an even larger number of content files. Every modern protocol design should include mechanisms for congestion avoidance and control [38]. Furthermore, the protocol should behave like a good citizen in the Internet, where the vast majority of traffic is carried by TCP streams [23]. Protocols that react to congestion and adjust the sending rates in a manner similar to TCP are known as *TCP friendly* [23, 25].

The problem of congestion control for multicast protocols is harder than the problem for point-to-point protocols. From one side, when global congestion is experienced in a session – for example when there is a single source and a congested path exists near it – the suitable reaction is to globally decrease the rate of the protocol, similar to what a TCP connection would do. On the other hand, if congestion is experienced only in some paths of the network and affects only a subset of the receivers in the group, a global decrease may not be the right decision to make, as the rate is unnecessarily decreased for all the receivers. Figure 2.5 illustrates the difference between global and local congestion, for an example session with a single source and a number of scattered receivers. In Figure 2.5(a), there is a congested path near the source, having a global impact on all receivers. In Figure 2.5(b), the problem affects only a few of the receivers. Things can get worse than this scenario, when there are multiple congested paths in the network with uncorrelated loss characteristics.

One of the first attempts in introducing congestion control for heterogeneous multicast

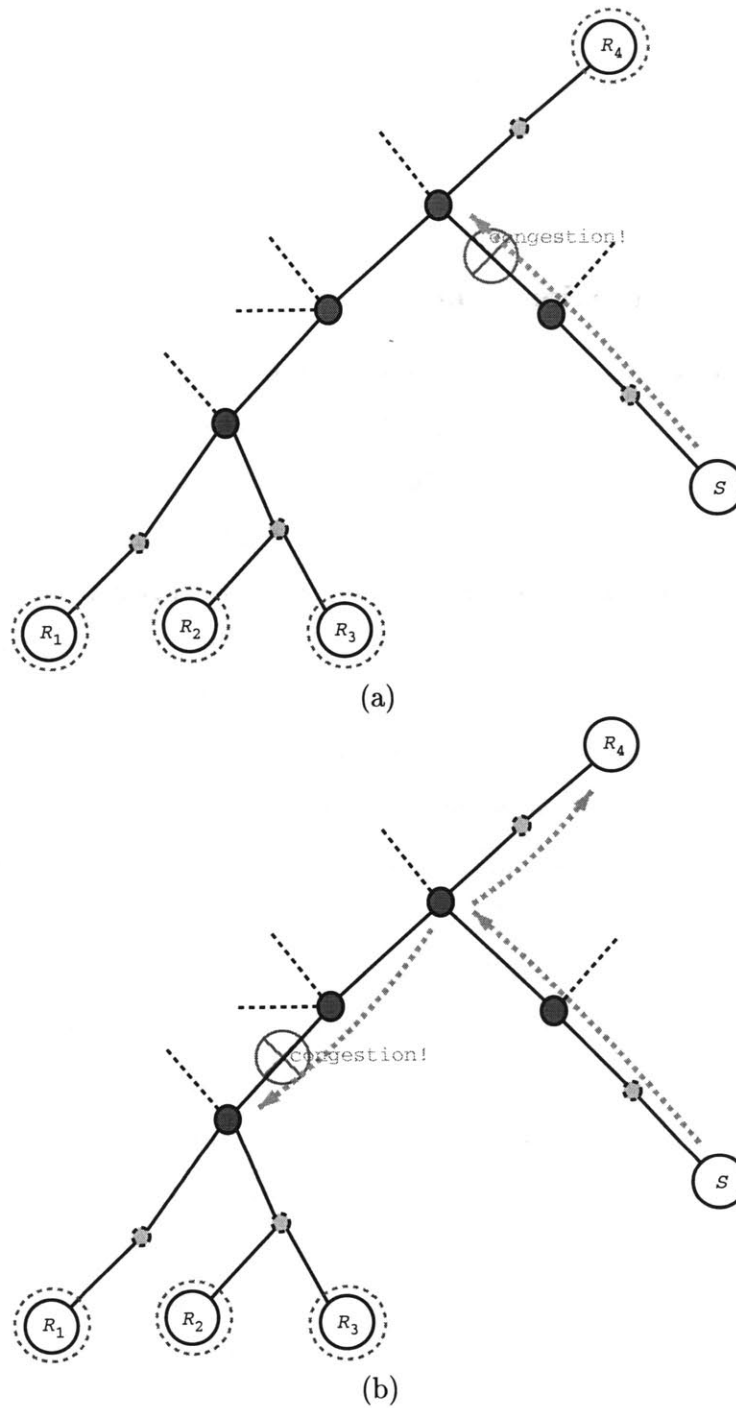


Figure 2.5: Congestion in multicast transport protocols, in a flow from the source S to sinks R_1, R_2, R_3, R_4 . (a) Global congestion caused by a congested link near the source. All sinks lose packets of the flow. (b) Local congestion for a receiver subset. Only sinks downstream the congested path (R_1, R_2, R_3) lose packets.

groups was Receiver-driven Layered Multicast (RLM) [49]. RLM was developed in conjunction with a layered video codec, and introduced layered transmission with multiple multicast groups. With RLM, there is a base layer carrying the traffic for basic video quality, and additional layers offering improved quality in a pyramid scheme. The more layers a receiver is able to tune in, the better the video quality. Initially, each receiver starts with the base layer and performs join experiments for additional layers. Receivers synchronise their join experiments with a randomised exponential back-off scheme and by announcing their intentions to other local receivers. While the join experiments are successful and all currently active layers are received without losses, receivers will keep adding layers until full quality video is achieved. If losses are perceived, the receiver deems the latest join unsuccessful and drops the highest layer, simultaneously backing off the interval for the next join experiment. Because experiments are announced to all receivers, the learning experience is shared; if a receiver joins a layer and other local receivers do not perceive any loss, then they also join the layer. If loss is perceived, other local receivers know that the source of the loss is another receiver performing a join layer and adjust their timers without dropping any currently joined layer. Unfortunately, there are some serious problems with this approach. First, if the base layer is causing congestion itself, its rate cannot be adapted, resulting in receivers entirely dropping out of the session. This is an artifact of real-time encoding, as explained in [68]. A second problem is not visible in the protocol design: router slack time. Tree pruning with IGMP is a slow process, and even when a layer has been dropped by all receivers in a network path, packets destined for the layer's multicast group will continue being forwarded for times in the order of minutes, causing packet losses to all layers. Finally, there is a problem of granularity in the layering scheme.

A partial solution for the problem of router slack time for layered rate adaptation is *dynamic layering* [19, 72]. With dynamic layering, the rate is temporarily increased within a particular layer to the rate of that would be achieved by joining the next layer, at well defined times. If a receiver perceives no losses, it can proceed and join the next layer. The scheme has been further extended to fine-grained layering [11], and is usually used in conjunction with Forward Error Correction (FEC) codes.

While these approaches solve the granularity of the layering, they cannot entirely fix the router slack time. When congestion is caused at some path because of sharing it with a

competing flow, even if the receiver perceives congestion and leaves the highest layer, packets will continue to be received. As a result, if the competing flow is a TCP connection, it will be effectively shut-down for the router slack time, while the actual traffic generated by the higher layer is simply lost. In addition, these approaches are unsuitable for our system model for a variety of reasons. Firstly, forward error correction is very resource intensive, with complexity increasing sharply when improving the efficiency of the code. Second, the digital carousel model employed by these schemes assumes a server constantly serving a stream of FEC packets. Thus, acting as a source requires significant resources. This limits the serving scalability, especially when we consider the fact that the number of content files in the system can be *very large*. Finally, in order for parts of the file to be decoded, complete code blocks must be received first, and if the beginning of the stream is missed, the receiver should wait for the carousel to rotate. Therefore, even if the achieved rate is sufficient for real-time operation for a node n receiving a file f , real-time playback will not be possible.

Despite the practical problems, dynamic layering is an interesting scheme for solving the path loss multiplicity problem. Coupled with TCP-friendly rate adaptation scheme for adjusting the throughput within the actual layers, we can leverage the density of the network and opportunistically use multiple sources for providing parts of the file. Of course, one problem remains: how to adjust the rate of a layer in a TCP-friendly manner. Fortunately, recent research has produced protocols with the desired characteristics using equation-based rate adaptation [25]. The cornerstone of these schemes is the approximation of TCP throughput through the equation

$$R = \frac{1}{t^{RTT} \sqrt{p} (\sqrt{2/3} + 6\sqrt{3/2} p (1 + 32p^2))} \quad (2.7)$$

where R is the throughput in packets per second, t^{RTT} is the round-trip time of the TCP connection, and p is the loss probability. The recently developed TCP-friendly Multicast Congestion Control (TFMCC) [77] scheme, explicitly uses the equivalent TCP throughput equation for a single-rate (non-layered) multicast congestion control. The scheme uses for the p and t^{RTT} values of the receiver which suffers the worse losses to globally adapt the rate. In a similar scheme, PGMCC [64] uses a dynamically designated receiver (the *acker*) in the group, as the floor for congestion control. The acker is selected again based on perceived losses by

the receivers, but the rate adaptation scheme is less smooth than TFMCC. These techniques are well-studied and are part of our protocol design for collaborative media distribution.

2.4 Programmable Networks

2.4.1 Session Discovery and Configuration

By using multicast as a low level transport mechanism, we can design a protocol for efficient distribution of information within a session. However, we have not yet discussed a suitable mechanism for session discovery and configuration. The session discovery problem is complicated by the necessity to submit high level queries for matching meta-data to actual session identifiers. Once a session identifier has been obtained, we need to configure participants in the session, so that data flow can begin.

The session discovery approach used in conjunction with SRM, is the Session Announcement Protocol (SAP) [32]. With SAP, a session directory maintains a list of current active sessions, together with a textual description, and periodically multicasts announcements containing both the description and configuration of the session. The payload of SAP messages is governed by the Session Description Protocol (SDP) [31]. Unfortunately, this approach cannot scale to a collaborative media distribution environment. The problem is twofold: first, announcements can cause significant network traffic, especially for large numbers of active sessions. And second, it suffers from the same registry scalability problems that napster-like systems suffer from. In addition, the session description protocol was designed for human users interacting in multicast conferences using SRM; as such, it does not contain any provision for automatic session configuration.

A more scalable solution for session location is to use the Service Location Protocol (SLP) [29]. SLP has provisions for scoped service discovery and multiple directories; however, the service description mechanism is intended for human users that require to access specific services by name. Hence, the naming problem is not resolved. Furthermore, the current design of the protocol is geared for enterprise users, which require to access relatively few and static services; it is not designed for a collaborative media distribution environment with many dynamic service access points, as we describe it in Section 2.1.

2.4.2 Active Networks and Active Service Frameworks

The main problem with session discovery and configuration is flexibility. The problem stems from constraints placed on applications by the Internet service model itself. On the one hand, the design of the Internet service model abstracts away how messages are forwarded through the network, divorcing applications from the complexity of the communication substrate. On the same time, in a system that follows the model and scale for collaborative media distribution, this design makes it hard for application to exploit detailed knowledge of the underlying network in order to enhance their performance and reduce network load.

For example, within the collaborative media distribution framework, we need to support dissemination of high level user queries which perform computation in order to deduce the suitability of a file description. Similarly, after locating a session identifier, we need to locate the closest potential sources in the network and perform configuration tasks. For scaling reasons, it is not possible to have preconfigured sessions for any given file in the network, have nodes constantly serving as sources for their locally stored files, or maintain a registry of active sessions. We need a mechanism for performing computation on demand, locally at each node of interest.

The active networks initiative [4, 78] sought to modify the Internet service model towards a programmable network architecture. In the active network model every entity in the network, especially routers, can be dynamically programmed to support new protocols. The IP service model is modified from a black box network transport to a fully configurable programming environment. The basic paradigm behind active networks is *mobile code* [27, 30]: executable code (usually interpreted) is transported in network packets and installed in routers. The installed code modifies the behavior of the router for specific types of packets, according to application specific semantics. Of the most successful active network toolkit is ANTS [76], which has been used for implementing a wide variety of routing and router-assisted transport protocols [75]. Of particular interest is the Active Reliable Multicast (ARM) [43] protocol. ARM implements a router-assisted variant of SRM, which increases the efficiency of local error recovery by leveraging router knowledge about network topology.

Despite the flexibility of the active network paradigm, it is not the only way that programmability can be introduced in the network. Active network bring a radical change in the IP service model, a model which has worked reliably for a long time and has been very widely

deployed. In addition, there are performance drawbacks, deployment problems, and serious security implications of enabling a fully programmable network core [74, 10].

Much of the flexibility of active networks can be accomplished by a less radical shift: a programmable service architecture [3]. With a programmable service architecture the core of the network and the IP service model remain unchanged. A programmable service architecture allows applications and user to download and execute code at strategic locations for the application. But instead of executing on routers, user code executes on normal hosts, preserving the end-to-end design principle of the Internet [28]. This model is known as *active services*.

In the context of our model of collaborative media distribution, an active service architecture can solve the session discovery and configuration problem, by turning all participating nodes into active service nodes. Instead of having a fixed interaction with the network, nodes allow mobile code to execute in their local address space. The code can carry high-level queries and session configuration parameters. In conjunction with the open model and scalability properties of IP-multicast we can construct a flexible and highly dynamic system to serve our purposes.

2.5 Concluding Remarks

In this chapter we set the background for the protocol architecture developed in this thesis. We developed a system model which captures the reality of digital media distribution in a highly networked world with a set theoretic approach. We discussed the peer-to-peer distribution paradigm, as set by first generation systems developed recently, and showed how the lack of a suitable transport protocol hampers the scalability and performance of the system. Then, we discussed research experience with multicast transport protocols and programmable networks. In the next chapter we show how we can reconcile this research experience with our system model, and develop a scalable high performance protocol architecture which leverages the redundancy and distributed nature of the model.

Chapter 3

A Protocol Architecture for Collaborative Media Distribution

We are now in position to outline our active protocol architecture for collaborative media distribution. Our architecture is heavily based on multicast and mobile code, and is built around an end-to-end programmable protocol for scalable media distribution. The reasons for desiring programmability in the protocol level stem from the active service paradigm. An application level protocol that supports embedding of code within messages is necessary for session discovery and configuration. At the same time, as we make no assumptions about the framing of the data or the security and privacy requirements of participating nodes, the protocol architecture must be flexible enough to support any choice for these options. Hence, instead of making a fixed protocol that operates on fixed data representation, we can construct protocol messages that contain mobile code for interpretation in the end nodes.

3.1 Building an Active Protocol Architecture

3.1.1 Protocol Design Considerations

There are a few considerations that need to be made with regards to protocol design. As we have mentioned, it is of interest to provide real-time streaming of media files, by meeting the framing schedule (Section 2.1). Real-time operation allows users to access media files as they are transported. For real-time streams, As we mentioned previously, Shenker argues in [68]

that the utility of the distribution has stepwise behaviour as a function of the actual transfer rate¹. If the transfer rate falls below a threshold, the utility sharply drops to 0. While this is certainly the case for the adaptation quality of current generation media codecs, this utility model is not directly applicable to our system model.

In our model, file access is not restricted to online playback. Rather, node users can access locally stored media offline. Similarly, nodes can act as future sources for the file. For these reasons, we constrain real-time operation to be a desired feature of the distribution protocol; the protocol should try to achieve reliability *and* real-time rate, but when this is not possible it should opt for reliable transmission of the file.

The distinction between real-time operation and reliability becomes sharper when network conditions are taken into account. Unfortunately, congestion is a phenomenon which occurs regularly in the Internet. Modern transport protocols are required to react to incipient congestion but decreasing the transmission rate in order to avoid congestion collapse. Real-time transport protocols that insist of maintaining their intrinsic rate in the presence of congestion are harmful for the network, and can have rippling effects that affect all nodes in an area of the network. Therefore the design of the protocol should incorporate congestion avoidance and control mechanism. Furthermore, routers are designed for giving congestion signals to network flows by dropping packets. To avoid further dropped packets and disruption of operation, the protocol should behave in a TCP-friendly manner.

In addition, we are designing for an operation environment with high node density and multiplicity of sources. The presence of multiple sources for a media file provide with an opportunity for traffic localisation, increasing protocol scalability. Hence the protocol should opt for local file transmission. The network effects of media provide for an additional incentive for traffic localisation. Popular files are expected to be widely propagated, providing ample opportunity for ad-hoc local caching.

Finally, care should be taken for the scalability of session discovery and configuration traffic. For the system to scale to large numbers of nodes, it must be able to sustain large number of session messages. Hence, a local caching mechanism is necessary for session information as well.

¹As experienced by the receiver.

3.1.2 Architecture Outline

With these design considerations in mind, we can outline the two components comprising our architecture:

- A dynamic session discovery and configuration protocol. The protocol allows us to locate session identifiers using high-level queries expressed as code, dynamically configure sources for a transport session, and locate information about session status. We achieve this by using a thin message specification; session messages do not specify the payload, and it is up to the end nodes to interpret them. The protocol operates in a completely decentralised manner, using scoped multicast packets for all messages. The protocol also uses opportunistic caching of query results at edge nodes and includes a traffic control mechanism for scaling with the number of sessions and nodes in the system.
- A distributed multicast transport protocol for dynamic M to N streaming. The protocol is designed with the semantics of the collaborative media distribution model in mind: multiple sources may be available in the system, some of them storing only a subset of frames for the target file, and multiple unsynchronised receivers may concurrently access it the same time. The protocol builds on ideas from SRM [24], TFMCC [77], and HPRTP [40], adapts to congestion in TCP-friendly manner, and on the same time attempts to leverage multiple source availability in order to achieve real-time operation. By using multiple sources in different network paths, the protocol can maintain the real-time rate with a dynamic layering scheme based on adaptive stream splitting and aggregation.

The protocol architecture is complemented by the MAST programming language. MAST is a new dynamic mostly functional language based on Scheme [41, 1], with transparently integrated support for mobile and distributed computation. The language is used in the prototype implementation as a scripting shell for the protocol architecture and a mechanism for expressing SDCP messages. However, we will not discuss it further as such a discussion is beyond the scope of the protocol architecture.

The remaining of the chapter discusses considerations behind protocol design and presents our basic protocols. We discuss the algorithms employed by the protocols and the exchange

of messages from a high level point of view. Details of protocol implementation are presented in the appendix.

3.2 Session Structure

Recall from the system model of Section 2.1 each content file is accessed in the network through a session, and that the set of active sessions is isomorphic to the set of active files. When a session is active it is not necessary that any data transport takes place or any node is configured to provide or receive data from it. Rather, an active session can be used for data transmission. Each session uses a multicast address for carrying configuration traffic between participating nodes. The multicast address is not statically assigned; when the need for accessing a session arises, a multicast address is allocated for the period of network activity and participating nodes are configured with the Session Discovery and Configuration Protocol (SDCP). SDCP is a soft-state protocol, that is the state of the session is retained as long as there is activity. In addition, node activation for a session is *localised*. Only nodes which have frames of the file in question locally stored in the neighbourhood of a node requesting session data need be configured and join the session.

Data flows in the session within *streams* of the Distributed Real-time Multicast Transport Protocol (DRMTP). Each stream represents a localised flow of file frames within a single multicast address. The frames that comprise the stream are a subset of the file frameset. That is, in a session $session(f)$ providing the file f , a stream s represents a flow for a frameset F_s^{stream} such that

$$F_s^{stream} \subseteq F_f \quad (3.1)$$

The set of streams in a sessions is denoted as $streams(session(f))$. At any given time, there is activity in a sessions if and only if $streams(session(f)) \neq \emptyset$.

Data in a stream flows in a single multicast address, locally allocated with SDCP. It should be stressed that streams are *local*; that is the data flow in a stream has a scope large enough to cover participating nodes only, and multicast addresses can be reused in different parts of the network for different streams. Each stream has a single source, which is aware of the frames that comprise it. The stream frame set, as perceived by the source, is updated as frames are transmitted. The frame set can also be explicitly modified by request of the sinks.

When the frame set becomes empty, that is all frames have been transmitted, the stream can be discarded.

A node can participate in a stream either as a *source* or a *sink*. Within the stream we maintain point-to-multipoint transmission semantics, with a source transmitting data and a sink receiving. A node however can participate in more than one streams at the same time, acting as a source in some streams and as a sink in others. Therefore, every sink in a stream can become a source in another one; and a node can *aggregate* data from multiple streams, acting as a sink.

There are two important consequences of this stream aggregation mechanism. First, we can consistently handle late joins in a stream, with local patching. And second, we can aggregate a number of slow interleaved streams to provide real-time operation for a node, even when none of the sources of each stream can support it. Figure 3.1 illustrates the first case, while Figure 3.2 illustrates the second.

In Figure 3.1, node N_1 is serving a stream S_1 for file f , where initially $F_{S_1}^{stream} \equiv F_f^{file}$. Nodes N_2 , N_3 and N_4 participate in the stream as sinks. Some time t after the stream flow has started, node N_5 joins the session for receiving the entire file. At the time of join, $F_{S_1}^{stream}(t) \subset F_f^{file}$, as some frames have already been served. N_5 joins S_1 , and on the same time receives the missing frames from N_3 in a new locally established stream S_2 . We explain later how this scenario is seamlessly accomodated with the DRMTP stream establishment algorithm. Note that the total number of packets in the network for transferring the file in this particular scenario is $E_p = 6|F_f^{file}| + 2|F_{s_1}^{stream}(t)|$. By comparison, if point-to-point connections were used, all provided by N_1 acting as a server, we would require $18|F_f^{file}|$ packets in the network.

Figure 3.2 illustrates real-time operation with aggregation of two half-rate streams. Recall from the model of Section 2.1 that a stream F_s^{stream} is real-time if the frames are delivered with a real-time schedule. Let F_f^{file} be the file of the session and R_f^{file} be the real-time schedule of the file and assume for the sake of the example that the frame-rate is uniform R frames per second, that is

$$\forall r_i, r_j \in R_f^{file}, r_{i+1} - r_i = r_{j+1} - r_j = \frac{1}{R} \quad (3.2)$$

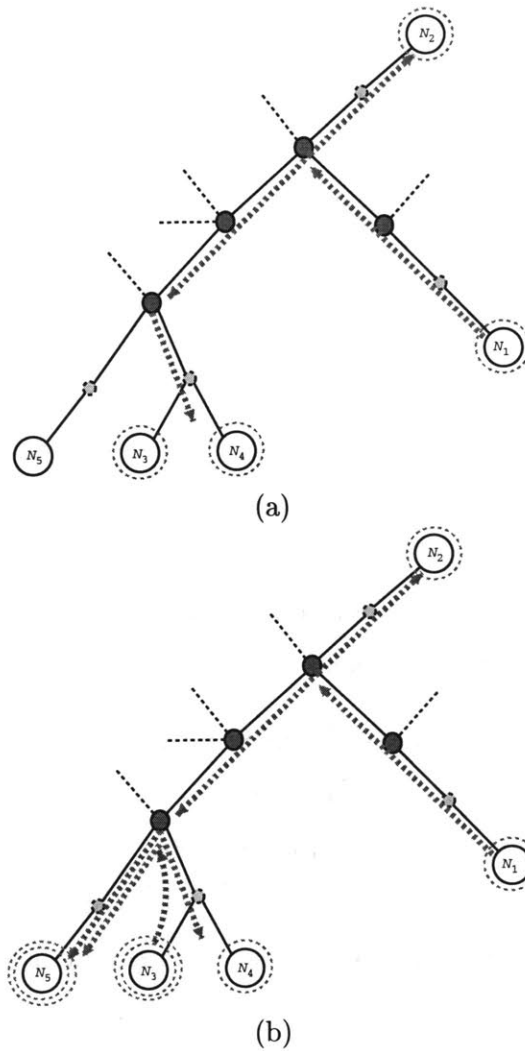


Figure 3.1: Local patching for late joins. (a) Initial configuration, with N_1 providing S_1 (red) to $N_2, N_3,$ and N_4 . (b) N_5 joins S_1 and N_3 locally patches with S_2 (blue).

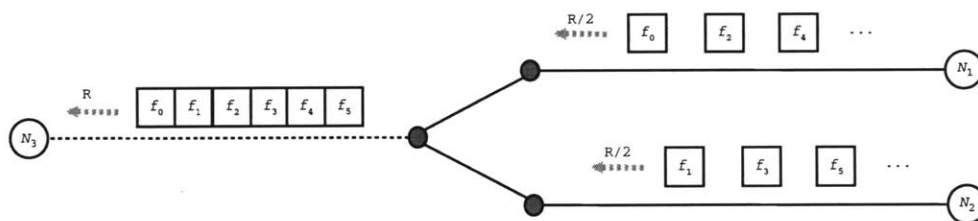


Figure 3.2: Real-time operation with aggregation of two half-rate streams.

Partition the file in two streams s_1 and s_2 , such that

$$F_{s_1}^{stream} = \{f_i | i = 2k \wedge f_i \in F_f^{file}\} \quad (3.3)$$

$$F_{s_2}^{stream} = \{f_i | i = 2k + 1 \wedge f_i \in F_f^{file}\} \quad (3.4)$$

If we deliver the two streams with the same schedule as the real-time schedule of the file, then each stream has a rate $R/2$. But the sink receives frames with an aggregate rate of R , because the two streams are interleaved. Hence, we can achieve real-time operation when the sink has enough downstream capacity, but the two sources have an upstream capacity $R/2$. Note that a similar scenario matches asymmetric residential connections. We later explain how stream splitting is handled by DRMTP as part of the congestion control algorithm.

3.3 Implosion Control

The key to scalability of multicast based systems is an effective mechanism for controlling implosion. Implosion can occur in several occasions, as we explain in later in this chapter:

- In a stream request by a sink, multiple sources may be able to satisfy the request.
- In establishing a new stream, multiple sources may concurrently request the same stream. This is especially the case when the new stream is established as a correction for missing frames.
- In providing feedback for controlling packet transmission within a stream.
- In congestion circumstances that affect multiple sinks.

The basic mechanism for controlling implosion is coordination among nodes. Unfortunately, nodes are not aware of each other, hence coordination needs to be performed without knowledge about competing nodes. We achieve these objectives by electing a single node as the feedback controller for scheduling packet transmission and using a randomised feedback suppression algorithm (Section 2.3.3).

Effectiveness of the feedback suppression algorithm depends on the distribution of timer intervals. As is shown in [53], the optimal timer for large groups is generated using a *truncated*

exponential distribution:

$$f_z(z) = \begin{cases} \frac{1}{e^\lambda - 1} \cdot \frac{\lambda}{T} e^{(\lambda/T)z}, & 0 \leq z \leq T \\ 0, & \text{otherwise} \end{cases} \quad (3.5)$$

where λ is a parameter of the distribution and T is the upper bound of the distribution. λ is chosen by an estimate of the number of nodes that may emit a message, while T is an upper bound on the *delay* of transmitting a response. The advantages of the exponential distribution are manifold. It is possible to avoid implosion while keeping the T interval small. In addition, the distribution is not very sensitive to the λ parameter, and can provide with good implosion control even if the parameter is misestimated by order of magnitudes. Finally, the effectiveness of implosion control is not sensitive to loss of feedback messages or heterogeneous delays between nodes. These characteristics make exponential distributed timers ideally suited for our system model, and are used for feedback suppression both on DRMTP and SDCP.

A timer interval with an exponential distribution can be generated with a uniformly distributed random value $\mathcal{U}[0, 1]$ as

$$\mathcal{T}[T, N] = T \cdot \log_N(1 + (N - 1)\mathcal{U}[0, 1]) \quad (3.6)$$

where N is chosen as an estimate of the number of competing nodes, as we explain in Chapter 4.

3.4 DRMTP: The Distributed Real-time Transport Protocol

DRMTP controls data flow in session streams. Here we present the algorithms employed by the protocol in terms of messages, assuming that the session has already been configured with SDCP (Section 3.5).

3.4.1 Stream Control

For each DRMTP stream, there is a single source and a primary controller elected during on stream establishment. The source (or sender) is passive and transmits frames in bursts according to a schedule provided to it by the primary controller. The primary controller is solely

responsible for scheduling. Therefore, even when there are more than one sinks (or receivers) receiving packets from a stream, we can maintain point-to-point semantics for feedback and control.

The controller schedules bursts by multicasting *schedule* messages to the stream group. The schedule message is of the form

$$\text{schedule} ::= (B \text{ schedule}(B) \hat{t}_{pc \rightarrow s}^{OT} \hat{v}_{pc \rightarrow s}^{OT})$$

Thus, each message contains the next burst to transmit, the schedule of the burst. $\hat{t}_{pc \rightarrow s}^{OT}$ is an estimate of the one way distance from the primary controller pc to the source s and $\hat{v}_{pc \rightarrow s}^{OT}$ is an estimate of its mean variance.

The schedule of a burst $B(f_{b_1} \cdots f_{b_n})$ containing frames $\{f_{b_1} \cdots f_{b_n}\}$ is of the form

$$\text{schedule}(B(f_{b_1} \cdots f_{b_n})) = t_1, \dots, t_n \quad (3.7)$$

where t_i is the transmission interval between frames $f_{b_{i-1}}$ and f_{b_i} , with t_1 the time interval from the receipt of the schedule.

After receiving the *schedule* message at time $t_{\text{schedule}(B)}^{rx}$ the source transmits the frames of the burst at times

$$t_{f_{b_i}}^{tx} = \max \{t_{f_{b_{i-1}}}^t, t_{\text{schedule}}^{rx}\} + \sum_{k=1}^i t_k \quad (3.8)$$

where $t_{f_{b_{i-1}}}^t$ is the transmission time of the last frame in the previous burst and t_{schedule}^{rx} is the arrival time of the schedule message. The schedule of frame transmissions in the burst follows the schedule of the file, unless the protocol is in congestion control mode. In the latter case the schedule is adjusted as we explain in Section 3.4.5. Hence, if R_f^{file} is the schedule of the file, the schedule of the burst will be

$$t_i = r_{b_i} - r_{b_{i-1}} \quad (3.9)$$

By multicasting the schedule to the entire group, we allow all sinks to compute distance samples from the source and the primary controller. The samples are used in selecting the schedule transmission times, setting time outs for detecting packet losses and primary controller or source failure.

With a burst size of at least two frames, the primary controller can obtain samples of

both the round-trip and one way trip time from the sender. Simultaneously, by observing the spacing of two sequential frames in a burst, every receiver in the stream can obtain a sample of the one-way trip time from the sender. We compute trip time estimates and mean deviation by using the same algorithm that TCP uses [59]. That is, for an estimator of the trip time \hat{t} , an estimator of the mean deviation \hat{v}_t and a sample m , we use the update rule

$$e = m - \hat{t} \quad (3.10)$$

$$\hat{t} \leftarrow \hat{t} + a \cdot e \quad (3.11)$$

$$\hat{v}_t \leftarrow a \cdot (|e| - \hat{v}_t) \quad (3.12)$$

where $a = \frac{1}{8}$ [38].

The controller computes a sample of the round-trip time sample from the source as

$$t_{pc \rightarrow s}^{RTT} = t_{f_{b_1}}^{rx} - t_{schedule}^{tx} - t_1 \quad (3.13)$$

where $t_{f_{b_1}}^{rx}$ is the arrival time of frame f_{b_1} , and $t_{schedule}^{tx}$ is the time when the schedule was sent to the group. Similarly, any receiver r can obtain a sample of its one-way trip time from the sender as

$$t_{s \rightarrow r}^{OT} = t_{f_{b_i}}^{rx} - t_{f_{b_{i-1}}}^{rx} - (t_i - t_{i-1}) \quad (3.14)$$

Notice that if a burst larger than two frames is used, a receiver can obtain multiple samples of the trip time. Hence, the primary controller can select the schedule transmission time by tracking the one-way trip time to the sender. If the last schedule has been transmitted at time $t_{schedule}^{tx}$, the next schedule $schedule'$ message will be transmitted at time

$$t_{schedule'}^{tx} = t_{schedule}^{tx} + \sum_{k: f_{b_k} \in B} t_k - (\hat{t}_{pc \rightarrow s}^{OT} + 4\hat{v}_{pc \rightarrow s}^{OT}) \quad (3.15)$$

to accommodate for one-way trip time variance.

A Receiver r can obtain a sample of its distance from the primary controller pc as

$$t_{r \rightarrow pc}^{OT} = t_{pc \rightarrow s}^{OT} + t_{s \rightarrow r}^{OT} \quad (3.16)$$

where $t_{pc \rightarrow s}^{OT}$ is the one-way trip time estimate from the primary controller to the sender, and $t_{s \rightarrow r}^{OT}$ is the one-way trip time estimate from the sender to r . Obtaining the first sample is simple, if the primary controller piggybacks distance estimates to schedule announcements.

3.4.2 Stream Establishment

A sink uses the SDCP to configure the session, and then proceeds to locate a stream for receiving frames. The session control protocol provides the node with the *session group*. The session group is a multicast group used for locating and establishing streams within the session. All sources configured in the session are members of the group, listening for stream requests while there is activity in the session. Sources use SDCP periodically for detecting when the session is inactive and they should stop waiting for stream requests.

The stream establishment algorithm operates in rounds. Each round has four phases: request, bid, accept, and announce/retract. The sink will engage up to a maximum number of rounds, until enough streams have been found to provide the entire frame set. For a receiver, this is Algorithm 1, where $F^{request}$ is the request frame set. The details of the algorithm, together with the respective actions by the actions of a source, are described in the sequel.

Algorithm 1 Stream establishment by a DRMTP receiver

```

 $d \leftarrow 1$ 
repeat
  request ( $F^{req}$   $d$ ) at depth  $d$ 
  receive set of bids,  $bids$ 
  if  $bids \neq \emptyset$  then
    select bids  $accept$ , such that  $\forall a, a' \in accept, a \cap a' = \emptyset$ 
    accept bids in  $accept$ 
    receive announcements,  $announce$ 
    join announced streams
     $F^{req} \leftarrow F^{req} - \bigcup_{s \in announcements} F_s^{announce}$ 
    if  $\forall b, b' \in bids, b \cap b' = \emptyset$  then
       $d \leftarrow d + 1$ 
    end if
  end if
until  $d > D_{session}$  or  $F^{req} = \emptyset$ 

```

Entering a request phase, the sink node is unaware of any sources present or any streams flowing in the network. Streams are discovered using *Expanding Ring Search* [15]: the node sends stream request messages with increasing scope, with a round completed within a scope

of the expanding ring search. Each successive *request* message contains the current scope of the request, and the set of frames F^{req} that the stream should provide:

$request ::= (F^{req} \ d)$

The source can locate one or more streams that will provide the requested frame set. After sending a request at scope depth d , the node deterministically waits for an interval $W_{max,d}$, where $W_{max,d}$ is a parameter used by the sources in randomising bid offers in scope d as we explain below. If at least one bid is received, the receiver proceeds to the accept phase. Otherwise the receiver increases the scope to $d+1$. If the scope has not exceeded the diameter of the session, as provided by the session control protocol, the receiver repeats the bidding process in the expanded scope.

In order to avoid request implosion, the request phase includes a request suppression algorithm (Algorithm 2). Before a receiver r tries to establish a stream for a frame set F^{req} , it waits for a random interval W_{req} . The timer is selected as $\mathcal{T}[W_{max,d}, N_d]$. For any request received by a receiver r' during this interval, the frame set $F_{r'}^{req}$ is extracted by the request frame set. Thus implosion is avoided when multiple receivers are trying to establish a stream for compatible frame sets.

Algorithm 2 Request suppression for DRMTP streams

```

 $F_r^{req} \leftarrow F^{req}$ 
set request timer  $W_{req} \leftarrow \mathcal{T}[W_{max,d}, N_d]$ 
repeat
  if receive request for  $F_{r'}^{req}$  and  $F_{r'}^{req} \cap F_r^{req} \neq \emptyset$  then
     $F_r^{req} \leftarrow F_r^{req} - F_{r'}^{req}$ 
  end if
until request timer expires
if  $F_r^{req} \neq \emptyset$  then
  send request  $F_r^{req}$ 
end if

```

On receiving a request, a source checks if it can provide some of the frames. If so, the bidding phase is entered. The node waits for a random interval and makes a bid for a stream. Each *bid* message contains a description of the set of frames F^{bid} that can be provided by the stream, and additional information whether the stream is already actively flowing in the network:

$bid ::= (F^{bid})$

If the source is already serving a *compatible* stream, that is a stream that includes some of the frames in the request frame set (i.e. a stream with a frame set that is not disjoint with the request frame set), it immediately re-announces the stream. Similarly, if there is an already pending bid that is compatible with the request, that bid is reused – the node need not submit two bids, as both requests will be covered by the first one. Otherwise, the source constructs a bid for a new stream that contains the maximal set of frames that are included in the request, can be provided by the source, and are not part of a compatible stream that is already being served or is in bidding process. This is described by the equation

$$F^{bid} = F^{req} - \bigcup_{s \in streams} F_s^{stream} - \bigcup_{b \in bids} F_b^{bid} \quad (3.17)$$

where *streams* denotes the set of active streams sourced at the node, and *bids* denotes the set of pending bids.

After deciding on the bid, the source sets a wait timer and submits the bid after it expires. The wait interval W_{bid} is again selected randomly:

$$W_{bid} \leftarrow \mathcal{T}[W_{max,d}, N_d] \quad (3.18)$$

where W_{max} is a maximum wait time and d is the depth of the scope. Randomisation is used for two purposes. First, there may be multiple stream requests that can be partially served by a single stream. Second, and most important, there may be multiple sources reachable in the scope. By randomising the response interval, we can avoid implosion by multicasting all bids to the stream control group. Other sources can hear a bid offer, and suppress their bid.

When a source m waiting to submit a bid for a frame set F_m^{bid} receives a competing bid message F_n^{bid} or announcement from a source n , it compares the stream description with the stream that it can provide. Depending on the description, it can choose to offer an alternate bid or suppress the bid altogether. If the source is not already serving the stream (i.e. the bid will create a new stream), the action taken by m is determined by Algorithm 3. If the stream is active, then the only modification is that the scope of the stream is increased to $\max d_{req}, d_{stream}$.²

²We assume that the forward and reverse path length from source to sink is the same. The increase can be adjusted to accommodate routing protocol and topology details.

Algorithm 3 Bid supression for new DRMTP streams

```

 $\Delta F \equiv F_m^{bid} - F_n^{bid}$ 
if  $\Delta F = \emptyset$  then
  suppress bid
else if  $F_n^{bid}$  is not for an active stream then
  if  $F_n^{bid} \cap F_m^{bid} \neq \emptyset$  then
     $F_m^{bid} \leftarrow \Delta F$ 
     $d_{bid} \leftarrow \max d_{bid}, d_{req}$ 
    maintain bid timer
  else
    maintain bid timer
  end if
end if

```

In the acceptance phase, the requesting node selects a subset of the offered streams to accept. Ideally, by the bid supression algorithm, the bid offers should contain disjoint subsets of the requested frame set. Nevertheless, due to the concurrency and randomisation of the bidding procedure, it may happen to have non-disjoint elements in the bid set. Non-disjoint bids are ignored, and the frame set they provide is left for resolution at the next round of the stream establishment algorithm. The depth of the scope for the next round will remain unchanged if the elements of the bid set were not pairwise disjoint, otherwise it will be increased by 1.

After selecting the set of bids to accept, the node waits for a random interval W_{accept} drawn with the same distribution as the bid wait time:

$$W_{accept} \leftarrow \mathcal{T}[W_{max,d}, N_d] \quad (3.19)$$

The reasons for the randomisation is once again implosion avoidance when multiple sinks accept the same bid. Any stream acceptance or announcement that is received during the wait interval supresses an acceptance that is in the acceptance set. After the interval expires, the node sends an *accept* message to the stream control group for each element of the acceptance set that has not been suppressed.

On receiving an acceptance for a submitted bid, the source of the bid needs to establish the stream, select the primary controler, and send an announcement for the stream. If a new stream is established, the frame set specified in the bid is used and the sender of the *accept* message is selected as a primary controler. Otherwise, the source checks the current status of

the stream and uses the current frame set and primary controller in the announcement. The announcement also includes the depth of the stream, so that the primary controller can adjust the scope of schedule transmission:

$announce ::= (\langle address \rangle \langle controller \rangle F^{stream} d)$

The announcement is delayed by

$$W_{announce} \leftarrow \mathcal{T}[W_{max,d}, N_d] \quad (3.20)$$

where d is the depth of the stream. The frame set of any compatible announcement received during the announce interval will be removed from the frame set of the announced stream. When the timer expires, if the frame set is not empty, the stream is announced and created at the node. If the frame set is empty, the announcement is suppressed, and the stream is not created. In addition, *actively* flowing streams can be *passively* announced by sinks participating in the stream. If a node receives a request compatible with a stream on which it is participating as a sink, it schedules an announcement for the stream with a delay $W_{announce}$. The frame set of the pending announcement is similarly modified by compatible announcements – if it becomes empty the announcement is suppressed.

Finally, each receiver ‘closes’ the announcement phase on its side by joining any announced streams in the acceptance set and obtaining control for those streams that has been designated as the primary controller.

3.4.3 Detection of Error Conditions

During the lifetime of a stream there are four classes of error conditions that can arise:

- Frame loss for some receivers. Frame loss occurs when packets are lost in some network links, but with a rate that does not signify congestion. In general, receivers will perceive different frame loss rates, as the paths from the source may differ.
- Persistent congestion for some receivers. Congestion occurs when losses for some receivers exceed a rate to be defined below.
- Primary controller failure. The primary controller may unexpectedly fail, leaving the stream without a scheduler. Frames cannot be transmitted from the source without a

primary controller providing a schedule.

- Source failure. Sources may also fail unexpectedly, ending the stream abruptly.

A receiver can detect frame loss in two ways: By time out, and by receiving a frame out of order with regards to the schedule. Missing frames are added to a list. Receivers can detect out-of-order delivery even if the schedule message has been lost, as the frame set of the stream is known by all participating nodes. Periodically, and if no persistent congestion has been detected, receivers attempt to recover the missing frames as we describe in Section 3.4.4.

The frame reception timeout is calculated conservatively, using the distance of the primary controller to the source and the distance of the receiver from the source. Upon receiving a schedule message from the controller, a receiver r computes the time-out for frame f_i in the schedule as

$$timeout_i^{rx} = t_i + \hat{t}_{pc \rightarrow s}^{OT} + 4\hat{v}_{pc \rightarrow s}^{OT} + \hat{t}_{s \rightarrow r}^{OT} + 4\hat{v}_{s \rightarrow r}^{OT} \quad (3.21)$$

The first three quantities are included in the schedule message, while the last two are calculated by the receiver. The primary controller itself has a more accurate timeout:

$$timeout_i^{pc} = t_i + \hat{t}_{pc \rightarrow s}^{RTT} + 4\hat{v}_{pc \rightarrow s}^{RTT} \quad (3.22)$$

When a lost frame is detected by a receiver, a *loss event interval* is initiated. The loss interval includes the number of successfully received frames between losses. Each receiver r maintains the length of past few loss intervals, and uses them for computing the average loss interval length $\overline{l_{r,k}}$ as a weighted moving average of m most recent loss intervals $l_k, \dots, l_{r,k-m+1}$:

$$\overline{l_{r,k}} = \frac{\sum_{i=0}^{m-1} w_i l_{r,k-i}}{\sum_{i=0}^{m-1} w_i} \quad (3.23)$$

This is the same as the approach is used by TFRC [25] and TFMCC [77]. From $\overline{l_{r,k}}$ we can compute a loss event rate at each receiver at any time as

$$\hat{p}_r = \frac{1}{\max\{\overline{l_{r,k}}, \overline{l_{r,k-1}}\}} \quad (3.24)$$

where the previous average loss interval length is included in the computation as the current loss interval may not yet be complete.

When intermittent packet loss occurs, there is no reason to adjust the transmission rate of the stream; receivers can simply recover the lost frames. But when packet loss occurs because of congestion, the stream rate should be reduced in a TCP-friendly manner. In order to detect congestion, we compute the equivalent throughput of a TCP stream operating under similar conditions. An approximation of the TCP throughput in relation to the loss rate and round-trip time is given by Equation 2.7 in packets per second:

$$R^{TCP} = \frac{1}{t^{RTT} \sqrt{p} (\sqrt{2/3} + 6\sqrt{3/2} p (1 + 32p^2))} \quad (3.25)$$

A problem with this equation is what the round-trip time actually is. In TCP, the round-trip time has well defined point-to-point semantics. We can't say the same for the multicast case though. Should the RTT be the RTT of a particular receiver? Or should each node use a separate RTT estimate? How do we define the RTT and how do we calculate it?

In DRMTP, similar to TFMCC, we choose to treat each receiver independently and decide that a receiver is in a congested path if the protocol throughput exceeds the throughput of an equivalent TCP connection for the same packet loss rate and RTT. Every receiver detects congestion separately, and based solely on local knowledge. This decision is compatible with the fact that only a subset of the receivers in the stream may experience congestion. Receivers compute the equivalent throughput of a TCP connection by assuming point-to-point semantics to the source. Thus, every receiver treats the stream as a point-to-point connection to the source. The primary controller uses the real RTT estimate, while other receivers assume path symmetry and compute the RTT as

$$\hat{t}_{s \rightarrow r}^{RTT} = 2\hat{t}_{s \rightarrow r}^{OT} \quad (3.26)$$

Hence, each receiver r computes an equivalent TCP connection throughput as

$$\hat{R}_r^{TCP} = \frac{1}{\hat{t}_{s \rightarrow r}^{RTT} \sqrt{\hat{p}_r} (\sqrt{2/3} + 6\sqrt{3/2} \hat{p}_r (1 + 32\hat{p}_r^2))} \quad (3.27)$$

A receiver decides that is suffering from congestion when the protocol rate exceeds the throughput of the equivalent TCP connection in the loss interval:

$$\hat{R}^{DRMTP} > \hat{R}_r^{TCP} \quad (3.28)$$

where \hat{R}^{DRMTP} is the DRMTP throughput in frames per second within the duration T_{loss} of the m most recent loss intervals. The DRMTP throughput can be computed as

$$\hat{R}^{DRMTP} = \frac{\# \text{ of frames in } T_{loss}}{T_{loss}} \quad (3.29)$$

This decision ensures that the protocol will not be more aggressive than an equivalent TCP connection in the same conditions. On the other hand, if the rate of the protocol is less than that of a competing TCP connection, the protocol should not yield to packet loss. Finally, we use the set of known transmitted packets in the throughput computation because the protocol is rate-based. Unlike TCP, which is self-clocked and only transmits a packet when an ACK is received, DRMTP will transmit packets with a specific rate in open loop. Therefore, any packet transmitted loads the network and should be taken into account in the rate computation regardless of whether it is acutally received.

The last two error conditions result to total disruption of the stream. When the primary controler fails, no schedule or frames are received in the stream. When the source fails, schedule messages are received in the stream, but no frames from the source. The primary controler publishes the schedule according to Equation 3.15. Therefore, a source failure can be detected when after a number of schedule messages have been received, no frames from the source have been received. Similarly, a receiver r expects to receive a new schedule $schedule'$ at most $t_{schedule'}^{rx}$ after the previous schedule message, with

$$t_{schedule'}^{rx} = t_{pc \rightarrow r}^{OT} + \sum_{f_i \in schedule} t_i \quad (3.30)$$

A receiver decides that the primary controler has failed if there are no frames or schedule received $\beta \cdot t_{schedule'}^{rx}$ after the last schedule, where β is a threshold parameter. If frames are received, but no schedule, the receiver deduces that the schedule message was lost and resets the timer.

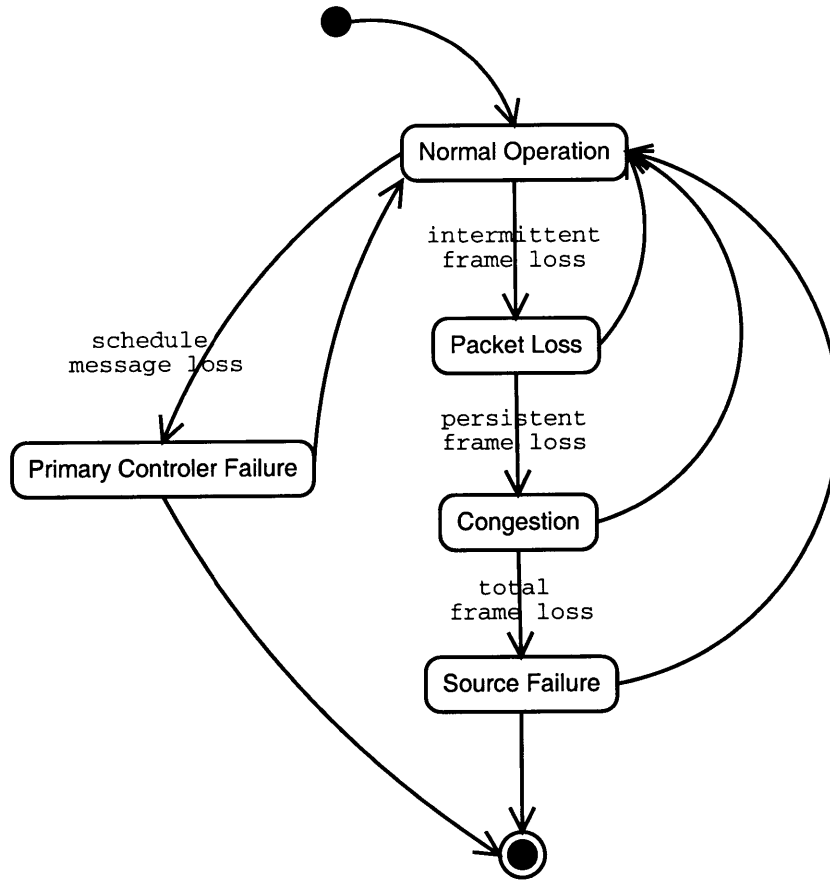


Figure 3.3: Stream error conditions.

Figure 3.3 summarises the relationship between error conditions in the stream, with a state transition diagram. The actions taken on each condition are described in the sequel.

3.4.4 Error Correction

Within the stream, there are never any frame retransmissions. A frame may be lost only for a subset of the receivers, so retransmitting the frame to all the receivers is unnecessary. Furthermore, after transmitting a frame, the source removes it from the stream specification. Thus, the stream can never revisit a frame that has been transmitted previously, clarifying both the semantics of the stream specification, and setting the stage for local error correction.

Error correction is handled by each receiver independently establishing a new stream for receiving missing packets. The scheme is a local error correction scheme, without any addi-

tional effort. Every node in the stream can act as a source, and by the stream establishment algorithm a local source will be discovered first.

When a frame loss is detected, a receiver initiates a loss interval. In order to distinguish intermittent error conditions from persistent congestion, a receiver will not attempt error correction before m loss intervals have been completed, where m is the same m used at average loss rate computation above. At the end of the interval, the receiver computes the protocol and loss rate of the interval and the equivalent TCP rate. If the protocol rate is higher than the equivalent TCP rate, the receiver decides that it is experiencing congestion (Equation 3.28), and congestion avoidance must be initiated for the stream. Congestion avoidance cannot be handled locally, as the rate of the stream must be adjusted. On the other hand, if the receiver decides that it does not experience congestion, it engages in error correction by establishing a correction stream. A delay of $W_{correct}$ is added before establishing the correction stream, to allow congestion information to be propagated in the stream, using *congestion* messages.

The process is summarised in Algorithm 4. Note that the request algorithm in stream establishment involves a supressions scheme (Algorithm 2 above). Therefore implosion is avoided even if multiple receivers perceive the same losses and simultaneously proceed to establish a correction stream. Note that since the node is maintaining an estimate of the distance to the source, it can fine-tune the selection of $W_{max,d}^{correction}$:

$$W_{max,d}^{correction} \propto \hat{t}_{s \rightarrow r}^{RTT} \quad (3.31)$$

3.4.5 Congestion Control

Although congestion may affect only a subset of the receivers in the stream, information is propagated with the *congestion* message so that it can be globally dealt with. The rate of the stream should be adjusted globally so that there are no congested receivers. An alternative would be to force the congested receivers out of the stream and have them establish an alternative one. This approach will not work however because of router slack time. That is, even if congested receivers leave the stream, routers keep forwarding packets for some additional time – which may be in the order of *minutes*.

Congestion avoidance and control is initiated by the primary controler. When a *congestion*

Algorithm 4 Error correction for a DRMTP receiver

```

if  $f$  is lost then
   $F_{err} \leftarrow \{f\}$ 
  initiate loss interval  $W_{loss}$ 
   $congestion \leftarrow false$ 
  repeat
    if  $f$  is lost then
       $F_{err} \leftarrow F_{err} \cup \{f\}$ 
    end if
    if receive congestion message then
       $congestion \leftarrow true$ 
    end if
  until  $congestion = true$  or loss interval ends
  if  $congestion = true$  then
    initiate congestion control
  else
    compute  $\hat{p}_{loss}$ ,  $\hat{R}_{loss}^{DRMTP}$ ,  $\hat{R}_{loss}^{TCP}$ 
    if  $\hat{R}_{loss}^{DRMTP} > \hat{R}_{loss}^{TCP}$  then
      send congestion message
      initiate congestion control
    else
      initiate congestion announcement interval  $W_{loss}$ 
      repeat
        if receive congestion message then
           $congestion \leftarrow true$ 
        end if
      until  $congestion = true$  or congestion announcement interval ends
      if  $congestion = true$  then
        initiate congestion control
      else
        establish stream for  $F_{err}$ 
         $F_{err} \leftarrow \emptyset$ 
      end if
    end if
  end if
end if

```

message is received, the controller adjusts the stream rate for a recovery period. After the recovery period, the original rate is reinstated. If congestion reappears within a $W_{recover}$ interval, then some of the $s \rightarrow r$ paths in the stream are permanently congested. The controller then adjusts the stream rate again and attempts to split the stream. If the stream split attempt is successful, the controller informs the receivers about the new stream and adjusts the frame set of the original split. If the split is unsuccessful, the controller continues with the original stream in reduced rate and periodically attempts to reinstate the original rate.

To be more specific, let $F^{stream} = \{f_i \cdots f_j\}$ be the stream frame set and $schedule(F^{stream}) = \{t_i, \dots, t_j\}$ the original long term stream schedule. On receiving a congestion announcement the primary controller schedules frames so that the rate of the stream is half the original rate. Thus, the schedule becomes

$$schedule^{(c,1)}(F^{stream}) = 2 \cdot schedule(F^{stream}) = \{2t_i, \dots, 2t_j\} \quad (3.32)$$

where the notation $schedule^{(c,1)}$ implies that this is the first congestion back-off for the stream. After $W_{recover}$, the controller resets the schedule to $T_{stream}^{(c,0)} \equiv T_{stream}$. If a new congestion report arrives, the rate is set again to $schedule^{(c,1)}(F^{stream})$ and a partition is prepared. A split $F_1^{stream}, F_2^{stream}$ is prepared so that

$$F_1^{stream} \cap F_2^{stream} \equiv \emptyset \quad (3.33)$$

$$F_1^{stream} \cup F_2^{stream} \equiv F^{stream} \quad (3.34)$$

and the rate of the two streams is approximately equal:

$$\forall i, j, f_i \in F_1^{stream}, f_j \in F_2^{stream}, t_{i+1} - t_i \approx t_{j+1} - t_j \quad (3.35)$$

The controller then sends a unicast *exclude* message to the sender, which instructs the sender to ignore stream requests for F_2^{stream} , and attempts to establish a new stream for F_2^{stream} using the stream establishment algorithm. Since the sender has been excluded from F_2^{stream} , a discovered stream will likely have different network paths than the original stream. To ensure that alternative sources in the same LAN as the same source do not bid for the stream, as they would have exactly the same network paths and suffer from the same conges-

tion conditions, the sender re-multicasts the exclude message with a TTL scope of 1^3 .

If the primary controller is successful at establishing the new stream, it multicasts a *join* message to the current stream, instructing the members of the stream to join a newly announce stream and abort the current. Simultaneously, the sender removes F_2^{stream} from the stream specification, converting the stream frameset to F_1^{stream} .

If the attempt fails, the controller enters a congestion control phase, where it attempts to restore the stream in decreasingly frequent intervals. On future congestion build ups, the controller will not attempt to split the stream. Rather, it will use directly the rate adaptation scheme. Again, let l be the current congestion level, with $l = 1$ after the failure to split the stream. The rate adaptation between level transitions is always is a division by 2 scheme:

$$\forall t_i \in schedule^{(c,l+1)}(F^{stream}), t'_i \in schedule^{(c,l)}(F^{stream}), t_i = 2t'_i \quad (3.36)$$

The adjustment attempts are made in exponentially decreasing intervals. If k is the number of failed increase attempts since the level was entered, then the next attempt will occur at

$$W_{recover}^{l,k+1} = 2^{l+k+1} W_{recover} \quad (3.37)$$

with the grace period always being $W_{recover}$

3.4.6 Source and Primary Controller Failure Recovery

Receivers react similarly to both primary controller and source failure: they abort the current stream and proceed to establish a new stream. If F^{rem} is the frame set remaining to be transmitted in the stream, and F^{err} is the error current error set, the receiver establishes a new stream for $F^{rem} \cup F^{err}$. The request suppression algorithm in the request phase ensures that implosion is controlled by the concurrent decision of all receivers in the stream to abort it.

An important detail in the case of primary controller failure, is that the source should also detect the failure and disband the stream. If the source has not decided that the primary controller has failed, it will bid for the existing stream without assigning a new controller. This

³Unless the exclude message is lost. The primary controller can detect loss of the exclude message if it receives a stream offer from the excluded source.

is handled if the source tracking schedule messages by the primary controller. When a schedule message has not been received for some time the source decides that the primary controller has failed and disbands the stream.

3.5 SDCP: The Session Discovery and Configuration Protocol

The Session Discovery and Configuration Protocol is a bootstrap protocol. It provides with a well known service access point, as a multicast group and port, for dynamically locating and configuring sessions in the system. SDCP provides a few basic operations: File discovery, address discovery and allocation, and source configuration. With these operations, a node can locate a file and a session identifier from a high level query, locate the session group and configure sources for the session.

3.5.1 File Discovery

Given a high level query, SDCP discovers the identifier $ident(f)$ of a matching file f . The file identifier can be then used as a session identifier for the remaining operations. In order to provide scalable operation for multiple concurrent discovery operations, the protocol uses a randomised feedback suppression algorithm and opportunistic local caching.

A query $query$ is a boolean function that operates on file descriptions:

$$query: desc(f) \Rightarrow \langle \text{boolean} \rangle$$

Queries are embedded in $find$ messages, which elicits $match$ responses from nodes where the query is successful:

$$find ::= (query\ d)$$

$$match ::= \{(ident(f)\ desc(f))\}^*$$

where d is the current scope of the find message. $find$ messages are transmitted with expanding scope until a desired number of responses has been located or a maximum scope has been reached. When a node receives a $find$ message, it evaluates the query function on the local descriptions. Local descriptions include the descriptions of all files in local store, and *cached* results from previous matches. On locating a match, a node multicasts a $match$ message to the SDCP group.

In order to avoid implosion from multiple matching nodes, a probabilistic feedback suppression mechanism is employed once again. Before transmitting a *match* message, a matching node sets an exponentially truncated timer (Equation 3.5). If any other matches are received during the wait, the node checks the identifier of the match message. If the identifier is part of the local match set, the *match* message is suppressed for the matching files. Otherwise, the node maintains the timer for the match and probabilistically caches the query result for future queries. The complete algorithm employed by a matching node is shown in Algorithm 5. Notice that we don't specify the form of the query. User code can be attached in the query messages, and be interpreted in the current node. The protocol is only interested about the result of evaluation.

Algorithm 5 Caching and suppression of SDCP *match* messages.

```

match(query)  $\leftarrow \{(ident(f), desc(f)) | \exists f_i \in F_f^{file}, store(n, f_i) \wedge query(desc(f))\}$ 
match(query)  $\leftarrow match(query) \cup \{(ident(f), desc(f)) | ((ident(f), desc(f)) \in cache(n) \wedge query(desc(f)))\}$ 
set timer  $W_{match} \leftarrow \mathcal{T}[W_{max,d}, N_d]$ 
repeat
  receive match',  $\{(ident(f) desc(f))*\}$ 
  for all  $(ident(f) desc(f)) \in match$  such that  $query(desc(f))$  do
    if  $(ident(f), desc(f)) \notin match(query)$  then
      with probability  $p$ ,  $cache(n) \leftarrow cache(n) \cup (ident(f) desc(f))$ 
    end if
  end for
   $match(query) \leftarrow match(query) - match'$ 
until  $match(query) = \emptyset$  or timer expires
if  $match(query) \neq \emptyset$  then
  send match(query)
end if

```

3.5.2 Address Discovery and Allocation, and Source Configuration

The address discovery and allocation operation is twofold. Given a session identifier $ident(f)$, the protocol attempts to locate the session group. If the attempt fails, the protocol selects a new address for the session and announces it to the network. The address allocation persists for as long as there is data flowing in DRMTP streams for the session. Like most of the algorithms employed by DRMTP and SDCP, address discovery proceeds with an expanding ring search and includes a randomised feedback suppression mechanism (Equation 3.5).

Source configuration is handled by an announcing the discovered address to the session group. A node within the scope of the announcement that has segments of the file in local scope becomes an active source, listening for stream requests in the session group. The node remains active as long as there are announcements pertaining to the session or is serving a data stream.

Note that we do not bind to a specific method for allocating the actual address. Multicast address allocation is an active area of development [33, 70] - any low level protocol that provides the necessary functionality can be used for this purpose.

3.6 Concluding Remarks

In this chapter we presented an active protocol architecture for collaborative media distribution. We outlined the basic components and delved into the details of the actual protocols and algorithms, preparing for the performance analysis and evaluation of Chapter 4. The basic component is the DRMTP protocol; the protocol allows for dynamic splitting and aggregation of multiple concurrent streams, flowing from multiple sources. Combined with TCP-friendly congestion control and scalable feedback mechanisms, the protocol offers a scalable substrate for soft real-time delivery of media streams. DRMTP is complemented by SDCP, which handles session discovery and address allocation. Both protocols provide programmable flexibility for the application: DRMTP scheduling and frame structure and SDCP query structure and interpretation are left to the application.

Chapter 4

Analysis and Evaluation

In this chapter we analyse and evaluate the protocol architecture characteristics. Initially, we analyse protocol scalability with the effectiveness of implosion by exponentially distributed timers. Then, we explore protocol behavior in a real network testbed, where we can show how the protocol scales with network effects of media distribution. We close the analysis with experiments performed with the *ns* [88] network simulator, evaluating congestion control and protocol fairness for competing TCP traffic.

4.1 Scalability and Latency Bounds

The most important factor in determining the scalability of our protocol architecture is the effectiveness of implosion control. We use implosion control in every part of the protocols where a message or condition may generate multiple new messages. These situations range from error correction and congestion control, to stream establishment and session discovery (Section 3.3).

The factor that determines the scalability of a feedback suppression algorithm is the timer distribution. In Section 3.3 we argued that the truncated exponential distribution is an optimal choice. We now quantify this choice and explain the derivation of Equation 3.6, based on the analysis of [53].

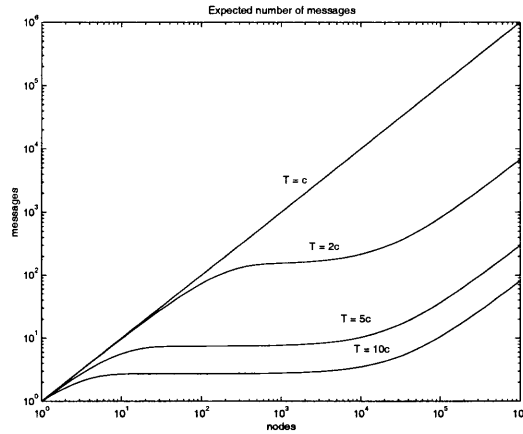


Figure 4.1: Expected number of feedback messages from an exponentially distributed timer, as a function of the number of feedback nodes, for $\lambda = 10$

4.1.1 Number of Feedback Messages

The expected number of feedback messages $E\{X\}$ from the distribution of Equation 3.5 is

$$E\{X\} = \begin{cases} N, & c \geq T > 0 \\ N \cdot \frac{e^{\lambda(c/T)} - 1}{e^{\lambda} - 1} - e^{\lambda(c/T)} \cdot \left(\left(\frac{1 - e^{-\lambda(c/T)}}{1 - e^{-\lambda}} \right)^N - 1 \right), & 0 < c < T \end{cases} \quad (4.1)$$

where c is the distance from the N nodes that may emit feedback messages. The assumption of homogeneous distance from feedback nodes results to an upper bound on the number of messages generated in heterogeneous cases, as explained in [53], and we will use it in the remaining of the analysis.

The number of feedback messages is almost exclusively controlled by the selection of the maximum interval T in relation to the trip time from the feedback nodes. Figure 4.1 draws the expected number of feedback messages with a parameter of T , and a value of $\lambda = 10$. As we can see, for an interval size of $T = 10c$, suppression is effective to $E\{X\} < 3.5$ for a range of up to 10^4 nodes.

The parameter choice of $\lambda = 10$ is not accidental. As the authors show in [53], the expected number of messages is a convex function of N , with a global minimum. The optimal choice of λ is close to

$$\lambda_0 = 1.1 \cdot \ln N + 0.8 \quad (4.2)$$

Furthermore, the function is not very sensitive to the actual choice of λ . Based on these observations, we can select the parameter of the distribution with a rough overestimate of the number of nodes that may emit feedback. For a rough estimate \hat{N} , we can simply choose

$$\tilde{\lambda} = \ln \hat{N} \quad (4.3)$$

Therefore, a choice of $\lambda = 10$ corresponds to a scale of 10^4 feedback nodes.

Hence, we can derive the generator function of Equation 3.6 as following:

$$F(z) = \mathcal{U}[0, 1] \Rightarrow \quad (4.4)$$

$$\mathbf{u} = \begin{cases} 0, & \mathbf{z} < 0 \\ \int_0^{\mathbf{z}} \frac{1}{e^\lambda - 1} (\lambda/T) e^{(\lambda/T)z} dz, & 0 \leq \mathbf{z} < T \\ 1, & \mathbf{z} \geq T \end{cases} \Rightarrow \quad (4.5)$$

$$\mathbf{u} = \frac{1}{e^\lambda - 1} (e^{(\lambda/T)\mathbf{z}} - 1) \Rightarrow \quad (4.6)$$

$$\mathbf{z} = \frac{T}{\lambda} \cdot \ln(1 + (e^\lambda - 1)\mathbf{u}) \Rightarrow \quad (4.7)$$

$$\mathbf{z} = \frac{T}{\ln N} \cdot \ln(1 + (e^{\ln N} - 1)\mathbf{u}) \Rightarrow \quad (4.8)$$

$$\mathbf{z} = T \cdot \log_N(1 + (N - 1)\mathbf{u}) \quad (4.9)$$

4.1.2 Feedback Latency

The other performance metric of interest is feedback latency. There is a tradeoff between the effectiveness of feedback suppression and latency. Specifically, the expected latency $E\{M\}$ for exponentially distributed timers is

$$E\{M\} = T \int_0^1 \left(1 - \frac{e^{\lambda m} - 1}{e^\lambda - 1}\right)^N dm \quad (4.10)$$

Figure 4.2 draws the expected feedback latency as a function of the number of nodes and parameter T . Once again, $\lambda = 10$. As we can see, there is a tradeoff between feedback latency and suppression: The better the suppression of feedback messages, the higher the latency. A selection of $T = 10c$ offers a good tradeoff: the latency is small – in the order of a few round-trip times – while the suppression is almost perfect. Hence, by using exponentially distributed timers we have a scalable and fast feedback mechanism. Dynamic changes in the number of

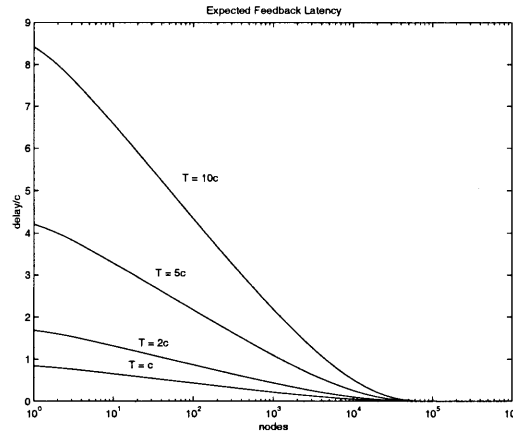


Figure 4.2: Expected feedback latency from an exponentially distributed timer, as a function of the number of feedback nodes, for $\lambda = 10$

nodes by orders of magnitude do not lead to feedback implosion and have only a minor effect on feedback latency.

We discuss the choices of distribution parameters for DRMTP and SDCP next.

4.1.3 Choosing the Parameters of the Distribution

Recall from Chapter 3 that the timers for both DRMTP and SDCP are adjusted as a function of scope depth. This is expressed by the wait maximum interval set to $W_{max,d}$, and the number of feedback nodes set to N_d . In the light of the previous discussion, these choices affect both the effectiveness of implosion control and the feedback latency. Our choices should take into account the increase of trip time with the increase of scope depth, and the increase of the number of reachable nodes [80].

We make the following two choices:

$$W_{max,d} = 10 \cdot d^{\sqrt{2}} \cdot c_0 \quad (4.11)$$

$$N_d = 10^{1/2 + \sqrt{d}} \quad (4.12)$$

where c_0 is the single hop latency. For terrestrial links, a reasonable estimate is in the order of milliseconds. Figure 4.3 plots the increase of the maximum wait interval as a function of the scope depth, with c_0 as a parameters. Similarly, N_d is plotted as a function of the scope

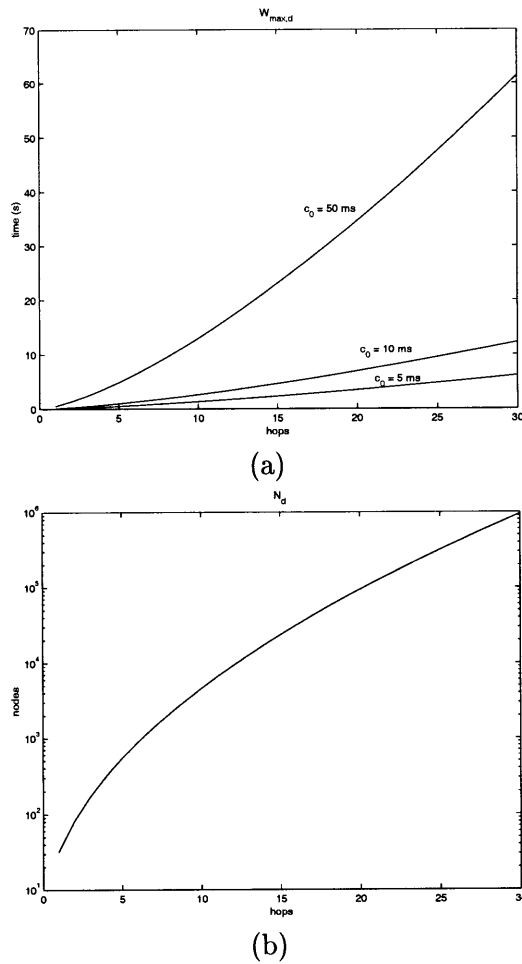
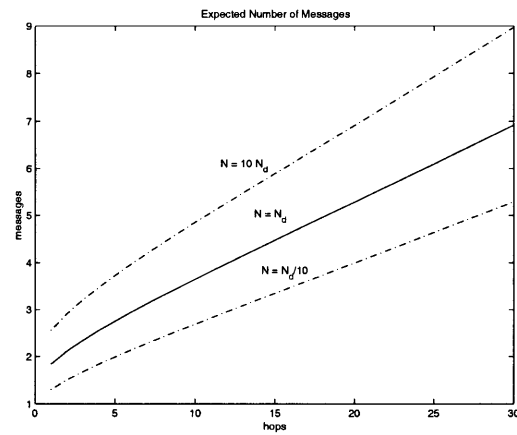


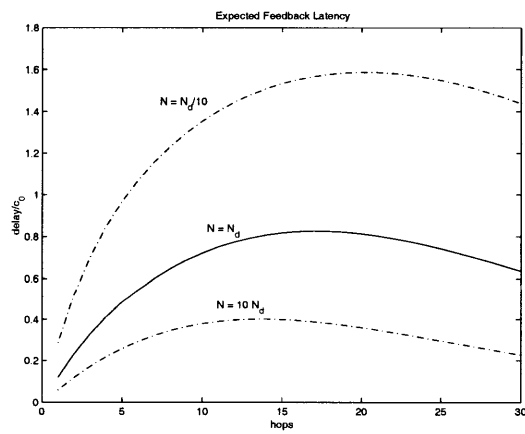
Figure 4.3: Growth of $W_{max,d}$ and N_d as a function of scope depth. (a) $W_{max,d}$ (b) N_d

depth. As we can see from the plots, both choices should be reasonable for the majority of networks that comprise the Internet. Nevertheless, the parameters can be adjusted if additional knowledge of the topology is available.

The expected number of feedback messages and latency is plotted as a function of d with these choices of $W_{max,d}$ and N_d in Figure 4.4. The figure also plots the same functions, but with an order of magnitude error in the estimation of N_d . As we can see, these choices provide good latency and effective implosion control, even if the estimation of N_d is off by an order of magnitude.



(a)



(b)

Figure 4.4: Expected number of messages and feedback latency as a function of d for our choice of $W_{max,d}$ and N_d , and effects of poor estimates (a) Expected number of feedback messages (b) Expected feedback latency

4.2 Experimental Evaluation in a Network Testbed

In order to determine the behavior and scalability of the protocol in good network conditions, we evaluated a prototype implementation in the internal Media Lab network. The objective of the experiments was twofold: test the behavior of the protocol in the presence of source failures, and evaluate traffic localisation and scalability for network effects. The behavior of the protocol in boundary conditions – in the presence of congestion – could not be evaluated in the network testbed, as the latter has limited topology. We illustrate the behavior of the protocol in such conditions with simulated experiments in Section 4.3

4.2.1 The Network Testbed

The experiments were carried in two subnets of the internal network with network distance of 2 hops (Figure 4.5). We used 5 hosts in the 18.85.45.x subnet, and 7 hosts in the 18.85.9.x subnet. On each host, we ran a number of virtual nodes as separate processes. Hence, we were able to evaluate the behavior of the protocol for dense constellations of active nodes.

For the experiments presented in this section, we used an MP3-encoded audio file with size 6994257 Bytes. The file was encoded with a fixed bit rate of 192 Kbps. The file consisted of 11156 data frames and a header frame, with total duration of 4 min 51 s (291 s). We used a burst size of 4 frames.

The machines in the 18.85.45.x subnet were dual 1GHz Intel PIII IBM x-330s, running Linux kernel 2.4.12. 18.85.9.{45,45,54} were dual 1.7GHz Intel Xeon IBM Intellistation ProMs, running Linux kernel 2.4.18. 18.85.9.{70,71,72} were 1 GHz Intel PIII machines running Linux kernel 2.4.12, and 18.85.9.60 was a dual 800 MHz Intel PIII machine running Linux kernel 2.4.17. All 18.85.45.x machines were connected with dual bonded 100BaseT ethernet cards to a 100Mbps ethernet. 18.85.9.{60,70,71,71} were connected with a single 100BaseT NIC each to a 100Mbps ethernet. 18.85.9.{45,46,54} were directly connected to the gigabit ethernet backbone with optical gigabit NICs. The link 18.85.45.1 - 18.85.9.1 was part of the gigabit backbone of the Media Lab network.

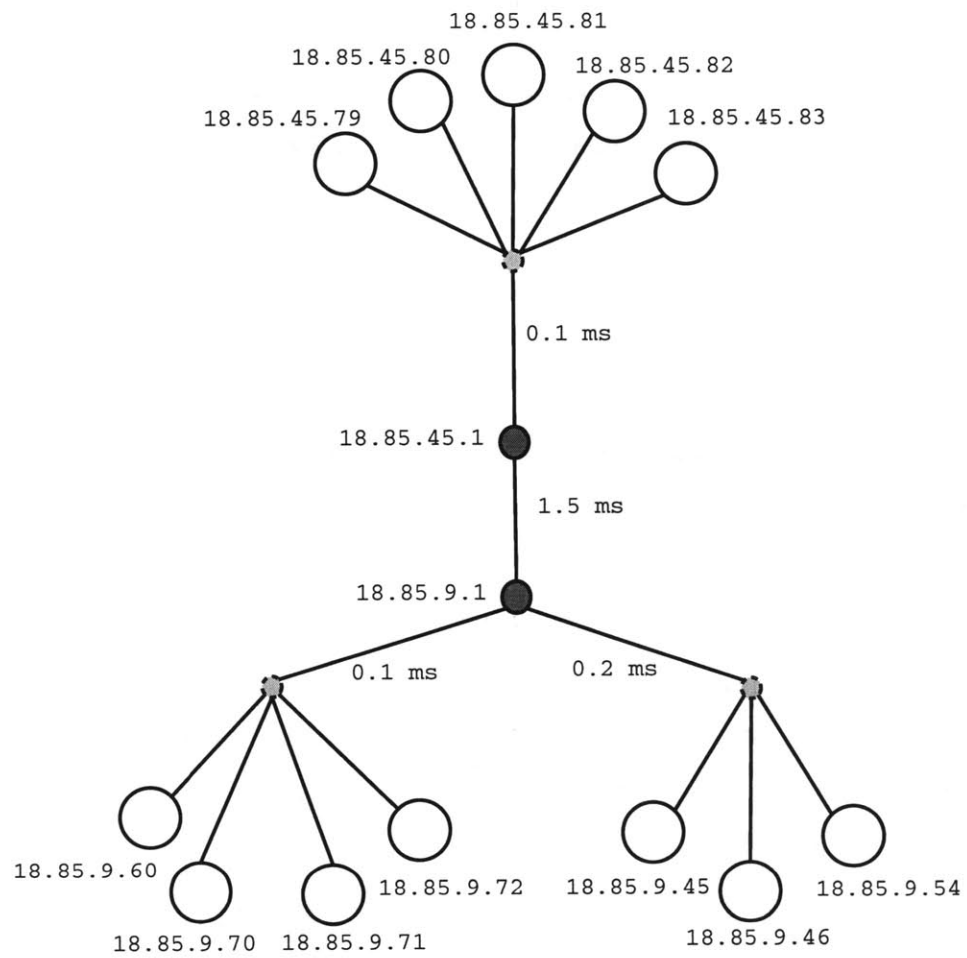


Figure 4.5: Network testbed

4.2.2 Traffic Localisation

The first experiment explores the traffic localisation behavior of the protocol. A source was initially placed at the 18.85.45.x subnet, and 3 sinks appeared consecutively in the 18.85.9.x subnet. The first sink joined the session after 13 s, the second after 137, and the last after 371 s. The arrival times were randomly generated by a poisson process with rate 0.01 arrivals per second.

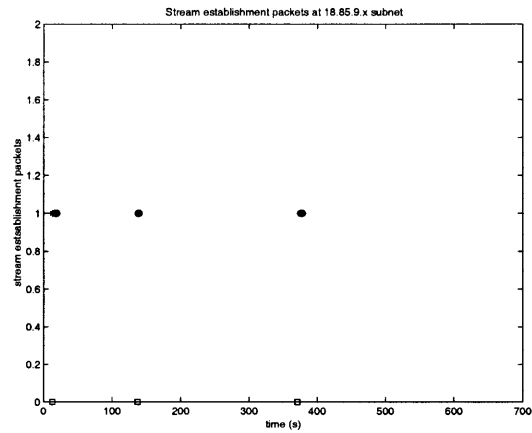
Figure 4.6 illustrates the packets exchanged on stream establishment by the three sinks. As we can see, the first sink initiated a new stream from the source. The second sink received a passive announcement by the first sink and joined the existing stream; the beginning of the stream was locally patched within the 18.85.9.x subnet. Finally, the last sink joined after the complete file had been received by the first two sinks, and was locally served by one of the two. Figure 4.7 depicts the data traffic in both subnets, exposing the traffic isolation between the two subnets and local stream patching.

4.2.3 Source Failure Recovery

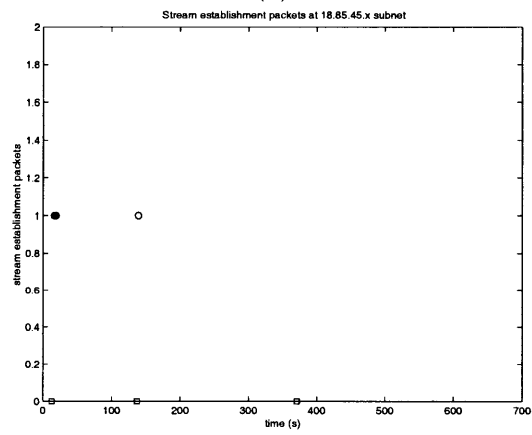
The second experiment explored the source failure recovery capabilities of the protocol. The five hosts in the 18.85.45.x subnet were configured as sources. Once a source started providing a stream, it would fail randomly with a mean failure time of 2 min. The sources in the 18.85.9.x subnet were configured as sinks, and started within milliseconds of each other, in order to be synchronised in the stream. The experiment was repeated 10 times, with similar results; we present the outcome of a single experiment, as it is representative of the behaviour of the protocol.

Figure 4.8 illustrates the control traffic generated in the network. Figure 4.8(a) is the control traffic for the 18.85.9.x subnet, and Figure 4.8(b) was taken from the 18.85.45.x subnet. The failure events are marked on the time axis. The behavior of the protocol is as expected: implosion is completely suppressed, and source failure is detected within a few seconds.

The data traffic is illustrated in Figure 4.9, with second scale. Since the sinks were synchronised, all flows were directed from the 18.85.45.x subnet to the 18.85.9.x subnet. The figure presents the trace from the 18.85.9.x subnet. As we can see, a single stream is flowing in the network – covering all 7 sinks. Failure is detected within a few seconds, as the schedule

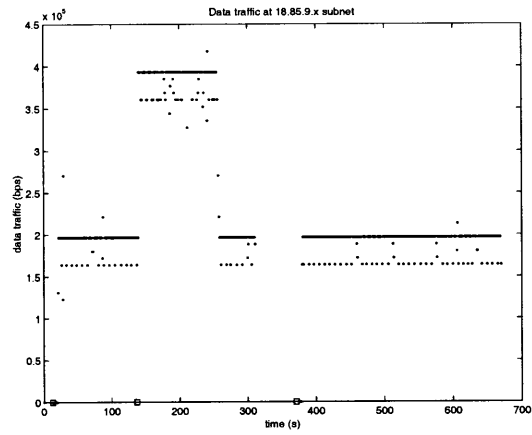


(a)

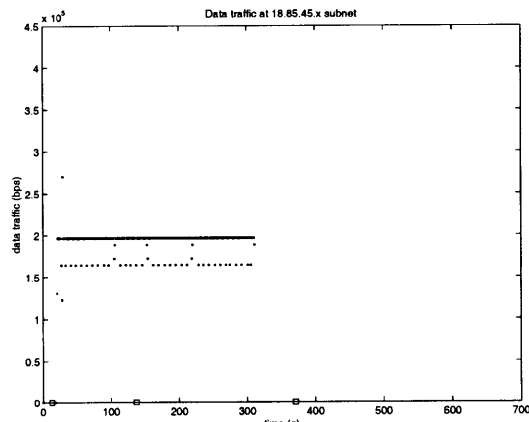


(b)

Figure 4.6: Stream establishment traffic in the first experiment. (a) 18.85.9.x subnet. (b) 18.85.45.x subnet.

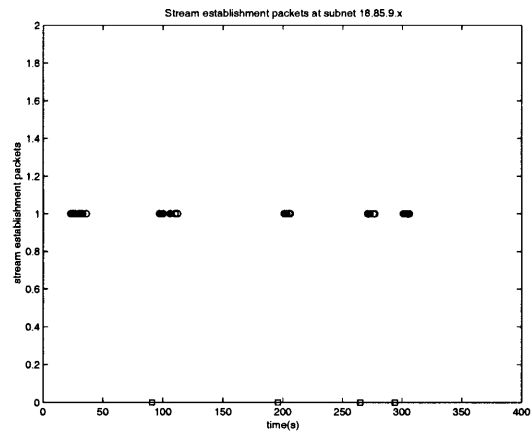


(a)

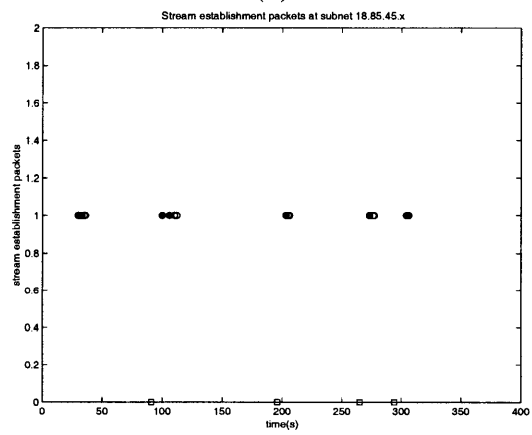


(b)

Figure 4.7: Data traffic in the first experiment. (a) 18.85.9.x subnet. (b) 18.85.45.x subnet.



(a)



(b)

Figure 4.8: Control traffic in the source failure experiment. (a) 18.85.9.x subnet. (b) 18.85.45.x subnet.

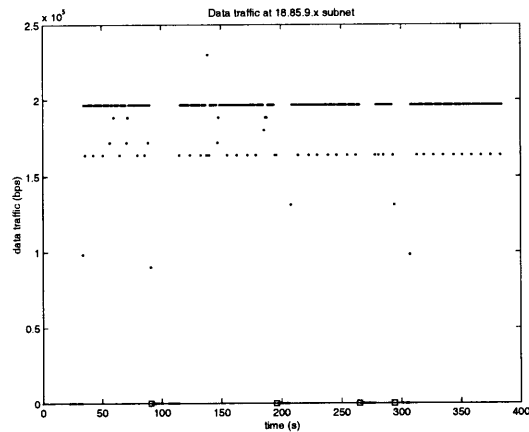
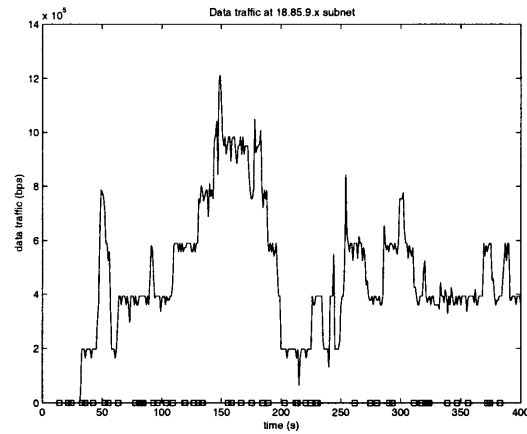


Figure 4.9: Data traffic in the source failure experiment. Trace taken from the 18.85.9.x subnet.

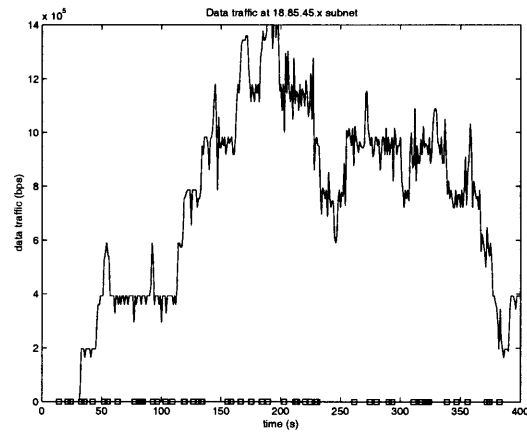
was transmitted 6 times a second. Stream failover is visible in the figure by the gaps in the data flow following the failure events, marked with squares in the time axis.

4.2.4 Scalability in the Presence of Network Effects

In this experiment we explore the behavior of the protocol in the propagation of a content file with network effects. Specifically, the content file was originally available at a single node in the 18.85.45.x subnet. In order to simulate large scale behavior, we added multiple virtual nodes on each host, arriving with a poisson process of rate $0.1s^{-1}$. The data traffic trace for the first 400 seconds of the experiment is present in Figure 4.10, with arrivals marked as squares in the time axis. As it is visible from the graph, a number of patch streams were independently established at each subnet. As a result, all nodes were able to achieve the real-time rate, while keeping the total consumed bandwidth at low levels. This is clearer in Figure 4.11, where we draw the evolution of the node population over time in conjunction with the number of streams reaching each subnet. As we can see, even as the population climbs over 60 within 400 seconds, the number of streams never exceeds 6 on any of the two subnets. The first stream established was serving all nodes in the network for its duration, with disjoint patching streams accommodating new arrivals.



(a)



(b)

Figure 4.10: Data trace for the network effect experiment. (a) 18.85.9.x subnet. (b) 18.85.45.x subnet.

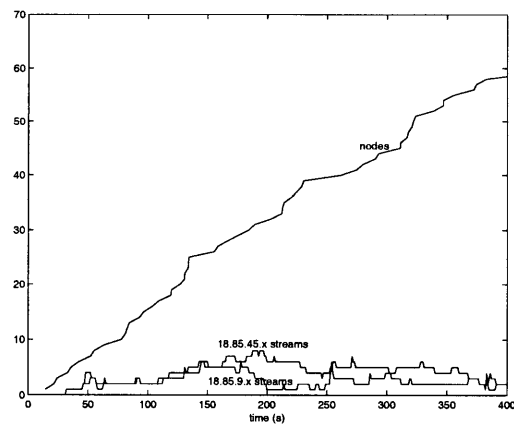


Figure 4.11: Evolution of session population and the number of streams in each subnet.

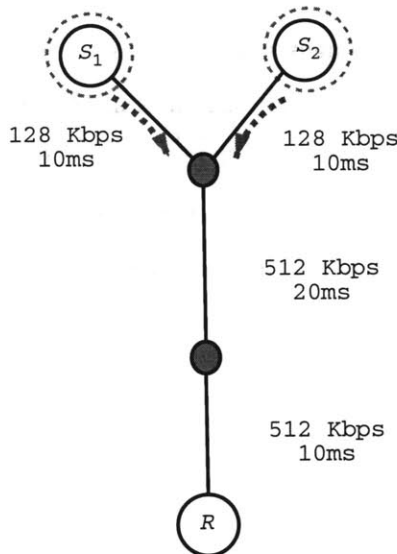


Figure 4.12: Source upstream congestion experiment scenario. R is the primary controller of the stream, and S_1 , S_2 are two sources with limited upstream capacity.

4.3 Protocol Behavior in Boundary Conditions

The experiments in the network testbed illustrated the behavior and performance of the protocol with regards to traffic localisation and session scalability. However, the limited topology of the testbed did not allow us to explore the congestion control capabilities of the protocol. For this purpose we describe the results from two simulated experiments.

4.3.1 Source Upstream Congestion Control

When the source of a multicast stream faces upstream congestion, the entire session is affected and must slow down. This is a significant problem, as it appears very frequently in residential broadband connections based on ADSL or cable modems.

A scenario that captures these conditions is depicted in Figure 4.12. There are two local sources for a session with a real-time rate of a 192Kbps constant bit rate – but both sources have a limit of upstream bandwidth at 128Kbps, and an arbitrary number of sinks condensed in the primary controller. The routers were drop-tail.

Figure 4.13 depicts a trace from a simulation performed using ns. At time 0, the sink R requests and establishes a stream from source S_1 . The sink acts as the primary controller

and attempts to drive the stream at the real-time rate of the session. The upstream capacity limitation appears as permanent congestion, forbidding a stream from achieving a rate higher than 128. When congestion is perceived among the sinks of a stream, the protocol temporarily slows down the rate, as described in Section 3.4.5. Then it proceeds to establish a new stream for a two-way partition of the frame set – the result is two interleaved streams flowing from both available sources at 96 *Kbps*, providing an aggregate rate that matches the real-time rate of the session.

4.3.2 Protocol Fairness

The second simulated experiment studies the behavior of the protocol in the presence of competing TCP traffic, which causes congestion in the path from a source. The simulation scenario is illustrated in Figure 4.14.

For the experiment, summarised in Figure 4.15, the session real-time rate is again 192 *Kbps*. Source S_1 provides a stream to R through a bottleneck link with capacity 256 *Kbps*. A TCP connection between nodes A and B is established, sharing the bottleneck link, and causing packet loss as the TCP sender attempts to expand its window. The protocol reacts with the congestion control mechanism, splitting the stream rate and establishing an interleaved stream from S_2 , which does not share the bottleneck link. As a result, the bottleneck bandwidth is fairly shared between the DRMTP stream and the TCP flow; on the same time, as a by-product of the aggregation mechanism, the protocol achieves the real-time rate by combining the streams from S_1 and S_2 .

4.4 Concluding Remarks

This chapter explored the basic properties of the protocol architecture. We analysed the implosion control algorithm, and showed how the choice of truncated exponential timer distribution provides as with a highly scalable and fast feedback mechanism. We showed the results of some experiments conducted in a network testbed in the internal Media Lab network, illustrating the basic scalability and performance properties of the DRMTP protocol. We also explored the properties of the congestion control algorithm with simulated experiments.

While this analysis and experiments are not a comprehensive evaluation of all aspects

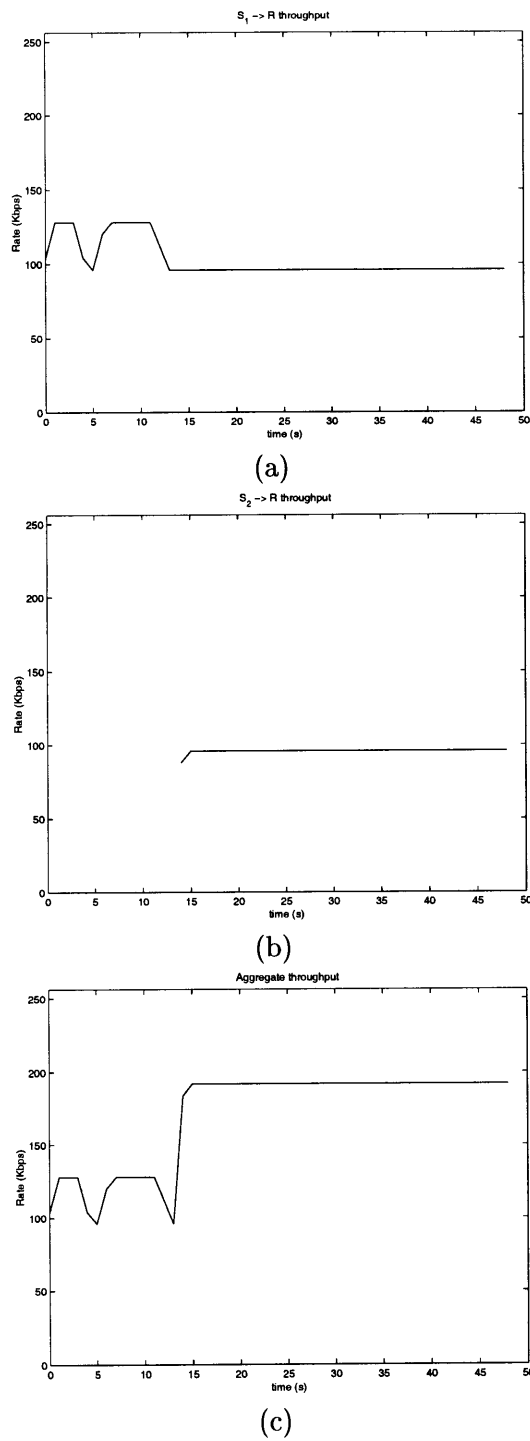


Figure 4.13: ns trace from the source upstream congestion experiment. (a) Throughput stream $S_1 \rightarrow R$ (b) Throughput for stream $S_2 \rightarrow R$ (c) Aggregate throughput.

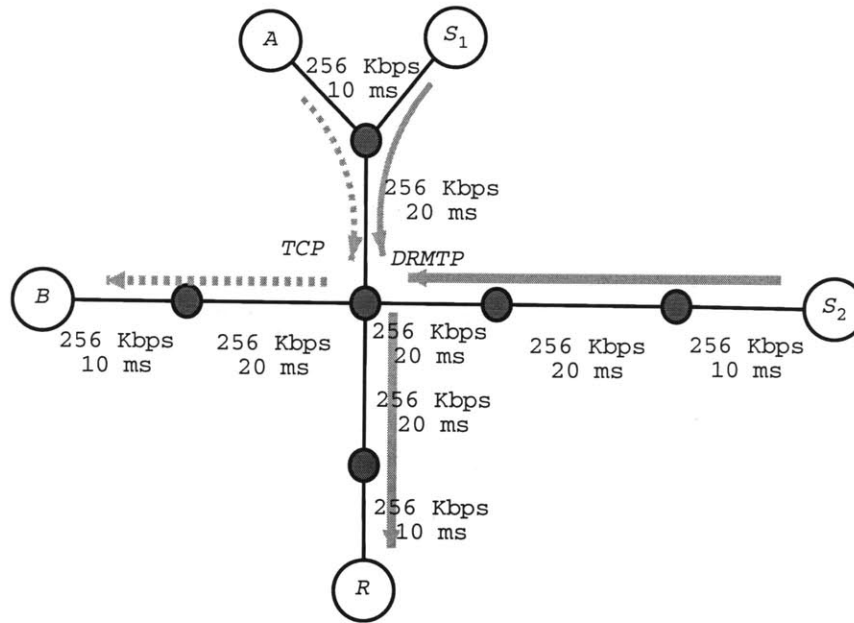


Figure 4.14: Source upstream congestion experiment scenario. R is the primary controller of the stream, and S_1 , S_2 are two sources. Nodes A and B establish a TCP flow, competing for the bottleneck link.

of our protocol architecture, they do offer a good indication on the scalability and performance properties. The protocol is able to quickly locate sources for providing the stream, and completely localise traffic only to the region of interest. Similarly, the stream establishment algorithm successfully groups all related request to a single stream – this aspect greatly contributes to the scalability of the protocol in the presence of network effects, serving the load with effectively a constant number of non-overlapping streams. Finally, from the simulated experiments, we see that the protocol can detect upstream capacity limitations of sources and dynamically split the stream to achieve the real-time rate when multiple sources are available. The protocol reacts similarly to congestion caused by competing traffic, and is able to achieve the real-time rate while competing fairly with other network flows.

Further experimentation and analysis is left for future work.

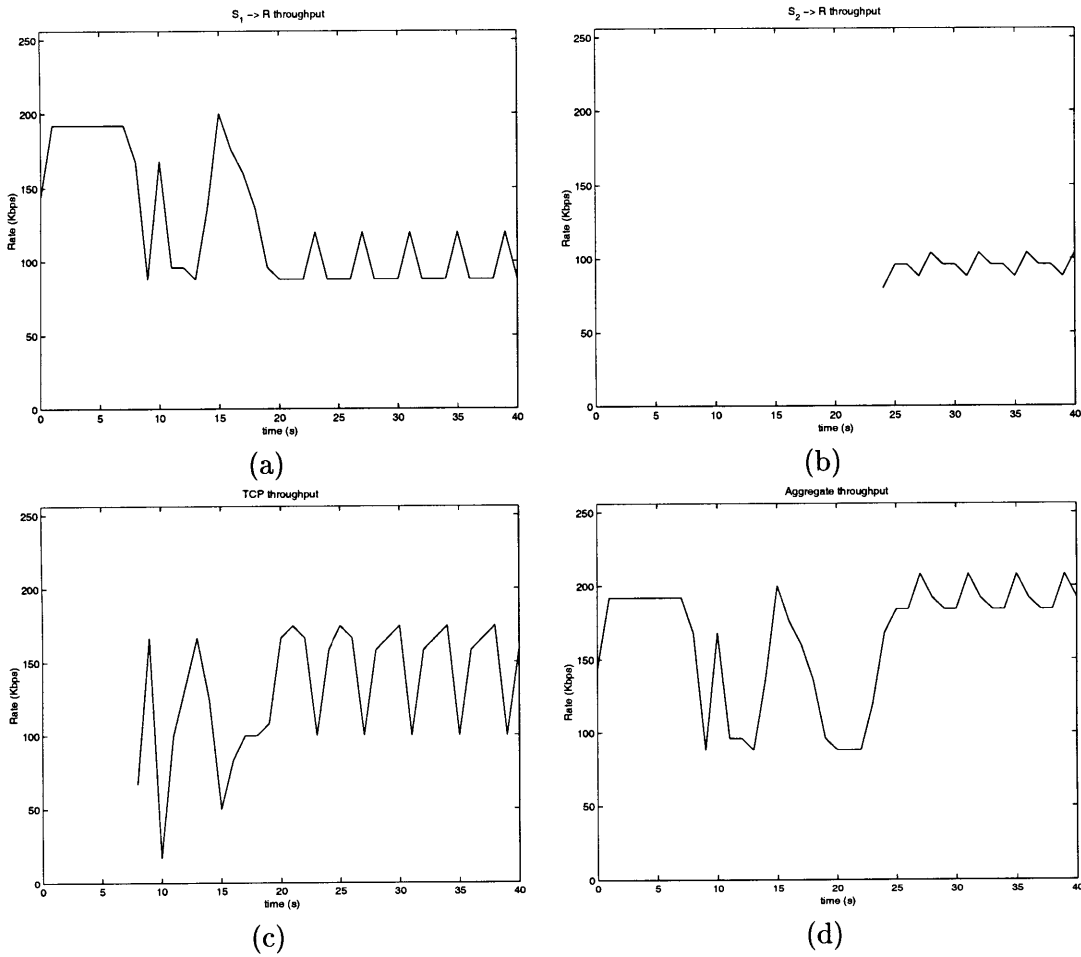


Figure 4.15: ns trace for the protocol fairness experiment. (a) Throughput for stream $S_1 \rightarrow R$ (b) Throughput for stream $S_2 \rightarrow R$ (c) TCP flow throughput (d) Aggregate throughput for R .

Chapter 5

An Economic Model for Collaborative Distribution of Media

The protocol architecture that we have developed in this thesis leverages the collaborative capacity of the network. In this manner, we provide a scalable programmable protocol that can support large numbers of nodes and media stream flows, by aggregating streams, localising traffic, and explicitly using the resources of any node participating in the system.

However, there is an exogenous economic factor that controls the applicability of the protocol architecture in real world media distribution. Digital media, as other forms of creative work, are usually copyrighted works. Authors rely on their work for sustaining an income, and the entire business model of the media distribution industry is based on paying royalties to the copyright owner – the publisher of the work. Hence, a system that is explicitly based on redistribution, in parts or whole, of copyrighted work is in direct conflict with the existing media distribution infrastructure.

In order to reconcile the economic realities of media distribution with the technical necessity of network scalability, a suitable economic model is required. We can outline such a model, for a media distribution system based on our protocol architecture, in the following terms:

- Any content file distributed in the system as copyrighted work is uniquely identified. Users storing files in a node must obtain a *license* for the file.
- Any node storing licensed files, is explicitly allowed to *redistribute* the file, in parts or

whole, to other nodes in the system.

- By redistributing a file, a node is granted *affinity points*. The affinity points propagate with further redistributions from the recipients, and are used by the copyright owner for providing an economic benefit to the redistributor. Affinity points are also granted by referral; that is, a user making a referral that results to further distribution of the file in the system is granted affinity points.
- There is an entity which interacts with the system as a *surrogate* of the copyright owner. The surrogate entity is responsible for handling licensing of the files and keeping track of the affinity points generated by the distribution.

With this model, it is possible to leverage our protocol architecture for scalable media distribution, without requiring any modification to the current end-to-end best effort service model of the Internet. Similarly, we provide a mechanism for interfacing with the existing economic structure of media distribution, without requiring a legal framework that imposes insurmountable obstacles to a scalable implementation. The interface to the existing media distribution infrastructure is the role of the surrogate of the copyright owner. The implementation of the surrogate is orthogonal to the distribution system – today there are already companies, such as Yaga [89], positioning themselves as transaction clearing houses for digital media licensing and distribution. Finally, by embedding an affinity point distribution mechanism, we provide an additional incentive for users to share and ensure the propagation the of files – users receive licensing discounts from collected affinity points. Users are *encouraged* to share by the licensing scheme and by doing so we increase the efficiency of the distribution with DRMTP while matching the cultural impetus of media distribution.

We fill the details of the economic model in the remaining of the chapter.

5.1 Licensing

In order to process a file f , a node n must obtain a license. The license is specific to a file and a node, described as $license(n, f)$. Note that the license covers any frame of the file:

$$license(n, f) \rightarrow \forall f_i \in F_f^{file}, license(n, f_i) \quad (5.1)$$

A node obtains a license by *registering* the file with the surrogate of the copyright owner. The surrogate computes the license for the node in such a way that it is uniquely identified and cryptographically hard to reconstruct. This can be handled by any cryptographic signature scheme [50]. The interaction is described by the following flow of messages, from a node n to the surrogate S_c :

$$\begin{aligned} n &\rightarrow S_c: (\text{register } I_n \text{ ident}(f)) \\ S_c &\rightarrow n: (\text{license } K_{S_c}\{I_n, \text{ident}(f)\}) \end{aligned}$$

where $K_{S_c}\{I_n, f\}$ denotes a cryptographic signature computed by the surrogate, and I_n is the identity of the node. The signature is the license for the file, for the specific node:

$$K_{S_c}\{I_n, \text{ident}(f)\} \rightarrow \text{license}(n, f) \quad (5.2)$$

The cryptographic signature can be verified by the copyright owner. The means of verifying the validity of a license by entities other than the copyright owner is outside the context of the protocol. It can be handled by using a public key cryptosystem, where the signature is signed by the private key of the surrogate and can be verified with the public key. For instance, a "secure" media player may contain the public key of the surrogate and verify the license before enabling playback of the file¹. Since the identity of the node is included in the signature, a player registered to a specific node can refuse the playback if the embedded node identity does not match the identity of the current node. Finally, since $\text{ident}(f)$ can be computed from the file itself, the contents *need not* be encrypted.

5.2 Affinity Points

A node license can be used as a reference from another node in a registration message. For example, n' may have provided a referral to n for accessing the file or have been a provider for some of the frames comprising the file. A claim of referral by n can be verified by the surrogate, if the registration message includes all the referral keys:

$$n \rightarrow S_c: (\text{register } I_n \ f \ R_{n,f})$$

¹Or allow only a limited number of playbacks for an unlicensed file

where $R_{n,f}$ is a referral set provided by n . The referral set includes all the license keys of nodes n_1, \dots, n_k that have provided a referral to n :

$$R_{n,f} \equiv \{K_{S_c}\{I_{n_1}, \text{ident}(f)\} \cdots K_{S_c}\{I_{n_k}, \text{ident}(f)\}\} \quad (5.3)$$

By using the referral set on the license registration message, the surrogate can compute and grant affinity points to each involved node. Let $\alpha_{f,t}$ be the affinity coefficient of a file. The coefficient can depend on both the file and the time of the distribution. Hence, a copyright owner can fine-tune the incentive for distribution by granting higher affinity coefficients for specific files that decay over time.

When a node appears in a referral set, it receives the affinity points associated with the file and the time of registration. Affinity points are maintained by the surrogate and *granted* on registration from another node. Hence, if $A(n, t)$ are the affinity points collected by node at time t , we have the following update rule:

$$\forall n_i, K_{S_c}\{I_{n_i}\} \in R_{n,f}, A(n_i, t) \leftarrow A(n_i, t) + \frac{1}{|R_{n,f}|} \cdot \alpha_{f,t} \quad (5.4)$$

that is, the actual affinity points granted depend on the size of the referral set.

Further, affinity points are *propagated* to nodes that have contributed as referrals to nodes n_i, \dots, n_k with a coefficient that decays with the depth of the distribution. Hence, if n'_1 was a first order (that is immediate) referral for node n_1 , it receives $\propto \alpha_{f,t}^2$ affinity points as a second order referral for node n . In general, for k th order referrals,

$$A(n_i^{(k)}, t) \leftarrow A(n_i^{(k)}, t) + \frac{1}{\prod_{j=1}^{k-1} |R_{n_i^{(j)}, f}|} \cdot \alpha_{f,t}^k \quad (5.5)$$

This model is a pyramid scheme, as illustrated in Figure 5.1. Note that in order for the scheme to be stable, $\alpha_{f,t} < 1$. Hence, the affinity points decay with the depth of the distribution. Similarly, if referral coefficients decay over time, the higher order affinity distribution further decreases. Hence, nodes have an incentive to increase *fast* distribution of files, even if they don't themselves receive first-order referral. In addition, this model of affinity distribution provides for a bounded total affinity. Even if distribution results in an infinite chain, the upper bound for total affinity is $\frac{\alpha_{f,t}}{1-\alpha_{f,t}}$.

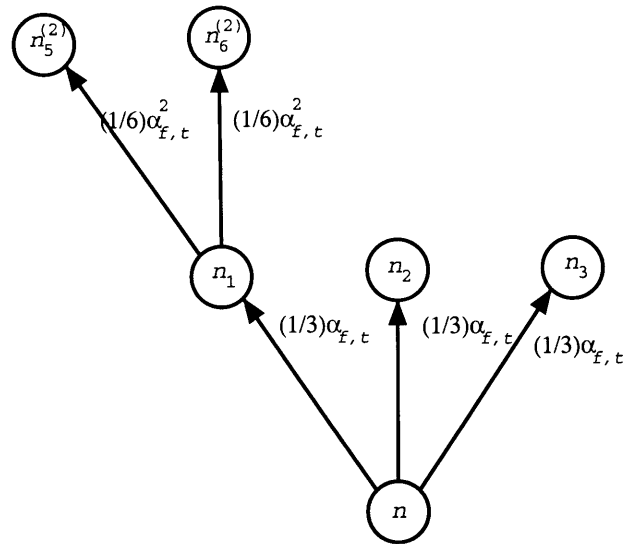


Figure 5.1: Affinity point distribution with a pyramid scheme. Nodes n_1, n_2, n_3 are the first order referral set of n , receiving $\frac{1}{3}\alpha_{f,t}$ affinity points. Nodes n_5, n_6 are the second order referral set, as first-order referrals to node n_1 , receiving $\frac{1}{6}\alpha_{f,t}^2$ affinity points.

Referrals can be either explicitly provided by the user or be provided by to the nodes that contributed frames in the distribution of the file with DRMTP, by embedding the node license in every frame transmitted by a source . However, for users *it doesn't matter when the bits come from*. Hence, the user should have control over the referral. If a user has received a recommendation by some other user for a file, he can include the other user in the referral set. In order to be included in the referral set, a node must provide its license key ensuring that referrals cannot be spoofed by fraudulent users. By providing a mechanism for the user to override the distribution referral with a recommendation referral, the users do not compete on bandwidth; rather, they compete in spreading the word about the system and specific media files. This behavior reinforces the network effects of media distribution, and on the same time increases the distribution efficiency of DRMTP by further spreading files in the network and increasing the number of nodes that concurrently access it.

5.3 Payment

The final detail about the economic model is the payment scheme. Payment should incorporate the affinity points – for instance in the form of a discount. On the same time, payment

should offer protection against affinity point spoofing by user collusion. The solution is to use micropayments in a pay-per-file setting. Payment is the prerequisite for obtaining the license: A node cannot obtain a license without payment, which is handed on registration.

When the surrogate receives the registration message from a node, it charges the node's account with the current price C of the file, adjusted for by a discount function computed on the node's accumulated affinity points. Affinity points are consumed by the charge:

$$C(n, f, t) \leftarrow C(n, f, t) - D(A(n, t)) \quad (5.6)$$

$$A(n, t) \leftarrow A(n, t) - D(A(n, t)) \quad (5.7)$$

The discount $D(A(n, t))$ cannot exceed a specific amount, as determined by the copyright owner. In addition, as affinity points are granted only on receiving a payment by an actual node, fraudulent behavior in the form of user collusion is defeated. Finally, since there is an upper bound in the total affinity model that can be generated by redistribution, the profit for the copyright owner is bounded from below by

$$\min profit(n, f, t) = C(n, f, t) - \frac{\alpha_{f,t}}{1 - \alpha_{f,t}} \quad (5.8)$$

Therefore, the pricing and affinity coefficient of a file can be adjusted according to the revenue that the owner should receive.

5.4 Concluding Remarks

In this chapter we outlined an economic model matched to the properties of the protocol architecture developed in this thesis. The economic model is based on licensing for redistribution. Instead of trying to restrict the user rights and try to criminalise sharing behavior, we try to reinforce it by providing a suitable licensing and affinity point distribution scheme. Users are encouraged to share, increasing the efficiency and scalability of distribution with our protocol architecture. At the same time the distribution cost for the copyright owner is decreased – as user resources are used for the distribution – and the revenue is increased by fostering network effects. The model is simple to implement: we only require a copyright owner surrogate entity for handling licensing and transactions and an license authentication

scheme on file playback. There is no need to engage in a content encryption arms race; the effort can be directed towards economic transaction authentication.

Chapter 6

Conclusion

This thesis embarked on distributing the distribution for real-time media. We developed a high level active protocol architecture, which adheres to the end-to-end design principle [66] of the Internet and provides the primitives for flexible and efficient location and distribution of information. On the same time we enabled security and privacy, accountability, and an embedded cost and loyalty distribution model. These primitives were used for developing a programmable protocol substrate, demonstrated with a prototype implementation and backed by an economic model for real-time collaborative media distribution.

The cornerstone of the architecture is DRMTP (*Distributed Real-time Multicast Transport Protocol*), an adaptive application-level [13] protocol core which allows collaborative multicasting of real-time streams. The protocol provides transparent semantics for loosely coupled multipoint interactions. It allows aggregation and interleaving of data fetched simultaneously from diverse machines and supports the location and coordination of named data among peer nodes, such as a record album or television program, without additional knowledge of network topology. The dynamic stream aggregation scheme employed by the protocol solves the problem of network asymmetry that plagues residential broadband networks. In addition, the stateless nature of the protocol allows for fast fail-over and adaptation to departure of source nodes from the network, mitigating the reliability problems of end-user machines. Coupled with well established techniques, like traffic localization [54], stream patching [40, 67, 37], and TCP-friendly congestion control [25, 77], we deliver a protocol that enables scalable real-time media distribution in a completely decentralised, serverless fashion.

DRMTP is supported by a dynamic content and source discovery protocol, which determines the properties of the network and availability of information based on high-level content description. This way, users are able to locate and access media without ever knowing about the existence of potential sources in the network and without noticing intermittent failures in the act of the distribution.

Along those lines, we have also developed a novel dynamic, mostly functional language named MAST (*Meta Abstract Syntax Trees*). The language has full support for mobile code and distributed computation and can be embedded in the payload of the content discovery protocol or even DRMTP itself. However, the presentation of the language was beyond the scope of the thesis.

The architecture is placed into context for real-world deployment by an economic model and embedded micropayment scheme for cost distribution and loyalty payment. Our scheme explicitly allows redistribution of content by end users and includes an affinity point computation algorithm which rewards end-users for redistribution. The ramification of this approach is that end-users are encouraged to provide access to their media store, thus maximizing the efficiency of the distribution with DRMTP. Simultaneously, the cost of distribution for copyright owners and content providers is drastically reduced, and availability of information is automatically determined by popularity, transcending the lifetime of the original host.

The presentation was based on the system model developed on Section 2.1, where a large number of autonomous nodes distributed over the Internet stores media files in part or whole. We evaluated the core of the architecture, the DRMTP protocol, on this model with mathematical analysis, experiments in a real network testbed, and simulation, and showed that it offers a scalable solution for decentralised media distribution.

The basic premise to scalability is a scalable randomised implosion control scheme and traffic localisation. We showed the scaling properties of the truncated exponential timer scheme in Section 4.1. We elaborated in particular choices, and showed that the protocols can theoretically scale to thousands of nodes per session, while maintaining low feedback latency. We evaluated traffic localisation, failure recovery, and scalability in the presence of network effects in Section 4.2. By experimentation with a real implementation in a Media Lab network testbed we showed that the protocol can successfully control implosion, quickly discover actively flowing streams, and isolate repair traffic to a local scope. Similarly, even in

the presence of adverse network effects, the protocol can maintain a constant low number of concurrent localised streams, which is only dependent on the arrival rate of new nodes. The congestion control scheme was examined in Section 4.3. There, we examined two simple but representative scenarios, and showed that the protocol successfully reacts to upstream source capacity limitations and competes fairly with cross TCP traffic.

A few pointers for future work can be provided among those lines. First, protocol analysis can be carried further, examining very large scale scenarios and a variety of rate schemes and congestion scenarios. Second, although the architecture provides a programmable protocol substrate, we barely scratched the surface on programmability. Further exploration of these aspects and a development of a full active service architecture for peer-to-peer systems can be the subject of future work.

Bibliography

- [1] H. Abelson, G.J. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, 2nd edition, 1996.
- [2] J.C. Bolot and T. Turetti and I. Wakeman. Scalable feedback control for multicast video distribution in the internet. In *Proc. Conference on Communications, Architectures, Protocols, and Applications*, pages 58–67, 1994.
- [3] E. Amir, S. McCanne, and R. Katz. An active service framework and its application to real-time multimedia transcoding. In *Proc. ACM SIGCOMM'98*, 1998.
- [4] D. Tennenhouse and. A survey of active networks research. *IEEE Communications*, 1997.
- [5] D.G. Andersen, H. Balakrishnan, M.F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. ACM SOSP'01*, 2001.
- [6] S. Ante. Inside Napster: How the music-sharing phenom began, where it went wrong, and what happens next. Business Week article, 2000. http://www.businessweek.com/2000/00_33/b3694001.htm.
- [7] S. Armstrong, A. Freier, and K. Marzullo. Multicast transport protocol. Internet Engineering Taskforce (IETF) RFC-1301, 1992.
- [8] A. Ballardie. Core based trees (CBT) multicast routing architecture. IETF RFC 2201, 1997.
- [9] J. Berst. Why technology can't stop music piracy. ZDNet News online, 2001. <http://www.zdnet.com/anchordesk/stories/story/0,10738,2677668,00.html>.

-
- [10] I. Brown. *End-to-End Security in Active Networks*. PhD thesis, University of London, 2001.
- [11] J. Byers, M. Luby, and M. Mitzenmacher. Fine-grained layered multicast. In *Proc. INFOCOM'01*, 2001.
- [12] J. Chang and N. Maxemchuk. Reliable broadcast protocols. *ACM Transactions on Computer Systems*, August 1984.
- [13] D. Clark and D. Tennenhouse. Architectural considerations for a new generation of protocols. In *ACM SIGCOMM'90*, 1990.
- [14] K. Dean. Instant messaging grows up. Wired news article, 2000. <http://www.wired.com/news/culture/0,1284,33736,00.html>.
- [15] S. Deering and D. Cheridon. Multicast routing in datagram networks and extended lans. *ACM Transactions on Computer Systems*, 1990.
- [16] J. Kubiawicz et. al. OceanStore: An architecture for global-scale persistent storage. In *Proc. of 9th international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, 2000.
- [17] D. Estrin et.al. Protocol independent multicast-sparse mode (PIM-SM): Protocol specification. IETF RFC 2362, 1998.
- [18] F. Dabek et.al. Wide-area cooperative storage with CFS. In *Proc. ACM SOSP'01*, 2001.
- [19] J. Byers et.al. FLID-DL: Congestion control for layered multicast. In *Proc. Networked Group Communication*, 2000.
- [20] J.W. Atwood et.al. Reliable multicasting in the xpress transport protocol. In *Proc. of the 21st Local Computer Networks Conference*, 1996.
- [21] S.E. Deering et.al. An architecture for wide-area multicast routing. In *Proc. ACM SIGCOMM'94*, 1994.
- [22] W. Fenner. Internet group management protocol, version 2. IETF RFC 2236, 1997.

-
- [23] S. Floyd and K. Fall. Promoting end-to-end congestion control in the internet. *IEEE/ACM Transactions on Networking*, 1999.
- [24] S. Floyd, V. Jacobson, and S. McCanne. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Trans. Networking*, 5(6):784–803, December 1997.
- [25] S. Floyd, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *ACM SIGCOMM'00*, 2000.
- [26] N.J. Garber and R. Cadiraju. Factors affecting speed variance and its influence on accidents. *Transportation Research Board*, 1998.
- [27] C. Ghezzi and G. Vigna. Mobile code paradigms and technologies: A case study. In K. Rothermel and R. Popescu-Zeletin, editors, *Mobile Agents - First International Workshop, MA '97*, number 1219 in Lecture Notes in Artificial Intelligence, pages 39–49, Berlin, DE, April 1997. Springer-Verlag.
- [28] R. Govindan, C. Alaettinoglu, and D. Estrin. A framework for distributed active services. Technical report, Information Sciences Institute, University of Southern California, 1998.
- [29] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. IETF RFC 2608, 1999.
- [30] D. Halls. *Applying Mobile Code to Distributed Systems*. PhD thesis, University of Cambridge, 1997.
- [31] M. Handley and V. Jacobson. SDP: Session description protocol. IETF RFC 2327, 1998.
- [32] M. Handley, C. Perkins, and E. Whelan. Session announcement protocol. IETF RFC 2974, 2000.
- [33] S. Hannah, B. Patel, and M. Shah. Multicast address dynamic client allocation protocol (madcap). IETF RFC 2730, 1999.
- [34] R. Hinden. IP next generation overview. *Communications of the ACM*, 1996.

- [35] M. Hoffman. A generic concept for large-scale multicast. In *International Zurich Seminar on Digital Communication*, 1996.
- [36] M. Hoffman. Adding scalability to transport level multicast. In *Proc. COST 237 Workshop - Multimedia Telecommunications and Applications*, 1997.
- [37] K.A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true video on-demand services. In *Proc. ACM Multimedia*, 1998.
- [38] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM'88*, 1988.
- [39] S.K. Kashera, J. Kurose, and D. Towsley. Scalable reliable multicast using multiple multicast groups. In *Proc. ACM SIGMETRICS International Conference on Measurement and Modelling of Computer Systems*, 1997.
- [40] R. Kermode. *Smart Caches: Localized Content and Application Negotiated Delivery for Multicast Media Distribution*. PhD thesis, Massachusetts Institute of Technology, 1998.
- [41] R. Kesley, W. Clinger, and J. Rees. The *Revised*⁵ report on the algorithmic language scheme. *Higher Order and Symbolic Computation*, September 1998.
- [42] J. Lardner. Hollywood vs. high-tech. *Business 2.0*, 2002. <http://www.business2.com/articles/mag/0,1640,39428,FF.html>.
- [43] L. Lehman, S. Garland, and D. Tennenhouse. Active reliable multicast. In *IEEE INFOCOM'98*, 1998.
- [44] L. Lessig. *The Future of Ideas*. Random House, 2001.
- [45] A. Lippman. Personal communication.
- [46] C.G. Liu, D. Estrin, S. Shenker, and L. Zhang. Local error recovery in SRM: Comparison of two approaches. *IEEE/ACM Transactions on Networking*, 1998.
- [47] J. Lyman. Report: Instant messenger use exploding. E-commerce Times article, 2001. <http://www.ecommercetimes.com/perl/story/14793.html>.
- [48] R. Mann. The need for speed. VU Magazine Article, 2000. http://the-vu.com/need_for_speed.htm.

- [49] S. McCanne, V. Jacobson, and M. Vetterli. Receiver driver layered multicast. In *Proc. ACM SIGCOMM'96*, 1996.
- [50] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
- [51] D. Meyer. Administratively scoped IP multicast. IETF RFC 2365, 1998.
- [52] J. Nonnenmacher and E.W. Biersack. Optimal multicast feedback. In *Proc. INFOCOM'98*, 1998.
- [53] J. Nonnenmacher and E.W. Biersack. Scalable feedback for large groups. *IEEE/ACM Transactions on Networking*, (August), 1999.
- [54] J. Nonnenmacher, M. Lacher, M. Jung, E. Biersack, and G. Carle. How bad is reliable multicat without local recovery? In *Proc. Annual Joint Conference of the IEEE Computer and Communications Societies*, 1997.
- [55] J. Padhye, V. Firoiu, D.F. Towsley, and J.F. Kurose. Modeling TCP performance: A simple model and its empirical evaluation. *IEEE/ACM Transactions on Networking*, April 2000.
- [56] T. Palmaffy. Don't break the big government. Policy Review Article, 1996. <http://www.policyreview.org/sept96/labs.html>.
- [57] C. Papadopoulos, G. Parulkar, and G. Varghese. An error control scheme for large-scale multicast applications. In *Proc. INFOCOM'98*, 1998.
- [58] S. Paul, K.K. Sabnani, J.C. Lin, and S. Bhattacharyya. Reliable multicast transport protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, 1997.
- [59] J.J. Postel. Transmission control protocol. IETF RFC 793, 1981.
- [60] S. Raman. *A Framework for Interactive Multicast Data Transport in the Internet*. PhD thesis, University of California at Berkeley, 2000.
- [61] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM'01*, 2001.

-
- [62] D. Reed. Weapon of math destruction. Context Magazine article, 1999. <http://www.contextmag.com/archives/199903/digitalstrategy.asp>.
- [63] R. Rivest. The md5 message-digest algorithm. Internet Engineering Taskforce (IETF) RFC-1321, 1992.
- [64] L. Rizzo. pgmcc: a TCP-friendly single-rate multicast congestion control scheme. In *Proc. ACM SIGCOMM'00*, 2000.
- [65] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large scale peer-to-peer systems. In *Proc. ACM SIGCOMM'01*, 2001.
- [66] J. Saltzer, D. Reed, and D. Clark. The end-to-end argument in system design. *ACM Transactions on Computer Systems*, 1984.
- [67] S. Sen, L. Gao, J. Rexford, and D. Towsley. Optimal patching schemes for efficient multimedia streaming. In *IEEE INFOCOM'99*, 1999.
- [68] S. Shenker. Fundamental design issues for the future internet. *IEEE Journal on Selected Areas in Communication*, 1995.
- [69] I. Stoika, R. Morris, D. Karger, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM'01*, 2001.
- [70] M. Thaler, M. Handley, and D. Estrin. The internet multicast address allocation architecture. IETF RFC 2908, 2000.
- [71] D. Towsley, J. Kurose, and S. Pingaly. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. *IEEE Journal on Selected Areas in Communication*, August 1997.
- [72] L. Vicisano, L. Rizzo, and J. Crowfort. TCP-like congestion control for layered multicast data transfer. In *IEEE INFOCOM'98*, 1998.
- [73] D. Waitzman, C. Partridge, and S.E. Deering. Distance vector multicast routing protocol. IETF RFC 1075, 1988.

- [74] D. Wetherall. Active network vision and reality: Lessons from a capsule-based system. In *ACM SOSP'99*, 1999.
- [75] D. Wetherall. *Service Introduction in an Active Network*. PhD thesis, Massachusetts Institute of Technology, 1999.
- [76] D. Wetherall, J. Guttag, and D. Tennenhouse. ANTS: A toolkit for building and dynamically deploying network protocols. In *IEEE OPENARCH'98*, 1998.
- [77] J. Widmer and M. Handley. Extending equation-based congestion control to multicast applications. In *ACM SIGCOMM'01*, 2001.
- [78] Y. Yemini and S. DaSilva. Towards programmable networks. In *Proc. IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, 1996.
- [79] B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U.C. Berkeley, 2001.
- [80] E. Zigura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *Proc. INFOCOM'96*, 1996.
- [81] The RIAA campaign against music piracy. <http://www.riaa.org/Protect-Campaign-1.cfm>.
- [82] Akamai. <http://www.akamai.com>.
- [83] EFF "intellectual property - security systems standards and certification act (SSSCA)" archive. <http://www.eff.org/IP/SSSCA-CBDTPA/>.
- [84] Gnutella. <http://gnutella.wego.com>.
- [85] Kazaa. <http://www.kazaa.com>.
- [86] Morpheus. <http://www.musiccity.com>.
- [87] Napster. <http://www.napster.com>.
- [88] The network simulator - ns-2. <http://www.isi.edu/nsnam/ns>.

[89] Yaga. <http://www.yaga.com>.

Appendices

Appendix A

Protocol Specification

The complete protocol architecture can be implemented as a library, which interacts with application-specific code. Application code provides the framing, stream scheduling, and evaluation of session messages. We specify the protocol in terms of external message representation, in the packet level, and the API provided to application level code in a node of the system.

A.1 Message Format

A.1.1 Message Envelope

All protocol messages are enclosed in a common message envelope. The envelope contains field for node identification and message demultiplexing. The message envelope is summarised in Table A.1.

The magic of the protocol is the character string "(AMTP)#0x00", standing for Active Message Transport Protocol. The version field identifies the protocol version, currently 1. The environment identifiers, with length 128 bits, allows the protocol engine to demultiplex between messages. Sessions and streams are identified as environments. Similarly, the node identifier is a unique node key – it can be used for encryption and authentication of messages, at the discretion of application code.

Each message is identified by a node specific identifier, and has a length specified by the length field. The maximum packet size is 31768 bytes – larger messages are frag-

Field	Length (bytes)	Purpose
magic	7	Protocol identification
version	1	Protocol version
destination-environment	16	Destination demultiplexing
source-node	16	Source identification
source-environment	16	Source environment reference
message-identifier	8	Identifier for reassembly of fragmented messages
length	4	Message size
packet-offset	4	Packet offset for fragmented frames.
packet-length	4	Packet payload length

Table A.1: Message Envelope Fields

mented in multiple packets and reassembled at the destination using the tuple (`source-node` `source-environment` `message-identifier`) as a reference. Finally, the envelope is padded with zeros at the end to a length of 96 bytes.

A.1.2 DRMTP Messages

DRMTP messages are demultiplexed using the `destination-environment` field of the AMTP envelope. Each session has a separate environment identifier, and so does each stream within the session. Each DRMTP message carries a DRMTP magic, the string "(DRMTP)", a version byte, and the message payload prepended by a type specifier byte. Table A.2 summarises the payload of DRMTP messages.

All integral types are represented in network byte order. Compound type representation is shown in Table A.3, except `sockaddr` which is the standard UNIX socket address specification.

A.1.3 SDCP Messages

Similar to DRMTP, SDCP messages are demultiplexed using the `destination-environment` field of the AMTP envelope. Each session has a separate environment identifier, and so does each stream within the session. Each message carries a SDCP magic, the string "(SDCP)#0x00", a version byte, and the message payload prepended by a type specifier byte. Table A.4 summarises the payload of SDCP messages.

Message	Type specifier	Parameter	Parameter type	Purpose
request	0x01	depth	u_int8_t	Scope depth
		frame-set	frame_set_t	Request frame set
bid	0x02	depth	u_int8_t	Scope depth
		frame-set	frame_set_t	Bid frame set
accept	0x03	depth	u_int8_t	Scope depth
		frame-set	frame_set_t	Accepted frame set
announce	0x04	depth	u_int8_t	Scope depth
		stream	byte[16]	Stream identifier
		address	sockaddr	Group address
		frame-set	frame_set_t	Stream frame set
schedule	0x11	depth	u_int8_t	Scope depth
		controler-source-t	u_int32_t	$\hat{t}_{pc \rightarrow s}^{OT}$ in ms
		source-controler-t	u_int32_t	$\hat{v}_{pc \rightarrow s}^{OT}$ in ms
		spec	schedule_spec_t	Schedule specification
congestion	0x12	depth	u_int8_t	Scope depth
		max-throughput	u_int32_t	\hat{R}_r^{TCP} in bps
exclude	0x13	frame-set	frame_set_t	Excluded frame set
join	0x14	depth	u_int8_t	Scope depth
		stream	byte[16]	Stream identifier
		address	sockaddr	Group address
		frame-set	frame_set_t	Stream frame set
data	0x21	depth	u_int8_t	Scope depth
		frame	u_int32_t	frame ordinal
		size	u_int32_t	frame size in bytes
		payload	byte[size]	frame payload

Table A.2: DRMTP messages

Type	Field	Field type
frame_set_t	size	u_int32_t
	range-set	frame_range_t[size]
frame_range_t	begin	u_int32_t
	end	u_int32_t
	modulo	u_int32_t
	index	u_int32_t
schedule_spec_t	size	u_int32_t
	frames	u_int32_t[size]
	delay	u_int32_t[size]

Table A.3: Compound types in DRMTP messages

Message	Type specifier	Parameter	Parameter type	Purpose
find	0x01	depth	u_int8_t	Scope depth
		size	u_int32_t	Query size
		query	byte[size]	Query
match	0x02	depth	u_int8_t	Scope depth
		size	u_int32_t	Description size
		desc	byte[size]	Match description
allocate	0x11	depth	u_int8_t	Scope depth
		stream	byte[16]	Target stream
propose	0x12	depth	u_int8_t	Scope depth
		stream	byte[16]	Target stream
		address	sockaddr	Group address
announce	0x13	depth	u_int8_t	Scope depth
		stream	byte[16]	Target stream
		address	sockaddr	Group address

Table A.4: SDCP messages

A.2 Node Application Programming Interface

The node API for participating in DRMTP and SDCP sessions is summarised in Figure A.1. Application code provides instances of the `drmtp_client` and `sdcp_client` interfaces. To participate in the system, the application must first instantiate an `sdcp_session`, and provide it with the `sdcp_client`. The latter receives upcalls from the node kernel on SDCP events. For each DRMTP session that the application participates, a separate `drmtp_session` must be instantiated and provided with a `drmtp_client` instance for receiving upcalls. SDCP content search is performed with the `find` method of `sdcp_session`, and data is fetched from a DRMTP session using the `fetch` method of `drmtp_session`.

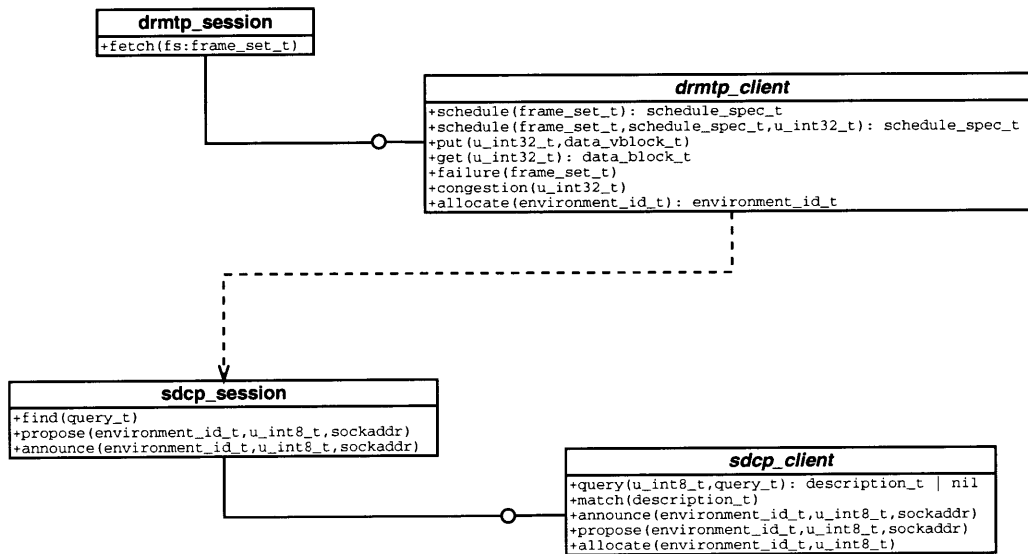


Figure A.1: Node Application Programming Interface