Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service

by

Yatin Dilip Chawathe

M.S. (University of California at Berkeley) 1998
B.Eng. (University of Bombay, India) 1995

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

in

Computer Science

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Fall 2000

The dissertation of Yatin Dilip Chawathe is approved:

Chair	Date
	Date
	Date
	Date

University of California at Berkeley

Fall 2000

Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service

Abstract

Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service

by

Yatin Dilip Chawathe
Doctor of Philosophy in Computer Science

University of California at Berkeley

Professor Eric A. Brewer, Chair

Despite the phenomenal success of the world wide web, one class of Internet applications that has yet to be satisfactorily realized is that of Internet broadcasting: the distribution of Internet content from one or more sources to a large number of simultaneous receivers. For large-scale broadcasting, the traditional Internet model of point-to-point unicast communication does not scale. So, the networking research community proposed IP multicast, a network layer service that allows a single source to distribute a data stream to many simultaneous receivers in an efficient manner. However, this network layer approach has met with limited success due to a number of factors including complexity of the network protocol itself, its inability to address Internet heterogeneity, and its lack of support for efficient and scalable transport protocols for reliability and congestion control. As a result, in spite of a decade of existence, the multicast protocol architecture remains just a research commodity with limited penetration into the commercially deployed Internet.

In this dissertation, we propose a new model for Internet broadcasting where we view multi-point delivery not as a network primitive but rather as an application-level infrastructure service. Our architecture relies on a collection of strategically placed network *agents* that collaborate with each other to form an overlay network composed of unicast interconnections. A data source distributes content to its receivers on top of this overlay network. We call this communication model *scattercast* and the network agents that are central to this model *ScatterCast proXies or SCXs.* The scattercast architecture effectively shifts the complexity associated with large-scale broadcasting away from the routing layer into a higher infrastructure service layer where it can be more easily managed. Moreover, by incorporating application-level intelligence within individual SCXs, scattercast can expand the traditional distribution model to optimize it for individual application needs by taking into account the effects of heterogeneity and application characteristics.

Such an infrastructural approach introduces two key questions: how do we construct the overlay network of SCXs in a distributed and dynamic fashion, and how do applications customize the overlay distribution framework to optimize it for their specific environment. To address the first problem, we present a protocol called Gossamer for grouping clients with SCXs and building an overlay network of unicast connections across SCXs, over which sources transmit data via efficient distribution trees. To address the second problem, scattercast develops a highly flexible and application-aware transport framework that incorporates the semantics of the application data into the transport protocol to allow the architecture to optimize the protocol for individual applications.

We demonstrate the efficacy of the scattercast approach using two complementary styles of evaluation. First, we use a set of simulations to demonstrate that the protocols and algorithms that underlie scattercast are viable and produce efficient overlay distribution networks. Second, we implement "real" applications on top of the architecture to demonstrate its ability to be customized for individual application requirements.

We believe that the scattercast approach of explicit application-level infrastructure embedded within the network is a new and promising direction for adaptive Internet applications such as network broadcasting. Similar approaches have recently been adopted by a number of commercial ventures for making the Internet broadcast-capable. In the future, we expect to see a greater proliferation of such infrastructural elements that enhance the Internet's capabilities while at the same time co-existing with the core IP architecture.

Professor Eric A. Brewer
Dissertation Committee Chair

"All are lunatics, but he who can analyze his delusions is called a philosopher."

—Ambrose Bierce.

Contents

List of Figures **vi**

List of Tables **x**

1 Introduction **1**

1.1 Motivation 2

1.2 IP Multicast: Not the Right Answer 3

1.2.1 Protocol Complexity 4

1.2.2 Network and End-system Heterogeneity 5

1.2.3 Layering Complex Transport Services 6

1.3 Our Solution: Infrastructure Service-oriented Approach 7

1.3.1 Pros and Cons of this Approach 9

1.3.2 Components of the Scattercast Architecture 10

1.4 Contributions 12

1.5 Thesis Organization 13

2 Network Infrastructure Services **15**

2.1 Evolving from a Plumbing System to a Richer Service Architecture 15

2.2 The Case for Clusters 19

2.3 Cluster Operating Systems 20

2.3.1 Scalable Network Services (SNS) 20

2.3.2 Active Services 22

2.4 Broadcast Distribution: An Application for Cluster Operating Systems . . . 22

3 Scattercast Architecture **24**

3.1 The Network Environment 24

3.1.1 The Internet Protocol 25

3.1.2 Locally Scoped Multicast 26

3.1.3 Opacity of underlying network topology 28

3.1.4 Infrastructure Service Clusters 29

3.2 An Architecture Overview 30

3.2.1 Scattercast in a Nutshell 30

3.2.2 The Service Model 32

3.2.3 Advantages of the Scattercast approach 33

3.2.4 Disadvantages of this approach 35
3.2.5 Scattercast Terminology 36
3.3 Discovering Scattercast Sessions: Scattercast Announcements 37
3.4 Joining a Session: Locating an Appropriate SCX 38
3.5 Joining a Session: Client Attachment 40
3.6 Summary of Scattercast Architecture 40

4 Gossamer: A Protocol for Application Level Multicasting 42
4.1 Introduction 42
4.2 A Practical Topology Construction and Management Algorithm 45
4.2.1 Protocol Overview 47
4.2.2 Node Discovery 49
4.2.3 Mesh Construction 52
4.2.4 Routing Layer 55
4.2.5 Data Distribution 57
4.3 Evaluation 57
4.3.1 Simulation Setup 58
4.3.2 Name Dropper++ Performance 60
4.3.3 Mesh Construction 61
4.3.4 Scaling Behavior 65
4.4 Discussion 68
4.5 Summary of Application Level Multicasting 69

5 Application-customizable Transport for Scattercast 70
5.1 Introduction 70
5.2 Application Level Framing for Semantic Transport 74
5.3 Data Naming: The Transport Independent Naming System 76
5.4 The Transport communication model 80
5.4.1 Data Delivery 83
5.4.2 End-to-end Reliability 84
5.5 Application-defined Semantic Transport 85
5.5.1 Application-customizable Reliability 86
5.5.2 Application-customizable Transmission Scheduling 87
5.5.3 Application-customizable Data Adaptation 89
5.6 The Programming API 90
5.7 Summary of Scattercast Transport 93

6 Scattercast Applications: Customizing the Scattercast Transport 95
6.1 Mediaboard: A Shared Electronic Whiteboard 96
6.1.1 Customized Namespace 97
6.1.2 Data Adaptation 98
6.1.3 Bandwidth Management 98
6.2 Mediapad: Whiteboard for the PalmPilot 99
6.2.1 A Split Programming Model 100
6.2.2 The Mediapad SCX 102

6.2.3 Protocol Conversion . . . 103
6.2.4 Data Transformation . . . 104
6.2.5 Intelligent Rate Limiting . . . 105
6.2.6 Implementation Experience . . . 105
6.3 InfoCast: Information Dissemination . . . 107
6.4 MusiCast: Radio over the Internet . . . 107
6.5 Summary . . . 109

7 Related Work **110**
7.1 Intelligent Network Infrastructure . . . 111
7.2 Application Level Multicasting and Overlay Networks . . . 113
7.3 Commercial Development . . . 115
7.4 Multi-point Reliable Transport . . . 116
7.5 Customizable Transport . . . 118
7.6 Adapting to Heterogeneity . . . 119
7.7 Summary of Related Work . . . 120

8 Conclusions and Future Directions **122**
8.1 Conclusions . . . 122
8.2 Lessons Learned . . . 124
8.2.1 Keep the network simple . . . 124
8.2.2 Limit IP multicast to domains where it is feasible and efficient . . . 125
8.2.3 Application-level intelligence is crucial . . . 125
8.2.4 Plan on throwing away at least one implementation . . . 126
8.3 Directions for Future Research . . . 126
8.3.1 Multi-level scattercast . . . 126
8.3.2 Source-rooted Trees vs. Shared Trees . . . 127
8.3.3 Infrastructure Administration . . . 127
8.3.4 An Improved Application-level Multicast Protocol . . . 128
8.3.5 Long-term Research Plan:
A Pervasive Application-aware Internet Middleware . . . 128
8.4 Summary . . . 129

A Hierarchical Start-time Fair Queuing (H-SFQ) Algorithm **131**

Bibliography **133**

List of Figures

1.1 End-client and Network Heterogeneity. No single data stream from the source can satisfy the entire range of client devices and networks within the session. 6
1.2 The scattercast architecture: Application-level infrastructure proxies (SCXs) form an overlay network of unicast interconnections to feed broadcast streams to a heterogeneous set of clients. Clients communicate with SCXs either via locally scoped multicast groups or via unicast. 9
1.3 The various components of the scattercast architecture. 10

2.1 The Akamai Infrastructure Service. 16
2.2 The Domain Name System: An illustration of how the DNS infrastructure resolves a name query for the host *mirage.cs.berkeley.edu*. The numbered arrows depict the sequence of steps involved in resolving the request. 17
2.3 Cluster Operating Systems and their use for hosting scattercast services: (a) The SNS Framework composed of a *frontend* to accept join requests from clients, a *manager* that spawns new *workers* on demand, and a *graphical monitor* to provide an administration console for the cluster. (b) The Active Services framework composed of *host managers* on each cluster node that manage one or more *service agents*. 21

3.1 Unicast vs. multicast transmission: In (a), source 1 unicasts a separate stream to each recipient; the packets of each stream explicitly specify the recipient's IP address. In (b), source 1 multicasts a single stream to a multicast group address (224.4.90.76) and the routers replicate the stream at appropriate branch points. The source does not need to know the recipient's IP address; the multicast group address provides a level of indirection. . . . 27
3.2 The layered Internet architecture and the composition of scattercast on top of this architecture 30
3.3 The scattercast architecture: Clients communicate with SCXs either via locally scoped multicast groups or via unicast. SCXs form a mesh of unicast interconnections between themselves. 31
3.4 Format of a scattercast announcement document. 38

4.1 Examples of mapping unicast, multicast, and Gossamer routing on to a physical network topology. (a) An example physical network topology consisting of 6 SCXs connected via 4 routers (b) Multicast routing from source SCX 1 using DVMRP-like reverse shortest path distribution tree (c) Naïve unicast distribution from source SCX 1 to all other SCXs (d) An illustration of how unicast distribution maps onto the underlying physical topology (e) A smart distribution tree composed of SCX-SCX unicast connections for Gossamer (f) Illustration of how the smart tree maps onto the underlying network topology. 43
4.2 A typical Gossamer overlay topology: (a) The topology consists of a mesh structure with a source-rooted spanning tree superimposed on top of the mesh. (b) The mesh+tree structure provides resilience to failures by automatically re-routing the tree around a failed SCX. (c) The mesh can be reused for multiple source-rooted trees without excessive additional computation. . 46
4.3 Gossamer Protocol Layers. 47
4.4 Gossamer's node discovery protocol: (a) Illustrates an example discovery graph where SCX 1 knows of nodes 2, 3, 4 and 5, while SCX 2 is aware of nodes 6 and 7. For the next discovery round, SCX 1 chooses a random other node, say SCX 2. (b) With Random Pointer Jump, SCX 1 retrieves SCX 2's membership information and as a result SCX 1 inserts edges to nodes 6 and 7 in the discovery graph. The new edges are illustrated by dashed lines. (c) With Name Dropper, SCX 1 transmits its own membership information to SCX 2, and SCX 2 adds the corresponding edges to the discovery graph including an edge back to SCX 1. (d) With Name Dropper++, both SCX 1 and SCX 2 exchange membership information and add the corresponding edges to the discovery graph. 49
4.5 Algorithm used by X_i to determine whether to accept X_j as a neighbor. . . 53
4.6 Algorithm used by X_i to compute the cost function for node X. 54
4.7 The Transit-Stub Model used by the Georgia Tech Topology Generator for constructing simulated internet topologies. This model builds topologies consisting of transit domains and stub domains analogous to the current Internet's backbone transit networks and edge networks. 58
4.8 Percentage of nodes that have discovered (a) at least 90% of the other nodes, and (b) all of the other nodes vs time. 60
4.9 Percentage of nodes discovered by a single SCX vs time. 60
4.10 Distribution of cost ratio ($\frac{C}{C_{mcast}}$) over a range of experiments. 61
4.11 Variation of cost ratio ($\frac{C}{C_{mcast}}$) vs time. 62
4.12 Cumulative number of edge changes per node in the mesh structure over time. Each edge that is added or removed is counted as two changes, one for each node in the edge. 63
4.13 Comparison of the number of physical links with packet duplication for the Gossamer topology vs direct unicast. 64
4.14 Name Dropper++ scaling: Time for everyone to discover at least 90% of the other nodes vs session size. 65

4.15 Variation of cost ratio ($\frac{C}{C_{mcast}}$) vs session size. 66
4.16 Variation of cost ratio ($\frac{C}{C_{mcast}}$) vs node degree ($<k_1,k_2>$ as defined in Section 4.2.1). 67
4.17 Time to stability vs session size. 68

5.1 Examples of adapting to heterogeneity: (a) Layered Media, and (b) Proxy-based approach. 72
5.2 An illustration of losses within a TCP data stream: the lost sequence numbers have no correlation to ADU boundaries and may cover only partial ADUs and/or span multiple ADUs; the flat sequence numbering with TCP provides no semantic information to the receiving application regarding the structure of the data that was lost. 76
5.3 A TINS Namespace: an online presentation consisting of multiple slides in HTML format. The example depicts a typical namespace consisting of containers, ADUs, and multiple instances. 77
5.4 An example of the use of primary and instance names in TINS: every version of the original image is assigned a different instance name that reflects the nature of transformation that was applied to generate the new version. . . . 79
5.5 The scattercast API between the transport framework and the application-level multicasting layer for providing the transport framework with information regarding the underlying topology of the overlay network. The direction of the arrows ($\uparrow$/$\downarrow$) next to the function definitions indicates whether the function call is an upcall or a downcall. 81
5.6 The internals of the scattercast transport framework. 82
5.7 End-to-end loss recovery scheme: (a) Propagation (and suppression) of recovery requests from receivers towards the source, (b) Efficient retransmission of lost data. 85
5.8 Customized reliability: An example of providing customized loss recovery for an online presentation application via the TINS hierarchy. Each container is assigned its own reliability policy based on application semantics. For example, a client with no Java support may decide to ignore losses in the applet container, but require total reliability for HTML. 86
5.9 Customized transmission scheduling: An example of providing customized bandwidth management for an online presentation application via a hierarchy of traffic classes. ADUs are assigned to specific traffic classes and each traffic class is assigned a priority or a portion of the available transmission bandwidth based on what the application deems to be the usefulness of the data in that class. 87
5.10 Customized data adaptation: Applications choose to apply data-type-specific transformations to the ADUs before transmission to adapt the ADUs to bottleneck bandwidth or to apply format conversion to better suit the receiving application's environment. For example, an image ADU may be transformed to reduce bandwidth consumption by discarding color information or resolution from the original image. 89

5.11 The API between applications and the scattercast transport framework. The direction of the arrows ($\uparrow/\downarrow$) next to the function definitions indicates whether the function call is an upcall or a downcall. 91

6.1 The mediaboard application: The client on the left is the source, and the one on the right is a low-bandwidth receiver. 99
6.2 The desktop mediaboard application interacting seamlessly with a PalmPilot mediapad client. The PalmPilot screenshot was taken from the xcopilot, a hardware emulator for the PalmPilot [31]. 102
6.3 The Infocast application: A stock ticker example (snapshot from August 2000).106
6.4 The MusiCast application: an illustration of the use of the scattercast architecture for online audio dissemination. 108

7.1 The BCFS Service Model. Reproduced from [144] 118

List of Tables

3.1 Multicast TTL scope control values 28

6.1 Comparison of device characteristics for a 3COM PalmPilot [1] and a high-end desktop workstation. 100

6.2 Examples of bandwidth savings with ALF-based SCX—PDA protocol. . . . 105

Acknowledgements

At the culmination of this journey through graduate life at Berkeley, it is time to look back and thank the countless people that have contributed to this work. Many people have told me that the Acknowledgments are the most fun part of writing a Ph.D. thesis, and now that I am actually doing it, I must say that I agree with that view. Over the course of this dissertation, I have had the good fortune of interacting with a number of brilliant people both at Berkeley and within the wider research community. Their constant input and comments on this work helped shape its design and implementation.

The single biggest influence during the course of my graduate career has been my thesis adviser, Eric Brewer. His guidance when I was a fresh and naïve young graduate student and his ability to foster independent research skills in all of his students have helped make me the researcher I am today. In this era of lucrative startup opportunities, he is one of the few people I know who has successfully managed both an immensely successful startup and a full-time faculty position. His sharp and clear insights into various aspects of the scattercast architecture and the design of its protocols proved incredibly valuable. I am grateful for his constant guidance, support, and encouragement. I look forward to continuing to have fruitful interactions with Eric long after my departure from Berkeley.

Steve McCanne originally helped me formulate the basis for this dissertation. His insights helped me develop the scattercast architecture from a preliminary prototype for PalmPilot-based whiteboards into the fully generalized system for Internet broadcasting. While he was a professor at Berkeley, Steve routinely held meetings with me to discuss and argue out various design decisions for the system architecture and the protocols and algorithms underlying the architecture. The combination of Eric as my adviser who was great at keeping me focused on the bigger picture and Steve as a co-adviser who delved into the intricacies of the system with me proved to be absolutely beneficial in terms of not only improving the quality of my dissertation but also for honing my research abilities.

In addition to Eric and Steve, the rest of my thesis committee—Randy Katz and Marti Hearst—gave me useful feedback both at the beginning of this thesis through an intensive qualifying exam and later when they read drafts of this work. I continue to be amazed by Randy's ability to quickly assimilate various aspects of the problem and give me sharp and precise feedback within a short amount of time. Marti's comments as an outsider to the field of networking research helped make the thesis more complete by pointing out various places where more clarification could help explain the issues clearly.

The collective group of graduate students at Berkeley was always a source of intellectual challenge and interaction. My almost daily conversations with friends in the department including Steve Gribble, Drew Roselli, Angela Schuett, and David Wagner gave me a lot of fodder for thought as well as valuable criticisms of my work. Without their incisive feedback, it would have been impossible to create high-quality presentations, paper submissions, and class projects. No matter where my research career takes me, these interactions and the skills that they helped me develop over the course of the past five years will always be with me and I am grateful for that.

Another excellent source of feedback throughout the course of my graduate life has been the semiannual Berkeley retreats that I have had the pleasure of attending and presenting at. At these retreats, researchers from other universities and industry are invited to listen to presentations on various students' research projects. Feedback from these retreats and interactions with researchers such as Sally Floyd, Mark Handley, and Steve Deering proved beneficial. I am glad to have been part of this Berkeley tradition and I hope it continues in the future to help other graduate students in the same way that it has helped me.

My past and current office-mates in 445 Soda—Nikita Borisov, Mike Chen, Armando Fox, Ian Goldberg, Steve Gribble, and David Wagner—made it a phenomenal place to work and develop my research ideas in. Even idle conversations with these folks could at times lead to insights into some specific problem that I might have been thinking about. I have been fortunate to have had such a great group of peers to work with and I am sure that this will not be the last opportunity for me to interact with them.

While at Berkeley, I have been affiliated with two research projects: the Daeladus/BARWAN wireless networking project and the MASH multimedia networking project. During the course of these projects, I had the pleasure of working with a stellar team of researchers: Elan Amir, Hari Balakrishnan, Gene Cheung, Armando Fox, Steve Gribble, Tom Henderson, Todd Hodes, Daniel Jiang, Ketan Mayer-Patel, Giao Nguyen, Venkat Padmanabhan, Matt Podolsky, Suchitra Raman, Sylvia Ratnasamy, Mark Stemm, Andrew Swan, Helen Wang, Tina Wong, and Tao Ye.

My decision to come to Berkeley to pursue a graduate degree was influenced by my undergraduate thesis adviser, Professor K. M. Kulkarni. His encouragement and advice during my final year at my undergraduate institution in Bombay and while I was applying to graduate school in the U.S. were instrumental in helping me get to Berkeley.

The support staff at Berkeley have done a tremendous job of helping me as a student and shielding me from the gnarly bureaucracy of the Berkeley administration. Terry Lessard-Smith, Bob Miller, Glenda Smith, Kathryn Crabtree, and Peggy Lau helped make life at Berkeley much smoother than it would have been without them.

My time at Berkeley has not been only about work. I would like to thank my friends for helping me maintain my sanity and keep a balance between work and play. I would especially like to thank the Soda 330 crowd for befriending me when I arrived at Berkeley and helping me ease my transition into a new life in a new country. The entire "Happy Family" has been a wonderful support and, true to its name, has been my family away from home. Finally, all of my friends at Trikone have been a great source of fun, entertainment, and support.

Although a number of people at Berkeley and elsewhere have helped shape my research career, the two people that I owe the most gratitude to for making me the person I am today are my parents. Thanks, Aayee and Baba, for teaching me the values of humanity and for instilling in me a keen sense of appreciation for others. Without your unwavering support and love, I could not have reached this point in my life. Thank you. Along with my parents, my sister, Monica, has been a great friend and an endless source of support and encouragement.

Finally, I would like to thank my partner, Tom, who has been caring, supportive, and patient with me. Tom, thanks for being there.

Chapter 1

Introduction

"Madonna's online concert encounters some static.

"Madonna's London gig, broadcast live on the Internet, was billed as chance for viewers around the world to take 'a virtual front row seat.' But as she burst into her first song, many fans were still queueing up outside the virtual venue, struggling to connect to the live feed."

—CNN News (November 29, 2000)[1]

The above news report from CNN News reflects the fact that in spite of the revolutionary global deployment and use of the Internet for information access, the current generation of Internet technology is not well-suited for high bandwidth *broadcast distribution*: the simultaneous delivery of content streams to a large audience. The overwhelmingly large number of online viewers—9 million recorded logons—at the Madonna concert resulted in numerous difficulties including slow, jerky and cut-off connections dampening what had been billed as one of the biggest live broadcast events ever over the Internet. This experience demonstrated the inability of the currently deployed Internet infrastructure to handle rich quality high bandwidth broadcast media. The Internet works well for point-to-point applications as exemplified by the dramatic success of the world wide web and the proliferation of a wide range of services and applications based on the web. However, simply extending the point-to-point architecture for broadcast distribution using multiple point-to-point streams is not scalable. As was acutely demonstrated by the Madonna broadcast, with millions of clients, neither the network fabric nor the broadcast servers can handle the excessive load presented to the system.

This dissertation is an effort to overcome the barriers to efficient broadcast distribution by using application-level infrastructure technology. Our solution emphasizes the use of application-defined semantics in building an adaptable system for broadcast distribution. We address two key challenges: the design of an architecture and a set of protocols

[1]`http://www.cnn.com/2000/TECH/computing/11/29/madonna.technology.01/index.html`

for efficient broadcasting, and the design of a framework for flexible transport services on top of this architecture that enable applications to tailor the semantics of the transport to their own requirements.

1.1 Motivation

The past decade has witnessed an explosive growth of the Internet. The Internet Software Consortium estimates that the number of hosts on the Internet has grown from less than 6 million in January 1995 to over 70 million in January 2000 [69]. This exponential growth has been propelled by the ever-increasing popularity of the world wide web [15]. Most traditional Internet applications including the web are based on point-to-point communication between a client and a server. As the Internet evolves, we expect to see the deployment of broadcast-oriented services that rely upon a multi-point (one-to-many or many-to-many) mode of communication. Examples of services that can utilize broadcast distribution[2] include Internet video, radio over the Internet, multi-party conferencing, and software distribution. We can classify these applications into the following categories:

Audio/video broadcasting: This involves applications such as Internet radio [100] and video/TV over the net. These applications typically have real-time constraints and involve a single source and many simultaneous receivers.

Multimedia collaboration: Audio and video conferencing applications [94] allow people to communicate with each other. In addition to traditional audio and video channels, the Internet also enables richer forms of interaction such as group drawing spaces, electronic whiteboards, and distributed games. Social considerations typically restrict these forms of applications to few-to-few or few-to-many communication.

Push services: Various forms of information dissemination applications have become popular recently [101, 111]. These services typically involve personal information tracking where a user receives a periodic feed of information such as stock quotes, news headlines, and weather/traffic reports according to his or her interests. A common trait of this class of applications is that most information is periodically updated on a regular basis.

File transfer: This includes applications such as software distribution, automatic software updates, content replication, and web-site mirroring or caching. These applications involve reliable transfer of a large amount of data to many sites.

Traditionally, one of the barriers to the spread of "killer" broadcasting applications such as audio and video has been the high bandwidth required for reasonable quality output

[2]In this dissertation, we use the term *broadcast distribution* to denote one-to-many or many-to-many forms of communication. We reserve the use of the traditional term *multicast* to refer to the specific set of protocols that implement the *IP multicast* network service [32, 34].

and the corresponding lack of last mile access capacity—the bandwidth available between the end client and their ISP. However, as broadband access technologies such as xDSL, cable modems, and broadband wireless become more prevalent, this problem will no longer exist. Commercial researchers predict that the number of Internet users accessing the net using broadband technology will exceed 50 million in the next two to three years [110]. Instead, the key challenge to broadcast distribution will be the delivery technology itself.

Traditional network delivery protocols are not amenable to efficient and scalable broadcast distribution. In particular, the naïve approach of using point-to-point *unicast* communication from the source results in transmitting a duplicate copy of the information generated by the source to each and every client of the broadcast. This is clearly not scalable to more than a small number of clients. Moreover, the IP multicast initiative [32, 34], which attempts to embed support for efficient broadcast distribution, has failed to take off due to its own set of problems. We look at some of the issues that have hampered the acceptance of IP multicast as a usable broadcast distribution service and discuss a solution that ameliorates the situation.

1.2 IP Multicast: Not the Right Answer

For over a decade, the research community has attempted to incorporate support for efficient broadcast distribution into the Internet architecture via a technology called the *multicast backbone* or *MBone* [32, 34]. The MBone extends the traditional point-to-point Internet datagram model to provide efficient multi-point communication using the "IP multicast forwarding service." In this model, each source's data flow is delivered efficiently to all interested receivers by forwarding copies of the source's data packets along a distribution tree rooted at the source (or, depending on the routing protocol, at a core router [12] or a rendezvous point [33]). A source sends a single copy of its data packets and the multicast routers within the network replicate the packets when necessary to forward them to all interested destinations. For large-scale group communication, the bandwidth savings afforded by multicast are enormous, and consequently, a number of multimedia conferencing tools [73, 89, 60, 87, 58] have been developed that exploit multicast and the MBone.

However, in spite of a decade of research on multicast protocols and applications, IP multicast is yet to take off. Although it has been available for research through the experimental MBone network, and has recently been implemented in many commercial routers, most ISPs are still reluctant to enable it in their domains. A number of crucial problems have impeded the global deployment of IP multicast. We identify three specific issues that have affected IP multicast's acceptability as a viable multi-point distribution protocol:

1. **Protocol complexity:** In [38] and [64], the authors cite a number of problems that are inherent in the current IP multicast service model. These problems, including group management, lack of access control, absence of a good inter-domain multicast

routing protocol, and distributed multicast address allocation, have proved to be a significant barrier to wide-spread commercial deployment of IP multicast.

2. **Network and end-system heterogeneity:** The heterogeneity in the Internet makes it difficult to build multicast applications that can simultaneously satisfy the conflicting requirements of the wide range of client devices and networks that span the entire Internet.

3. **Layering complex transport services:** Finally, like IP unicast, the multicast service model provides only best-effort packet delivery, that is, the service makes no guarantee that packets necessarily reach their destinations or that they are delivered in the order they were generated. Instead, richer services such as reliable, sequenced delivery and congestion control are relegated to higher transport or application layers. However, unlike in the unicast world where TCP addresses these issues for most applications, in the multicast domain, these problems are far more complex and much harder to address in the context of a single generic transport protocol.

We next discuss each of these issues in more detail.

1.2.1 Protocol Complexity

One of the key reasons for the success of the Internet as a commercial infrastructure is its simplicity and consequent robustness. In keeping with the principles of end-to-end design [120], the Internet was explicitly designed to leave the core network layer technology simple, robust, and easy to understand, and to migrate all complex services to higher layers. As a result the unicast datagram forwarding service provides a simple best-effort service model and relegates all richer functionality such as reliability, congestion management and flow control to higher layers at the end systems.

However, unlike the unicast model, the IP multicast service model suffers from several drawbacks that have made the protocol quite complex and prevented it from being widely deployed. First, the current multicast service model has no support for effective group management. Primarily, the lack of access restrictions in the service model implies that any malicious participant can cause extensive denial-of-service attacks by flooding useless data on to a popular multicast session. Although this is a problem for unicast as well, it is much more severe in the multicast case since packets can be replicated many times within the network. Similarly, the service model has no mechanisms to prevent unauthorized receivers from tuning in to restricted sessions. A second concern with IP multicast is the per-group state that is required to be maintained by routers. This not only increases the complexity of the routers but also presents serious scaling issues for the routing infrastructure. Third, the service model requires allocating a globally unique multicast address to each session. The lack of a globally coordinated address allocation policy implies that sessions may have address collisions and result in extraneous cross-traffic across sessions. Fourth, unlike IP unicast where intra- and inter-domain routing are well-defined and cleanly separated, there is still no efficient and scalable deployed solution for inter-domain multicast routing. Finally, manageability of the multicast infrastructure is another key issue that troubles ISPs. Very few network monitoring and debugging tools exist for IP multicast. Most of these tools are academic prototypes and are not robust enough for commercial deployment. They only partially address the various problems associated with multicast monitoring and debugging.

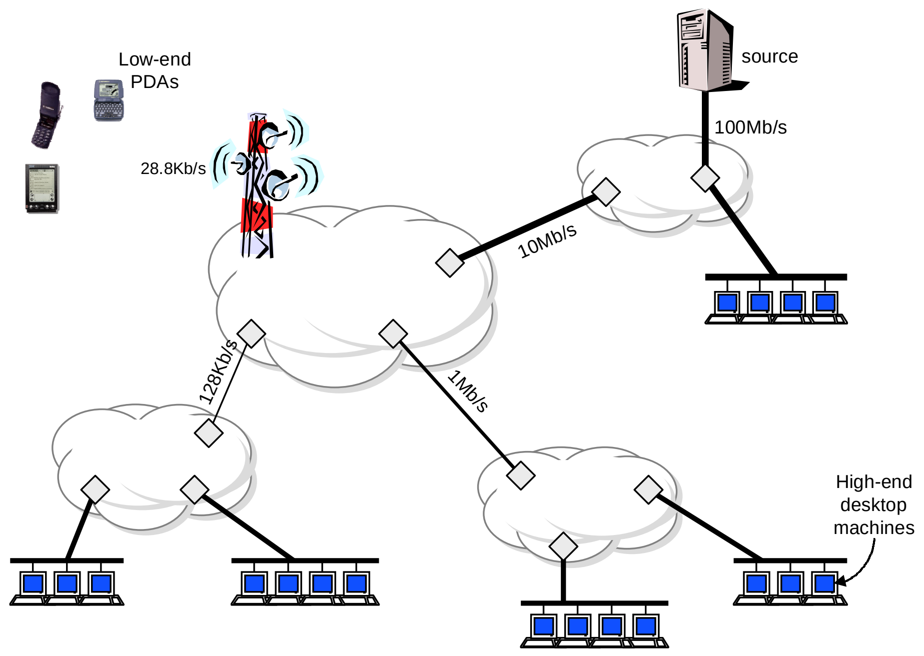
1.2.2 Network and End-system Heterogeneity

Though multicast applications reap enormous performance benefits from the underlying multicast service, they are fundamentally challenged by the heterogeneity inherent in the disparate technologies that comprise the Internet, both within the end systems and across the network infrastructure. Figure 1.1 shows the high variance in client and network capabilities today. End devices range from simple palm-top personal digital assistants (PDAs) to powerful high-end desktop PCs, while network link characteristics can vary by many orders of magnitude in terms of delay, capacity, and error rate. Although technology continually advances the low end of the heterogeneity spectrum, the gap between low-end and high-end systems will inevitably exist far into the future. Hence, any software system designed to function well across such a wide range of characteristics must adapt to the needs of its environment.

When network heterogeneity convolves with the multicast communication model, a communication source is potentially confronted with a wide range of path characteristics to each receiver, e.g., different delays, link rates, and packet losses. Consequently, that source cannot easily modulate its data stream in a uniform fashion to best match the resource constraints in the network. For example, if the source sends at the most constrained bit rate among all paths to all receivers, then many high-bandwidth receivers experience performance below the network's capability, whereas if the source sends at the maximum possible bit-rate, then low-bandwidth paths become congested and receivers behind these congested links suffer. A source cannot simply transmit a stream at a uniform rate and simultaneously satisfy the conflicting requirements of a heterogeneous set of receivers.

Moreover, the wide variation in client capabilities (ranging from extremely limited devices such as PDAs, pagers and cell phones to powerful desktop "work horses") makes it difficult for a single source to adapt its data stream to best suit the needs of all receivers. Although the low-end devices will no doubt get more powerful in the future, the high-end machines too will advance in technology and capabilities, and the gap between the two will persist. Any broadcasting architecture will inevitably have to deal with this extreme degree of heterogeneity.

Although layered media streams [126, 105, 20, 90] have often been proposed as an end-to-end solution for heterogeneity in the context of multicast, this approach is not sufficient. In this approach, a source encodes its data into a number of layers, each of which is transmitted over a separate multicast group. Despite its attractiveness, end-to-end



System Characteristic	Low-end	High-end
Machine	Hand-held PDA	High-end desktop machine
CPU speed	16 MHz	750 MHz
Screen resolution	160x160 2-bit grayscale	1600x1200 24-bit true-color
Memory capacity	2MB physical 64KB address space	512MB physical 4GB address space
Network bandwidth	28.8Kb/s modem	100Mb/s ethernet
Network latency	200-400ms wireless	1ms ethernet

Figure 1.1: End-client and Network Heterogeneity. No single data stream from the source can satisfy the entire range of client devices and networks within the session.

layering is not sufficient to address the extreme forms of heterogeneity that can exist in a broadcast session. The adaptation granularity is on the order of a single data layer, which is not sufficient to encompass the extent of heterogeneity that is possible within the session. Moreover, for a source to have to encode an independent layer to accommodate each of the full range of heterogeneous clients and networks participating in the session is simply not feasible.

1.2.3 Layering Complex Transport Services

Like IP unicast, IP multicast is a best effort service. However, providing higher level services such as reliability, congestion control and flow control for multicast is inherently more difficult than for unicast. In the unicast world, requirements for reliable, sequenced and congestion-controlled data delivery are fairly general, and TCP addresses these requirements for most applications. On the other hand, in the multicast domain, different applications have widely different requirements for reliability, congestion management, and flow control. As a result, these problems are far more complex and much harder to address in the context of a single generic end-to-end multicast transport protocol. Reliable multicast applications are challenged to provide efficient and scalable loss recovery especially in the face of the extreme client and network heterogeneity present in the Internet. In addition, although schemes exist to provide TCP-friendly congestion control in the multicast realm [136, 140], these approaches have limitations. They work only with single-source sessions, and none satisfactorily accommodates bandwidth heterogeneity across the multicast distribution tree.

Recently, protocols such as IPv6, BGMP/MASC [78], and GLOP addressing [93] have attempted to address some of these issues. Researchers have also proposed changing the underlying multicast service model itself (EXPRESS [64] and Simple Multicast [107]) to better manage some of the above problems. However, none of these solutions address the crucial issues of heterogeneity, reliability, and congestion control, which remain a stumbling block for the success of multicast services. Moreover, as new protocols are invented to patch problems inherent in the multicast service model, it has the undesirable effect of making the underlying network layer more and more complex.

1.3 Our Solution: Infrastructure Service-oriented Approach

As described above, the IP multicast service model is too complex to be implemented satisfactorily entirely as a network primitive. In this thesis, we thus distinguish between the notion of IP multicast as a network layer primitive and multi-point data delivery as a higher-level network service. Rather than assume the existence of a global multicast "dial-tone," we view IP multicast as an efficient protocol building block that need not be available everywhere. We instead suggest that broadcast delivery is a service that is best provided by moving up the protocol stack. Thus, instead of providing network layer IP multicast, we move the broadcasting functionality from network routers into higher application layer components. This notion of "application-level multicast" has recently generated a great deal of interest both within the research community [49, 65] and in the commercial world [41, 29].

One possible solution is to migrate the multi-point delivery functionality entirely to the end-clients that participate in the multicast session without any support from the network [49, 65, 62]. Such systems build an on-the-fly dynamic distribution tree consisting of unicast connections across the end-clients participating in the session. However, such an approach inevitably suffers from scalability concerns since building a distribution tree adaptively across a large number of clients fast enough is a difficult problem and cannot scale beyond a few thousand clients.

To scale an Internet broadcasting service to truly large audiences, we will inevitably need support from the network infrastructure. However, instead of relying on the traditional approach of integrating such support into network routers, we advocate building multipoint delivery as an infrastructure service that leverages support from strategically placed application-level components. These infrastructure components rely on well-understood and robust *unicast* transport protocols and couple them with IP multicast for efficient multipoint data delivery. Moreover, the infrastructure components provide a convenient point to embed application-level semantics into the distribution protocol.

To this end, we propose a twofold solution for Internet broadcast distribution: (1) an application-level infrastructure to provide the routing and forwarding services, and (2) a customizable transport framework on top of this infrastructure that leverages application-defined semantics to tune the transport protocol. Our architecture is grounded in a hybrid communication model that partitions a heterogeneous set of session participants into disjoint *data groups*. Each data group is an independent locally scoped IP multicast region consisting of a topologically co-located group of participants. Every data group is serviced by a strategically located network agent. A collection of network agents collaboratively provides the multicast service for a session. Clients locate a nearby agent and tap into the multicast session via that agent. Agents organize themselves into an overlay network of unicast connections and build data distribution trees on top of this overlay structure. The agents are application-aware and use detailed knowledge of application semantics to adapt to the heterogeneity constraints within the session. This allows the infrastructure to adapt a source's data stream with fine-grained control to fit the needs of the variety of end-clients and networks in the session. We call this communication model *scattercast*[3] and the network agents that are central to this model *ScatterCast proXies (SCXs)*[4].

Figure 1.2 motivates this approach by illustrating the components of the architecture and their use to provide efficient broadcast distribution to a heterogeneous range of clients connected to the Internet via a variety of networks as depicted in Figure 1.1. SCXs collaborate with each other to provide an application-aware distribution service that adapts the source's data stream using detailed knowledge of application semantics. This, for example, allows local high bandwidth receivers to continue receiving data at a high rate from the source, while data sent to remote receivers is congestion-controlled by the SCXs.

The scattercast architecture relies on three key concepts. First, to mitigate the hard multicast problems such as bandwidth allocation, address assignment, inter-domain routing, and scalable loss recovery, we partition the large wide-area heterogeneous session into many smaller and simpler homogeneous data groups and provide an *application-level multicast* service by leveraging infrastructure support in the form of SCXs. This divide-and-conquer approach effectively decouples each data group from the vagaries associated with the rest of the session participants. Second, as data flows through an SCX, the SCX

[3]The term *scattercast* is borrowed from prior work by Ratnasamy et al. [116] on a delivery-based model for multicast communication.

[4]In an earlier incarnation of this work, scattercast proxies were called Reliable Multicast proXies or RMXs.

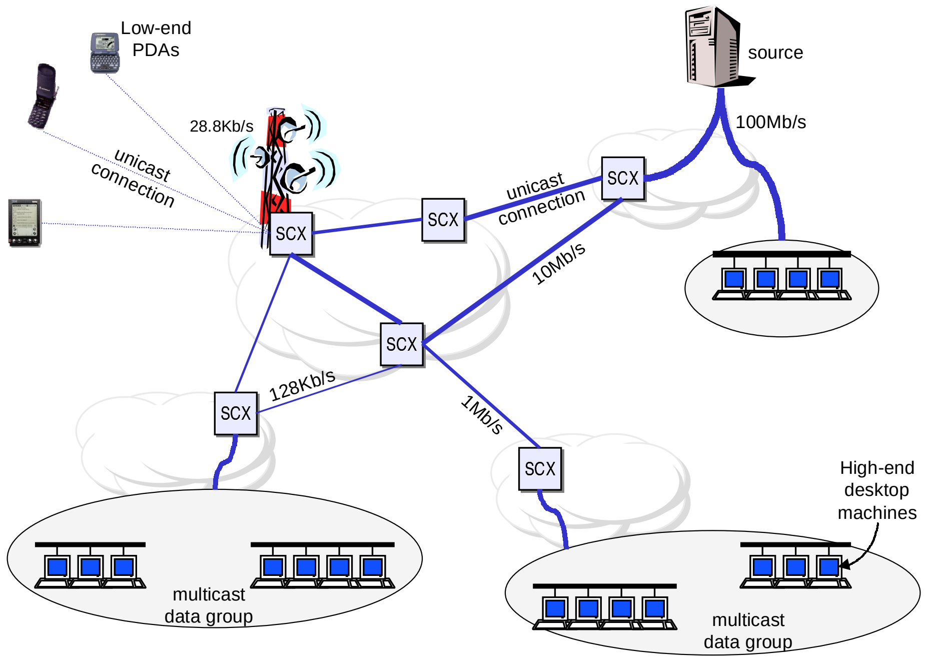


Figure 1.2: The scattercast architecture: Application-level infrastructure proxies (SCXs) form an overlay network of unicast interconnections to feed broadcast streams to a heterogeneous set of clients. Clients communicate with SCXs either via locally scoped multicast groups or via unicast.

uses application-level knowledge to alter the content of the data dynamically or to adapt the rate and ordering of data objects. The SCX allows for the notion of *semantic transport* as opposed to bit-level data transport, that is the transport of information rather than that of the representation of the information. In this manner we lift the constraint that all receivers advance uniformly with a sender's data stream; each receiver defines its own level of reliability for the stream and decides how and to what degree individual data objects might be transformed and compressed. Finally, to support these semantics, we leverage the *Application Level Framing* (ALF) [30] protocol architecture, which argues that application performance can be substantially enhanced by reflecting the application's semantics into the design of its network protocol. This approach to protocol design benefits overall performance since the application is optimized for the network and vice versa.

1.3.1 Pros and Cons of this Approach

A comparison of scattercast with global IP multicast serves to highlight the advantages of our approach. First, by migrating the broadcast distribution service to higher layers, scattercast keeps the underlying network model simple and straightforward. The

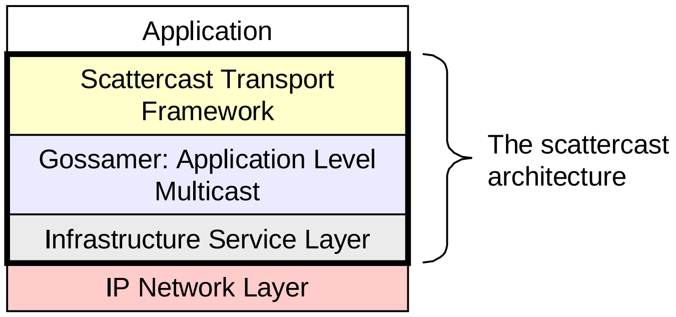

Application
Scattercast Transport Framework
Gossamer: Application Level Multicast
Infrastructure Service Layer
IP Network Layer
The scattercast architecture

Figure 1.3: The various components of the scattercast architecture.

divide-and-conquer tactics of scattercast allow us to isolate the use of IP multicast to well-defined locally scoped data groups thereby eliminating or mitigating the problems associated with wide-area multicast. Second, by explicitly using application-level agents in the network, scattercast also allows for a scenario where SCXs can use application semantics to adaptively modify the content in order to suit the needs of the clients. This property of scattercast is very useful to tackle the heterogeneity that plagues IP multicast applications. Third, building transport level services such as reliability and congestion control is simplified since scattercast can leverage the robustness of well-understood unicast protocols such as TCP for wide-area communication thereby inheriting the reliability and congestion management mechanisms built into TCP. We elaborate on these advantages in Section 3.2.3.

A potential question regarding scattercast is the apparent similarity of its overlay architecture to that of the MBone, which builds an overlay network comprised of local clouds of native multicast connectivity interconnected by unicast tunnels. Scattercast, however, is fundamentally different from the MBone in that it explicitly incorporates application-level intelligence into the design of its forwarding and transport algorithms. Moreover, as we shall see in the following chapters, the overlay structure within scattercast is dynamic and self-configurable as opposed to the static hand-configured structure of the MBone.

Although scattercast does provide a number of advantages for efficient broadcast delivery, these advantages come at a price. The scattercast architecture introduces new components into the network (SCXs) that are potential points of failure that can disrupt the service. Moreover unlike multicast routers, which typically execute the routing and forwarding algorithms in fast specialized hardware, SCXs are software entities that can give rise to scalability concerns. We look at these issues in more detail in Section 3.2.4.

1.3.2 Components of the Scattercast Architecture

The scattercast model for broadcast communication embraces a dynamic and flexible architecture that hinges on application-level infrastructure support for multi-point distribution. As depicted in Figure 1.3, this architecture is composed of three core components: an infrastructure service layer that hosts that SCXs, an application-level multicast service that provides the routing and forwarding functionality across the session, and an adaptable transport framework that includes application-semantics to tune the data transport.

Infrastructure Service Layer

The Internet as it stands today does not have any built-in support for including application-programmable software components such as SCXs into the network infrastructure. The current Internet infrastructure consists mostly of routing components and some higher level services such as DNS, e-mail servers, and web services. There is, however, no explicit support for hosting and ensuring the fault-tolerance, scalability and availability of programmable application-level network services such as SCXs. Although this is a crucial component for deploying the scattercast architecture, the issues it raises and their solutions are essentially orthogonal to the architecture itself. Hence, rather than design and implement a new platform for hosting the SCX infrastructure service, we present some of the design issues that must be addressed by such a platform and our experiences with some such platforms in Chapter 2. In particular, we have experimented with building general-purpose service deployment platforms such as the Scalable Network Services framework [22] and the Active Services infrastructure [8, 25]. In Chapter 2, we discuss how these platforms can be used to host robust SCX services.

Application Level Multicast

One of the key tenets of scattercast is the migration of the network-layer packet forwarding service up to the application layer by incorporating application-aware SCXs into the forwarding framework. We need to define a set of protocols and mechanisms that SCXs can use to communicate with each other and forward packets from SCX to SCX for disseminating a source's data to the entire session. Just as the network layer Internet architecture provides a well-defined structure for IP routing and for peering of IP networks, so also this new scattercast service requires a framework that imposes structure on the peering model for SCXs and the interaction across SCXs, and between SCXs and clients. As a result, at the core of scattercast is an application-level multicast protocol called *Gossamer* that SCXs use to locate each other in a decentralized manner, to self-configure themselves into an adaptive and efficient overlay mesh of unicast interconnections, and to forward data to the entire session. SCXs run a variant of a distance-vector routing protocol [32, 137] on top of this overlay structure and effectively build source-rooted reverse-shortest-path distribution trees[5].

[5]Distance-vector protocols for multicast build source-rooted trees that are composed of the shortest paths from the receivers towards the source, that is, the "reverse shortest paths" from the source to all receivers.

Application-aware Transport Framework

The Gossamer protocol provides a basic application-level multi-point forwarding service. However, for practical applications, scattercast must additionally address the question of providing higher-level transport services such as reliability and congestion management. These questions are further complicated by the problem of heterogeneity as outlined in Section 1.2.2. What is needed is a flexible transport framework on top of the scattercast architecture that not only enhances the forwarding service by incorporating support for reliable congestion-controlled communication, but also is capable of adapting the transport protocol to address the heterogeneity constraints inherent in the session and to tune the protocol to the needs of the specific application and client environments. The transport framework needs to address three crucial issues: how to provide efficient and scalable end-to-end reliability across the entire session, how to deal with bandwidth management and congestion control across the wide area, and how to adapt to the heterogeneity that exists in the Internet?

In Chapter 5, we present our scattercast transport framework which addresses these issues and effectively accommodates a wide range of applications. The scattercast transport deliberately exposes the Gossamer overlay topology to the transport layer and flexibly embeds higher-level application semantics into the transport protocol to adapt it to individual applications' or clients' requirements.

1.4 Contributions

The core focus of this dissertation is the design and implementation of a framework for broadcast distribution that encompasses a wide range of application environments and can support a heterogeneous assortment of client devices and network characteristics. Our main contributions are:

The Scattercast Architecture: We present the design and implementation of scattercast, an infrastructure-service-based architecture for efficient broadcasting. The scattercast architecture embodies the principles of application-aware network computation by incorporating application-level network agents—SCXs—in the design of the broadcast protocols.

Gossamer—A Protocol for Application Level Multicasting: We present a complete design, implementation, and evaluation of Gossamer, a protocol for organizing a collection of SCXs into an overlay distribution structure and routing and forwarding data flows from a source to the rest of the session. Gossamer uses decentralized adaptive algorithms to allow SCXs to configure themselves into an overlay of unicast interconnections and to incrementally enhance the overlay structure for efficient distribution. A detailed set of simulations demonstrate the effectiveness of the Gossamer algorithms for building application-level distribution trees. Gossamer is certainly not the only application-level multicasting protocol that is possible for scattercast. It is the result of our initial experimentation with building the various components of the scattercast architecture.

An Application-customizable Transport Framework for Scattercast: We have designed and implemented a framework for providing flexible transport services for scattercast. By entrenching application-awareness within the transport framework, we allow scattercast applications to customize the behavior of the framework to suit their own data semantics and to provide application-specific adaptation of the data stream to adjust it to varying heterogeneity constraints. Although significant work has been carried out with respect to proxy-based transport for specific applications such as web access [47] and real-time media gateways [9], to our knowledge the scattercast architecture is the first to provide a single generic application-aware transport framework that can be applied to a wide range of application requirements such as flexible data reliability, customizable congestion management, and real-time constraints.

Real Application Prototypes: To demonstrate the usability and flexibility of the scattercast architecture, we have experimented with a number of different applications on top of this architecture. We present our experiences with these applications and the different ways in which they customize the scattercast architecture.

Although this dissertation addresses the specific problem of multi-point data distribution, we believe that our solution of using application-aware infrastructure services represents a radical new approach for building network protocols. Hence, the most important contribution of this thesis is to open a new arena for network research that attempts to enrich the traditional Internet architecture by embedding application-level services within the network infrastructure.

1.5 Thesis Organization

The rest of this dissertation is organized as follows: In the next chapter, we discuss the issues that arise as a result of the infrastructure-service nature of the scattercast architecture. We present a case for using clusters of workstations as the implementation platform for building infrastructure services.

In Chapter 3, we present a detailed overview of the scattercast architecture. We discuss the network environment in which scattercast operates and the components of the Internet architecture that scattercast builds upon. We present the scattercast service model and evaluate the advantages and disadvantages of this model. This chapter concludes with a discussion of some of the components that underlie the architecture.

Chapter 4 presents the Gossamer protocol that scattercast uses to perform application-level packet routing and distribution. We discuss the various components of Gossamer and present a detailed evaluation of the protocol with respect to its performance and scalability.

On top of Gossamer, scattercast builds an application-customizable transport framework that can be adapted to individual applications' constraints. Chapter 5 presents the details of the scattercast transport framework and demonstrates the ability of the framework to tune the transport protocol to different application environments.

The key demonstration of the flexibility of scattercast is its use in a range of different applications. To this end, in Chapter 6 presents an overview of a number of applications that we have experimented with on top of the scattercast architecture and discusses how these applications customize the architecture components to match their data semantics.

Finally, Chapter 7 presents a survey of related work in this area and describes how the various related research has contributed to the design of scattercast. In conclusion, Chapter 8 describes future directions for this work and presents a summary of our contributions.

Chapter 2

Network Infrastructure Services

The scattercast architecture as proposed in Chapter 1 represents a radical departure from traditional models of Internet communication. The Internet has typically been looked upon as a bit-level transport infrastructure that provides a uniform interface and a set of protocols for sending data packets from one point to another across the network. This simplistic model for the Internet provides a best-effort IP dial-tone service effectively abstracting away the details of how packets are forwarded through the network. More complex services such as reliable in-order packet delivery and congestion and flow control are layered on top of this underlying network model. However, as articulated earlier, this simple model is not sufficient to provide robust, scalable, and efficient broadcast data delivery, and we instead rely on an infrastructural overlay architecture that sits on top of the IP network and is specifically designed for Internet broadcasting.

In this chapter, we investigate the implications of this design choice. In particular, we note that introducing a new infrastructure layer into the Internet introduces additional points of failure and can lead to decreased robustness unless carefully architected to avoid faults and to automatically recover from faults when they do occur.[1] Moreover, we must ensure that this infrastructure layer is designed in a way that allows it to scale depending on the offered load. We describe how these properties of fault tolerance and scalability can be achieved in a service-independent manner that allows us to focus on the specifics of the broadcasting architecture while shielding us from the service management details.

2.1 Evolving from a Plumbing System to a Richer Service Architecture

The Internet infrastructure started as a collection of low level routers, traffic management components and network "pipes"—cables and wires—that provided the "plumb-

[1]Failure in this context refers to faults and bugs within individual nodes that can cause the service to get disrupted. In addition to such local node failures, the scattercast architecture must also deal with failures due to wide-area network partitions. This issue is addressed in Chapter 4.

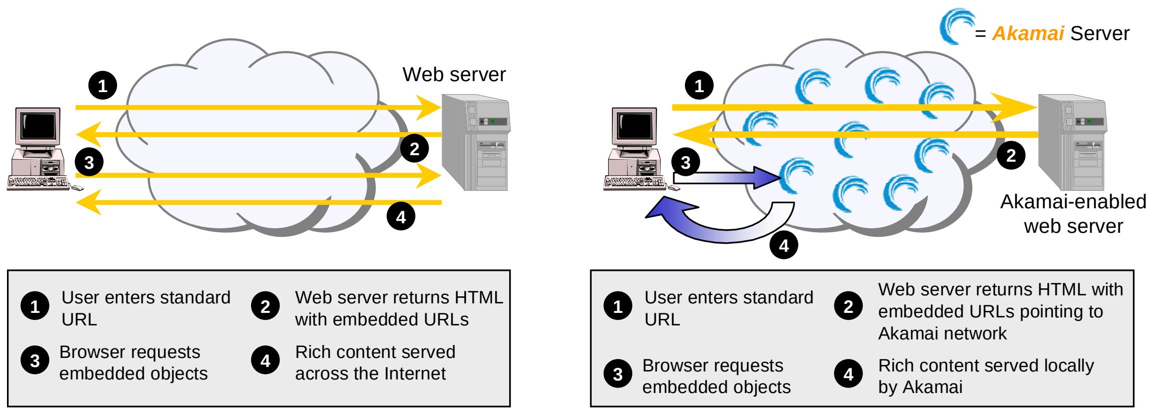

= Akamai Server
Web server
Akamai-enabled web server
1
2
3
4
1 User enters standard URL
2 Web server returns HTML with embedded URLs
3 Browser requests embedded objects
4 Rich content served across the Internet
1 User enters standard URL
2 Web server returns HTML with embedded URLs pointing to Akamai network
3 Browser requests embedded objects
4 Rich content served locally by Akamai

Figure 2.1: The Akamai Infrastructure Service.

ing" to make data flow electronically from one computer to another. The success of the Internet architecture can be attributed in part to the set of protocols and technologies such as the Internet Protocol (IP) that enabled us to build large, highly heterogeneous networks with reasonably efficient routing. At the network level, the Internet Protocol provides a basic connectionless best-effort packet delivery service. Network end-points send packets to each other simply by including the destination Internet address in the packet headers, and the underlying network infrastructure takes care of figuring out the most efficient route towards the destination. Because the connectionless IP service routes each packet separately, it does not guarantee reliable, in-order delivery. Instead, such services are provided at a higher level in the form of a richer transport protocol such as TCP. This combination of a simple packet delivery service and a richer higher-level transport service provided a convenient abstraction for application programmers and quickly gave rise to network applications such as electronic mail, file transfer, and remote login.

This basic network plumbing infrastructure has served us well and has resulted in an information revolution especially with the arrival of the world wide web [15]. The web has provided a platform for developing a wide range of application-level services such as information portals, online traffic and weather reports, online banking, e-commerce and web-based e-mail.

However, as the Internet evolves and as applications get more complex, they demand richer network functionality. Many applications are no longer satisfied with just a simple IP dial-tone service from the network, but actually demand more complex and application-aware computation from within the network. A number of ISPs, for example, already rely on transparent web caching services [67] to provide quick and efficient access to commonly visited web sites. Another example of a system that transparently enriches existing network infrastructure is the FreeFlow architecture from Akamai Technology [3]. Akamai attempts to solve a crucial problem facing many commercial web sites today: how

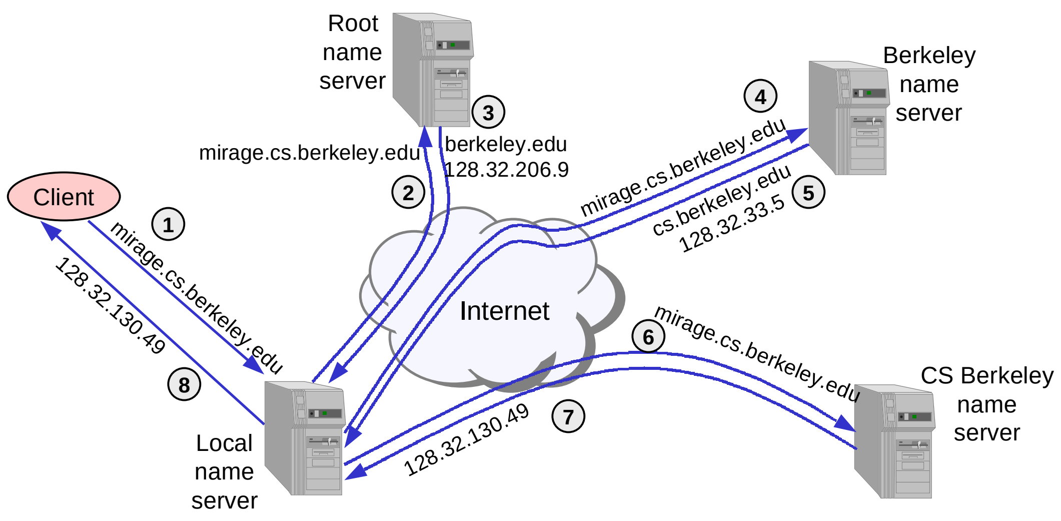


Figure 2.2: The Domain Name System: An illustration of how the DNS infrastructure resolves a name query for the host *mirage.cs.berkeley.edu*. The numbered arrows depict the sequence of steps involved in resolving the request.

to provision their sites to deal with peak crowds and provide speedy performance. Akamai's solution is to provide universal content replication by building a global network of machines consisting of a few thousand nodes located at major access points around the world. When a user requests a page from a site that relies on Akamai, his or her browser is redirected to an Akamai server to download the high-bandwidth components of the web site such as images, banners, etc. Based on a real-time network map, Akamai directs requests to the server best able to satisfy each request, resulting in better performance and reliability. Figure 2.1 depicts a schematic of the Akamai architecture. The Akamai service effectively provides a transparent mechanism for content providers to enhance performance and deal with traffic bursts via an infrastructural solution.

An even more commonplace example of an infrastructure service that provides value-added functionality is the ubiquitous Domain Name System (DNS) [95] that all users of the Internet (knowingly or not) rely on to translate user-friendly host names to Internet addresses useful for routing data across the network. This valuable service allows users to identify network computers with meaningful high-level names (such as `www.berkeley.edu`) instead of the obscure 32-bit IP numbers used by the underlying routing infrastructure. In the early days of the Internet, there was no comprehensive system for mapping host names to IP addresses; instead, a central authority maintained a flat table of name-to-address bindings for all the hosts on the Internet. As the Internet grew beyond a few hundred nodes, this centralized naming system proved ineffective and the Internet adopted DNS, which uses a hierarchical decentralized name space rather than a flat name space. Name servers spread all across the Internet are responsible for portions of this hierarchical name space. The name servers themselves are organized into a hierarchy that is similar to the naming system hierarchy. Figure 2.2 shows the organization of the Internet domain name system. DNS has proven to be one of the many valuable infrastructure components that have propelled the evolution of the Internet from a primitive research and academic network to a full-blown commercial scale system.

The scattercast architecture outlined in Chapter 1 takes the notion of network infrastructure support one step further. Traditionally, the task of delivering data packets from a source to its destination has been performed transparently by the network plumbing layer. In scattercast, we instead move this functionality up the protocol stack and provide broadcast delivery of packets via application-level components that are embedded within the network infrastructure.

A natural question that arises out of this movement towards introducing additional infrastructure components into the Internet is that of the robustness and availability of these new components. Moreover, when these components provide crucial functionality such as name resolution, content delivery, or broadcast distribution, they must be carefully architected to ensure that they can deal with the offered load and potentially scale to absorb any increase in load. We thus identify two critical issues that any infrastructure component for the Internet must address:

Availability and fault tolerance: As much as possible, the infrastructure service must remain available in the face of system crashes, network failure, and hardware upgrades. The service as a whole must be available 24×7 despite transient partial hardware or software failures.

System scalability: The service must be able to scale incrementally with the offered load. When the load offered to the service increases, the service should be able to gracefully absorb the excess load.

Going back to the DNS example, the architects of the system took specific precautions to ensure that failures of one or more name servers would not bring the entire system down. If a DNS server fails, it can render a portion of the name space unavailable for name resolution. To avoid this situation, DNS requires that each domain or sub-domain of the name space be served by at least one replica in addition to the primary name server for that domain. In particular, the root name server of the domain name hierarchy can potentially be a single logical point of failure for the entire name system. To avoid such a catastrophe, the root of the Internet namespace is held in thirteen geographically distributed name servers operated by nine independent organizations. In a worst case scenario, all thirteen of the root name servers would have to fail to cause significant disruption to Internet operation. Another issue that the root name servers in particular need to address is that of scalability. Since all inter-domain name resolutions need to access the root server, this represents a significant bottleneck. To improve overall performance, all Internet name servers use name caching to optimize their search costs and to avoid overloading popular name servers.

Although DNS implements a custom solution to the availability and scalability problems faced by infrastructure services, it provides lessons that can be incorporated into a generic platform for supporting such services. DNS uses resource replication to avoid loss of availability due to hardware or software faults, and load balancing and caching techniques to provide good performance. Some of these ideas can be generalized to provide a programmable infrastructure service platform for building services such as broadcast distribution while satisfying the requirements of availability and scalability.

2.2 The Case for Clusters

In [46], we demonstrated that clusters of workstations have some fundamental properties that can be exploited to meet the above requirements. A cluster consists of a potentially heterogeneous collection of independent commodity workstations linked together by an extremely high-speed local area network. Clusters provide three primary benefits over single larger machines such as SMPs: high availability, incremental scalability, and the cost/performance and maintenance benefits of commodity PCs. We elaborate on each of these in turn.

High Availability: Clusters have natural redundancy due to the independence of the nodes: each node has its own buses, power supply, disks, etc., so it is "merely" a matter of software to mask (possibly multiple simultaneous) transient faults. Network services can exploit this redundancy to avoid service interruption by relying on software techniques such as shadow processes (secondary backup process in case the primary process fails) and component replication. A natural extension of this capability is to temporarily disable a subset of nodes and then upgrade them in place ("hot upgrade"). Such capabilities are essential for network services, whose users expect 24-hour uptime despite the inevitable reality of hardware and software faults due to rapid system evolution.

Scalability: Clusters are well suited to network service workloads which are highly parallelizable. For example, in broadcast distribution, each broadcast channel or session is essentially independent of all other sessions. For such workloads, large clusters can dwarf the power of the largest of single unit machines. The workload associated with the various broadcast sessions can be easily distributed across the entire cluster.

Furthermore, the ability to grow clusters incrementally over time is a tremendous advantage for service deployment. If the existing cluster cannot cope with the load presented to the system, it can be grown to handle the excess load "simply" by adding more machines to the cluster. Of course, this requires careful architecting of the cluster management software on the part of the cluster administrators. With network services where capacity planning depends on a large number of unknown variables, incremental scalability replaces this sort of capacity planning with relatively fluid reactionary scaling: add a machine when you detect excess load. In relation to traditional SMPs, clusters correspondingly eliminate the "forklift upgrade," in which you must throw out the current machine (and related investments) and replace it with an even larger one.

Commodity Building Blocks: The final set of advantages of clustering follows from the use of commodity building blocks rather than high-end, low-volume machines. The obvious advantage is cost/performance since memory, disks, and nodes can all track the leading edge. Furthermore, since many commodity vendors compete on service (particularly for PC hardware), it is easy to get high-quality configured nodes in 48 hours or less. In contrast, large SMPs typically have a lead time of 45 days, are more cumbersome to purchase, install, and upgrade, and are supported by a single vendor, so it is much harder to get help when difficulties arise. Once again, it is a "simple matter of software" to tie together a collection of possibly heterogeneous commodity building blocks.

To summarize, clusters have significant advantages in terms of scalability, growth, availability, and cost. Although fundamental, these advantages are not easy to realize. Developing cluster software and administering a running cluster remain complex. Cluster-based software must deal with issues such as partial failure, state management across machines, and administration and monitoring of a large collection of machines. Fortunately, it is possible to isolate the software necessary to run and manage clusters in a cleanly separated middleware substrate that service authors build on top of. Such a substrate provides the key functions of scalability, load balancing and fault tolerance. In effect, we can view this substrate as a "cluster operating system," a software layer that isolates the service author from the details of maintaining a large-scale network service in a manner similar to conventional operating systems, which isolate users and application developers from having to deal with the intricacies of the underlying computer hardware.

2.3 Cluster Operating Systems

In the past few years, there has been a substantial amount of research effort poured into developing cluster operating system platforms specifically designed for network services. We have experimented with two cluster operating system platforms: Scalable Network Services (SNS) [22, 46] and Active Services [8, 25]. We now present an overview of these systems.

2.3.1 Scalable Network Services (SNS)

The SNS framework was originally designed to provide an implementation platform for web-based services such as web proxies [47]. But the basic framework can be utilized for other forms of network services as well. SNS effectively decomposes a network service into

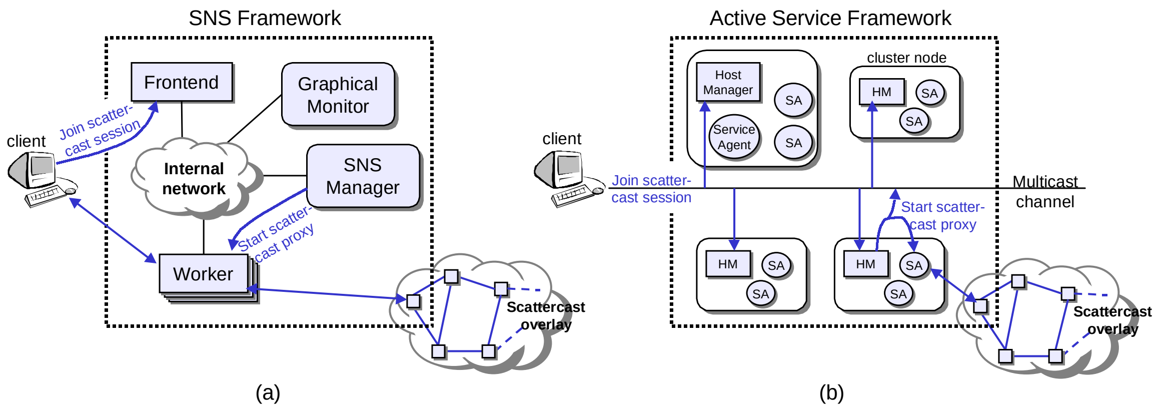


Figure 2.3: Cluster Operating Systems and their use for hosting scattercast services: (a) The SNS Framework composed of a *frontend* to accept join requests from clients, a *manager* that spawns new *workers* on demand, and a *graphical monitor* to provide an administration console for the cluster. (b) The Active Services framework composed of *host managers* on each cluster node that manage one or more *service agents*.

a number of "workers." A pool of workers potentially composed into a UNIX-like pipeline collectively provides the network service. Each type of worker may be instantiated one or more times depending on offered load. Workers are assumed to be stateless, so fault recovery is straightforward. If a worker fails, another one is started possibly on a different node to take over the tasks of the failed worker. The offered load is balanced across the entire cluster via fine-grained load balancing and replicating workers on multiple machines. The system scales gracefully when it detects overload by spawning new workers on additional nodes. A frontend node provides the interface to the SNS cluster as seen by the outside world. It shepherds incoming requests by matching them with the appropriate workers, waiting for the workers' response, and returning the result to the requesting client. An SNS Manager coordinates the operation of the entire cluster and a graphical monitor allows a human administrator to investigate the state of the cluster. Figure 2.3 (a) shows a schematic of the SNS framework.

The design of the SNS framework reflects its heritage as a platform for web-based services. Tasks in SNS are built around the request-response paradigm: a client sends a request, the frontend intercepts it, forwards it through a chain of workers, and returns the result. This model is clearly unsuitable for streaming applications such as audio/video broadcasting. The built-in load balancing rules for SNS workers are assume short-lived tasks as opposed to the session-duration tasks performed by a typical streaming broadcast worker. To build a viable broadcasting system on top of SNS, these rules and the use of the frontend as a single bottleneck point for data flow would have to be modified.

2.3.2 Active Services

The Active Services platform [8, 25] is another example of a cluster operating system designed for building efficient scalable network services. Like SNS, Active Services decomposes its service implementation into components known as service agents. Unlike SNS, however, it does not rely on a frontend node to be the single aggregation point for all clients' requests. Instead, a group of host managers (one on each node in the cluster) coordinate in a distributed fashion to direct requests to appropriate service agents, to create new agents when necessary, and to recover from system faults. Service authors implement their service as one or more agents. Agents are dynamically introduced into the cluster by the host managers when clients make requests for specific services. Figure 2.3 (b) depicts the basic architecture of an Active Services cluster.

The Active Services design centers around the principle of *soft state* operation [115]. All components within the system—host managers and service agents—utilize only soft state with periodic refresh messages to keep the state up-to-date. This design simplifies fault recovery and service management since no complicated state recovery procedures are required after failure.

The Active Services programming model allows clients to specify explicitly which agents are run on their behalf. Digitally signed agent code repositories may reside on the web and clients may request a service cluster to execute a specific agent based on a URL provided by the client. Clients can also be explicitly involved in the fault tolerance mechanisms by incorporating smarts into the clients that can detect agent failure and take proactive steps to notify the cluster to start a new agent.

This platform has been used successfully in the design of a media gateway system to allow impoverished clients to access bandwidth-heavy multimedia sessions on the MBone [4], and as a base for building a multimedia archive server [124, 123].

2.4 Broadcast Distribution: An Application for Cluster Operating Systems

In various prototype implementations of the scattercast architecture, we have used one or both of the cluster operating system frameworks described above to build scattercast proxies. Scattercast proxies were implemented as workers in the SNS framework and as service agents on top of the Active Services platform.

Although Active Services is more amenable to streaming session-oriented applications such as scattercast, it does not have the fine-grained level of load balancing that is designed into the SNS framework, nor does it have mechanisms for automatic incremental scaling of the cluster without having to first bring the entire service down. We expect that a realistic implementation of a cluster operating system for commercially deployable scattercast proxies would combine features from both SNS and Active Services. However, the design of such an ideal cluster operating system is orthogonal to the scattercast architecture itself. For the purpose of this dissertation, we assume that there exists a cluster operating system that deals with the actual details of launching scattercast proxies when required, monitoring them for faults, and recovering from failures when necessary. The cluster operating system allows us to view the scattercast cluster logically as a single large virtual machine. Internally, the cluster operating system may harness multiple nodes to run scattercast proxies on depending on the load on the system. The cluster operating system masks failures within the cluster as much as possible—if a proxy fails, it is automatically restarted possibly on a different node within the cluster.

In the following chapter, we will look at the details of the scattercast architecture. We shall present an overview of the service model associated with scattercast and discuss some of the issues that arise as a result of the service model.

Chapter 3

Scattercast Architecture

In this chapter, we present a detailed overview of the scattercast architecture. We discuss the various architectural components that form the foundation of our broadcast distribution framework and define the service model that scattercast presents to its clients and applications. We continue with a discussion of the pros and cons of the scattercast distribution model and present the details of some of the components that underlie the architecture.

3.1 The Network Environment

The scattercast framework builds upon the Internet architecture and its delivery semantics for point-to-point (unicast) and multi-point (multicast) transmission. In this section we present an overview of the Internet architecture and the assumptions that we make of the architecture in the design of scattercast. Our framework relies upon the following key Internet concepts:

The Internet Protocol: The Internet protocol architecture and its robust layered design for best-effort and reliable data delivery is the bedrock for the scattercast framework. Scattercast constructs its broadcast delivery service by leveraging the various Internet protocols and components to construct a more complex higher level distribution service.

Locally scoped IP multicast: To provide efficient broadcast delivery, scattercast leverages the support of local- or site-area IP multicast where it is available. In combination with wide-area unicast distribution, scattercast provides local-area efficiency as well as wide-area robustness and scalability.

Opacity of the IP routing topology: Rather than embed scattercast components within the IP routing architecture, we leave the routing layer untouched and make no assumptions about the packet routing and forwarding algorithms and the topologies associated with the various internetworks that span the global Internet.

Infrastructure service clusters: To provide efficient large-scale broadcasting, we leverage application-level intelligence within the network in the form of clusters of machines embedded within the infrastructure to host the scattercast service.

We now discuss each of these concepts and their relation to the scattercast architecture.

3.1.1 The Internet Protocol

The Internet architecture encompasses the collection of mechanisms and protocols that allow machines on the Internet to communicate seamlessly with one another. The Internet Protocol (IP) provides a network-level communication technology to interconnect a diverse collection of networks and to route data packets from network to network. The nodes that interconnect the networks and forward packets from one network to another are called *routers*. The routers across the Internet collaborate with each other to construct a routing topology over which the packets can be efficiently forwarded from a source to its destination.

The two underlying principles that guide the design of the Internet Protocol are (1) network technology independence, and (2) universal interconnection. Although IP is based on conventional packet switching technology, it is independent of any particular underlying network hardware configuration. The Internet is composed of a variety of network technologies such as ethernet, ATM networks and fiber optic cables. IP provides a common interface layer that allows hosts connected to this diverse Internet to send data packets without having to understand the details of the underlying network technologies. Secondly, IP provides universal interconnection by allowing any pair of machines within the Internet to communicate with each other. Each machine has a unique IP address that identifies the machine across the entire Internet. Every data packet embeds the addresses of its source and destination within its headers.

The core philosophy associated with the IP service model is that of a *connectionless delivery system*. An IP *datagram* is the fundamental unit of data delivery. Each datagram includes in its header sufficient information to allow the network to deliver it to its appropriate destination without requiring any initial setup to tell the network how to handle the datagram. This connectionless delivery model ensures that the forwarding technology used within the Internet protocol architecture remains simple, stateless and straightforward. A second key concept that defines the IP service model is its *best effort* delivery semantics. Although IP makes every effort to deliver datagrams to their destinations, it makes no guarantees. Since each datagram is forwarded independently, the network infrastructure may drop packets due to bit errors or congestion or reorder packets causing them to arrive at their destination out of order.

To provide connectionless datagram forwarding, each router in the Internet maintains a *routing table* that allows it to map destination networks to one of its many interfaces.

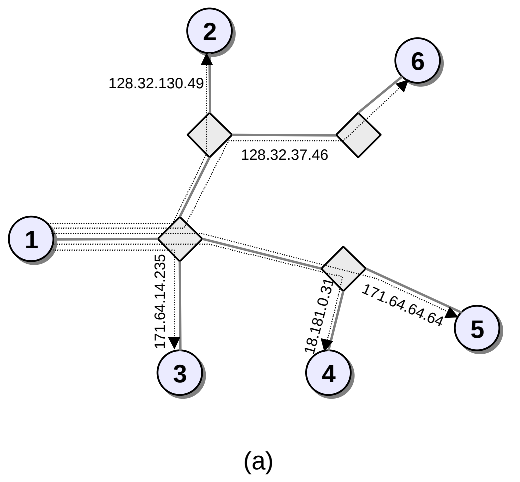
When a packet arrives at a router, it looks up the destination in its routing table to determine the outgoing interface to use to direct the packet towards its destination. Routers execute a distributed routing protocol to discover efficient paths across the network. Within a single administrative domain, routers may use distance vector protocols such as Routing Information Protocol (RIP) [63] or link-state protocols such as Open Shortest Path First (OSPF) [96, 91], while interdomain routing relies on protocols such as the Border Gateway Protocol (BGP) [85].

Since IP only provides an unordered best effort host-to-host communication model, the Internet architecture builds more complex services such as application-level demultiplexing and reliable or real-time packet transport on top of the IP layer. The User Datagram Protocol (UDP) provides a simple wrapper around the IP service to allow applications on the source and destination machines to directly identify each other via an abstract per-host locator called a *port*. Similarly, the Transmission Control Protocol (TCP) provides a connection-oriented reliable byte stream service across applications on top of the IP layer. TCP guarantees the reliable in-order delivery of a stream of bytes while providing effective flow-control and congestion-control [70] mechanisms. In addition, the Real-time Transport Protocol (RTP) [125] provides transport-level support for real-time media such as audio and video.

3.1.2 Locally Scoped Multicast

The TCP/IP protocol architecture described above supports a point-to-point or *unicast* mode of communication. Scattercast builds its multi-point broadcast service out of this basic unicast transmission model. But, to provide large-scale efficient broadcasting, scattercast also leverages *multicast* transmission wherever possible. Unlike unicast, multicast provides an efficient network-level delivery service that allows a source to send a single data packet that is then replicated at appropriate branch points by the routers within the network so as to reach all of its destinations efficiently. However, as described in Section 1.2, global IP multicast is not an appropriate solution for wide-area broadcasting. We instead leverage the efficiency of multicast only within the local area when available.

The IP multicast service model provides an abstraction called the *host group* [27]. The is a Class D IP address (224.0.0.0–240.255.255.255) that provides a level of indirection by allowing sources to send data to a group of multicast receivers without having to explicitly know or specify the list of receivers in the multicast packets. As depicted in Figure 3.1 (b), a source simply addresses its packets to a multicast group address and the routers within the network transmit the packets along a distribution tree to all interested receivers. Contrast this with multiple-way unicast (Figure 3.1 (a)), which requires the source to address each individual receiver explicitly in independent unicast packets. To provide the host group abstraction, multicast routing protocols such as Distance Vector Multicast Routing (DVMRP) [32, 137], Protocol Independent Multicast (PIM) [33], and Core-Based Trees (CBT) [12] construct efficient spanning trees across the network rooted either at the



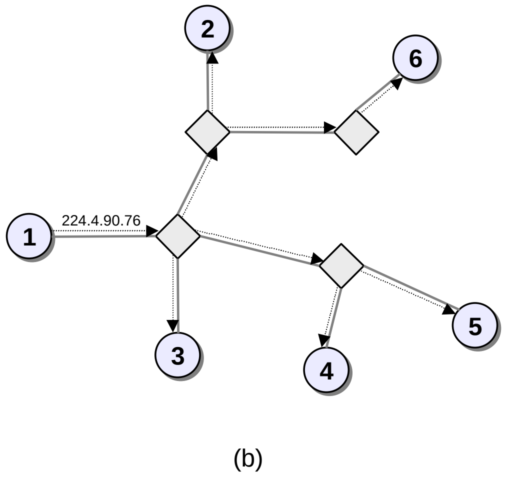


Figure 3.1: Unicast vs. multicast transmission: In (a), source 1 unicasts a separate stream to each recipient; the packets of each stream explicitly specify the recipient's IP address. In (b), source 1 multicasts a single stream to a multicast group address (224.4.90.76) and the routers replicate the stream at appropriate branch points. The source does not need to know the recipient's IP address; the multicast group address provides a level of indirection.

source or at a well-known "core" within the network. Like unicast, IP multicast only provides best-effort delivery and protocols such as Scalable Reliable Multicast (SRM) [43] and Reliable Multicast Transport Protocol [82] build reliable transport on top of this best-effort model.

Scattercast leverages IP multicast only within a local domain. The key mechanism to restrict a multicast flow to a local domain is known as *multicast scoping.* A multicast source may use two forms of scope control: TTL-based scopes and administrative scopes. Traditionally, IP multicast uses the time-to-live (TTL) field in the IP header to limit the range of a multicast transmission. As in unicast, every time a multicast router forwards a multicast packet, the router decreases the packet's TTL by 1. By default, a router does not forward packets with a TTL value of 1. This threshold can be modified to another value for each interface in a multicast router. If a multicast packet's TTL is below the threshold, the packet is dropped. For example, if we set the TTL threshold to 15 on a router interface, only packets with a TTL value that is greater than 15 can be forwarded via that router interface. Table 3.1 summarizes the conventional TTL values that are used to restrict the scope of an IP multicast flow.

TTL-based scoping has a few shortcomings. It cannot resolve conflicts that arise when attempting to enforce simultaneous limits on topology, geography, and bandwidth. In particular, TTL-based scoping results in expanding rings of scope regions and as a consequence cannot handle overlapping scope regions, which is a necessary characteristic of ad-

TTL threshold	Scope
0	Restricted to the same host
1	Restricted to the same subnetwork
15	Restricted to the same site
63	Restricted to the same region
127	Worldwide
191	Worldwide, limited bandwidth
255	Unrestricted in scope

Table 3.1: Multicast TTL scope control values

ministrative domains. A more serious architectural problem concerns the interaction of TTL scoping with broadcast and prune multicast routing protocols such as DVMRP [32, 137]. The particular problem is that TTL scoping can prevent pruning from being effective in many cases. To address these issues, a new mechanism known as administrative scoping [71, 92] was proposed.

Administrative scoping lets us limit a multicast flow to a certain network boundary (e.g., within the organization) by using special administratively scoped addresses. To configure an administrative scope zone, all the multicast routers bordering that zone are configured to be a boundary to that zone for a range of multicast addresses. Administrative scope boundaries are defined to be bi-directional. As opposed to TTL scoping, administrative scoping provides clear and simple semantics for scoped IP multicast. Packets that are addressed to administratively scoped multicast addresses do not cross configured administrative boundaries. Moreover, administratively scoped multicast addresses are assigned locally, and so are not required to be unique across administrative boundaries. The IETF (Internet Engineering Task Force) has designated IP multicast addresses between 239.0.0.0 and 239.255.255.255 as administratively scoped addresses for local use within intranets.

By using administrative scoping, scattercast can effectively control the range over which it uses IP multicast. To cross administrative scope boundaries and extend the broadcast to the wide area, scattercast relies on an overlay of unicast connections.

3.1.3 Opacity of underlying network topology

The scattercast broadcast distribution architecture attempts to construct optimal broadcast paths between the source and the receivers. These paths inherently depend upon the nature of the underlying Internet topology. However, scattercast makes no assumptions about the topology and structure of the underlying Internet. Although embedding support for scattercast in the IP routing infrastructure would allow us to build optimal distribution paths, scattercast maintains a clean separation between the packet routing infrastructure and the broadcast framework that it builds on top of the IP architecture. In doing so it avoids the pitfalls associated with global IP multicast as explained in Section 1.2.

Rather than require the IP layers to expose the physical network topology to the scattercast framework, we assume an opaque network topology, that is, scattercast requires no knowledge of the physical network structure. Instead, scattercast components dynamically attempt to "discover" the topology by using end-to-end measurements between the components without directly involving the underlying routing infrastructure. The only assumption scattercast makes of IP is that it provides a unicast communication service between any two IP end-points and a locally scoped multicast distribution service. Underneath these service models, the IP architecture is free to use any mechanisms and policies it pleases to perform route computation and packet forwarding. In fact, the routing protocols may dynamically adapt the routes associated with a data flow to adjust to changes in network conditions or the physical network topology. Scattercast is impervious to these changes in the underlying topology except for the fact that it may in turn dynamically discover newer and potentially better broadcast paths as the underlying topology changes.

By maintaining the separation between the IP architecture layers and the broadcasting infrastructure components, scattercast may end up with broadcast paths that are worse than the optimal paths that would be possible by embedding branching points within the routing infrastructure. However, we believe that this separation is essential to the successful design and implementation of the scattercast architecture. In particular, scattercast remains isolated from transient changes or updates to the routing topology. Moreover, not only is the topology of the global Internet too large to expose it to scattercast in a scalable manner, but also since Internet routing is typically hierarchical in nature, no single network or routing entity can possibly have detailed knowledge of the entire topology. Finally, this separation of the basic routing and forwarding functionality from higher-level complex systems such as scattercast is in tune with the original design principles of the Internet architecture, which advocated for maintaining the simplicity and robustness of the basic routing infrastructure by pushing all complex functionality to higher layers at the edges of the routing infrastructure.

3.1.4 Infrastructure Service Clusters

Since the IP architecture only provides the very basic delivery semantics to the higher layers, scattercast relies on intelligent computing embedded within the network to provide efficient broadcasting. However, unlike IP multicast, which uses support from routers to build multicast distribution trees, scattercast leverages application-level intelligent via components that are strategically placed across the network, potentially even co-located with IP routers. The scattercast infrastructure components are typically hosted at ISP points-of-presence (POPs). To provide a robust, scalable and available service, the scattercast components are materialized out of clusters of machines as described in Chapter 2. These clusters provide the necessary service properties of scalability, load balancing and fault tolerance. We emphasize, however, that the design of the cluster operating system is orthogonal to the scattercast architecture itself. In this dissertation, we merely assume

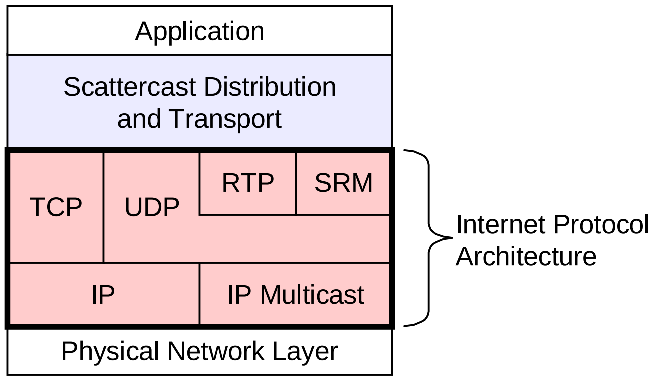

Application
Scattercast Distribution and Transport
TCP
UDP
RTP
SRM
IP
IP Multicast
Physical Network Layer
Internet Protocol Architecture

Figure 3.2: The layered Internet architecture and the composition of scattercast on top of this architecture

that such a cluster platform exists and can dynamically initialize and manage the various scattercast service components. Although internally the cluster may harness many nodes to run the scattercast components based on system load, the cluster operating system effectively allows us to view the scattercast cluster logically as a single large virtual machine.

3.2 An Architecture Overview

We now present an overview of the scattercast architecture and describe how it leverages the underlying Internet architecture for efficient broadcasting. We present the scattercast service model and discuss the advantages and disadvantages of this approach.

3.2.1 Scattercast in a Nutshell

The scattercast architecture employs the traditional unicast and multicast protocols as building blocks towards an efficient broadcasting infrastructure. We build upon the robust and congestion-friendly behavior of wide-area unicast protocols while at the same time relying on local area IP multicast for efficient data delivery without the complexities associated with wide-area network-layer multicast. In keeping with the principles of end-to-end design [120], we make the explicit decision to leave the core network layer technology untouched and migrate the scattercast functionality to components entirely at the application level. We build multi-point delivery as an infrastructure service that leverages well-understood and robust *unicast* transport protocols and couples them with locally scoped IP multicast. Separating multi-point delivery into a higher-level infrastructure service allows us to keep the network layer primitives simple and easy to manage.

Figure 3.2 shows the composition of the scattercast framework on top of the traditional layered IP architecture. Scattercast relies on well-understood unicast protocols such

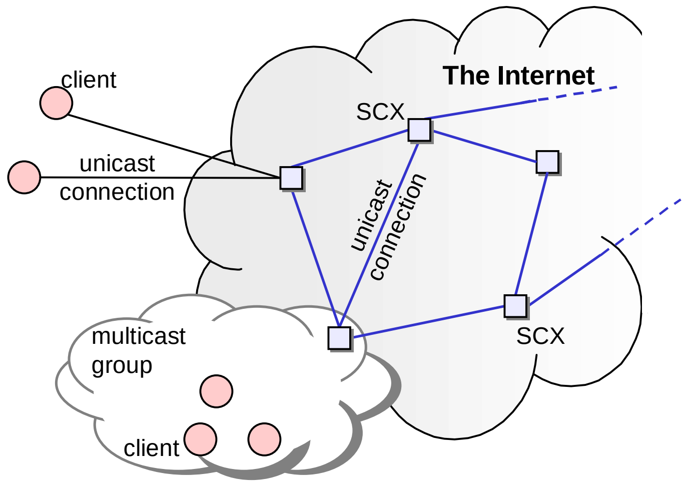


Figure 3.3: The scattercast architecture: Clients communicate with SCXs either via locally scoped multicast groups or via unicast. SCXs form a mesh of unicast interconnections between themselves.

as TCP, UDP, and RTP for communication across the wide area. This allows scattercast to, for example, inherit the reliability and congestion management features of TCP. TCP works well across the wide area because it adaptively accommodates a large range of network rates, delays and losses. TCP employs a feedback loop between the sender and the receiver that detects and reacts to congestion. Scattercast can leverage this behavior of TCP by using it for communication across the wide area. Similarly, a protocol such as RTP can provide scattercast with transport functions suitable for applications transmitting real-time data, such as audio and video. RTP itself is augmented by a control protocol (RTCP) that allows applications to monitor data delivery and provide quality feedback reports from the receivers to sources to allow them to adjust the data stream based upon perceived quality at receivers. In addition to using commonplace unicast protocols for wide-area communication, scattercast harnesses the efficiency of IP multicast in well-defined locally scoped regions. This allows scattercast to leverage the benefits of multicast in terms of efficient network utilization in the local area while at the same avoiding the complexities that arise with wide-area multicast.

This model of communication effectively partitions a scattercast session into a number of independent locally scoped IP multicast regions. We call these regions *data groups*. To extend the broadcast channel across these disjoint data groups, scattercast embeds in the network infrastructure a collection of agents that together provide the scattercast service. Figure 3.3 illustrates the components of this infrastructure architecture. At the core of the architecture is a network of *ScatterCast proXies* or *SCXs* that configure themselves into an overlay structure composed of wide-area unicast interconnections. This overlay network forms the basis for scattercast distribution. Clients (sources or receivers) wishing to participate in a scattercast session communicate with a nearby SCX and tap into the session via that SCX. Clients join the session either by subscribing to a local multicast channel for the session or by initiating a direct unicast connection to the local SCX. Each locally scoped multicast channel represents an independent data group of the session; each data group contains a representative SCX, which is strategically placed in the network to service its data group. The SCXs spread across the various data groups organize themselves into an overlay structure via unicast interconnections and provide efficient data distribution over this application-level network.

Thus, rather than rely on a single global multicast communication channel, scattercast partitions the logical channel into a number of smaller data groups. This divide-and-conquer approach results in a topological clustering of clients into homogeneous clusters thereby reducing the wide-area heterogeneity problem into a simpler problem of rate adaptation within homogeneous clouds and point-to-point rate adaptation across clouds.

The collection of SCXs participating in a session coordinate with each other to build an application-level overlay distribution tree composed on unicast interconnections across SCXs. Sources send packets to their local SCX which in turn forwards the packets to the rest of the session over this application-level distribution tree. Moreover, SCXs embed application-aware semantics into the data forwarding path to allow individual applications to customize the transport protocols to suit their own environments. In effect, the scattercast architecture exposes a flexible transport framework to the applications that can be used to tune the network protocols for application-defined efficiency constraints.

3.2.2 The Service Model

Each scattercast session has an explicit URL-like unique name. The name is used to identify the session and to distinguish between sessions. Session names are of the form `scattercast://creator-identity/session-name`. The creator identity is used to avoid collisions in the session name-space. The simplest form of creator identity is the domain name of the agency that creates the session. For example, a multimedia seminar announcement at UC Berkeley may have the name `scattercast://cs.berkeley.edu/multimedia-seminar/`.

A single scattercast session may consist of multiple independent data flows, each with its own transport requirements. For example, some flows may require reliable data delivery while others may be satisfied with best-effort performance. Applications may also wish to split different media types into separate flows. For example, a real-time Internet broadcast may be composed of a video and an audio flow. Scattercast uses the notion of *data channels* to separate such independent flows. Each data channel has an associated well-known numeric identifier. When sources transmit data packets, they include the channel identifier in the packet header. Scattercast uses this identifier to route the data over the appropriate channel. This notion of multiple channels within the same scattercast session allows us to reuse the same scattercast overlay network for different types of data flows.

Unlike IP multicast, the scattercast service model requires both sources and receivers to explicitly join the session. Moreover, sources must explicitly announce their intent to send data. This allows the underlying scattercast protocols to use this information to build efficient source-rooted data distribution trees. A single session may have multiple independent sources; each source typically results in a separate source-rooted distribution tree.

Each client (source or receiver) attaches to a nearby SCX and interacts with the rest of the session via that SCX. Each SCX, in turn, simultaneously serves many clients. As shown in Figure 3.3, clients communicate with their SCX using multicast if possible, otherwise they revert to unicast connections to the SCX. Across SCXs, data is transmitted using an overlay structure of unicast connections. These data transport connections may be UDP, TCP, or some other unicast transport protocol depending upon the requirements of the channels for that session. The overlay structures are constructed on a per-session basis, that is, the overlay is composed only of those SCXs that actually have clients participating in that particular session.

Typically, SCX lifetimes are limited to those of their clients. An SCX is created on demand for a specific session and it dies when all of its clients leave the session. We note that although clients may join and leave a scattercast session at a rapid rate, since an SCX serves a number of clients, it remains part of the session as long as it has at least one client to serve. We assume that, in general, SCXs join and leave a scattercast session at a relatively slow rate.

Finally, rather than act as dumb relay devices that simply forward data packets across the session, SCXs provide application-level intelligence to the scattercast transport service by embedding application semantics within the transport protocol. SCXs can provide on-the-fly adaptation to customize the level of reliability, congestion management, and data formats and resolution in order to adjust the content generated by the source to the heterogeneous range of client devices and networks.

3.2.3 Advantages of the Scattercast approach

In the following discussion, we present the advantages and disadvantages of the scattercast approach with respect to global IP multicast.

Alleviating network complexity

The divide-and-conquer tactics of scattercast allow us to isolate the use of IP multicast to well-defined locally scoped data groups while leveraging the robustness of well-understood unicast protocols for wide-area communication. By eliminating the use of multicast across the wide-area, in particular within the Internet backbones, we reduce the forwarding state problems associated with backbone multicast routers. Since wide-area data forwarding takes place via application-level proxies (SCXs), the forwarding state is migrated out of the routing infrastructure to the individual SCXs. A second advantage of the absence of wide-area IP multicast is that no complex inter-domain multicast routing protocol is needed. Inter-domain routing instead occurs across SCXs at the application-level. We note, however, that by simply migrating state and protocol complexity out of the network layer to the application layer does not eliminate those problems. Yet, we believe that this migration is crucial to simplifying the network layer since these issues can be more flexibly managed by application-level entities than by hardware in the fast-path of network routers.

Another advantage of scattercast is that it does not share IP multicast's address allocation problems. Since each scattercast session is identified by an explicit and globally unique URL-like name, no complex address/name assignment schemes are necessary to avoid traffic collision in scattercast. Finally, the application-awareness of SCXs enables us to embed application-specific access control policies with the SCXs to restrict access to a scattercast session. In particular, SCXs may control who is allowed to send or receive data within a session via strong authentication and encryption techniques.

Dealing with heterogeneity

Unlike IP multicast, which strains to support a heterogeneous range of receiver devices and networks, scattercast incorporates explicit support for dealing with heterogeneity into its transport framework. SCXs embed application-level semantics into the transport protocol by allowing for on-the-fly adaptation of the data stream to provide application-defined reliability and congestion semantics as well as to transform data dynamically to better suit the needs of the receivers.

Reliable broadcasting

Unlike IP multicast, where researchers have struggled to build efficient and scalable reliable distribution protocols on top of the best effort service model, scattercast simplifies end-to-end reliability by leveraging well-understood unicast protocols as building blocks for constructing an end-to-end reliability protocol. TCP is one example of a reliable unicast protocol that can be used provide hop-by-hop reliability across SCXs. By making each hop along the overlay distribution structure reliable via robust unicast protocols, we improve the reliability of the entire session. We must note however that hop-by-hop reliability does not automatically lead to end-to-end reliability. Packets may be lost due to lack of buffer space within SCXs or due to transient changes in the overlay structure. To overcome this, scattercast builds an end-to-end reliability protocol on top of the piecemeal solution provided by the unicast reliable protocols.

Similarly, the use of protocols such as TCP across the wide-area permits scattercast to leverage the congestion-friendly behavior of those protocols. Scattercast thus exhibits the same congestion tolerance characteristics as the wide-area unicast protocols. Once again, to provide end-to-end flow- and congestion-control, scattercast implements higher-level bandwidth management and congestion backoff policies on top of the basic functionality provided by protocols such as TCP.

TCP is not the only unicast protocol that scattercast can use across SCXs. Depending upon the individual application's requirements, scattercast may trade off reliability for real-time performance by using RTP on top of UDP for communicating across SCXs. In general, inter-SCX communication can trade off various application-specific semantics such as in-order delivery and bit-level reliability while at the same time providing efficient congestion-controlled behavior by using the appropriate unicast communication protocol between SCXs.

3.2.4 Disadvantages of this approach

Small local groups

Scattercast is geared towards large wide-area broadcasting applications. For limited local-area groups, IP multicast is a much simpler and cleaner solution than scattercast. This is especially obvious given that even scattercast relies upon local area IP multicast (if available) within a local domain. Scattercast's primary use is to address the concerns that arise out of wide-area multicasting such as dealing with network complexity and heterogeneity.

Additional points of failure

By introducing a new infrastructure service layer and new service components embedded within the network, scattercast gives rise to additional failure points. Network administrators now not only have to manage the routing infrastructure but also the new SCX components associated with scattercast. However, we believe that by using the principles of clustering and cluster operating systems introduced in Chapter 2, we can alleviate the management and robustness problems associated with the scattercast components. A well-engineered cluster platform can provide scalable, available and robust support for scattercast.

No level of indirection for resource discovery

IP multicast provides a convenient level of indirection for discovering nearby machines running specific services. For example, a TTL-based expanding ring search via multicast is an excellent way to discover services without having to know their exact location ahead of time. The host group abstraction of IP multicast allows clients to communicate with services over an IP multicast group without ever having to know the exact IP location of the service. Scattercast is not well-suited for these forms of applications. Although scattercast does provide an abstraction that can hide the exact location of clients from each other, quickly discovering new services with scattercast is difficult especially since the overlay structure used by scattercast for data distribution only forms and stabilizes over time. Moreover, most of such resource discovery applications are restricted to the local area where scattercast is not nearly as useful as direct IP multicast.

Scalability

Another issue that arises as a result of the migration of the distribution service from the network layer to infrastructure services is that of the ability of the service to withstand the load that may be offered by the various scattercast data streams. Unlike multicast routers that typically make routing and forwarding decisions entirely in hardware, scattercast relies on application-level software entities for the data distribution process. As a result, a single scattercast component clearly cannot handle the same load as a hard ware-level multicast router. However, we believe that the benefits accrued due to this approach far outweigh the scalability concerns of using software components for distribution. Moreover, by using clusters of machines to host the scattercast components, we can leverage the advantages of clusters in terms of incremental scaling by distributing the SCX service across a number of machines within the cluster rather than being hosted entirely by a single machine. In [46], we demonstrated this ability of clusters to grow a service dynamically when the offered load increases, and those results directly apply to the scattercast service as well.

3.2.5 Scattercast Terminology

Before delving into the details of the scattercast architecture, we summarize some of the terminology involved:

SCX (ScatterCast proXy): The core component of the scattercast architecture. A collection of SCXs collaboratively provides the broadcast distribution service.

Source SCX: An SCX with an attached source client. By explicitly identifying source SCXs, scattercast can build efficient overlay paths from the source to the rest of the session.

Overlay: The structure composed of unicast connections between SCXs that scattercast uses to route data packets across the entire session.

Data group: A locally scoped IP multicast group with a representative SCX that scattercast uses for efficient local area broadcasting.

Application-level Multicast: A term used to identify the set of protocols and mechanisms that scattercast uses to build the application-level overlay structure across SCXs and to route data from sources to the rest of the session.

Scattercast Transport: An application-customizable transport layer that scattercast provides on top of the application-level multicasting layer for efficient and adaptive data transport.

The scattercast architecture as described earlier in this section results in a two-tiered communication model. The collection of SCXs serving a scattercast session collaboratively form an overlay distribution network within the infrastructure. In addition, at the edges of this overlay network, clients attach to the session either via direct unicast connections to local SCXs or via locally scoped multicast data groups. In the rest of this chapter, we address some of the issues raised by this model. In particular, we describe how clients discover scattercast sessions, how they locate a nearby SCX, and how they attach to the session via that SCX. Within the infrastructure, SCXs use an application-level multicast protocol to self-organize into an overlay structure. We leave the discussion of this protocol to Chapter 4.

3.3 Discovering Scattercast Sessions: Scattercast Announcements

In the MBone world, IP multicast sessions are advertised via the Session Announcement Protocol (SAP) [59] and the associated session directory application *sdr* [58]. *sdr* clients listen to a well-known multicast group for session announcements which are periodically announced by the creators of the sessions.

In scattercast, session announcements are represented in the form of announcement documents. These announcement documents may be advertised to the world via a special well-known scattercast session in a manner similar to that used for the MBone, but more typically we expect the announcement documents to be published on the world wide web. For example, a broadcast of a seminar from the Computer Science department at Berkeley may be announced via a document posted on the departmental web server.

The announcement documents are composed using the Extended Markup Language (XML) [17]. The document contains all the necessary parameters pertaining to the session. Figure 3.4 shows the format of a scattercast announcement. Each announcement includes the URL-like name of the session, one or more <CHANNEL> sections, and a <RENDEZVOUS> section. As mentioned in Section 3.2.2, the scattercast service model allows for multiple data channels within any session. Each <CHANNEL> section includes the numeric identifier associated with the channel. In addition, each channel may use its own set of network transport protocols for communicating across the unicast connections between SCXs and within the locally scoped multicast data groups. Accordingly, the <CHANNEL> section lists the types of transport protocols that the channel is required to use. Unicast protocol descriptors may be UDP, TCP, RTP or some other unicast transport protocol, while multicast protocol descriptors may be UDP, RTP, or a reliable multicast protocol

```
<SCATTERCAST
        name="scattercast://creator-identity/session-name">
    <DESCRIPTION>
        An optional textual description of the session
        goes here
    </DESCRIPTION>

    <CHANNEL id="numeric-identifier">
        <DESCRIPTION>
            An optional textual description for this channel
        </DESCRIPTION>
        <TRANSPORT unicast="unicast-protocol-name"
                   multicast="multicast-protocol-name"/>
    </CHANNEL>

    <CHANNEL id="numeric-identifier">
        ...
    </CHANNEL>

    <RENDEZVOUS>
        rendezvous-point-location
        ...
    </RENDEZVOUS>
</SCATTERCAST>
```

Figure 3.4: Format of a scattercast announcement document.

such as SRM [43]. The <RENDEZVOUS> section lists one or more *rendezvous points* that SCXs use to find each other and to form the overlay structure for application-level multicasting. We discuss the details of the rendezvous mechanisms and their use for constructing the application-level overlay structure in Chapter 4.

3.4 Joining a Session: Locating an Appropriate SCX

As described in the previous section, announcement documents published on the web allow clients to discover a scattercast session. But before clients can start receiving data from the session efficiently, it is imperative that they attach themselves to an SCX that is close to them. With a potentially large number of SCX-capable cluster platforms spread across the Internet, clients need a way to locate the closest SCX. This problem is similar to that faced today in the context of the world wide web, especially for mobile web clients, where web browsers need to be configured with the location of the best available web-caching proxy. This is a well-studied research problem and solutions in this arena can be applied to the scattercast architecture as well. We discuss some potential solutions:

Static configuration: Clients may be statically configured with the location of their closest cluster platform. This mechanism is simple to implement, but does not permit automatic discovery of new nearby cluster platforms.

Auto configuration: A modification to the static configuration scheme is to use a statically configured DNS name to identify the local cluster platform (e.g. `scattercast.mydomain.edu`), or a script akin to web-proxy auto-configuration scripts [86]. The WPAD (Web Proxy Auto Discovery) Draft [51] describes a number of mechanisms for discovery of network services based on DHCP [39], SLP [135], or DNS queries [57, 54]. These mechanisms do not require the client to know the exact names of the cluster platform machines, but still do not account for dynamic network changes.

Transparent DNS redirection: A more sophisticated approach relies on using special DNS names that are resolved differently for different clients based on the clients' location. This approach is used by the Digital Island's Footprint web caching service [36]. The basic idea is to construct a *redirection framework* that manages a special DNS domain, say `redirect.scattercast.net`, and resolves client queries for that domain into an address for a scattercast cluster that is closest to the client. The redirection framework is composed of an elaborate network of probes that monitor the Internet building a real-time network map that identifies the delays between different parts of the network. Using this map, the redirection framework can easily identify the closest scattercast cluster for any client. Thus the client always manages to find a nearby cluster without any pre-configuration.

Explicit application-level redirection: Using DNS resolution for redirecting clients to appropriate clusters can be plagued by problems due to clients caching stale addresses. Old clusters may no longer be offering the scattercast service, new clusters may have cropped up that are closer to the client, or network conditions might have changed. This can be addressed by using an explicit application-level redirection mechanism such as that used by HTTP.

Our prototype scattercast implementation relies either on static client configuration via a file in /etc on UNIX or a registry entry in Windows, or on dynamic configuration via DHCP where available. These mechanisms are simple to implement, but are not as flexible as the redirection framework approach. The Digital Island Footprint service [36] has implemented a proprietary system that includes a well-designed redirection framework, and a practical deployed scattercast architecture should utilize that work.

3.5 Joining a Session: Client Attachment

The previous section describes mechanisms for clients to locate a nearby scattercast cluster. Once a client has discovered the nearest cluster, it contacts the cluster and makes a request for an SCX. The request includes the session announcement for the session that the client is interested in and an indication of whether the client is a source of data or not. The cluster creates a new SCX if needed and returns the location of the SCX, including a unicast IP address and port number as well as a locally scoped IP multicast group address that can be used if multicast connectivity is available between the client and the SCX. The client initially attempts to communicate with the SCX over the IP multicast group, but reverts to unicast communication if that fails. This allows us to leverage the efficiency of IP multicast in the local domain when it is available.

As long as the client is part of the session, it sends periodic KEEP_ALIVE messages to the SCX. It announces its imminent departure via an AM_LEAVING message. The SCX uses this message (or the loss of KEEP_ALIVE messages) as an indication of the client's death. When all clients of the SCX have left the session, the SCX too leaves the session.

The interface that clients use to communicate with their scattercast cluster and to attach to the appropriate SCX is actually more application-specific than the description above. For example, if an audio broadcasting service wishes to use scattercast to deliver its content to standard audio clients such as WinAmp [103] or RealAudio [118], it must be able to provide an interface that the clients are capable of interacting with. For example, if the client uses HTTP streaming as its download interface, the scattercast cluster should be able to expose such an interface to the client. Such a client would make a request for a session by embedding a pointer to the session announcement within an HTTP GET request to the scattercast cluster. The cluster would in turn return an HTTP redirect to point the client to the appropriate SCX for the session. The client can then directly connect to that SCX to access the audio stream.

3.6 Summary of Scattercast Architecture

In this chapter, we presented an overview of the scattercast architecture, its service model, and the network environment underlying the architecture. We described details of how scattercast clients discover sessions, locate appropriate scattercast proxies, and attach to them to participate in the session.

The scattercast architecture relies on application-level intelligence embedded within the network infrastructure rather than on network layer multicast primitives to provide efficient multi-point data distribution. Our architecture makes use of a collection of intelligent network agents (ScatterCast proXies or SCXs) that collaboratively provide the multicast service for a session. Clients participate in the session via a nearby SCX by either using locally scoped IP multicast groups or direct unicast connections to the local SCX. SCXs organize themselves into an overlay network of unicast interconnections and build data distribution trees on top of the overlay structure.

By migrating the multi-point delivery functionality out of the network layer to a higher infrastructure service layer, scattercast maintains the simplicity of the underlying network model. Although scattercast does not eliminate the state scaling issues or protocol complexity associated with network level IP multicast, it moves the complexity to a higher layer where it can be dealt with more easily. We believe that the benefits that arise out of the scattercast model far outweigh the penalties that we have to pay for performing broadcasting at an application level rather than within the network layer.

In Chapter 4, we will discuss the details of how scattercast performs application-level multicast by building and maintaining an efficient self-configurable overlay network across SCXs. Then, in Chapter 5, we will evaluate how higher level transport services such as adaptive reliability, customizable bandwidth management and end-to-end information distribution can be layered on top of this application-level multicast component. This combination of application-level multicast and a generic customizable transport framework are key to the success of scattercast as a flexible broadcasting architecture.

Chapter 4

Gossamer: A Protocol for Application Level Multicasting

Instead of relying on global multicast capability at the network layer, scattercast performs explicit application level multicasting. The collection of SCXs participating in a session coordinate with each other to build an application-level multicast distribution tree. Sources send packets to their local SCX which in turn forwards the packets to the rest of the session over the application-level distribution tree. The protocol for dynamically building and maintaining this tree structure forms the core of the scattercast architecture. In this chapter, we will introduce the protocol, which we call Gossamer, present a detailed description of the various components involved, and demonstrate its efficacy via a comprehensive set of evaluations.

4.1 Introduction

Our goal with Gossamer is to construct an efficient data distribution tree from the source of data. Unlike IP multicast, Gossamer does not rely on intelligence within routers to build the distribution tree. Instead, it constructs the tree entirely at the application level. Only SCXs are permitted to act as tree nodes and branch points. SCXs communicate with each other via unicast connections and the underlying physical network topology remains opaque to Gossamer. Although Gossamer makes no assumptions about the underlying network topology, it should build distribution trees that map on to the physical topology as efficiently as possible. However, the restriction on Gossamer trees to not rely on optimal branch points within network routers implies that a distribution tree constructed using Gossamer will potentially not be as efficient as one built via native multicast. The following example illustrates this by comparing distribution trees constructed by IP multicast with a typical tree that relies only on application-level multicast. Using this example we identify the key requirements that will be required of an application-level multicast protocol such as Gossamer.

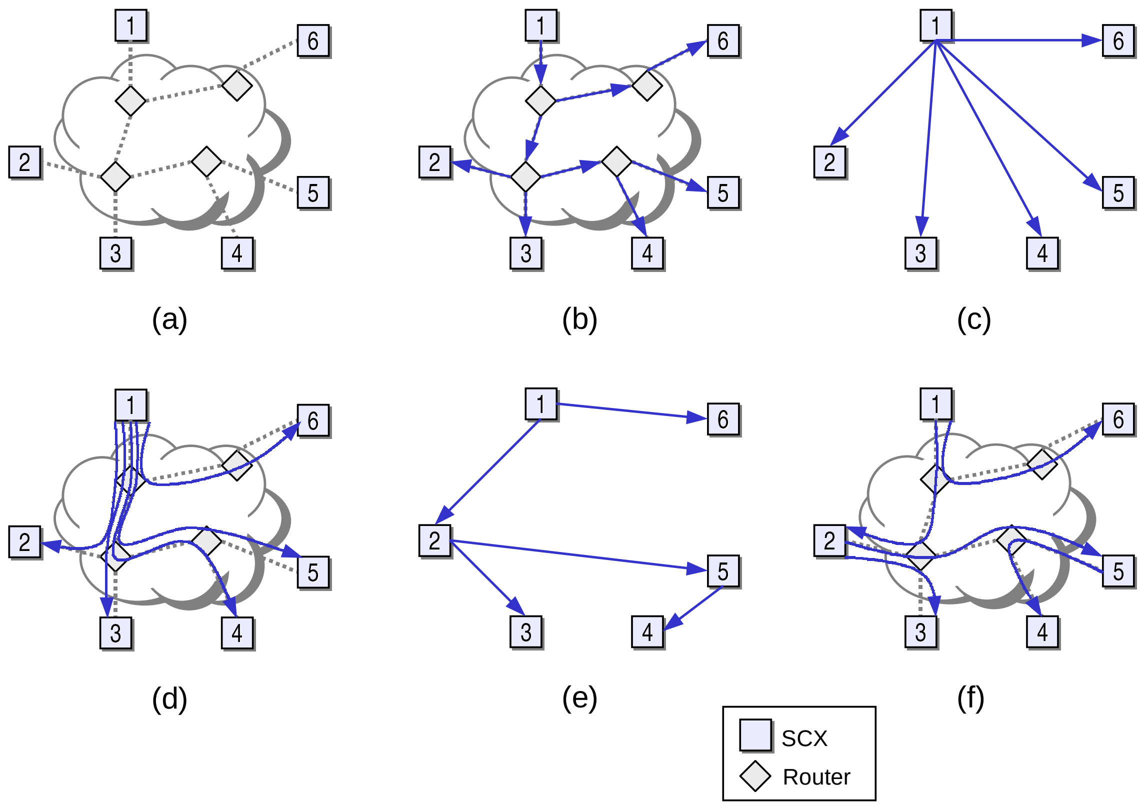


Figure 4.1: Examples of mapping unicast, multicast, and Gossamer routing on to a physical network topology. (a) An example physical network topology consisting of 6 SCXs connected via 4 routers (b) Multicast routing from source SCX 1 using DVMRP-like reverse shortest path distribution tree (c) Naïve unicast distribution from source SCX 1 to all other SCXs (d) An illustration of how unicast distribution maps onto the underlying physical topology (e) A smart distribution tree composed of SCX-SCX unicast connections for Gossamer (f) Illustration of how the smart tree maps onto the underlying network topology.

Figure 4.1 (a) shows an example physical network consisting of six SCXs interlinked via four routers. Consider a scenario in which SCX 1 has a broadcast source attached to it. The IP multicast distribution tree from SCX 1 to the rest of the session constructed using a routing protocol such as DVMRP (Distance Vector Multicast Routing Protocol) [32, 137] is depicted in Figure 4.1 (b). DVMRP is a source-based routing protocol that constructs a reverse-shortest-path multicast distribution tree by computing the shortest (reverse) paths from the source to all possible recipients. In [32], the authors describe the routing and forwarding algorithms used by DVMRP in detail. As shown in Figure 4.1 (b), the DVMRP tree makes most efficient use of the physical network since each physical link carries at most one copy of a packet. Packets are duplicated by the routers along the tree and forwarded along multiple interfaces. Assuming symmetric routing, the path taken by a data packet along the DVMRP tree from the source to any destination is the same as the direct unicast path from the source to that destination.

On the other hand, since Gossamer does not rely on routers to construct a distribution tree, we need to build the tree entirely out of unicast connections between SCXs. A naïve approach for such a tree would be to simply construct a unicast star topology rooted at the source SCX as shown in Figure 4.1 (c). Figure 4.1 (d) shows how this star maps on to the underlying physical network. Although the unicast star structure results in source-destination latencies that are identical to those for the DVMRP tree, we note that this simplistic approach has the dangerous effect of excessive network load near the source. With a star topology, the source SCX simply performs an n-way unicast transmission of the data to all destination SCXs. Since each packet is duplicated multiple times at the source SCX, the bandwidth requirements on the physical Internet links near the source SCX can be excessive.

To limit the amount of duplicate packets traversing any physical link across the network, Gossamer should build smarter distribution trees where the source SCX transmits data only to a handful of nearby SCXs which in turn forward the data towards the rest of the session. In other words, Gossamer distribution trees should restrict the degree of any single SCX node depending upon its bandwidth capabilities. Figure 4.1 (e) shows an example of a smart distribution tree where the maximum degree of any node is 3 (for SCX 2). As expected, the immediate effect of such a degree restriction is that the physical links near the source are no longer overloaded. This is evident from Figure 4.1 (f) which depicts the mapping of the smart tree on to the physical network topology. We note however that such a degree-restricted tree will result in longer delays for certain SCXs than the corresponding delays in the original multicast tree or the unicast star. For example, in Figure 4.1 (f) the delay for SCX 3 is increased because packets to it are now routed via SCX 2 instead of directly via the shortest routing path.

The above example demonstrates that Gossamer has two conflicting goals:

- To provide efficient data transmission by minimizing average latency between the source and all destinations.
- To cause minimal bandwidth overhead, that is, to limit the number of duplicate packets traversing any physical Internet link by restricting the degree of all nodes within the distribution tree.

We thus wish to build a degree-restricted spanning tree across SCXs while at the same time keeping the average delay between the source and all destinations at a minimum. This problem can be defined more formally as follows:

GIVEN: A set of Internet nodes V that represent SCXs participating in a scattercast session, a source SCX $s \in V$, and node degree constraints $k_i \geq 2$ $(\forall v_i \in V)$. We can build an *abstract distance graph* $G = (V, E)$ that is the complete graph over the set of SCX nodes. The cost of edge $\{v_i, v_j\} \in E$ is set to the unicast distance between nodes v_i and v_j (assuming shortest path symmetric Internet routing).

FIND: A *distribution tree* T, which is a spanning tree of the graph G such that δ_i, the degree of node $v_i \in V$ in T is at most k_i, and T's total cost C is the minimum[1] among all possible such trees. Here, the tree cost C is defined as $\sum_{v \in V, v \neq s} d_T(s, v)$, where $d_T(s, v)$ is the path length in T between the source s and the node v.

The problem of constructing minimal degree-constrained spanning trees of graphs is known to be NP-hard [50]. Moreover, the problem remains NP-hard even in the specific case of complete graphs [104]. Hence we need to rely on heuristics to solve the above problem. However, it is difficult to compare the performance of the heuristic approach to the optimal solution, since computing the optimal tree is prohibitively expensive. But, we do know that the cost of the optimal tree is bounded by the cost, C_{mcast}, of the corresponding native multicast tree T_{mcast} rooted at the source s. Analogous to the definition for the Gossamer tree cost, C_{mcast} is defined as $\sum_{v \in V, v \neq s} d_{T_{mcast}}(s, v)$, where $d_{T_{mcast}}(s, v)$ is the path length in the multicast tree between the source s and the node v. By definition, $C \geq C_{mcast}$ and we can use this bound as a metric for evaluating the performance of the heuristic approach.

4.2 A Practical Topology Construction and Management Algorithm

To be practically deployable, any heuristic that we develop must satisfy the following concerns. The protocol should be completely decentralized and distributed across the group of SCXs participating in the session. A centralized "tree construction server" would be an inevitable failure point. In addition, the protocol must be able to cope with a dynamically changing membership of the set V, that is, SCXs may join or leave the session at any time. The protocol should be capable of quickly admitting new SCXs into the overlay structure and re-routing the overlay around failed SCXs or SCXs that decide to leave the session. Note, however, that in general SCX lifetimes are considerably more stable than those of individual end clients. Although clients may join and leave the session at a very high rate, an SCX remains part of the session as long as it has at least one client to serve. A third requirement for Gossamer is that it should incrementally attempt to optimize the overlay by dynamically adding and removing overlay links over time in a manner that results in lowering the overall tree cost C. Finally, the protocol should take into account multiple simultaneous sources rather than being restricted to single-source sessions. Given these requirements, we look at the design of a practical protocol that can address all of them.

[1]Note that minimizing the total cost C is equivalent to minimizing the average delay.

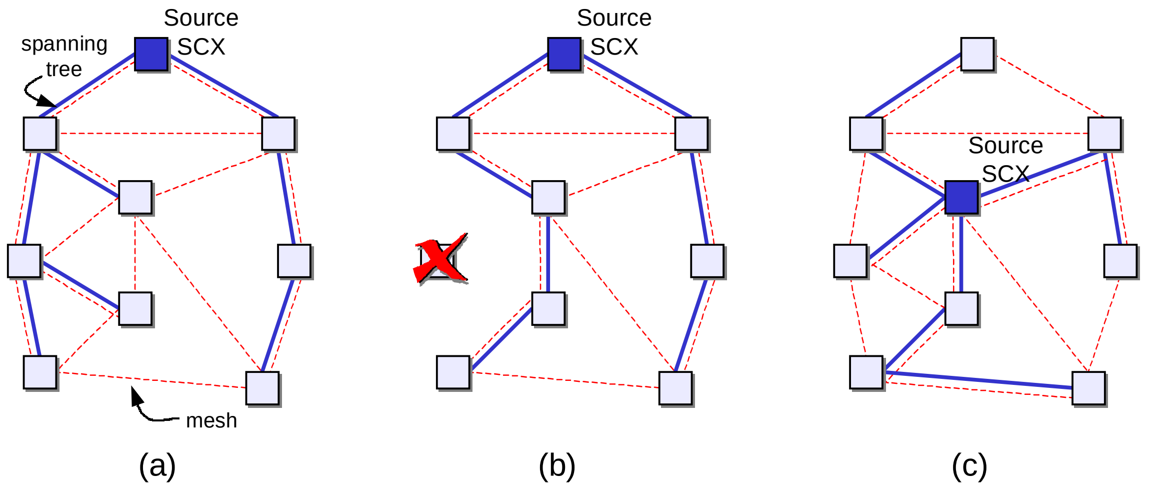


Figure 4.2: A typical Gossamer overlay topology: (a) The topology consists of a mesh structure with a source-rooted spanning tree superimposed on top of the mesh. (b) The mesh+tree structure provides resilience to failures by automatically re-routing the tree around a failed SCX. (c) The mesh can be reused for multiple source-rooted trees without excessive additional computation.

The ultimate goal of Gossamer is to construct spanning trees for data distribution. A possible approach is to construct a tree directly by having each SCX independently pick an appropriate parent [49, 62]. Although a tree-based approach seems like a natural design choice, we believe that a more robust approach would be to construct the distribution tree in two logical steps. First, we build a strongly connected graph structure—the *mesh*—and then superimpose source-rooted trees on top of this structure. Gossamer builds the mesh out of unicast connections across SCXs. On top of this mesh, it runs an application-level routing protocol similar to DVMRP [32, 137] to compute source-rooted reverse shortest path distribution trees. Figure 4.2 (a) shows an example of this two-step mesh+tree approach.

The reasons for this two-step protocol are three-fold. First, the mesh provides redundancy to the scattercast topology, making it more resilient to failures than a fragile tree structure. As shown in Figure 4.2 (b), if an edge or node in the topology fails, the routing algorithm automatically constructs new paths by routing around the failure. Second, routing algorithms have built-in mechanisms to deal with detection and avoidance of loops in the distribution paths. This makes construction of loop-free distribution trees much simpler. Lastly, it is relatively straightforward to reuse the existing mesh for building additional source-rooted trees in the event that the session has many simultaneous sources. Information gathered in the routing step can easily allow us to put together new trees on top of the same mesh structure as depicted in Figure 4.2 (c). This is in contrast to the tree-based approach where the entire protocol would have to be duplicated for any additional

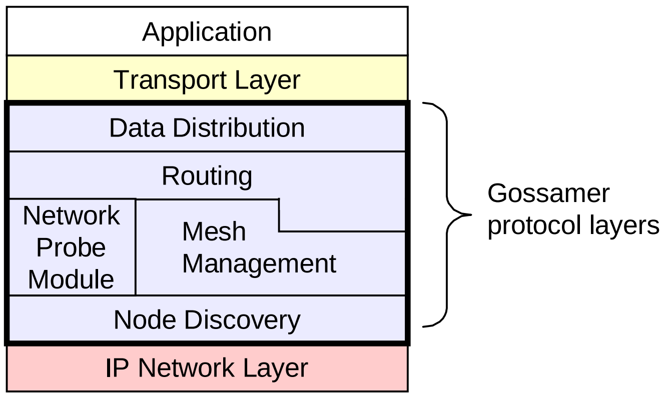


Figure 4.3: Gossamer Protocol Layers.

tree.

As described earlier, Gossamer has two goals: to construct efficient distribution trees with minimal average latency, and to limit bandwidth overhead by restricting the number of neighbors each tree node has. Since our two-step approach automatically builds trees out of routing information on top of the mesh, we enforce our requirements on the underlying mesh itself. To minimize latency, we attempt to insert mesh links in a manner that tries to reduce the routing cost between the sources and the rest of the SCXs. Similarly, since the distribution trees have degree constraints, we impose similar constraints while constructing the mesh. This ensures that the shortest path trees that are built by the routing protocols on top of this mesh automatically satisfy the degree constraints. Each node is assigned a degree constraint based on its bandwidth capacity and the expected bandwidth for the session. For example, a node at the edge of the networks will typically have a lower degree as compared to a node within the Internet backbone.

4.2.1 Protocol Overview

Figure 4.3 shows the different layers involved in the Gossamer protocol. At the bottom is a node discovery layer that SCXs use to dynamically locate other SCXs participating in the session. The mesh management layer deals with constructing and maintaining the basic mesh topology. It relies on the network probe module to conduct periodic network measurements to other SCXs to determine its connectivity to the rest of the session and to arrive at an optimal overlay structure. The routing layer runs a distance vector routing protocol on top of the mesh and provides input to the mesh management layer to assist in optimizing the mesh, and as a consequence, the paths between the sources and the receivers. The data distribution layer constructs distribution trees based on routing information extracted from the routing layer and deals with the forwarding algorithms that are used to disseminate the data. The Gossamer protocol layers effectively provide a basic broadcast distribution service. More complex transport protocols and application-specific computation may be layered on top of the Gossamer layers. We now look at the details of the Gossamer protocol starting with a brief overview of the entire protocol.

Gossamer uses a gossip-style resource discovery protocol to disseminate session membership information. In gossip protocols, a participant periodically forwards new information to other randomly chosen participants thereby propagating information rapidly across the entire system. Such protocols were initially used in the Clearinghouse project [35] to maintain database consistency and have since been used to achieve fault tolerance and detection [106, 134] and consistent data replication [2]. In Gossamer, gossip-style discovery occurs as follows. When an SCX joins a session, it bootstraps itself via a well-known list of rendezvous SCXs, and then relies on gossip-style discovery to locate other SCXs that are part of the session. As the SCX encounters new nodes, it selects some of them to be its neighbors in the mesh. As defined by the degree constraints, each SCX has a target number of neighbors that it attempts to connect to in the mesh. To ensure that nodes can insert edges in the mesh without requiring any explicit coordination across nodes, we split the degree constraint at each node into two: a maximum number of edges (k_1) that the node is allowed to insert from it to other nodes, and a maximum number of edges (k_2) that it is willing to accept from other nodes. As long as we ensure that $k_2 > k_1$, any new SCX node joining the mesh will eventually find some k_1 nodes that have room to accept connections from it. We use the notation $<k_1,k_2>$ to represent these degree constraints, effectively resulting in a total degree constraint of $k_1 + k_2$.

Rather than pick a random set of k_1 neighbors, each node uses a local optimization algorithm to choose neighbors that will result in better distribution trees. The SCX uses the Network Probe Module to periodically execute simple measurement experiments to determine latencies between it and other SCXs. Based on the results of these experiments, the SCX decides whether or not to accept new neighbors into the mesh. Finally, distribution trees are constructed by running a distance vector routing protocol on top of the mesh topology. Each node maintains a routing table with an entry for each source SCX. The data distribution layer uses this routing information to construct source-rooted reverse shortest path distribution trees. Thus, Gossamer produces a separate source-rooted tree for each source SCX within the session.

Although the protocol consists of three distinct stages—node discovery, mesh construction, and tree building—we note that all three stages actually occur in parallel. When an SCX joins a session, it immediately discovers a small subset of mesh members, attaches itself to the existing mesh, and starts receiving data from the session. Although this initial configuration may not be optimal, as we will see in the following sections the SCX gradually discovers all the other session members and over time figures out its optimal position in the mesh and the distribution tree.

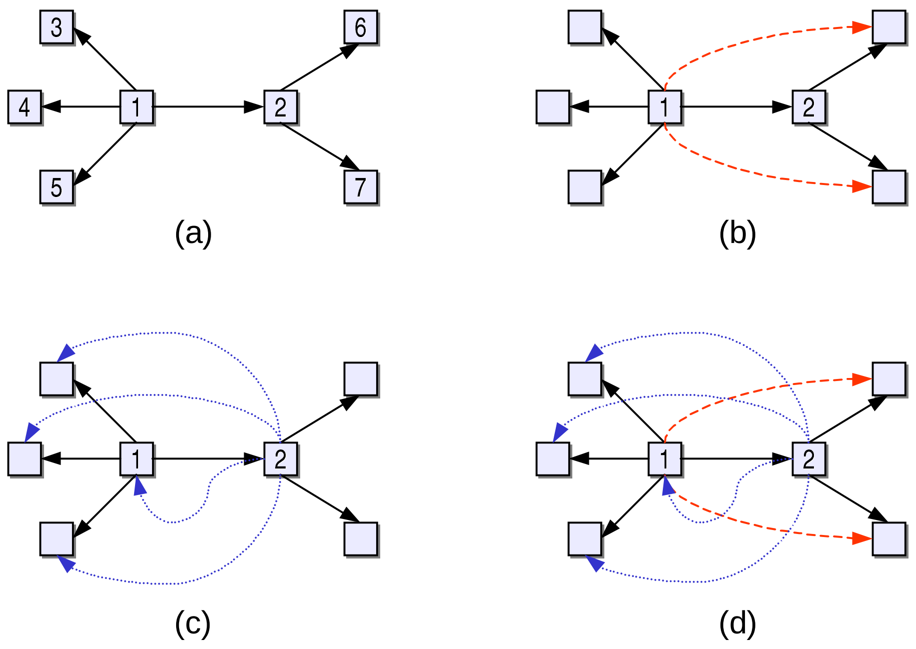


Figure 4.4: Gossamer's node discovery protocol: (a) Illustrates an example discovery graph where SCX 1 knows of nodes 2, 3, 4 and 5, while SCX 2 is aware of nodes 6 and 7. For the next discovery round, SCX 1 chooses a random other node, say SCX 2. (b) With Random Pointer Jump, SCX 1 retrieves SCX 2's membership information and as a result SCX 1 inserts edges to nodes 6 and 7 in the discovery graph. The new edges are illustrated by dashed lines. (c) With Name Dropper, SCX 1 transmits its own membership information to SCX 2, and SCX 2 adds the corresponding edges to the discovery graph including an edge back to SCX 1. (d) With Name Dropper++, both SCX 1 and SCX 2 exchange membership information and add the corresponding edges to the discovery graph.

4.2.2 Node Discovery

An SCX joins a scattercast session on behalf of its clients. To attach itself to the overlay mesh, the SCX must first locate other SCXs that are part of the session. A naïve mechanism would be to rely on a centralized directory of session participants. However, this centralized approach is undesirable because it can become a potential bottleneck. Instead, we rely on a distributed gossip-style protocol to find other SCXs participating in the session. The basic underlying principle in a gossip protocol is that nodes start off with some limited initial knowledge of the system and gradually discover more information by periodically exchanging messages with other known nodes.

We can model the node discovery problem as a directed graph. Each SCX in the session is represented by a node of the graph. An edge between nodes X_i and X_j in the graph represents the fact that X_i knows of the existence of X_j. Over time, as SCXs discover each other, more and more edges are inserted into this discovery graph. The goal of the discovery algorithm should be to build a complete graph as soon as possible. Figure 4.4 (a) shows an example of the discovery graph where SCX 1 knows of nodes 2, 3, 4 and 5, and SCX 2 knows of nodes 6 and 7.

When an SCX joins a session, it initiates the gossip process by discovering a small set of mesh members through some startup rendezvous mechanism. Let us denote by $\Gamma(X_i)$ the set of other SCXs that an SCX X_i knows of. We note that it is not required for SCXs to have complete and accurate information of mesh membership at all times. In fact, they gradually build their information (the set $\Gamma(X_i)$) by "gossiping" with each other. Our only assumption is that the entire group of nodes be at least weakly connected, that is, if we ignore edge directions, then the discovery graph is connected.

Periodically, each node independently performs a discovery round. During each discovery round, a node (say X_i) picks a random node $X_j \in \Gamma(X_i)$ and exchanges discovery messages with it. We look at two different algorithms for discovery before presenting our solution which is a hybrid of these two.

4.2.2.1 Random Pointer Jump

In this algorithm, during each discovery round, X_i contacts a random $X_j \in \Gamma(X_i)$ and asks it for its membership set. X_j transmits $\Gamma(X_j)$ back to X_i which then merges $\Gamma(X_i)$ with $\Gamma(X_j)$, that is, it sets $\Gamma(X_i) = \Gamma(X_i) \cup \Gamma(X_j)$. Figure 4.4 (b) depicts an example of the Random Pointer Jump operation. Node 1 picks SCX 2 for the discovery round, SCX 2 transmits its membership information to SCX 1, as a result of which SCX 1 too points to those nodes. The new edges are represented in the figure with dotted lines.

Conceptually, the algorithm is fairly simple: a node contacts another node and attempts to discover members that the other node knows of. However, it turns out that this algorithm is not actually a good choice for two reasons. First, it requires that the discovery graph be strongly connected, that is, there should exist a directed path between every pair of nodes for the graph to converge to a complete graph. A simple example illustrates the problem: consider a discovery graph with two nodes and a single edge from one node to the other. The second node can never discover the first one. Secondly, for many example configurations, Random Pointer Jump requires too many rounds ($\Omega(n)$) to converge to a complete graph [61].

4.2.2.2 Name Dropper

This algorithm proposed by Harchol, et al. [61] overcomes the limitations of the Random Pointer Jump algorithm. This algorithm works as follows. During each discovery round, a node X_i contacts a random node $X_j \in \Gamma(X_i)$ and transmits $\Gamma(X_i)$ to X_j. X_j merges this information with its own membership set and at the same time learns of the existence of X_i: $\Gamma(X_j) = \Gamma(X_j) \cup \Gamma(X_i) \cup X_i$. The Name Dropper operation is depicted in Figure 4.4 (c) where SCX 1 randomly picks SCX 2, transmits all of its membership information to SCX 2, and causes SCX 2 to add edges to these nodes. In addition, SCX 2 learns of SCX 1. The Name Dropper algorithm converges to a complete discovery graph with high probability in $O(log^2 n)$ rounds [61].

Although Name Dropper improves the worst case convergence time as compared to Random Pointer Jump, it can potentially take a new SCX a while before it can locate other members. We note that X_j updates its membership set $\Gamma(X_j)$ only when another node randomly picks X_j and transmits its membership set to X_j. As a result, when a new SCX joins the session, it cannot discover nearby SCXs and attach itself to them in the mesh structure until other SCXs have contacted it and updated its membership set. Secondly, as the session size increases, the communication overhead for Name Dropper (as well as for Random Pointer Jump) increases rapidly. This is particularly important given that session membership changes dynamically—new SCXs may join the session at any time. As a result, the Name Dropper algorithm needs to be continuously executed and cannot be halted after the $O(log^2 n)$ rounds expected for completion with a static membership set. To address these issues, we rely on a modified version of the Name Dropper algorithm.

4.2.2.3 Name Dropper++

Our discovery algorithm differs from the original Name Dropper proposal in two ways. Name Dropper transmits membership information in only one direction ($\Gamma(X_i)$ sent to X_j) during a round. We instead incorporate an exchange in both directions (X_i transmits $\Gamma(X_i)$ to X_j and vice versa) thereby allowing a newly joining X_i to quickly discover a number of other SCXs. This, however, comes at the cost of increased communication cost. The second difference is that, to minimize communication overhead, we limit the size of the lists exchanged at each round. These limits may be predetermined for each session, or may be dynamically changed based on current estimates of session size. This mechanism to adapt the overhead associated with the discovery process is similar to approaches to limit control traffic for the Real-time Transport Protocol (RTP/RTCP) [125] and the Session Announcement Protocol (SAP) [59]. The modified algorithm, Name Dropper++, is described below:

An SCX (say X_i) initiates Name Dropper++ by discovering a small set of mesh members through a rendezvous mechanism. Periodically, X_i performs a discovery round. During this round, it picks a random node $X_j \in \Gamma(X_i)$ and sends a DISCOVERY message to it. This message includes a *bounded random list* $\gamma(X_i) \subseteq \Gamma(X_i)$. When X_j receives the message, it merges this list into its own set $\Gamma(X_j)$ of known nodes. In addition, it returns a DISCOVERY_RESPONSE message that includes its own list $\gamma(X_j) \subseteq \Gamma(X_j)$. X_i in turn merges this list into its own set $\Gamma(X_i)$, and thus gradually learns of all or most of the other nodes in the system. Figure 4.4 (d) shows the results of one round of Name Dropper++. SCX 1 picks SCX 2, they exchange their membership sets, and add the corresponding edges to the discovery graph.

Although our modifications to Name Dropper effectively limit the control overhead associated with the discovery process, it increases the amount of time that SCXs require to discover everyone else in the session. We have not analyzed the effects of these modifications on the performance of Name Dropper++. However, in practice, our simulation results described in Section 4.3.2 demonstrate the practical usability of the algorithm.

4.2.2.4 Rendezvous Points

To bootstrap Name Dropper++, an SCX needs to first locate some initial group of other SCXs to start the discovery process. We rely on explicit rendezvous points as the bootstrapping mechanism. Each scattercast session has associated with it one or more rendezvous SCXs that are advertised in the session announcement (see Figure 3.4). For example, a video distribution server may configure itself as a rendezvous SCX for its session. All other session participants contact the rendezvous point in order to participate in the session. When a new SCX joins the session, it initializes its mesh membership set $\Gamma(X)$ to the list of rendezvous points for the session, and then proceeds to use Name Dropper++ to eventually discover all the other nodes in the mesh. In addition to bootstrapping the discovery process, the well-known rendezvous points provide a mechanism to detect and repair partitions that may occur in the mesh structure. We discuss this mechanism in Section 4.2.3.3.

To ensure high availability of the rendezvous points, we replicate them across the entire session. The redundancy introduced by multiple rendezvous points ensures that new SCXs can join the mesh even if one of the rendezvous points fails. We note, however, that even if all rendezvous points in the session fail, existing mesh members can continue to operate. The only functionality that is lost is the ability for new SCXs to join the session and the ability to repair potential mesh partitions until the rendezvous points come back up.

4.2.3 Mesh Construction

When a new SCX joins the session, it gradually encounters new nodes via the node discovery protocol. It selects some of these nodes to be its neighbors in the mesh. As described in Section 4.2.1, each SCX has an associated target node degree of $<k_1,k_2>$, where k_1 is the maximum number of edges that the SCX is allowed to insert from it to other SCXs and k_2 refers to the maximum number of edges that it is willing to accept from other SCXs. Thus each SCX gradually attempts to attach itself to the mesh by setting up connections to up to k_1 other nodes. To avoid explicit coordination across nodes, if an SCX X_i attempts to insert a connection to another SCX X_j, we require that X_j accept the connection as long as it has not reached its limit k_2 of the maximum number of edges that it is willing to accept.

Ideally, we would like the mesh constructed by Gossamer to result in optimal dis-

```
optimize($X_j$) {
        Let $\Lambda$ = (set of neighbors of $X_i$) $\cup$ $X_j$
        For each $X \in \Lambda$ {
                Let C.F.[$X$] = compute_cost_function($X$)
        }
        Let $Y (\in \Lambda)$ = node with maximum C.F.

        Let $H$ = hysteresis value
        If ($Y == X_j$) then
                reject $X_j$
        Else if (C.F.[$Y$] $-$ C.F.[$X_j$] $> H$) then {
              accept $X_j$
              reject $Y$
        }
        Else reject $X_j$
}
```

Figure 4.5: Algorithm used by X_i to determine whether to accept X_j as a neighbor.

tribution trees. However, the initial edges that a new SCX inserts into the mesh will be sub-optimal. This is because when the SCX joins the session, it does not have complete membership information and may not know of nodes that would provide better distribution paths. Moreover, since SCXs may join and leave the session over time, the mesh should adapt to the changes in session membership. We therefore allow for incremental optimization of the mesh structure.

4.2.3.1 Mesh Optimization

In our discussion of the algorithms that Gossamer employs to optimize the mesh over time, we assume that the routing layer on top of the mesh executes a limited form of distance vector routing [63], where the routing table contains only a small number of entries: one for each source SCX. Since the eventual goal of Gossamer is to build efficient data distribution trees, the optimization algorithm attempts to add edges that will result in an effective improvement of the routes towards the sources of data and remove edges that are not as useful.

SCXs periodically probe other mesh nodes to evaluate the usefulness of adding new edges. A node X_i probes another node X_j using a REQUEST_STATUS message. The STATUS response from X_j contains a copy of the current routing table of X_j and a CAN_ACCEPT flag that indicates whether X_j has room to accept a connection from X_i. The state of this flag depends upon whether X_j has reached its limit, k_2, of connections it is willing to ac-

```
compute_cost_function($X$) {
        Let C.F.[$X$] = 0.0
        For each source $S$ in $X_i$'s routing table {
                Let C.F.[$X$] += normalized cost* of routing to $S$ via $X$
        }

        If ($X_i$'s routing table is empty) then {
                Let C.F.[$X$] = normalized cost* of the edge between $X_i$ and $X$
        }
        Return C.F.[$X$]
}
```

***Note:** Normalized routing cost is defined as the cost of the route to S via X divided by the maximum of the corresponding such costs for all $X' \in \Lambda$ (see Figure 4.5 for definition of Λ). Similarly the normalized edge cost is defined as the ratio of the cost of the edge between X_i and X to the maximum of the corresponding edge costs for all $X' \in \Lambda$. We note that the normalized cost is always a value between 0.0 and 1.0.

Figure 4.6: Algorithm used by X_i to compute the cost function for node X.

cept from other nodes. X_i uses this status information to evaluate the usefulness of X_j as a neighbor over its current set of neighbors.

If X_i has not yet filled its limit k_1 of edges it is allowed to add, it will accept X_j as a neighbor. But, if X_i already has k_1 neighbors, then to accept X_j, it will have to remove one of its existing neighbors. X_i runs an optimization algorithm that evaluates the "cost" of all of its neighbors and X_j. To realize efficient data distribution trees, the mesh needs to be optimized for efficient routes between receivers and source SCXs. The cost function takes this into account and computes the cost of routing to the various sources via the individual neighbors. Figure 4.6 shows the cost function used as input to the optimization algorithm which itself is described in Figure 4.5. The SCX will accept X_j as a neighbor only if X_j's cost function is less than that of one of its existing neighbors by at least H. H is a hysteresis value that allows us to trade off the stability of the mesh versus the level of optimization. A higher value of H will result in fewer changes to the mesh structure, but may result in a less efficient mesh. After preliminary experiments, we have set the hysteresis value to 0.15 times the number of known source SCXs.

4.2.3.2 Mesh Updates: SCX Leaves and Failures

When an SCX leaves the session, it floods a time-stamped notification to the rest of the mesh. This notifies the SCX's neighbors in the mesh of its impending departure and allows them to reconfigure their edges to other existing SCXs. It also allows the remaining SCXs to remove the departing SCX from their membership set $\Gamma(X)$. The departing SCX sends the notification to its immediate mesh neighbors which in turn propagate it to the rest of the session. Since the mesh is not loop-free, SCXs use the time-stamp in the notification to detect duplicate copies of the notification and stop forwarding the copies. In addition to leaving a session explicitly, an SCX may fail without any warning. In such a situation, its neighbors in the mesh will detect the failure and notify the rest of the session. To detect neighbor failure, neighboring nodes in the mesh exchange periodic KEEP_ALIVE messages. Loss of these messages is an indication of failure. Upon receiving an SCX leave/failure notification, other SCXs mark that SCX as dead in their membership list, and trigger updates in the routing layer.

4.2.3.3 Mesh Partition

It is possible that the death of an SCX causes the mesh to be partitioned. Although such an occurrence will be rare, it must be dealt with and the mesh repaired. To detect mesh partitions, we rely on a periodic HEARTBEAT that is generated by one of the rendezvous points and propagated over the mesh. The rendezvous points run a simple distributed election algorithm and pick one of themselves as the heartbeat generator. As long as every SCX in the session continues to receive this heartbeat, the entire mesh is connected. Loss of heartbeat messages indicate a potential mesh partition, and the SCX that detects the loss attempts to heal the partition by re-contacting the heartbeat generator. It is possible that a large number of SCXs that are partitioned from the heartbeat generator detect the partition at the same time. To prevent all of them from contacting the heartbeat generator simultaneously, we use a randomized damping interval before the SCX attempts to heal the partition. In the event that the heartbeat generator itself has died, the remaining rendezvous points elect a new heartbeat generator and the healing process continues.

4.2.4 Routing Layer

The mesh management layer constructs the basic overlay structure on top of which sources distribute their data. To build data distribution trees, Gossamer relies on the routing layer to run a variant of a distance vector routing protocol [63]. Routing is performed at the application level on the dynamically constructed mesh topology as opposed to standard network routing protocols which run on the physical network topology. The distance metric used by Gossamer for routing is the unicast distance between SCXs.

We describe our application level routing protocol below. Standard distance vector routing algorithms maintain routing tables at each node with an entry for every other node in the routing network. Gossamer routing, however, builds restricted routing tables. Only data sources in the Gossamer topology actively advertise route information. Sources announce their intent to send data to their SCX. The source SCX immediately starts advertising a zero-length route for itself to the rest of the session. All other SCXs simply collect this route information passively. Thus each SCX maintains a routing table that contains one entry per source SCX. To detect routing loops and avoid the counting-to-infinity problem [26], each routing table entry contains not only the routing cost to the source SCX, but also the complete path associated with that cost. The routing cost for a path is defined as the length of the path or the sum of unicast distances along the path.

Periodically, each SCX sends a routing update to every mesh neighbor. The update is a set of messages that contain all of the information from the routing table. It contains an entry for each source SCX that includes the routing cost to that source SCX and the associated path over the mesh. When an SCX X_j receives a routing update from a neighbor X_i, X_j examines the set of source SCXs listed and the path length to each. If X_i knows of a shorter route to a source SCX, or if X_i has an entry that X_j does not have in its table, or if X_j currently uses X_i as the next hop to a source SCX and X_i's path length to that source has changed, X_j updates its table entry. Before selecting a better route based on the routing update, X_j first checks the path that X_i reported towards the source to ensure that X_j is not already on the path, thus avoiding routing loops. This approach is based on a similar mechanism used by the Border Gateway Protocol (BGP) [85] to prevent routing loops across IP domains.

4.2.4.1 Route measurements: The Network Probe Module

The routing protocol relies on unicast distances between nodes as the metric for the routing protocol. Each SCX in the mesh runs a simple *ping* experiment to determine its distance to its mesh neighbors. A ping experiment consists of a small sequence of time-stamped packets that the node sends to its neighbor. When a neighbor receives the ping packets, it simply reflects them back to the sender. The sender uses the average time difference between sending the packets and receiving the responses to compute the round-trip times and thus the one-way distances.

SCXs use shared learning when determining unicast distances. As mentioned in Chapter 2, an SCX runs as an infrastructure service that is hosted on a cluster of work-stations. The same cluster may host multiple SCXs simultaneously for different sessions. Rather than repeat the ping experiments across all SCXs within a cluster, we isolate the measurement process within the Network Probe Module. The probe module is shared across all SCXs within a cluster. It is responsible for actually executing the ping experiments and maintaining a soft state cache of previously made measurements. Since machines within a cluster are connected via a low latency high speed network or more typically are located on the same subnet with a single router, we assume that a single network probe module per cluster is sufficient to measure unicast distances to and from the cluster.

4.2.5 Data Distribution

Gossamer uses the routing tables generated by the routing layer to construct source-rooted reverse shortest path data distribution trees. The per-source trees rely on the (reverse) shortest path routes between all destination SCXs and the source SCXs. This form of distribution trees is similar to the Distance Vector Multicast Routing Protocol (DVMRP) [32, 137] used by IP multicast networks. The distribution trees are built out of an independent set of transport connections that are separate from the control connections used by the mesh construction and routing protocols. As described in Chapter 3, a session potentially consists of many channels. The session announcement specifies the form of transport connections that each channel requires. Gossamer constructs a separate tree for each of the channels. The transport protocols used for the channels depend upon the information specified in the session announcement.

Data forwarding occurs as follows. Each data packet consists of a Gossamer header that identifies the source of data, the source SCX, the numeric channel identifier, and the length of the payload. At each SCX X_i, when a packet is received on a channel from a neighbor X_j, it is forwarded only if X_j is the next hop in X_i's route towards the source SCX. Every packet that passes this reverse-path check is forwarded to all those neighbors that use X_i as their next hop for routing towards the source SCX. To ensure that packets do not loop forever within the scattercast network due to misconfigured SCXs or transient loops in the distribution tree, the packets are stamped with a Time-To-Live (TTL) field that gets decremented by each SCX before the packet is forwarded. If the TTL reaches 0, the packet is dropped.

Transient changes in the distribution tree due to routing updates or mesh changes may result in temporary disruption of data flow. To minimize any data loss during a route change, data continues to be forwarded along the old route for a short while until the downstream SCX establishes the new tree connections and starts receiving data along the new route.

4.3 Evaluation

In this section, we evaluate the behavior of our architecture with respect to the mesh structure and the distribution trees that Gossamer produces. We rely on simulation experiments to evaluate the operation of our protocol. The main metric that we use for evaluation is the total cost C of routing from a source over the Gossamer distribution network. As described in Section 4.1, C is defined as the sum of path lengths in the distribution tree between the source SCX and all other SCXs. The length of each path in the tree is the sum of the unicast distances between the pairs of nodes that make up the path. This can be expressed as:

$$C = \sum_{v \in V, v \neq s} d_T(s, v)$$

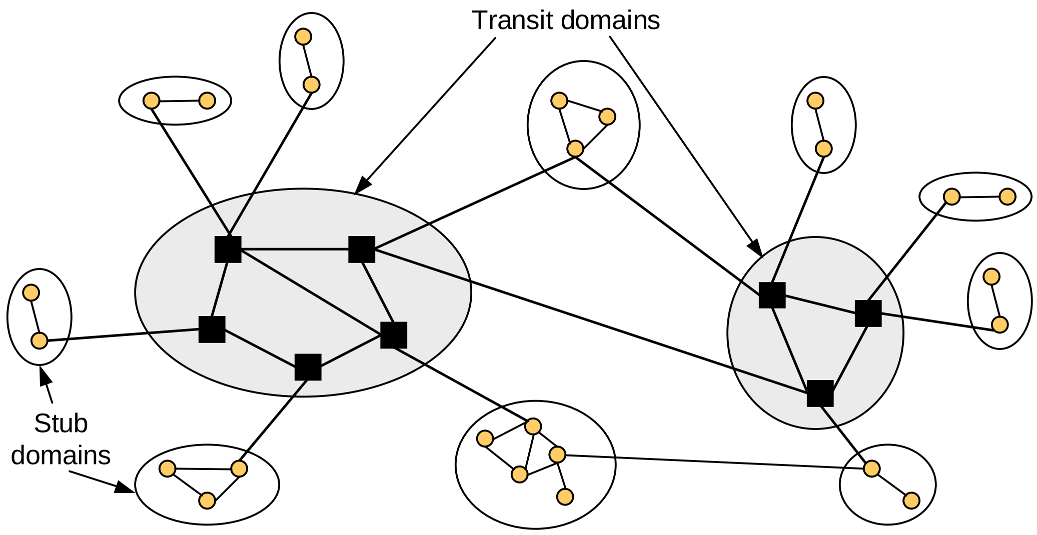


Figure 4.7: The Transit-Stub Model used by the Georgia Tech Topology Generator for constructing simulated internet topologies. This model builds topologies consisting of transit domains and stub domains analogous to the current Internet's backbone transit networks and edge networks.

where s represents the source SCX, and $d_T(s, v)$ is the path length in T between the source s and the SCX node v. We compare this cost to the cost C_{mcast} of routing over the corresponding native multicast tree T_{mcast} rooted at the source s. We recall the definition of T_{mcast} as $\sum\limits_{v \in V, v \neq s} d_{T_{mcast}}(s, v)$, where $d_{T_{mcast}}(s, v)$ is the path length in the multicast tree between the source s and the node v. We note that, in terms of path lengths, the cost of a unicast star topology (a tree comprised of direct unicast connections between the source SCX and each of the other SCXs) is the same as the cost C_{mcast} incurred for the source-rooted multicast routing tree (assuming shortest-path and symmetric internet routing). On the other hand, a degree-restricted tree constructed by Gossamer will have a cost that is worse than that for the multicast tree or the unicast star.

We use the *cost ratio* $\frac{C}{C_{mcast}}$ as our performance metric. The cost ratio represents how close the latencies using the Gossamer distribution tree are to the native multicast routing latencies. For example, a cost ratio of 1.7 means that the average latency using Gossamer is 1.7 times worse than multicast. Our evaluations will demonstrate that typical latencies with Gossamer are within a factor of 2 of multicast latencies.

4.3.1 Simulation Setup

We implemented Gossamer in an experimental protocol simulator. The simulator implements the Name Dropper++ discovery algorithm, the mesh construction and optimization algorithms, and a distance vector routing protocol for building data distribution trees. The input to the simulator is an internet topology generated using the Georgia Tech Topology Generator [146]. We used the Transit-Stub model to generate our experimental topologies.

The Transit-Stub model provides a good approximation of today's Internet topology. The Internet can be viewed as a collection of independent routing domains interconnected by a number of backbone networks. An important characteristic of the domains is that paths between any two nodes within the same domain are entirely contained within the domain. Moreover we can classify domains as either *stub* or *transit* domains based on the kind of traffic that they are willing to carry. If we define local traffic as traffic that originates at or terminates on nodes within a domain, and transit traffic as traffic that passes through a domain, then we can correspondingly define stub domains as those that allow only local traffic, that is, any route that goes through that domain either originates or terminates in that domain. Transit domains on the other hand do not have this restriction: they allow transit traffic through. A transit domain typically represents a collection of backbone Internet nodes that are very well connected to each other. Stub domains connect to the transit domains and to other stub domains via one or more gateway nodes. In keeping with these characteristics of the current Internet topology, the Transit-Stub model within the Georgia Tech Topology Generator produces topologies that are composed of interconnected transit and stub domains. Figure 4.7 shows an example topology constructed by this model.

We constructed 50 different topologies using the Transit-Stub model. Each topology consisted of 1000 nodes and approximately 4200 edges. We used these topologies as input to the simulator. The simulator assumes shortest path internet routing and accordingly computes unicast distances between nodes in the topology. Some of these nodes are selected at random as SCX nodes and the Gossamer protocol is run across these nodes. The simulator does not take into account the effects of any cross traffic and queueing delays on the behavior of the protocol. In the next few sections, we present the results of our experiments to evaluate the performance of Gossamer in a range of environments. In each of our experiments, there is a well known rendezvous SCX. All remaining SCXs join the session at a random instant within the first five seconds of the experiment. Unless mentioned otherwise, each session consists of a hundred SCXs and one randomly chosen source SCX, and each SCX has a node degree constraint of <3,4>[2]. SCXs execute the Gossamer algorithms every 5 seconds. The routing layer performs routing updates every 30 seconds. The experiment ends when there are no changes to the mesh structure for at least 100 seconds.

Our experiments evaluate the performance of Gossamer with respect to its ability to discover other session members rapidly and to form stable efficient overlay meshes. We show how various factors such as session size and node degrees affect the performance of Gossamer.

[2]The degree constraint is composed of two components, k_1 and k_2 as described in Section 4.2.1.

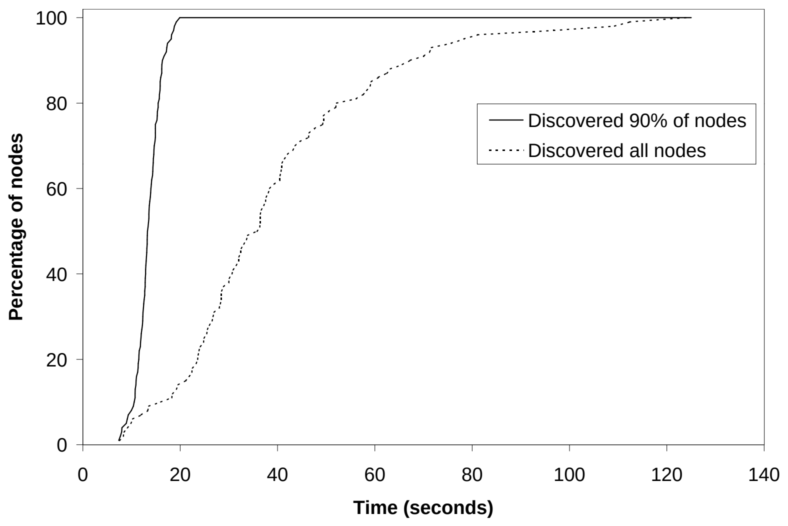


Figure 4.8: Percentage of nodes that have discovered (a) at least 90% of the other nodes, and (b) all of the other nodes vs time.

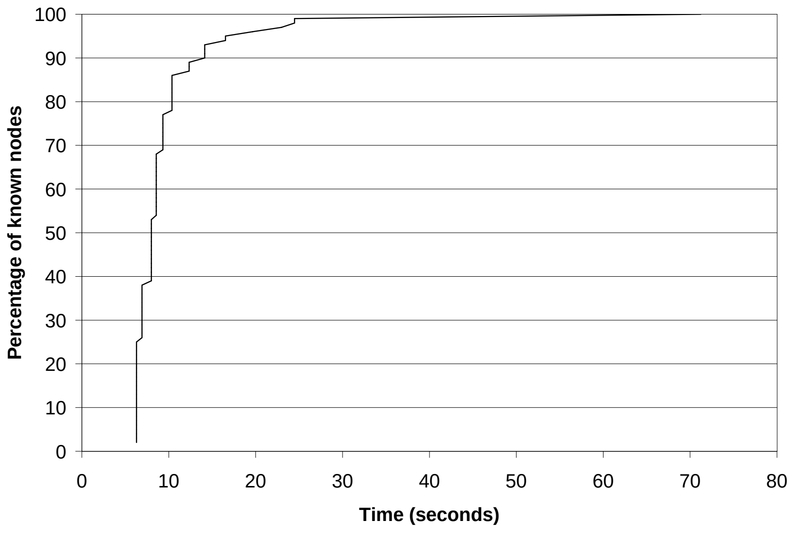


Figure 4.9: Percentage of nodes discovered by a single SCX vs time.

4.3.2 Name Dropper++ Performance

Figures 4.8 and 4.9 depict the performance of the Name Dropper++ node discovery algorithm during an experiment consisting of 100 SCXs. In our experiment, we restricted

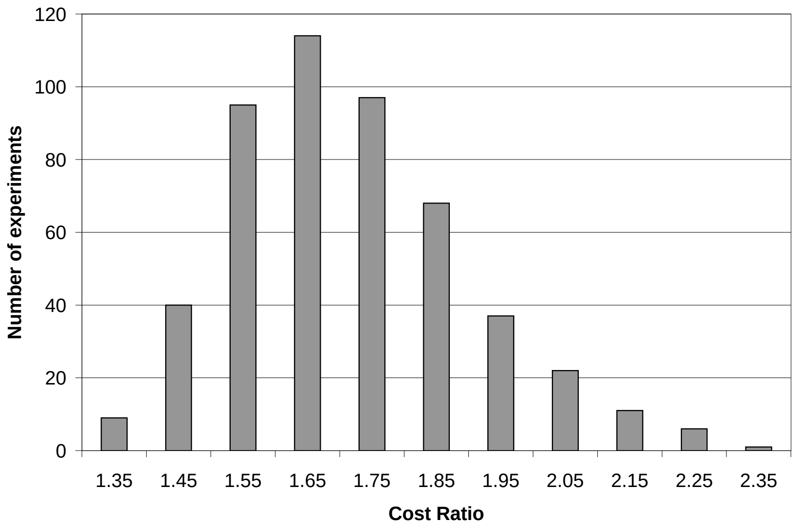


Figure 4.10: Distribution of cost ratio ($\frac{C}{C_{mcast}}$) over a range of experiments.

the size of the membership set $\gamma(X)$ exchanged during each DISCOVERY round to 30. From Figure 4.8, we see that all nodes discover at least 90% of the rest of the nodes within the first 20 seconds of the experiment. Over 85% of the nodes discover everyone else within the first minute of the experiment. The rest of the nodes discover everyone else within the first 2 minutes. Figure 4.9 shows the behavior of the Name Dropper++ algorithm for a single randomly chosen SCX in the experiment. The SCX rapidly discovers most of the other nodes in the session. The rate of discovery tapers off for the last 5% of the nodes. This indicates that the session participants quickly discover each other and can start forming a mesh structure. We will see how the discovery time for Name Dropper++ scales with increasing session size in Section 4.3.4. Let us now see how the mesh construction algorithms behave.

4.3.3 Mesh Construction

This section presents results to prove the effectiveness of Gossamer as a smart overlay construction protocol. The results highlight various properties of Gossamer such as the stability of the mesh structure, the variation of cost ratio across independent runs of the protocol, and the level of packet duplication experienced using Gossamer.

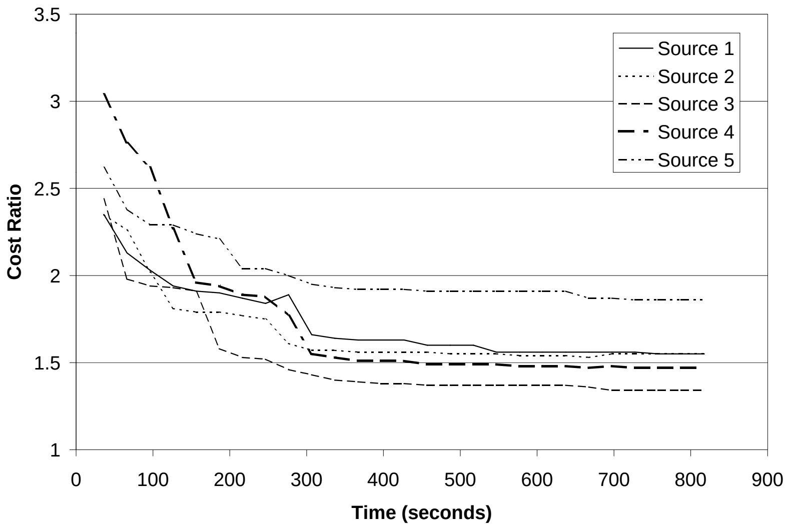


Figure 4.11: Variation of cost ratio $(\frac{C}{C_{mcast}})$ vs time.

Cost Ratio Distribution

Figure 4.10 demonstrates the distribution of cost ratios $(\frac{C}{C_{mcast}})$ for stable meshes over a large number of experiments. We ran 100 experiments over 25 different topologies. Each experiment was comprised of 100 SCXs and 5 sources. The cost ratios were measured for each source once the mesh structure had stabilized. The y-axis represents the number of experiments that resulted in a cost ratio between ± 0.05 of the corresponding x-axis value. As seen from Figure 4.10, the distribution of cost ratios is centered around 1.65, that is, the cost of routing data on the scattercast distribution tree is typically 1.65 times that of directly multicasting the data from the source SCX to the other nodes. For a small number of experiments, the cost ratio was as low as 1.35 or as high as 2.35. This plot indicates that Gossamer results in similar stable mesh structures over repeated runs of the experiment with similar final cost ratios.

Mesh Stabilization

We study the variation in mesh structure with respect to the cost ratio and the edge changes that occur within the mesh over time. Figure 4.11 shows the variation of cost ratio $\frac{C}{C_{mcast}}$ for each distribution tree during the progress of an experiment involving 5 source nodes. We notice that over time, the cost ratio progressively decreases for each of the five distribution trees. Within about 300 seconds all of the cost ratios mostly stabilize to their final value. Initially, the SCXs attempt to locate other nodes and to determine

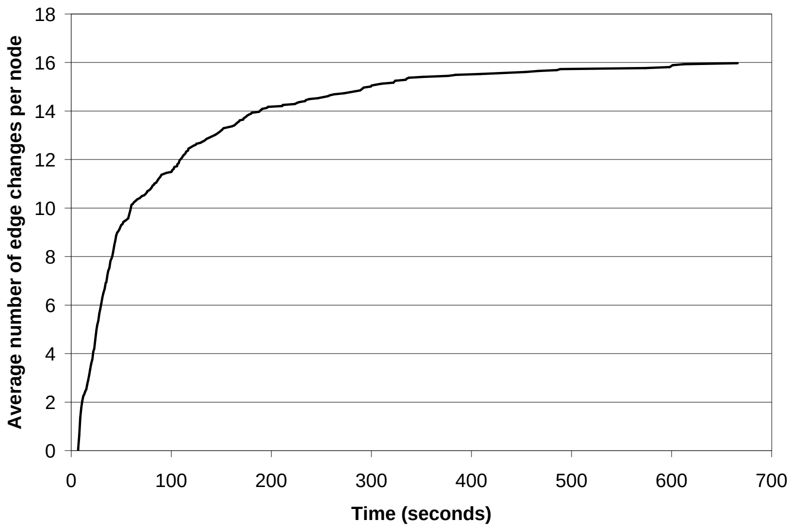


Figure 4.12: Cumulative number of edge changes per node in the mesh structure over time. Each edge that is added or removed is counted as two changes, one for each node in the edge.

their utility as neighbors. Over time, as they discover their optimal neighbors, the mesh stabilizes into its final overlay structure.

In Figure 4.12, we study the stabilization properties of Gossamer with respect to the edge changes that occur in the mesh over the lifetime of the experiment. The figure shows the cumulative distribution of the average number of changes made to the mesh per node. Each edge that is added to or removed from the mesh is counted as two changes, one for each node in the edge. As in the previous experiment, we notice that most of the changes to the mesh topology occur in the initial stages of the experiment. Within about 300 seconds, the mesh stabilizes to almost its final structure. The above two experiments demonstrate that the Gossamer algorithms are effective in rapidly converging to a stable mesh structure.

Packet Duplication Overhead

We now demonstrate the effectiveness of the Gossamer distribution trees in limiting the number of duplicate copies of data that any internet link needs to carry. We ran a single experiment and computed the number of physical links in the underlying internet topology that carried duplicate data copies. We note that, by definition, source-rooted multicast distribution will result in no packet duplication: all links carry exactly one copy of the data packets. Since Gossamer does not rely on router support to provide optimal tree branching

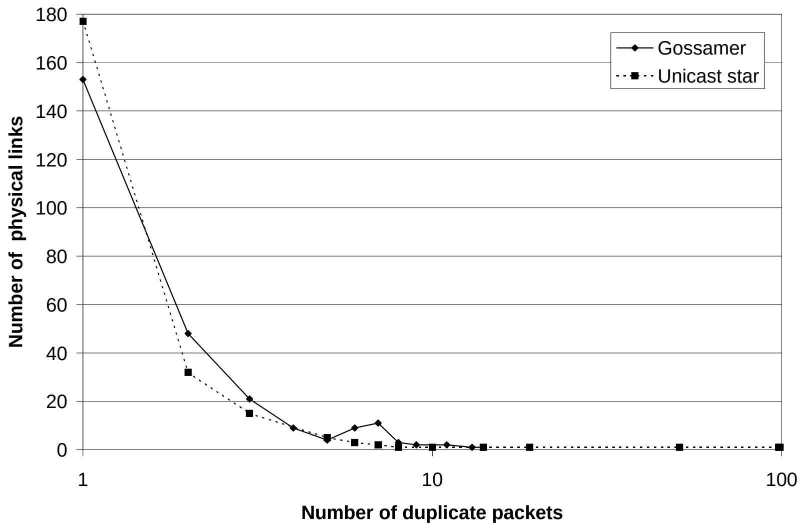


Figure 4.13: Comparison of the number of physical links with packet duplication for the Gossamer topology vs direct unicast.

points, it will necessarily result in a few internet links having to carry multiple copies of the data packets. We compare the packet duplication overhead for Gossamer with that for the extreme case of direct unicast.

The plot in Figure 4.13 depicts the distribution of packet duplication across the physical Internet links where the x-axis represents the number of data copies that any link may see, and the y-axis represents the number of links that carry a given number of copies. We notice that most links carry only one copy of the data: in the Gossamer distribution tree, 153 of the physical links carry only one copy, and in the unicast star topology 177 links carry a single copy. However, the tail of the plot shows that with unicast at least two links carry over 95 copies of each data packet and many other links still have a substantially high level of packet duplication. The worst duplication occurs at the links near the source. In comparison, with the Gossamer mesh structure, no link carries more than 14 copies of the data. Thus, we see that Gossamer is quite effective in limiting the packet duplication overhead in comparison to naïve unicast. Even though the packet duplication overhead is high when compared to native multicast, it is evident that a smart distribution protocol such as Gossamer can provide substantial improvement over naïve unicast. We note, however, that the amount of packet duplication in the Gossamer overlay network is a function of the node degree ($<k_1,k_2>$) associated with the various SCXs in the network. The node degree is not a fixed constant for the entire network. SCXs that are connected to the Internet via "fat" pipes can tolerate higher packet duplication and can hence be assigned a higher node

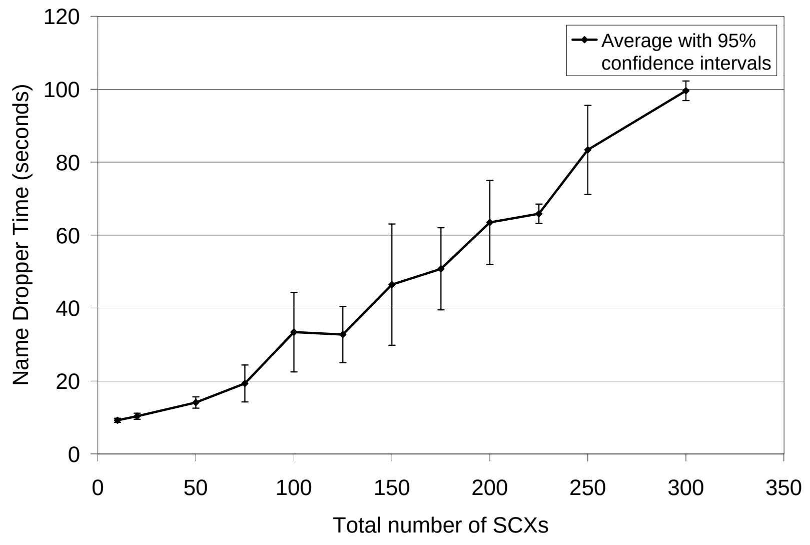


Figure 4.14: Name Dropper++ scaling: Time for everyone to discover at least 90% of the other nodes vs session size.

degree. This allows Gossamer to improve latencies within the overlay structure by reducing the number of hops that a packet has to encounter before reaching its destination.

4.3.4 Scaling Behavior

The above experiments depict the behavior of Gossamer for a fixed number of SCXs and a fixed node degree. We now look at how the protocol operates as we vary these parameters. For all of the following experiments, we compute each data point by running 25 independent simulations and calculating the average and the 95% confidence interval.

Name Dropper++ Scaling

Figure 4.14 shows the scaling behavior of the Name Dropper++ algorithm. The x-axis plots the session size in terms of the number of SCXs and the y-axis plots the time when all of the SCXs in the session have discovered at least 90% of the other SCXs. As expected, the time to completion of the Name Dropper++ algorithm increases with increasing session size. We note that Name Dropper++ performance scales essentially linearly as opposed to the $O(log^2 n)$ performance for the original algorithm in [61]. This is due to the fact that we use a bounded list size during each DISCOVERY round unlike the original algorithm which exchanges the entire membership set $\Gamma(X)$ in each round. This penalty is incurred to limit the communication overhead in each round. A higher (or infinite) bound on the

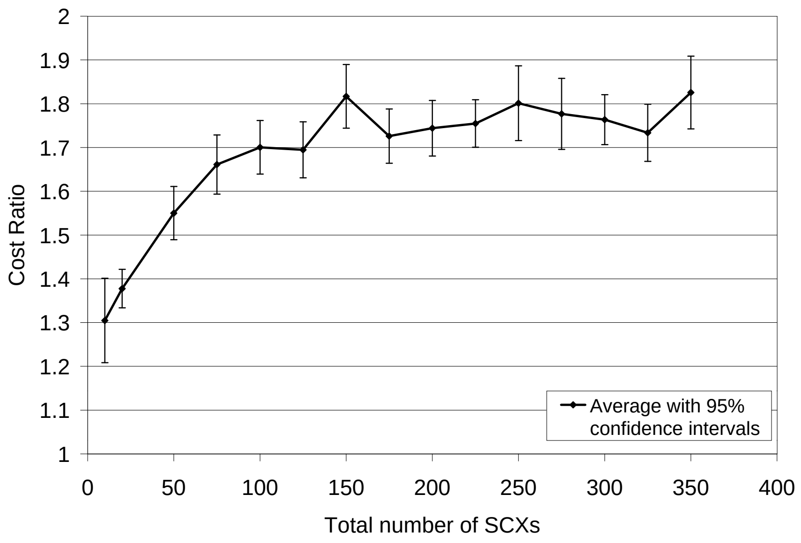


Figure 4.15: Variation of cost ratio ($\frac{C}{C_{mcast}}$) vs session size.

size of the membership list that is exchanged during each DISCOVERY round would reduce the amount of time that SCXs require to discover everyone else. However this reduction in time would come at the cost of increased control traffic as a result of larger DISCOVERY and DISCOVERY_RESPONSE messages. We hypothesize that a reasonable solution would be for an SCX to start with a high value for the bound on the size of its DISCOVERY messages when it initially joins the session and gradually reduce the bound as it discovers more and more SCXs. Although we have not experimented with this form of adaptive bounds, intuition suggests that it will allow SCXs to discover most of the other session members quickly while still keeping the communication overhead low.

Scaling of Cost Ratio with Session Size and Node Degree

Figure 4.15 shows the variation in cost ratio across a range of session sizes. For a small number of SCXs, most of the SCXs are directly connected to the source SCX, and hence the cost ratio is low. As the number of SCXs increases, most SCXs receive data through other transit SCXs, thereby resulting in a higher cost ratio. We measured the cost ratio for sessions with up to 350 SCXs. For most sessions, the cost ratio remains within 1.6 and 1.9. This implies that typically the average latency associated with a Gossamer distribution tree is within a factor of 2 of the corresponding latencies for native source-rooted multicast.

Figure 4.16 shows how the cost ratio varies with node degree for sessions with 100

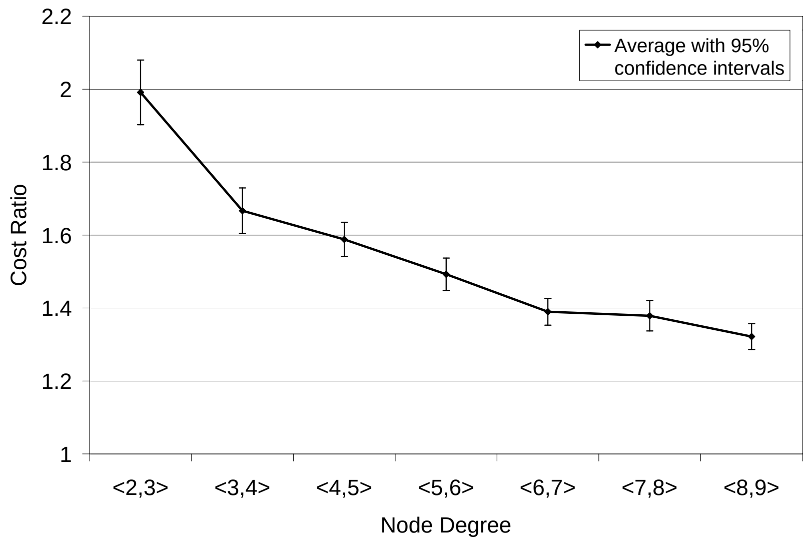


Figure 4.16: Variation of cost ratio ($\frac{C}{C_{mcast}}$) vs node degree ($<k_1,k_2>$ as defined in Section 4.2.1).

SCXs. As expected, the cost ratio decreases with increasing node degrees. As the node degree increases, the depth of the distribution trees decreases, thereby decreasing the cost of routing over the tree. In the extreme case, Gossamer degenerates to n-way unicast and the cost ratio falls to 1.0.

Scaling of Stabilization Time

Finally, Figure 4.17 shows the variation in the running time of the Gossamer protocol with respect to the total number of SCXs. We notice that the time it takes for the mesh to stabilize increases with increasing number of SCXs. This is expected since a larger session size implies more SCXs to discover and more SCXs to attempt to optimize for. We note, however, that as shown in Figure 4.12, although the mesh may not stabilize for a long time, most mesh changes occur early on and subside fairly quickly; only a small number of nodes continue to optimize their connections for a while. Moreover, SCXs and clients do not have to wait until the mesh has completely stabilized to start receiving data from the sources. Clients start receiving data as soon as any path is established between the source and the destination. This initial path may be suboptimal, and the SCXs will eventually reconfigure themselves into a more optimal overlay structure.

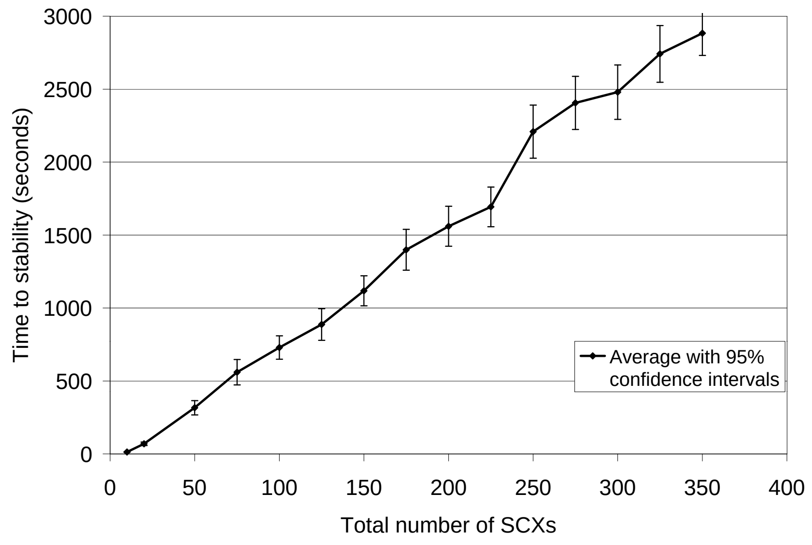


Figure 4.17: Time to stability vs session size.

4.4 Discussion

In the previous section, we provided detailed measurements of the Gossamer protocol that confirmed the effectiveness of our design and algorithms. Our measurements relied on an implementation of Gossamer within a simulated environment. We have also implemented a preliminary prototype of Gossamer for use on the Internet and have built applications on top of the architecture. Examples of these applications and our experience building them are described in Chapter 6.

We note that Gossamer is certainly not the only application level multicasting protocol that is possible for scattercast. It is the result of our initial experimentation with building the various components of the scattercast architecture. In fact, this area of research, namely moving multicast functionality out of routers into end-systems or application level infrastructure, has attracted significant interest in the past couple of years. Gossamer shares a common thread with a few other research projects [65, 49, 62] that have designed or are in the process of designing application level multicasting protocols and architectures. We discuss this related research in more detail in Chapter 7.

Gossamer uses unicast distances and latencies from the source as the metric for performing application level routing. However, in practice, a number of other factors can affect routing decisions. For example, we may wish to bypass a heavily loaded SCX to provide better throughput to applications. Bandwidth capacity across links and estimated loss rates may be other criteria that can impact the underlying routing. Moreover, application-specific policy decisions may also affect our choice of routes. To incorporate such a diverse set of characteristics into the routing decision, Gossamer should generalize the routing and mesh optimization algorithms into a more customizable framework. Applications should be able to choose routing policies that are most appropriate for their needs. This is a challenging problem that needs further research.

4.5 Summary of Application Level Multicasting

In this chapter, we have described Gossamer, a dynamic self-configurable protocol for application-level multicasting across SCXs. The Gossamer protocol forms the core of the scattercast architecture by enabling SCXs to configure themselves automatically into an overlay distribution structure. By exploiting a two-step approach, SCXs first build a richly connected mesh, and then run an application-level routing protocol on top of the mesh to construct source-rooted data distribution trees. Gossamer embraces the gossip paradigm for SCXs to discover each other when they join a session. Our adaptation of the Name Dropper discovery algorithm [61] incorporates network-friendly behavior by limiting the communication overhead due to the discovery process.

By using an adaptive mesh construction algorithm, we allow SCXs to optimize the mesh structure incrementally and as a result build efficient distribution trees. SCXs build a degree-restricted mesh in a distributed fashion. A variant of distance vector routing run across SCXs provides the necessary information to build source-rooted reverse shortest path distribution trees. The distribution trees are fashioned out of application-level unicast transport connections between SCXs.

Our simulation results show that the latencies incurred by transmitting data over the Gossamer mesh are typically within 1.6 to 1.9 times those associated with directly multicasting or unicasting the data from the source to the various destinations. At the same time, the mesh generated by Gossamer substantially limits the bandwidth usage of the physical Internet links in comparison to naïve n-way unicast.

We believe that Gossamer and other application level multicasting protocols like it are an important new direction towards solving the multi-point distribution problem. The benefits provided as a result of moving this functionality out of the routing infrastructure to the application level far outweigh the performance penalties that arise as a result of less-than-optimal distribution.

Chapter 5

Application-customizable Transport for Scattercast

In the previous chapter, we presented a protocol for application-level multicasting across SCXs. We now turn to the problem of assembling a flexible multi-point transport framework on top of this architecture. The transport framework needs to address three crucial issues: how to provide efficient and scalable end-to-end reliability across the entire session, how to deal with bandwidth management and congestion control across the wide area, and how to adapt to the heterogeneity that exists in the Internet? In this chapter, we develop a customizable transport framework that addresses these issues and effectively accommodates a wide range of applications.

5.1 Introduction

Gossamer provides the underlying communication components to enable rich broadcast distribution applications across the wide area. Rather than relying purely on a network-layer infrastructure to provide multi-point delivery semantics, Gossamer moves the complexity up to the application layer and establishes an application-level distribution tree. Although the mechanisms provided by Gossamer may be sufficient for some broadcasting applications such as real-time audio and video delivery [73, 89, 58], which are tolerant to losses, a number of applications and services that could benefit from efficient broadcasting require more complex transport mechanisms such as reliable delivery. Examples of such applications include multi-player games, shared electronic whiteboards [87], software distribution and webcasting [111].

For traditional point-to-point unicast applications, TCP is a generic transport protocol that provides reliable transmission. For most unicast applications, TCP works well across the wide area because it adaptively accommodates a large range of network rates, delays and losses. However, the reliability problem for multi-point broadcasting applications is much more complex. Different applications may have widely different requirements for reliability. For example, some applications such as audio and video conferencing may care more about real-time delivery than about absolute reliability, while other applications are not capable of tolerating packet losses. Similarly, some applications may require in-order delivery of the sender's data while others may be capable of tolerating out-of-order arrival of packets. In addition, applications such as stock and news tickers may not care about recovering losses of obsolete data due to the periodically updating nature of their information. Moreover, it is possible to envision environments that have differing reliability requirements for different portions of the data space within the same application. This range of requirements reflects the difficulty of providing a generic transport protocol that can optimally serve the needs of a wide range of broadcasting applications. Although one could design a protocol for the worst-case requirements—complete reliability with in-order delivery—such an approach will impose unwarranted overhead for applications with more modest requirements.

In addition to the issue of customizable reliability policies, an important challenge with providing reliability for broadcast applications is the implosion problem at the source [109]. When receivers in a large scattercast session discover a packet loss and react to it simultaneously by sending a loss report to the source, the resulting implosion of control traffic can not only overwhelm the source but also cause congestion in the network. A good transport protocol should be designed to take this implosion effect into account and suppress redundant control traffic.

A third issue that needs to be addressed is that of effective bandwidth management. For any protocol to be viable as a "good Internet citizen," it must have sound congestion control built in. Again, in the unicast scenario, for most non-real-time applications, TCP provides effective congestion control. By employing an active feedback loop between the source and the receiver, TCP can detect and react to congestion. On the other hand, for multi-point broadcasting, congestion control proves to be a more challenging problem. Although schemes have been built to provide TCP-friendly congestion control in the multi-point distribution realm [136, 140], these approaches have limitations. They work only with single-source sessions, and none satisfactorily accommodate bandwidth heterogeneity across the distribution tree. When network heterogeneity convolves with the broadcast communication model, a communication source is potentially confronted with a wide range of path characteristics to each receiver, e.g., different delays, link rates, and packet losses. Consequently, that source cannot easily modulate its data stream in a uniform fashion to best match the resource constraints in the network. For example, if the source sends at the most constrained bit rate among all paths to all receivers, then many high-bandwidth receivers experience performance below the network's capability, whereas if the source sends at the maximum possible bit-rate, then low-bandwidth paths become congested and receivers behind these congested links suffer. A source cannot simply transmit a stream at a uniform rate and simultaneously satisfy the conflicting requirements of a heterogeneous set of receivers.

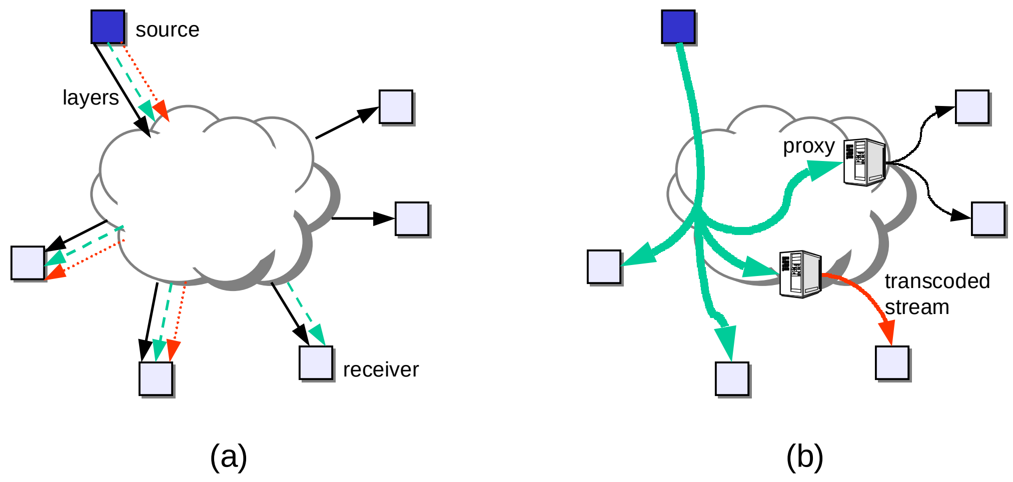


Figure 5.1: Examples of adapting to heterogeneity: (a) Layered Media, and (b) Proxy-based approach.

This heterogeneity that is inherent in the Internet fundamentally challenges the design of a flexible transport protocol for the scattercast architecture. A number of promising research projects have addressed the problem of heterogeneity for multi-point distribution in the particular case of real-time audio/video data, and each of these solutions generally falls into one of two categories: end-to-end adaptation based on layered media [126, 105, 20, 90] or proxy-based transcoding embedded within the network [132, 9]. Figure 5.1 depicts examples of the two approaches. In the former approach, a source encodes its signal in a layered representation and stripes these layers across multiple multicast groups. In turn, receivers individually tune their reception rates by adjusting the number of groups they receive. As a result, heterogeneity is accommodated since each receiver sustains the maximum rate that the network supports.

In the proxy model, media gateways are situated at strategic points within the network and actively transform media streams to mitigate bandwidth heterogeneity and client diversity. By placing a proxy between the source and sink of data, we can accommodate network bandwidth variation through format "distillation" [48] and optimize the allocation of bandwidth across flows using intelligent rate adaptation [7, 9]. Moreover, the proxy can translate the underlying media representations to enable communication among otherwise incompatible clients.

Unfortunately, not all applications are amenable either to layered representation or to transformational compression. Traditional streaming applications such as audio and video are composed of ephemeral streams and can gracefully accommodate packet loss by momentarily degrading quality. However, many applications like group whiteboards or shared text editors rely upon "persistent state" and thus require that all data eventually reaches all interested receivers, i.e., such applications require a reliable transport. Coping with network heterogeneity in these cases is more challenging compared to the unreliable case because the goals for reliability imply that information cannot be discarded to create a heterogeneous set of transmission rates. In other words, the source is fated to run at an average rate at or below the most constrained receiver's rate.

This problem of reliable data transport in the context of multi-point distribution applications has attracted the attention of a number of researchers [43, 82]. Traditionally researchers have appealed to the end-to-end design principle [120] for building protocols to deal with reliability, congestion control, and rate adaptation. However, unlike reliable point-to-point protocols like TCP, reliable multi-point protocols must deal with scale and heterogeneity. In spite of almost a decade of research, no viable solution that solves these problems completely has emerged.

To address these problems, we present a twofold solution by (1) relaxing the semantics of reliability, and (2) pushing complexity away from the end-points into the infrastructure SCXs[1] to assist in the transport protocol. In relaxing the semantics of reliability, we lift the constraint that all receivers advance uniformly with a source's data stream. To achieve this, we leverage the Application Level Framing (ALF) protocol architecture [30], which says that application performance can be substantially enhanced by reflecting the application's semantics into the design of its network protocol. Thus, to accommodate network heterogeneity, we allow each receiver to define its own level of reliability by explicitly requesting retransmission of lost data *if* and *when* it wishes to do so. We also allow the application to decide how and to what degree individual application data units (ADUs) might be transformed and compressed thereby admitting a scenario where receivers "tap" into the session at a variety of rates. As a result, different receivers can implement different levels of reliability and tune the transport protocol to their users' requirements or to suit their application environment.

Our experience with building reliable transport protocols for multicast applications [113] has demonstrated the difficulty of building purely end-to-end protocols that are efficient, scalable and capable of dealing with heterogeneity. With traditional IP multicast protocols, architecting reliable transport in an end-to-end fashion proves to be inefficient, because when loss recovery is required, sending retransmissions to the entire session does not scale to large sessions. Moreover, even if we wish to localize loss recovery to only parts of the session, the underlying network provides no support to leverage its knowledge of the network topology for limiting the loss recovery traffic. We instead take a radically different approach in scattercast by explicitly involving the distribution infrastructure in the transport protocol and exposing the topology of the overlay distribution tree to the transport protocol. By moving complexity away from end clients into the infrastructure, we simplify the implementation of the transport protocol at the clients and enable the incorporation

[1]We note that in previous work on the scattercast transport [24], SCXs were referred to as Reliable Multicast proXies or RMXs.

of a wide range of clients from impoverished devices such as PDAs to high-end desktop workstations. SCXs explicitly participate in the transport protocol and assist in providing efficient loss recovery and congestion management. Moreover, by incorporating application-specific adaptation into the SCXs, we can address the heterogeneity that arises across the entire session.

The scattercast transport protocol leverages the robustness and congestion-friendly behavior of unicast transport protocols such as TCP to assist in wide-area distribution. Communication across SCXs takes place using well-understood unicast protocols and we construct end-to-end transport on top of these robust building blocks. The use of infrastructure support and point-to-point rate and content adaptation across SCXs simplifies the higher level transport protocol.

The rest of this chapter is organized as follows. In Section 5.2, we discuss the relevance of the Application Level Framing arguments to our framework. Section 5.3 presents a hierarchical naming protocol called Transport Independent Naming System (TINS) that we use to customize the transport framework to the individual application requirements. Section 5.4 describes a communication model for the transport protocol and presents the details of the end-to-end data distribution and loss recovery strategies. In Section 5.5, we show how we make heavy use of ALF at all levels. Finally, we summarize our experiences.

5.2 Application Level Framing for Semantic Transport

As we described in the previous section, different broadcasting applications may have different reliability requirements. Similarly, the mechanisms that applications use to adapt to congestion and changes in available bandwidth vary from application to application. For example, a real-time video distribution application can accept intermittent packet loss and can adapt to congestion by reducing its frame rate or encoding resolution. On the other hand, a software or file distribution application is not tolerant of packet losses and needs to actively recover any lost packets. Moreover, unlike video transmission, it cannot easily adapt the amount of data it is transmitting to deal with congestion; instead it will simply transmit at a slower rate. These examples demonstrate that it is difficult to build a single transport protocol that is capable of dealing with this range of application requirements.

In 1990, Clark and Tenenhouse proposed a new protocol architecture called Application Level Framing (ALF) [30] which espouses the inclusion of application semantics in the design of the underlying protocols. The philosophy of ALF is that in order to satisfy the diverse range of application requirements, the transport protocol should leave as much functionality and flexibility as possible to the application. The application is best suited to make the most intelligent decisions with respect to what portions of the data space are required to be reliable, what sort of ordering constraints are needed, and how the application should react to changes in available bandwidth.

To realize the principles of Application Level Framing, rather than allowing the transport protocol to define its own arbitrary notions of packets and packet boundaries, we allow applications to define and enforce their own concept of data boundaries in terms of Application Data Units or ADUs. An ADU is the smallest unit of data that can be processed independently by an application, such as a block of a video frame or a scan of a progressive JPEG image. Application Level Framing requires that the underlying transport protocol then adhere to these application-defined data boundaries. Even though the transport protocol may under the covers fragment an ADU into smaller subunits for transmission and error recovery, it must reassemble the ADU and present it atomically up to the receiving application.

ADUs define not only useful packet boundaries but also the context in which that ADU is meaningful to the application. ADUs can be used as the unit of error recovery by the transport protocol based upon input from the application. If the application deems a lost (or partially lost) ADU significant for recovery, it can instruct the transport protocol to commence the recovery process. Similarly, when bandwidth becomes scarce, the application can determine which ADUs are more important than other and accordingly schedule the transmission of ADUs across the network in a more efficient manner.

The principles of ALF thus allow us to define the notion of *semantic transport* as opposed to bit-level data transport, that is, transport of information rather than that of the representation of the information. In addition to performing application-driven loss recovery and bandwidth management, this approach enables us to leverage the infrastructure SCXs to perform application-specific adaptation and transformation of the data as it flows from SCX to SCX. For example, if an incoming data object is a high-resolution image, the SCX may transform it to a lower-resolution representation for faster delivery across low-bandwidth links. Alternatively, it may transform the image into a progressive representation (such as progressive JPEG) and send only as many scans of the progressive image in the form of independent ADUs as can be handled by the downstream participants. Depending on the capabilities of the receiving client devices or applications, the SCX may also convert the data to a different format that is compatible with the clients. Clients, in turn, can request the original high resolution data from the source or their local SCXs if they so desire.

Although the concepts of application-customizable transport protocols are fairly straightforward, realizing them using traditional building blocks in a generic framework for scattercast is not an easy task. ALF presents two conflicting requirements for the design of a generic framework. On the one hand, we need to specialize the transport layer and the SCXs according to specific application needs, but on the other hand, we do not wish to implement a new custom protocol for every new application and client-type. What is required is a common well-defined interface between the application and the transport protocol that can be used to specify application policies to the transport layer. Unfortunately, traditional transport protocols do not provide the necessary primitives to enable this form of customization. For example, TCP's interface provides a flat sequenced byte stream. Such

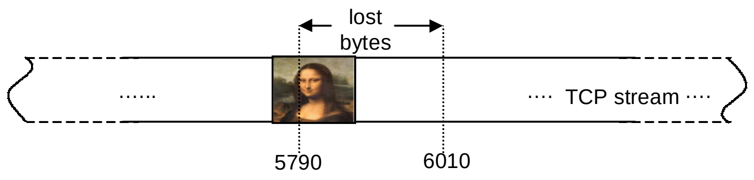


Figure 5.2: An illustration of losses within a TCP data stream: the lost sequence numbers have no correlation to ADU boundaries and may cover only partial ADUs and/or span multiple ADUs; the flat sequence numbering with TCP provides no semantic information to the receiving application regarding the structure of the data that was lost.

a flat naming scheme for data within a TCP stream destroys any semblance of semantic structure that an application may attempt to impose on its data. With TCP, if a receiving application loses the data between sequence numbers 5790 and 6010, the application has no way of knowing what ADU was associated with those sequence numbers and hence cannot make any intelligent decisions about whether or not to recover those lost bytes. This is depicted in Figure 5.2 which shows that the lost sequence numbers may span multiple ADUs in the application's data stream and may correspond to only partial losses of individual ADUs. In other words, the mapping of the ADUs onto a flat numbered sequence space inhibits the receiving application from extracting semantically useful information from the byte stream. What an ALF-capable protocol ought to provide instead is a mechanism by which a source can compute a name for each ADU that permits the receiver to understand its place in the sequence of ADUs produced by the source, and a way for the source to transmit these ADU names in a manner that preserves the semantic structure of the name space at the receiver. In the next section, we present our data naming model for scattercast transport that enables us to provide application-customizable reliability, transmission, and adaptation.

5.3 Data Naming: The Transport Independent Naming System

As we discussed in Section 5.2, traditional transport primitives such as TCP sequence numbers are not sufficient to enable an application-customizable transport framework. Sequence numbers end up eliminating the semantic structure associated with an application's data. Yet this structure is precisely the missing link that would allow applications to intelligently optimize the transport protocol in accordance with ALF. Thus we need a richer naming system that is capable of preserving the semantic structure of the application data through the transport protocol all the way to the receiving application. The transport protocol should be explicitly aware of the ADU naming system and incorporate it into the transport level decisions of loss recovery, bandwidth management, and data

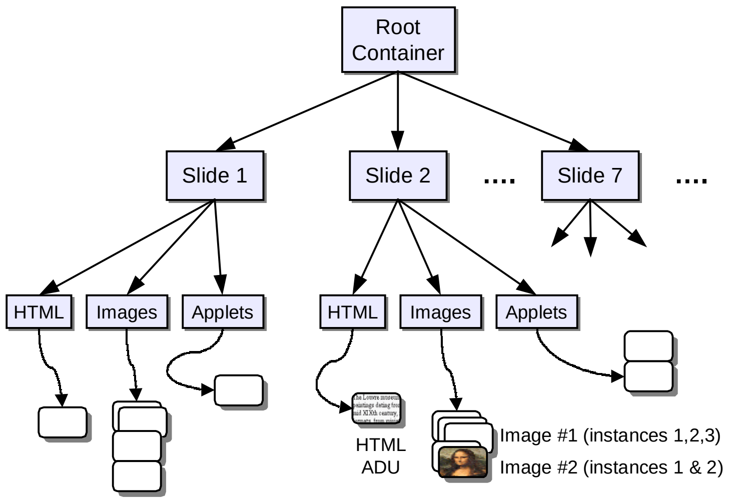


Figure 5.3: A TINS Namespace: an online presentation consisting of multiple slides in HTML format. The example depicts a typical namespace consisting of containers, ADUs, and multiple instances.

adaptation.

A crucial issue that impacts the design of a flexible naming system for scattercast is that of content adaptation and transformation. Our notion of semantic transport permits SCXs to transform data before forwarding. This implies that an ADU generated by a source may have multiple different representations. The transport protocol should allow receiving applications to distinguish between the various representations and to request specific representations of the ADU if they so desire. Hence, our solution provides a well-defined *namespace* that allows applications to reflect the semantic structure of the data within the namespace and exposes the notion of multiple versions or representations of the same data object all the way from the sending application via the transport layer to the receivers. We call this scheme *TINS* or Transport-Independent Naming System.

We derive our naming scheme from SNAP (Scalable Naming and Announcement Protocol) [114], a hierarchical naming protocol designed for use in the Scalable Reliable Multicast (SRM) transport framework. A TINS namespace is a hierarchical structure that applications use as a syntax for representing the semantics of their data. All ADUs of an application are assigned a specific position within the hierarchy and this position reflects the semantics of the application. Nodes within the namespace hierarchy are called *containers*. A container encompasses a group of related ADUs and may point to other sub-containers.

Figure 5.3 illustrates an example of a TINS namespace for an online presentation application. Consider such an application where a sender generates multiple slides for a presentation with each slide represented as a web document that is composed of an

HTML page and embedded image objects and applets. As shown in the figure, the online presentation application creates a namespace for the entire presentation where the root container for the presentation is subdivided into many containers, one for each slide of the presentation. Within each slide container itself, there are separate containers for each data type—HTML, images, and applets. The individual HTML, image, and applet containers point to the actual ADUs associated with the session. Another example of a namespace is one for disseminating news reports: the various sections of the news, such as headlines, business news, sports, etc., are grouped into separate containers, and news items within each section are disseminated as ADUs within the corresponding container.

Each data source within a session constructs its own independent namespace. This is to ensure that multiple sources cannot modify the same namespace and cause inconsistencies due to simultaneous conflicting operations. This approach eliminates the need for conflict resolution across sources within a namespace and simplifies consistency issues that may otherwise arise with multiple sources trying to update the same namespace.

Within a namespace, applications assign their own names to containers. The syntax for these container names is left entirely to the application level. As long as the sending and receiving applications agree on the syntax, the underlying transport layer is oblivious to the semantics embedded within the name. In fact, for efficiency, the transport protocol converts the application-defined container names into numeric identifiers. It uses the numeric identifiers to refer to containers within the transport layer and preserves a mapping between the application-defined name and the container identifier. When a container is presented back to the application layer at the receiving end, the numeric identifier is automatically translated into the application-specified name.

Within a container, data objects are organized into a flat sequential numerical space. Thus each object has a unique sequence number within its container. This effectively produces an application-specific name for each object that is generated by a source. This name, composed of a {*container name, sequence number*} pair, is called the *primary name* of the object. Although the primary name can identify any object associated with a session, it is not sufficient to differentiate between the various versions of that object that might have been generated by the source or by intermediate SCXs within the session. Hence, in addition to the primary name, a source or an SCX can attach a secondary or *instance name* to a specific representation of the object. Each different representation of the object must have a unique instance name. However, the syntax of the instance name itself is opaque to the transport protocol. As in the case of container names, this syntax is determined by the controlling application, and sending and receiving applications are expected to agree upon and be able to decipher the meaning of the instance names for objects in their session. As an example, suppose a source in the online presentation application generates a high-resolution image object. If an SCX transforms this image to a lower resolution, then the SCX must assign a new instance name to the transformed representation. We note that even though a new version of the object was created with a new instance name, the original primary

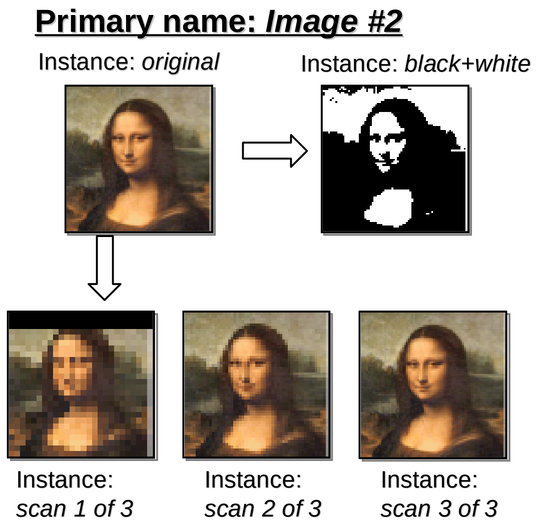


Figure 5.4: An example of the use of primary and instance names in TINS: every version of the original image is assigned a different instance name that reflects the nature of transformation that was applied to generate the new version.

name generated by the source remains the same for all versions of the object. Figure 5.4 shows examples of the use of primary and instance names within the online presentation application.

The TINS namespace is propagated by the transport layer from the source across the network of SCXs to all receivers. When a source constructs a new container, a message is transmitted to the rest of the session announcing the existence of the new container. All ADUs that are transmitted by the source include a header identifying the primary name of the ADU (i.e., the container that the ADU belongs to and its sequence number within the container) as well as its instance name. To allow receivers to detect and repair any losses of container announcements or ADUs in the namespace hierarchy, the source periodically transmits a concise representation of the current state of the namespace. These periodic transmissions act as source-driven announcements of the structure of the TINS namespace. The exact structure of the namespace can be described by traversing the entire namespace tree and including a description of each container in the tree as part of the announcement. However, in the case of sessions with large namespaces, the cost of periodically transmitting the entire namespace in such a manner would be prohibitive. We instead choose a more compact representation known as a *signature* that summarizes the state of the namespace in a short announcement. The signature of the namespace is defined as the signature of the root container of that namespace. In turn, the signature of a container itself is defined recursively as an MD5-hash function of the sequence number of the last ADU transmitted within that container and the signatures of all its children containers. A receiver can compare the signature announcement with its own state to detect ADU losses and to discover missing pieces of the namespace hierarchy. This protocol for announcing the TINS namespace using signatures is based on the protocol used by Raman et al. to transmit the SNAP namespace in SRM. Details of this namespace announcement protocol are described in [114].

When a receiver detects a missing data object, it can decide whether or not to recover that object based on application-specific needs. It can issue a recovery request to the transport layer by specifying the primary name of the object. The transport layer in turn attempts to recover the lost object; the details of the recovery protocol will be described in Section 5.4.2. The recovery request may include an instance name if the receiver wishes to recover a specific version of the object, or just the primary name in which case any version of the object can satisfy the request. Thus an application that receives a low-resolution or transformed representation of an object can ask for the original highest-quality instance with an explicit request.

A key point to note from this discussion is that although the nature of the TINS namespace translates semantic knowledge of the application's data structure into a representation that the transport protocol can understand, the transport protocol itself is unaware of the meaning that the application associates with these semantics. It simply interprets the namespace as a hierarchical tree structure that is to be preserved while transporting the data across the network to the receiving applications. In Section 5.5, we will show how applications can use the TINS namespace to customize the behavior of the transport protocol.

5.4 The Transport communication model

The TINS namespace provides the primary mechanism for including application semantics in the transport protocol. We now turn our attention to the underlying communication model that the transport protocol uses for efficient data distribution. As described in Section 5.1, rather than hide the topology of the overlay distribution tree from the transport layer, we explicitly involve the SCXs embedded within the distribution infrastructure into the decision making process at the scattercast transport layer. This allows us to customize the transport not only to the needs of the various applications that use it, but also to adapt it to the vagaries associated with the heterogeneity that is present across the session.

As we illustrated in Chapter 3, a scattercast session is composed of a number of locally scoped IP multicast groups interconnected via SCXs by wide-area unicast connections. The scattercast transport layer leverages well-understood and robust unicast protocols as building blocks to construct a robust end-to-end transport protocol for broadcast distribution. For example, if an application cares about reliability of its data, scattercast can build its overlay network using TCP connections across SCXs. On the other hand, real-time sessions such as audio/video broadcasting would more likely use UDP connections with a real-time protocol such as RTP [125] layered on top. By relying on well-understood unicast

```
↑ void scast_add_link(SCastLink new_link);
↑ void scast_remove_link(SCastLink old_link);

↓ SCastLink scast_get_link_to_source(SCastSource source);
↓ void scast_get_forward_links(SCastSource source, SCastLink *links,
                               int *num_links_ptr);
```

Figure 5.5: The scattercast API between the transport framework and the application-level multicasting layer for providing the transport framework with information regarding the underlying topology of the overlay network. The direction of the arrows (↑/↓) next to the function definitions indicates whether the function call is an upcall or a downcall.

protocols for wide-area communication, scattercast automatically inherits their robust behavior to assist in wide-area broadcasting. At the same time, we limit the scope of IP-level multicast to the local area. We can use simple local area reliable multicast protocols such as SRM [43] and simple multicast congestion control schemes within the local groups. The actual protocols that are used by the session for communication across SCXs and within locally scoped multicast groups are specified as part of the session announcement (see Figure 3.4).

This overlay topology composed of unicast and multicast transport connections is exposed to the scattercast transport layer. At each SCX, the underlying application-level multicasting protocol (Gossamer) keeps the transport layer informed about the set of *links* associated with that SCX. A link is defined as the interface to either a locally scoped multicast group that the SCX uses for communication with local end-clients, or a unicast connection between the SCX and another SCX or an end-client. Whenever a change occurs in the underlying distribution topology, the application-level multicasting protocol informs the transport layer of the change and accordingly the set of available links changes. Each link has an associated link agent whose job it is to manage the communication over that link. This exposure of the distribution topology provides the transport layer with fine-grained control over how data distribution occurs across the network. Packets may be distributed either along the forward path, that is, along all links along the distribution tree heading away from the source, or along the reverse path back towards the source. Data typically flows along the forward path. The reverse path is used to transmit control information such as congestion notifications and packet loss reports from the receivers or intermediate SCXs back towards the source. Figure 5.5 illustrates the API between the transport layer and the underlying application-level multicasting protocol. The transport layer receives upcalls (scast_add_link() and scast_remove_link()) whenever the underlying topology changes. In addition, the transport layer can query the application-level multicasting layer to discover

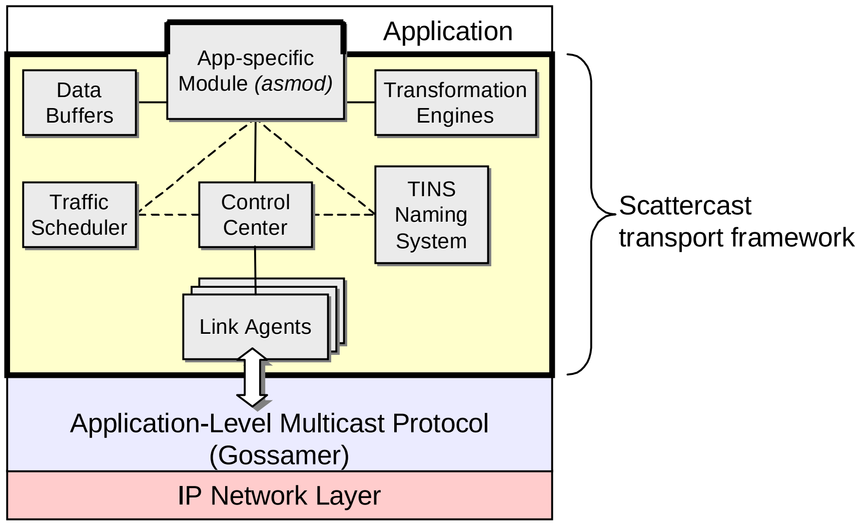


Figure 5.6: The internals of the scattercast transport framework.

the forward and reverse paths along a distribution tree for a source.

To determine the reverse path towards the source, SCXs use a scheme similar to that used by *learning bridges* to discover the route towards hosts on a LAN [108]. Each SCX maintains a table that maps a source to the link along which that source's downstream packets typically arrive. When data is received on an incoming link from a source, the SCX makes an entry in the table mapping that source to the incoming link along with a timestamp indicating the time when the entry was last updated. Every time new data arrives on that link from the same source, the timestamp is updated. Periodically, a process sweeps through the table purging old entries. This ensures that recovery proceeds in the right direction even in the face of dynamic reconfiguration of sources and SCXs.

Figure 5.6 shows the various components of the scattercast transport layer. Each link into the SCX has an associated *link agent* that implements the specific details of the (unicast or multicast) protocol used on that link. The *control center* is the coordination and dispatch unit that shuttles data coming in from any link through the rest of the system. Each SCX has a customizable *Application-Specific MODule* or *asmod* that implements the application-specific pieces of the SCX. The *asmod* is a dynamic code library that is loaded into the SCX at run-time. The control center passes incoming data to the *asmod* which, in turn, decides how the data is forwarded to the other links. The *asmod* relies on specialized *transformation engines* to convert the data when needed into a different representation before forwarding it. A cache of data buffers stores data temporarily before it is forwarded.

5.4.1 Data Delivery

This section details how the transport protocol provides end-to-end communication across the scattercast network. Although the architecture does not require any specific multicast protocol within the locally scoped multicast groups or unicast protocol across SCXs, for this discussion we assume that Scalable Reliable Multicast (SRM) [43] and TCP are the underlying distribution protocols.

Data delivery over the SCX network uses scattercast forwarding. A source that generates data distributes it to its local data group using SRM. The local group members including the representative SCX use SRM's built-in data recovery mechanisms to recover any lost data. The SCX forwards data from the local group towards the participants in the other groups. Whenever the SCX receives data on a link, it forwards the data to the downward links along the distribution tree including its own local data group. As the data flows from SCX to SCX it eventually propagates to the entire session.

Since all participants in a single data group have essentially similar network characteristics, bandwidth management and congestion control are straightforward. Over the wide-area, data flows via TCP connections which have built-in congestion- and flow-control. However, given the heterogeneity over the wide area, different TCP links in the SCX network are likely to have different bit-rates and delays. Thus data may flow into an SCX faster than it can be forwarded. The SCX addresses this by using a combination of the following mechanisms:

- If we assume infinite buffer-space in the SCX, the SCX can buffer incoming data while it tries to forward it along the low-bandwidth connections.
- With finite buffering, the SCX may drop incoming data as its buffers fill up and rely upon end-hosts to recover the dropped data.
- It can adopt an end-to-end flow control scheme whereby it informs the SCX along the upward stream towards the source to slow down its sending rate. Such a feedback mechanism can percolate all the way up to the source and eventually bring the source transmission down to a rate that is suitable for the SCXs to handle.
- The SCX can transform incoming data and forward lower bit-rate versions instead. Our notions of semantic transport and ALF enable application-specific adaptation and transformation of the data that arrives into an SCX before it is forwarded on to the other links. For example, if an incoming data object is a high-resolution image, the SCX may transform it to a lower resolution for faster delivery across low-bandwidth links.

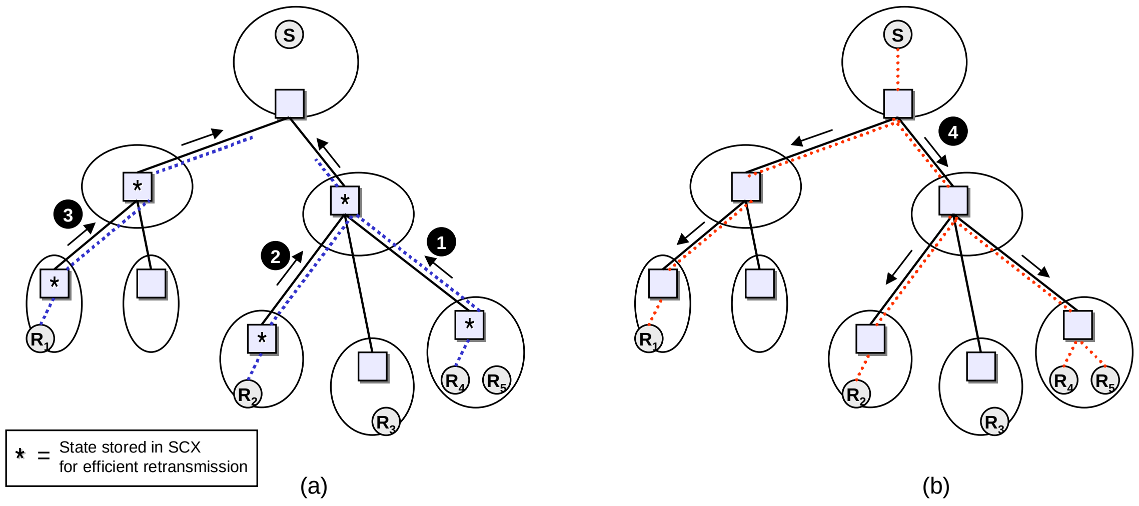
In addition to forwarding data from the source to the receivers intelligently, the transport framework can also act as a bridging layer between the scattercast overlay and standard scattercast-unaware end-clients. For example, many Internet radio applications rely on end clients that download the radio stream over standard unicast protocols such as HTTP [68]. A number of popular clients ranging from RealAudio [118] to WinAmp [103] support protocols such as HTTP. Rather than require that these clients understand and implement the scattercast protocols, we incorporate support into the SCX to translate to a standard HTTP interface that such clients can talk to. The local link agents within the SCX implement the details of these protocols and provide a seamless mechanism for standard clients to hook into the scattercast network.

5.4.2 End-to-end Reliability

Although scattercast can use TCP for reliable inter-SCX communication, we still need an end-to-end data recovery scheme that allows applications to request missing data or to request a different (more refined) representation of a data object. Each hop from the source to the receiver in the scattercast overlay structure is reliable (using either SRM or TCP). If we assume infinite buffering capabilities at SCXs, then there are no drops within the SCXs themselves and all data eventually propagates to all receivers. However, if an SCX has finite buffer space, and the incoming data rate exceeds the rate at which the SCX can forward outgoing data objects, then the SCX must drop data. Even if SCXs do not drop any data, receivers may still wish to recover a different representation of an object from the one that the local SCX forwarded. This requires an end-to-end recovery mechanism that allows receivers to recover missing data from the source or across the SCX network.

When a receiver requests a piece of data, the request is multicast to the receiver's data group using SRM. If the data cannot be recovered from the local group, then the group's SCX forwards the recovery request upward along the SCX hierarchy towards the source of the data. Intermediate SCXs first attempt to recover the data from their local data group, and if that fails, forward the request toward the source.

To limit the scope of retransmitted data, we use a scheme based on PGM (PraGmatic Multicast) [127] and BCFS (Bread-Crumb Forwarding Service) [144]. When an SCX forwards a recovery request towards the source, it records the link on which the retransmitted data must be returned. If another recovery request for the same data arrives, the SCX suppresses the new request and records the new link for that request in its previously saved state. When retransmitted data arrives, the SCX forwards it only along those links that had requested that data. The per-link state is "soft" and is eventually timed out to ensure correct operation in the event of loss of the retransmitted data. Figure 5.7 depicts the loss recovery protocol employed by scattercast. Figure 5.7 (a) shows an example of how recovery requests are forwarded and suppressed by an SCX. All recovery requests result in instantiating state in the SCX that is used for restricting retransmissions of the lost data to only those links that requested the retransmissions. This is shown in Figure 5.7 (b).



1. R_4 sends recovery request towards S; intermediate routers keep state representing this request
2. R_2 sends recovery request towards S; intermediate router suppresses this request
3. R_1 sends recovery request towards S; this request travels all the way to the source
4. S sends retransmission along those paths that made a recovery request; no retransmission to R_3

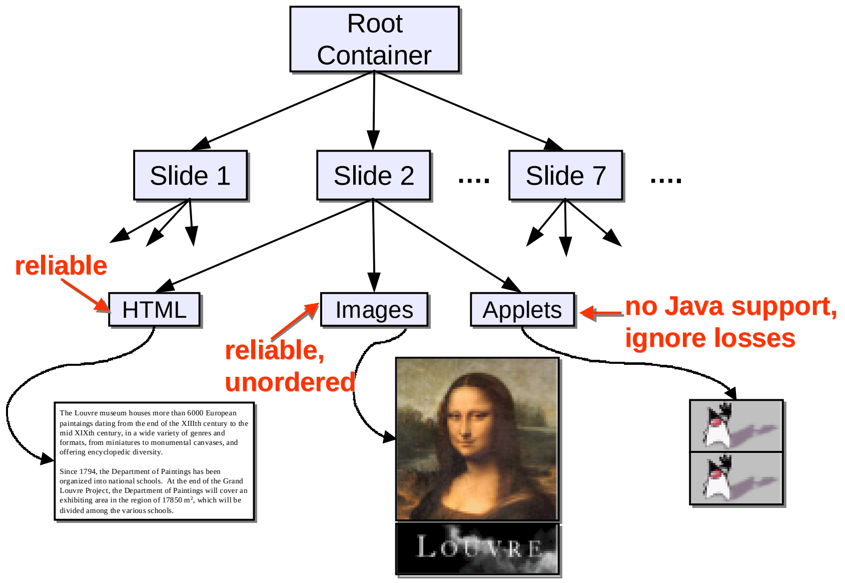
Figure 5.7: End-to-end loss recovery scheme: (a) Propagation (and suppression) of recovery requests from receivers towards the source, (b) Efficient retransmission of lost data.

5.5 Application-defined Semantic Transport

As depicted in Figure 5.6, each SCX uses an application-specific module or *asmod* to customize the transport protocol for each individual SCX. Scattercast isolates the customizable policy decisions into the *asmod* and provides the core mechanisms necessary to achieve this customization. The *asmod* uses two key mechanisms: the notion of hierarchical namespaces, and the knowledge of the transport links in and out of the SCX. The namespace provides the syntax that applications can use to embed reliability and transmission policies. The explicit knowledge of the links connecting the SCX to the rest of the session allows the *asmod* to perform application-specific adaptation on each outgoing link. We thus identify three ways in which the *asmod* can customize SCX behavior:

- customizable data reliability,
- customized transmission scheduling, and
- dynamic data adaptation.

We discuss each of these in detail.



The Louvre museum houses more than 6000 European paintaings dating from the end of the XIIIth century to the mid XIXth century, in a wide variety of genres and formats, from miniatures to monumental canvases, and offering encyclopedic diversity.

Since 1794, the Department of Paintings has been organized into national schools. At the end of the Grand Louvre Project, the Department of Paintings will cover an exhibiting area in the region of 17850 m^2, which will be divided among the various schools.

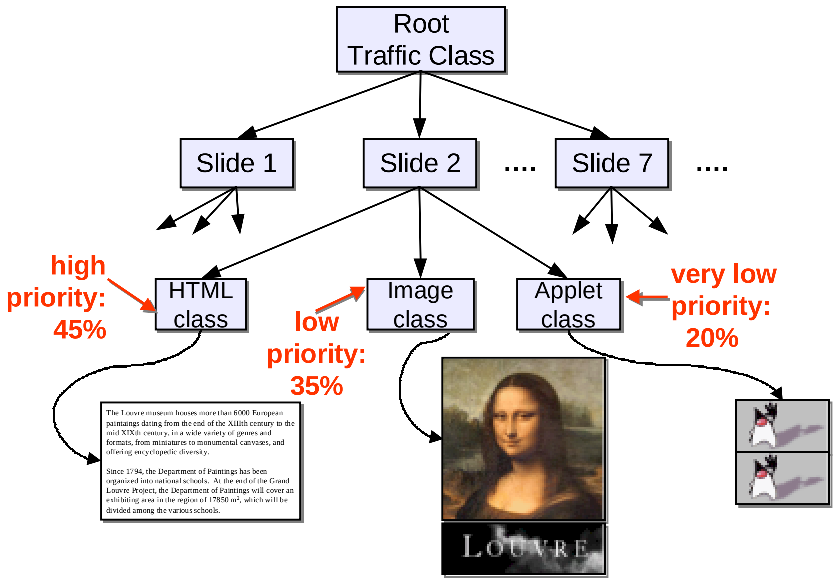
Figure 5.8: Customized reliability: An example of providing customized loss recovery for an online presentation application via the TINS hierarchy. Each container is assigned its own reliability policy based on application semantics. For example, a client with no Java support may decide to ignore losses in the applet container, but require total reliability for HTML.

5.5.1 Application-customizable Reliability

Based upon the principles of ALF, our hierarchical naming system TINS allows sources to compute names for each ADU that permit the receiver to understand the ADU's place in the sequence of data produced by the source. In doing so, TINS allows applications and SCXs to control the behavior of the reliability protocol.

Containers within a TINS namespace are the unit of selective reliability: an application can choose to apply different reliability and ordering semantics to different containers. If an application does not care about certain sub-spaces of the data namespace, then the *asmod* may decide to never issue recovery requests for missing data in that sub-space. When the transport protocol detects a loss of an ADU (or part of an ADU), it issues a notification to the application level that includes the primary name—{*container name, sequence number*} pair—associated with the ADU. This information is sufficient for the *asmod* to decide whether or not to recover the lost ADU based on the application environment's needs. Similarly, it is possible to specify flexible ordering constraints on the ADUs on a per-container basis.

For example, in the case of the online presentation application, a receiver or SCX may assign different reliability policies to the various containers of the namespace. Since a



The Louvre museum houses more than 6000 European paintaings dating from the end of the XIIIth century to the mid XIXth century, in a wide variety of genres and formats, from miniatures to monumental canvases, and offering encyclopedic diversity.

Since 1794, the Department of Paintings has been organized into national schools. At the end of the Grand Louvre Project, the Department of Paintings will cover an exhibiting area in the region of 17850 m^2, which will be divided among the various schools.

Figure 5.9: Customized transmission scheduling: An example of providing customized bandwidth management for an online presentation application via a hierarchy of traffic classes. ADUs are assigned to specific traffic classes and each traffic class is assigned a priority or a portion of the available transmission bandwidth based on what the application deems to be the usefulness of the data in that class.

presentation may consist of a large number of slides, only one of which is active and visible at any time, the namespace can be set up to ignore losses on older slides until a user specifically backtracks to look at previous slides. Moreover, the various images within any particular slide are not required to be delivered in order. In fact, out of order delivery of inline images will speed up rendering of the slide on the receiver's display. Hence the application informs the transport protocol to ignore ordering constraints for the image container. Finally, an impoverished device such as a hand-held PDA may not even be capable of handling any of the applets that may be present on the page, in which case it can be configured to ignore losses in the 'applets' container. The mapping of these reliability policies on to the application data is depicted in Figure 5.8.

5.5.2 Application-customizable Transmission Scheduling

In addition to customizing data recovery in an application-specific manner, the SCX provides hooks to the *asmod*s to schedule transmission or forwarding of data depending on the application's and receivers' requirements. Applications may prioritize data objects based on receiver interest or capabilities.

As with application-driven reliability, we use a hierarchical data organization to allow the SCX to flexibly schedule data for transmission. The traffic scheduler in the SCX constructs a hierarchy of *traffic classes* for bandwidth allocation. The *asmod* assigns each traffic class a certain percentage of the available bandwidth. When the *asmod* has a data object to transmit, it assigns the object to a specific traffic class. The scheduler honors the bandwidth restrictions for each traffic class while scheduling the data for transmission over the network. The control traffic such as namespace dissemination and recovery requests are assigned their own independent traffic class to ensure that they do not conflict with application data.

Figure 5.9 shows an example traffic class hierarchy and bandwidth allocations for the online presentation application. Assuming that most of the content in the presentation is embedded in the HTML for the presentation, we allocate a large percentage of the available bandwidth to the HTML traffic class and split the remaining bandwidth between the images and the applets. In addition, at the higher level in the traffic class hierarchy, bandwidth may be distributed across the individual slides in the presentation. We may allocate maximum bandwidth to the currently active slide and use the remaining bandwidth to trickle in content from the obsolete slides. The traffic class hierarchy may be further customized to accommodate progressive image representations. For example, the image traffic class may be further split into two: one for the first scan of the progressive images and another for the refinement scans. This would allow the source to quickly transmit the base scan of a progressive image while deferring the refinement scans until more bandwidth is available.

Although our examples use the same hierarchical structure for the container namespace and the traffic class hierarchy, the scattercast framework does not require that. To maximize flexibility, scattercast allows applications to define entirely unrelated hierarchies for the TINS namespace and for traffic scheduling. The TINS hierarchy defines the ADU names that receiving applications and intermediate SCXs can use to issue recovery requests and for dynamic data adaptation. On the other hand, the traffic class hierarchy characterizes the scheduling policies associated with transmission of the ADU. Each ADU is assigned to a specific traffic class and is appropriately scheduled for transmission by the scheduler.

We can use any suitable traffic scheduler in the SCX such as Hierarchical Round Robin (HRR) [76], Start-time Fair Queuing (SFQ) [53], Hierarchical Fair Service Curve (HFSC) [128], or Class-Based Queuing (CBQ) [42]. Our current implementation relies on a hierarchical SFQ scheduler. The SFQ scheduling algorithm was first proposed by Goyal et al. in [53]. Our reasons for selecting SFQ as the scheduling algorithm for scattercast are fourfold. First, SFQ provides fair allocation of bandwidth across traffic classes even in the presence of variation in the total available bandwidth. This is important since the bandwidth available across the wide area fluctuates over time in the presence of congestion and transient flows. SFQ can allocate the available bandwidth across its classes and sub-classes such that each sub-class receives its specified share. In addition, if certain sub-classes are idle (i.e., there is no data to transmit in that class), SFQ allocates the residual bandwidth fairly among the remaining sub-classes. Secondly, SFQ does not require a priori knowledge of the packet size to be scheduled. As a result an application can defer its

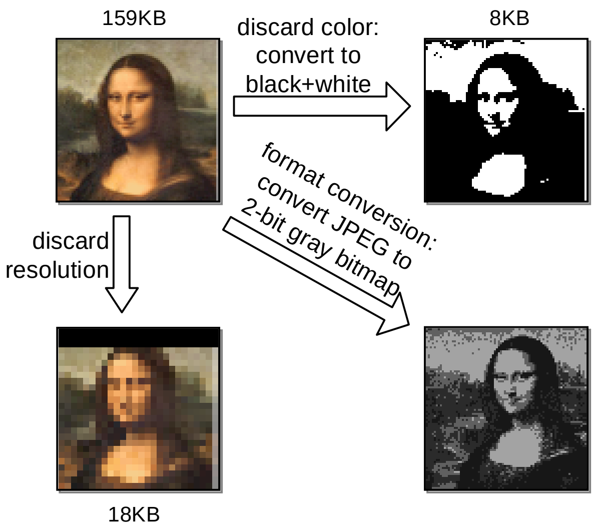


Figure 5.10: Customized data adaptation: Applications choose to apply data-type-specific transformations to the ADUs before transmission to adapt the ADUs to bottleneck bandwidth or to apply format conversion to better suit the receiving application's environment. For example, an image ADU may be transformed to reduce bandwidth consumption by discarding color information or resolution from the original image.

decision as to which ADU to transmit until the last moment when the scheduler is ready to accept the ADU. Third, SFQ provides bounds on the maximum delay and minimum throughput across the traffic classes. These bounds are described in [53]. Finally, SFQ is computationally efficient, so it can be easily used in scheduling high bandwidth traffic without excessive overhead. The details of the SFQ scheduling algorithm are described in [53]. The interested reader can refer to Appendix A for a summary of the algorithm.

5.5.3 Application-customizable Data Adaptation

Our third mechanism for injecting application semantics into the SCX is the use of specialization code for performing application- and data-specific transformation operations to convert data objects on the fly inside the SCX. These transformations depend heavily on the individual applications under consideration.

Figure 5.10 shows the forms of transformations that may typically be applied to image data in an online presentation application. Adaptation has two purposes: to reduce the amount of network bandwidth consumed due to redundant information, and to perform format conversion to adjust the data to an encoding that is better suited to the receiving application's or user's environment. For example, to conserve bandwidth across a bottleneck link, an image ADU may be transformed via lossy compression either by eliminating unnecessary color information or by throwing away excess resolution. The resulting image is an approximate representation of the original, but it may be sufficient to convey the information contained in the image to the receiver. As depicted in Figure 5.10, lossy compression can potentially reduce the bandwidth usage of an image by an order of magnitude. This represents substantial bandwidth savings that can be utilized to transmit more important data across bottleneck links. A second use of dynamic transformation is to adapt the format of the ADU to an encoding that the receiving client can understand. This is especially useful in environments that support impoverished clients such as handheld PDAs and cell-phones that do not necessarily have the programming capability to decode complex data formats such as JPEG images. In such a situation, the SCX can convert the JPEG into a simpler representation such as a 2-bit-per-pixel grayscale bitmap that the end-client understands.

Within the SCX, the *asmod* decides what kinds of transformations are to be applied to data before it is forwarded. It may do so in a statically determined manner (e.g. convert all images to progressive format), or rely on feedback from the traffic scheduler or the link agents to determine the extent of transformation required. For example, if a link agent can notify the SCX whenever it detects congestion in the network, the *asmod* can use this information to transform high-resolution data aggressively to lower resolutions. Similarly, *asmod*s can adjust their transformation granularity based on TCP throughput measurements across SCXs.

Although the TCP layer does not provide sufficient feedback to the application, one can envisage using a more ALF-like unicast transport protocol based on the *Congestion Manager* work at M.I.T. [10]. Such a transport protocol can be better integrated into the SCX: it would allow the *asmod* to dynamically determine the available bandwidth constraints and can notify the *asmod* whenever it detects congestion in the network. The *asmod* can in turn use this information to determine the amount of transformation to apply to incoming data.

5.6 The Programming API

In this section, we present the relevant details of the API that applications use to communicate with the scattercast transport protocol and to customize the protocol to their requirements. Figure 5.11 highlights the important functions that are part of the scattercast API. The API can be classified into four categories: functions for namespace and traffic class manipulation, functions for sending data, functions to handle incoming data, and functions for customizable loss recovery.

```
typedef struct ByteArray { unsigned char *name; int len; } ByteArray;
const ByteArray NullArray = { NULL, -1 };

↓ SCastSession scast_create_session(char *session_name, int channel_id);
↓ ScastSource scast_create_source(SCastSession session, ByteArray source_name);

↓ Bool scast_create_container(SCastSource source, ByteArray parent_container,
                              SCastName new_container);

↓ Bool scast_create_traffic_class(SCastSource source, ByteArray parent,
                                  ByteArray new_class);
↓ void scast_destroy_traffic_class(ByteArray tclass);
↓ double scast_set_class_weight(ByteArray tclass, double weight);

↓ Bool scast_send(SCastSource source, SCastLink link, ByteArray data,
                  ByteArray container, int seqno, ByteArray instance, ByteArray tclass);
↓ Bool scast_request_to_send(SCastSource source, SCastLink link, ByteArray tclass);
↑ void scast_ready_to_send(SCastSource source, SCastLink link, ByteArray tclass);

↓ int scast_delay_until_full_packet(int flag);

↑ void scast_recv(SCastSource source, ByteArray data, ByteArray container, int seqno,
                  ByteArray instance);
↑ void scast_recv_container(SCastSource source, ByteArray parent_container,
                            ByteArray container_name);

↑ void scast_notify_loss(SCastSource source, ByteArray data, ByteArray container,
                         int start_seq, int end_seq, ByteArray instance);
↓ Bool scast_recover(SCastSource source, ByteArray data, ByteArray container,
                     int start_seq, int end_seq, ByteArray instance);
↓ Bool scast_cancel_recovery(SCastSource source, ByteArray data, ByteArray container,
                             int start_seq, int end_seq, ByteArray instance);
↑ void scast_read_adu(SCastSource source, ByteArray data, ByteArray container,
                      int seqno, ByteArray instance);
```

Figure 5.11: The API between applications and the scattercast transport framework. The direction of the arrows (↑/↓) next to the function definitions indicates whether the function call is an upcall or a downcall.

Namespace & Traffic Class Manipulation

The scattercast transport API creates a separate session object for each channel[2] within a scattercast session via the scast_create_session() function. An application can create one or more source objects by invoking scast_create_source(). A source may have an associated application-defined name specified in the source_name argument. Within a session, scattercast provides two separate hierarchies at the transport layer: a namespace hierarchy for providing ADUs with a structured naming system, and a traffic class hierarchy to provide flexible scheduling of ADUs for transmission. The following functions allow applications to create and manage these hierarchies. Applications can use scast_create_container() to allocate a container in the TINS namespace. parent_container refers to the parent container for this new container, and new_container is the name that the application wishes to associate with the new container. The function returns a boolean value to indicate success or failure. The scast_create_traffic_class() function is analogous to scast_create_container(), but it operates on the traffic class hierarchy instead of the TINS hierarchy. To destroy an existing traffic class, applications use scast_destroy_container(); the tclass argument refers to the name of the traffic class that is to be destroyed. To set or modify the weight associated with a traffic class, an application can use the scast_set_class_weight() function. The new weight is passed in via the weight argument while tclass refers to the name of the class that being updated. The function returns the old weight associated with the class.

Data Transmission

A source or SCX uses the scast_send() function to inject ADUs into the session. The link argument refers to the unicast/multicast connection over which this ADU is to be transmitted. data points to the ADU payload. The container, seqno, and instance arguments specify the primary and instance names associated with the ADU, and tclass specifies the traffic class under which this ADU is to be transmitted. Typically, scast_send() will buffer the ADU until the traffic scheduler is ready to accept it for transmission via the specified traffic class. Scattercast also provides an alternative network-driven API for applications to transmit data without buffering it within the transport layer. With this API, the application uses scast_request_to_send() to notify the scheduler that it has data to send in a specific traffic class. The scheduler in turn makes an upcall scast_ready_to_send() to the application when it has room to transmit an ADU via that class. The application can then use scast_send() to transmit the ADU.

Some applications may deal with a large number of very small ADUs. Rather than require the underlying transport protocol to transmit each ADU as an independent packet and result in inefficient network usage, the application can force the transport layer to delay transmitting the ADUs until a reasonably sized packet is available. The transport library uses the delay_until_full_packet flag to control this behavior. Applications typically set this

[2]See Section 3.2.2 for a definition of scattercast channels.

flag, issue a number of scast_send requests, and then reset the flag to cause all previous scast_sends to be bunched into one or more larger network packets.

Data Reception

Data reception in scattercast is driven by the transport library via upcalls to the application layer whenever significant events occur. The library issues an scast_recv_container() upcall whenever the receiver detects a new container generated by the source. The scast_recv() upcall notifies the application when an ADU is available for consumption by the application. The application is expected to copy the incoming ADU from the library buffer pointed to by data into its own application buffers.

Loss Recovery

When the transport library detects a loss of a complete ADU or part of an ADU, it notifies the application via the scast_notify_loss upcall. The loss notification may be for a specific version of a data object in which case the instance argument is set to the instance name of that version, otherwise that argument is set to NullArray. The transport library uses the same notification upcall to also indicate losses of container announcements within the namespace hierarchy. When an application receives a loss notification, it must decide whether or not to issue a recovery request for the lost ADUs (or container announcements). The scast_recover() call can be used to initiate loss recovery for a single specific ADU or a range of ADUs. To cancel a previously issued recovery request, applications use the scast_cancel_recovery() function call. To respond to a recovery request, the transport library at the source or an intermediate SCX issues an scast_read_adu() upcall to access the payload of the missing ADU.

This basic set of APIs provides applications with all the necessary hooks required to customize the behavior of the transport protocol. In Chapter 6, we will present examples of how real applications can be built on top of the scattercast transport framework.

5.7 Summary of Scattercast Transport

In this chapter, we presented a detailed description of how scattercast provides a flexible transport framework that applications can customize to optimize performance for their specific environments. Our transport framework relies on two key concepts to achieve this extent of flexibility. First, we leverage the principles of Application Level Framing to provide the notion of *semantic transport* as opposed to bit-level data transport, that is, transport of information rather than that of the representation of the information. Second, we allow the underlying application-level multicasting layer to expose the structure of the overlay network up to the transport layer. This allows the transport layer to tailor the distribution stream to suit the characteristics of each individual link in the overlay network.

In providing semantic transport, scattercast allows applications to customize their use of various reliability policies and transmission scheduling decisions. In addition, it allows for dynamic on-the-fly transformation of ADUs within SCXs to adapt the data to the vagaries of the network. On-the-fly adaptation allows SCXs to react to congestion by aggressively transforming ADUs and throwing away redundant information. Moreover, SCXs can use this mechanism to serve content in formats that end-clients can understand in the event that they cannot decipher the source's original format.

Scattercast relies on a hierarchical naming structure that allows applications to encode reliability and transmission policies within the transport layer. Our hierarchical naming scheme TINS is derived from a similar scheme called SNAP used for the Scalable Reliable Multicast protocol. It groups related ADUs into a hierarchically organized collection of containers that applications can use to assign specific reliability constraints to different portions of their data namespace.

In the next chapter, we present an evaluation of the scattercast transport framework in terms of its ability to be customized for individual application needs. We discuss how the scattercast transport framework is actually utilized by real applications and optimized for the specific data streams and application environments. By presenting a range of applications that are built on top of the scattercast transport API, we demonstrate the flexibility of the API to accommodate different kinds of applications.

By moving transport-level intelligence into the network infrastructure, the scattercast transport framework provides a new direction for resolving heterogeneity and creating flexible transport mechanisms in the context of broadcast distribution. This model not only simplifies the design of efficient broadcast transport but also enables rich new avenues for adapting the transport protocol to deal with the vagaries of Internet heterogeneity.

Chapter 6

Scattercast Applications: Customizing the Scattercast Transport

In this chapter we discuss a few scattercast applications that we have designed and demonstrate how these applications exploit the customization features of the scattercast model. In particular, we look at how the semantics of the various applications affect their decisions regarding the policies and mechanisms that they use to optimize the distribution protocol.

We implemented a prototype of the scattercast framework using the MASH toolkit [88] as our development platform. This toolkit is a Tcl/C++-based programming framework for multimedia networking applications developed by the MASH research project at UC Berkeley. Our early efforts at building a prototype scattercast application did not include a well-defined transport-independent naming scheme. Our experiences with that prototype led to the evolution of TINS, the transport-independent naming system that we presented in Section 5.3. With a structured naming protocol that exposed application structure to the SCXs and SCX data transformations to the applications, the design of the applications and *asmods* was greatly simplified.

Our range of applications demonstrates the flexibility of the scattercast framework to accommodate not only a wide variety of client devices and network characteristics but also different classes of application data ranging from persistent reliable data to real-time content. The scattercast architecture provides these applications with three key benefits that are not typically available with traditional unicast or multicast-based broadcasting schemes. First, scattercast eliminates the need for a globally deployed IP multicast network. Scattercast applications can leverage multicast if and where it is available, but can still function efficiently even in networks that do not have native multicast support. Second, unlike multiple-unicast distribution, scattercast builds an intelligent distribution topology that makes more efficient use of network bandwidth and allows applications to scale gracefully with increasing numbers of simultaneous receivers. Third, scattercast has explicit support for embedding application-aware intelligent computing within the distribution infrastructure that applications can use to customize the behavior of the broadcast to suit their specific requirements. This allows us to fully optimize the distribution protocol for individual applications and to provide fine grained control over transport policies for bandwidth management and content reliability. In the rest of this chapter, we focus on this ability of scattercast to customize the transport protocol for a range of applications. In particular, while presenting our application design experience, we characterize the behavior of these applications with respect to the following concerns:

Rate adaptation: We demonstrate how the various applications adapt to changes in available network bandwidth and processing capabilities by intelligently adjusting their sending rates.

Customized reliability: Each application uses detailed knowledge of the semantics of its data to determine the extent and form of reliability that it requires. Applications use the flexible scattercast transport API to reflect these reliability requirements into the transport protocol.

Data adaptation: The forms of data adaptation algorithms that the applications use to perform on-the-fly transformations depend entirely on the nature of the application data and the goals of the application in terms of adapting to network or client heterogeneity. Based on the knowledge of the application or end user environment, applications make intelligent decisions regarding what and how to transform.

Scattercast applications typically consist of three separate components: the receiver application, the data source or server, and the customization modules that are embedded within the SCXs. Simple applications may not require any customization modules. Similarly, the receiving applications need not be scattercast-aware. They may simply be programmed to interact with the scattercast infrastructure via a standard interface such as HTTP. In such cases, all the smarts are embedded within the infrastructure and the receiver has no knowledge of such smarts. However, in order to take full advantage of the range of customization offered by scattercast and to achieve end-to-end adaptation, reliability and congestion control, receiving applications must actively participate in the scattercast transport protocol.

6.1 Mediaboard: A Shared Electronic Whiteboard

Our first prototype application is the mediaboard, a collaborative drawing application whose main use is as a shared electronic whiteboard in multi-party conferencing scenarios. It is based on the original shared electronic whiteboard application *wb* [72, 87], which was developed at the Lawrence Berkeley Laboratory. *wb* was one of the earliest applications to explore the space of customizable data transport using Application Level Framing. Based on our experiences with *wb*, we built a new version of the shared whiteboard called mediaboard [131, 21]. Unlike the original whiteboard application in which the application's protocol was tightly integrated into the transport layer without any flexible mechanisms for general-purpose customizability, the latest version of the mediaboard relies on the programmable scattercast transport API. This allows us to provide a clean separation between the application protocols and the underlying transport functionality, while at the same time allows mediaboard to customize the transport protocol to optimize it according to the semantics of the application data.

The mediaboard application presents a shared drawing space interface to its end users. The drawing space is divided into a number of canvas pages. Any participant in the shared application can create a new page in the session. Each page supports a variety of graphic objects and media such as link drawings, text, geometric figures, images, and postscript files. The application allows a group of users to interactively create, display, and modify a collection of objects, and to skim through the various pages of content generated over the lifetime of the session.

Incorporating this application into our framework involved two tasks: making the application aware of the scattercast transport framework including its hierarchical naming system, TINS, and implementing an *asmod* that understood mediaboard semantics. Unlike traditional multicast-based whiteboard applications, the scattercast framework allowed us to provide flexible bandwidth management for mediaboard by embedding mediaboard intelligence in the form of data-aware adaptation algorithms and scheduling policies within the SCXs. To achieve this, the mediaboard *asmod* was customized in the following ways:

6.1.1 Customized Namespace

Mediaboard specializes the TINS namespace into a two-level hierarchy consisting of a root container and many second-level containers, one for each page in the session. Operations on a particular page are ADUs within the corresponding container. When ADUs are created by the source, they are assigned a null instance name. If an SCX transforms an ADU, it assigns the new representation a different instance name that describes the transformation that was applied.

By encoding the structure of the application namespace in such a manner, mediaboard can take advantage of the selective reliability mechanisms provided by scattercast. The application deems only the page that is currently in view to be worthy of loss recovery. If a loss is discovered in the container associated with that page, the application triggers a recovery request for the lost ADUs. Losses in any of the other containers are ignored until the user switches his or her view to a different page, at which time the application issues recovery requests for missing ADUs within the container associated with that page. No ordering constraints are imposed on the delivery of ADUs to the application. The application is capable of processing ADUs out of order, thereby speeding up the rendering of large complex pages within the session.

In this manner, rather than imposing strict total in-order reliability on the application, the scattercast transport framework allows the mediaboard to choose which portions of its namespace are critical enough to request retransmissions of lost data.

6.1.2 Data Adaptation

The *asmod* converts all high-resolution images in the session to progressive JPEG representations. Each scan of the progressive JPEG is identified as a separate instance of the original data object. This allows the *asmod* to send only a certain number of scans selectively depending on available bandwidth and to prioritize the base scans over the refinement scans. The receiving application accumulates successive scans and displays them as they arrive from the network. Each scan is assigned an instance name that defines the scan as part of a progressive representation and includes the scan number of this scan and the total number of scans that make up the image.

6.1.3 Bandwidth Management

The *asmod* prioritizes low-bandwidth operations such as lines, geometric shapes, move, copy, delete, etc. over high-bandwidth data such as images. In addition, it incorporates receiver feedback while allocating bandwidth. Receivers periodically send reports to their local SCX informing it of the page that they are currently viewing. The *asmod* at the SCX aggregates these reports and propagates them to upstream *asmod*s. The *asmod*s use these reports to assign bandwidths to pages that are proportional to the number of receivers that are looking at that page. This form of receiver-driven bandwidth adaptation is based on consensus-driven schemes described in [7].

The *asmod* relies on the traffic scheduler to perform traffic prioritization and bandwidth allocation. Each page on the mediaboard has a corresponding traffic class. The page traffic class is sub-divided into two classes, one for the low-bandwidth data and the other for high-bandwidth objects. The high-bandwidth traffic class generally tends to contain image data which the *asmod* transforms into progressive scans; hence, this class is further sub-divided into classes for the base scans and for the refinement scans. The traffic class for each page is allocated a bandwidth that is proportional to the number of receivers looking at that page. Thus, in the common case, with all local receivers viewing the page with current activity, most of the bandwidth is allocated to that page.

Figure 6.1 shows an example of two mediaboard applications: the source and a receiver behind a low bandwidth link. The source generates a high resolution image on the mediaboard which is translated into multiple progressive scans at the SCX. The scans trickle down to the receiver over the low bandwidth connection. In the figure, the receiver has accumulated only the first two scans of the original image. Over time, the rest of the scans eventually arrive at the receiver and it can reconstruct the full-quality image. With

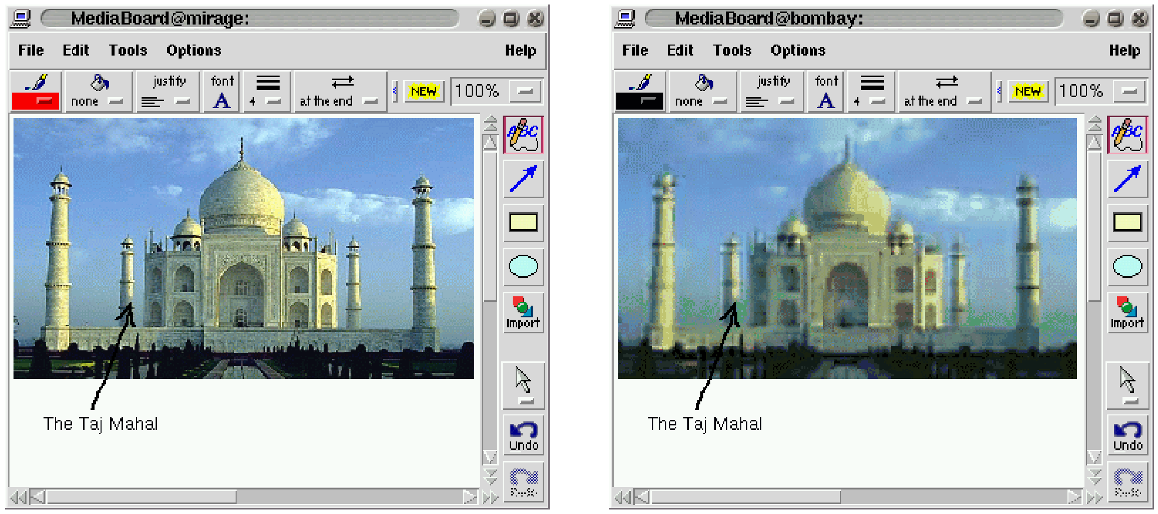


Figure 6.1: The mediaboard application: The client on the left is the source, and the one on the right is a low-bandwidth receiver.

the SCX, the receiver is able to render an approximate version of the image as soon as the first scan arrives; the approximation is refined as more scans reach the receiver.

The mediaboard application was one of the earliest applications designed using the scattercast framework. Our lessons from a building customizable transport layer for mediaboard provided valuable input into the design process for the scattercast transport framework. After a few iterations the generalization of the mediaboard transport resulted in the programmable scattercast API that was presented in Section 5.6.

6.2 Mediapad: Whiteboard for the PalmPilot

The mediaboard application demonstrates the customizability of the scattercast framework. We now look at the use of scattercast to adapt to an extreme form of heterogeneity. This example illustrates the ability of scattercast to support a wide range of client devices simultaneously.

The mediaboard application as described above meets the needs of desktop and laptop PC users. It is not well-suited for impoverished environments such as hand-held devices or personal digital assistants (PDAs). Most PDAs are too limited in their capabilities to be able to handle the complexities of the mediaboard protocol on their own. For example, while the 3COM PalmPilot PDA [1] has a 64 kilobyte code size limit, the binary for the desktop version of *mediaboard* is several megabytes in size. Given these technical limitations of PDAs, it is not feasible to create a stand-alone mediaboard client on current generation PDAs. However, such a client does enable a variety of interesting applications:

System Characteristic	**PalmPilot**	**Typical high-end desktop**
CPU Speed	16MHz	750 MHz
Screen Resolution	160x160 2-bit gray-scale	1600x1200 24-bit true-color
Memory Capacity	2MB physical 64KB address space	512MB physical 4GB address space
Network Bandwidth	28.8Kb/s modem	100Mb/s ethernet
Network Latency	200-400ms wireless [6]	1ms ethernet

Table 6.1: Comparison of device characteristics for a 3COM PalmPilot [1] and a high-end desktop workstation.

- *Meeting support*: Mediaboard-equipped PDAs can be used as collaboration tools to annotate or write on a shared screen that may be projected into the room using an overhead projector or a LiveBoard [142].
- *Smart cell phones*: Smart phones, such as the Nokia 9000 [102] are another interesting example of PDAs. Such smart phones can enhance communication between people: for example, a person trying to locate a friend's house could use a shared map on a small whiteboard on the phone's screen to interact with the friend.

Although the desktop mediaboard application is not amenable to direct implementation on current handheld devices, we were able to implement a simplified version of the mediaboard client—the Mediapad—within the severe limitations of our PDA platform by making extensive use of the ALF principles [23]. The SCXs within the scattercast framework handle most of the complexity of the mediaboard protocol requiring little more from the PDA than a simple drawing canvas.

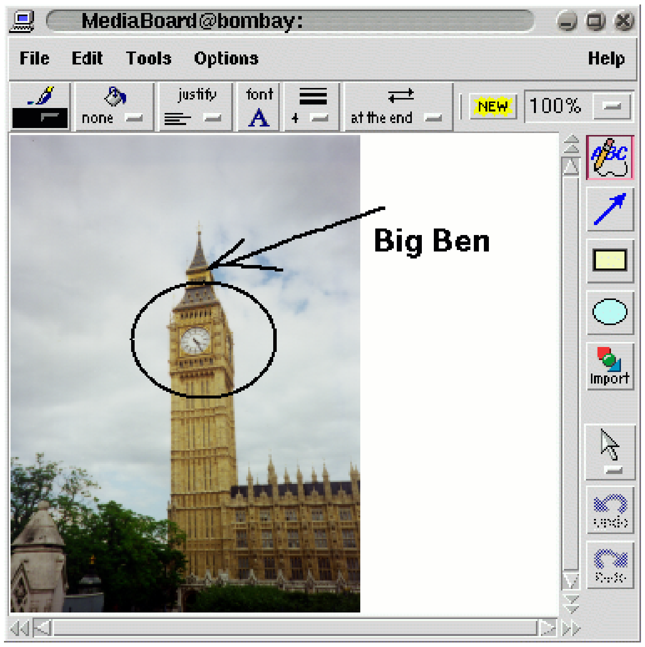
6.2.1 A Split Programming Model

The primary goal for the mediapad application was to maintain full interoperability with the desktop mediaboard and to retain as much of the mediaboard functionality as possible in the simplified PDA client. We used the 3COM PalmPilot [1] as our implementation platform. Table 6.1 summarizes the extreme degree of heterogeneity that the mediapad application is expected to cope with. Given the restricted programming environment available on the PalmPilot, implementing the entire mediapad application on the PalmPilot is not just a formidable task, but we actually expect that it would be impossible to reproduce a realistic efficient version of the mediaboard entirely on the PalmPilot without quite some loss of functionality. We instead take a different approach where we partition the application into a simple user interface component that is implemented on the PalmPilot and a separate backend component that implements the complexities of the mediaboard protocol and translates them into simpler UI commands for transmission to the PalmPilot. The backend is integrated into the scattercast infrastructure via a customized link agent within the SCX that specializes in communicating with the PalmPilot.

An extreme approach is to move all intelligence to the SCX and use the PDA merely as a dumb terminal [66]. Although a viable solution in certain situations, this is entirely impractical in most wireless environments. Wireless network connectivity tends to be highly unreliable, and rendering a PDA useless when disconnected from the network is unacceptable. We would like to have some limited functionality on the client even when it is not connected to the SCX. For example, the client should be able to continue browsing a stored session while disconnected from the network. In addition, given the fact that wireless connectivity to PDAs is quite limited, the latency incurred by the client if it were to communicate with the SCX for each and every operation it wishes to perform would be high enough to render it practically unusable as an interactive application. We instead use a smarter client application that relies on local computing power for the basic interactive components and pushes the more complex operations to the powerful machines hosting the SCX.

This approach of splitting the application program allows us to keep the PDA client simple while at the same time retaining most of the mediaboard functionality. The simplicity of the PDA application directly translates into reduced complexity and increased robustness. Since designing and implementing applications in traditional desktop environments is much easier than the cumbersome and bug-prone programming environment associated with the PalmPilot, the split programming model results in less code to debug on the PalmPilot which in turn implies fewer chances for bugs.

By moving complexity away from the PalmPilot into the infrastructure, we allow mediapad clients to interact with existing mediaboard applications without requiring any modifications to the desktop versions. The mediapad client supports a simple drawing canvas that can display and manipulate simple objects such as lines, text, geometric shapes, and bitmaps. Rather than overloading the canvas with complex operations, we rely on the scattercast infrastructure to translate the complex mediaboard protocol into a sequence of simple draw operations or *draw-ops*. The edge SCX within the scattercast overlay network implements a custom *asmod* and a custom link agent for handling mediapad clients. To provide the mediapad clients with a user experience that is similar to traditional mediaboard users, we have implemented all the standard whiteboard features such as creating, cutting, pasting, and moving objects within the shared drawing space. The mediapad allows users to browse through existing pages without requiring extensive communication with the SCX at each step. Moreover, given the physical limitations of the PalmPilot screen, the mediapad in collaboration with its SCX allows its users to pan around the canvas efficiently and to zoom in and out to different levels of granularity. Figure 6.2 shows screenshots of the desktop mediaboard and the PalmPilot mediapad applications. We note that the same scattercast infrastructure integrates both applications and allows for a seamless experience across this range of device and network heterogeneity.



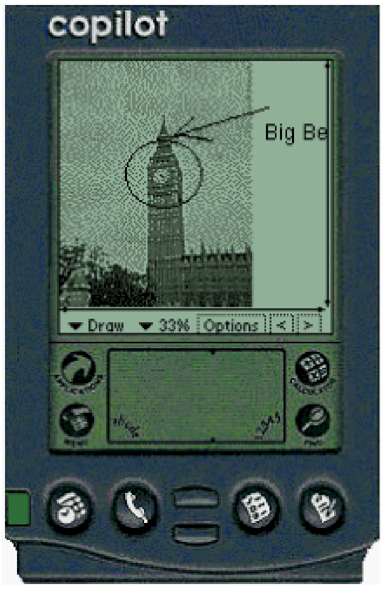


Figure 6.2: The desktop mediaboard application interacting seamlessly with a PalmPilot mediapad client. The PalmPilot screenshot was taken from the xcopilot, a hardware emulator for the PalmPilot [31].

6.2.2 The Mediapad SCX

In this section, we demonstrate how we specialize the SCX to accommodate the PalmPilot mediapad application. Most PDA clients, including the PalmPilot, do not support multicast; hence the link agent for the mediapad clients relies on a direct unicast connection to each of the client devices. To preserve reliability, we use TCP for communication between the clients and the SCX. For every client connected to the SCX, the link agent maintains a connection object which encapsulates the per-client state at the SCX. It contains up-to-date information about the client's device characteristics, the current page, and the current zoom level. The *asmod* uses this information to assist it in the adaptation process.

Rather than require the PDA clients to participate directly in the scattercast transport protocol, the SCX deals with the transport issues on their behalf and shields the clients from having to implement the complexities of the protocol. The SCX deals with losses that occur in the session and uses the reliability mechanisms in the scattercast transport protocol to request lost ADUs and repair them. Each ADU in the mediaboard session is a "command" that performs a certain action on the shared drawing space. Commands are associated with a specific page and client in the session. As described in Section 6.1, the commands are organized into a TINS hierarchy that consists of a separate container for each page in the session.

When the mediapad SCX receives ADUs from the mediaboard session, it stores them in local data buffers. The ADUs are then translated to simple draw-ops for transmission to the mediapad clients. Similarly when the SCX receives data from a mediapad client, it converts the draw-ops into corresponding mediaboard commands, stores them in the local data buffers and forwards copies to the rest of the session.

In adapting the mediaboard data stream for the PDA clients, the SCX leverages detailed knowledge of not only the mediaboard application semantics, but also the application environment—the PDA—for which the data is destined. It effectively performs three forms of adaptation:

1. protocol conversion to maintain the simplicity of the PDA client,
2. data transformation to convert ADU formats to representations that the PDA can understand and to conserve precious bandwidth by dropping useless information, and
3. rate limiting to manage available bandwidth between the PDA and the SCX.

We now discuss each of these in detail.

6.2.3 Protocol Conversion

The mediapad client uses a specialized TCP-based protocol to communicate with its SCX. The SCX effectively provides a bridge between the mediapad protocol and the rest of the scattercast session. To ensure that the client implementation is as straightforward as possible, the SCX handles all the complexities of the mediaboard session. The client receives only a sequence of simple draw operations (*draw-ops*). The SCX transforms the entire session's data into a "pseudo-canvas" by executing each command and storing its result in the canvas. The draw-ops on the pseudo-canvas are what is transmitted to the PDA. This protocol conversion results in simplifying the mediapad client. For example, to eliminate any unnecessary state at the client, all undo operations are performed entirely by the SCX and are converted into appropriate draw-ops before sending them to the client.

Since a mediapad client may join a mediaboard session at any time in the life of the session, the SCX must be able to replay all past events that have happened on the pseudo-canvas. Hence, the canvas caches a history of the effects of all mediaboard commands in memory. When a new client joins the session, it can replay this history. This form of application-aware interaction between the PDA and the SCX allows us to retain within the PDA as much (or as little) of the application complexity as is possible depending upon the PDA's capabilities.

6.2.4 Data Transformation

In addition to converting mediaboard commands into simpler draw-ops, the SCX also converts individual data objects according to the requirements of the clients. The PalmPilot can handle simple draw operations such as lines, circles, rectangles, text, etc. However more complex objects such as images and postscript are too difficult for the PDA to digest on its own. We look at each of these in the following sections.

Image and Postscript conversion: The mediaboard uses the standard GIF and JPEG formats for images, which the PalmPilot cannot understand. Implementing decoders for these formats on the PDA is too complex and time-consuming. Instead, we rely on decoders in the proxy. Internally, the PalmPilot uses a simple bitmap representation for images. The SCX converts mediaboard images directly to the PDA's native representation before sending them. Similarly, the SCX must convert postscript data either to images in the PDA's native format or into plain text that can be easily displayed by the client.

The SCX uses specialized image transformation engines to assist it in the conversion. We have implemented an image converter using code developed by Paul Haeberli [56]. The image converter is optimized for the PalmPilot's screen characteristics. In addition to format conversion, it performs lossy compression by scaling down the images according to the zoom level on the client, the screen resolution of the client, and the color depth of the client's screen. The processing steps consist of image resizing, sharpening, adding noise, and dithering.

Other data types: The SCX also assists the client for seemingly simpler data types such as arrows and fonts. Drawing an arrow requires trigonometric calculations using floating point numbers. The PalmPilot has no built-in floating point hardware and emulation software is either not usually installed or too slow. Hence the protocol adapter computes the arrow coordinates and sends them as part of the draw-op to the client. Similarly, since the client cannot understand the X Windows-based fonts that are used by the mediaboard protocol for text objects, the SCX converts these font names into reasonable native PDA fonts.

Zooming: Since most PDA screens are extremely small, we support zooming to multiple levels on the client canvas. This enables the user to view the session data at different levels of refinement. The client can handle scaling of simple objects (lines, rectangles and ellipses) on its own. For scaling complex objects, it relies on the SCX. Whenever the user switches zoom levels on the client, it communicates this state change to the SCX, which in turn recomputes new font mappings for the new zoom level. In addition, the client may request the SCX to send some or all of the displayed images and postscript at the new zoom level. The SCX recomputes the new bitmap representations at the new zoom level and sends them over to the client.

Data size (bytes)	Mediaboard Protocol	Simplified PDA Protocol
Image	52651	4704
Free-hand drawing	11004	812 (max compression) 10580 (max interactivity)
Arrow	84	76

Table 6.2: Examples of bandwidth savings with ALF-based SCX—PDA protocol.

6.2.5 Intelligent Rate Limiting

Since the SCX has complete knowledge of the client's state, it can perform intelligent forwarding of data from the mediaboard session to the client. Lossy image compression is one such mechanism. By eliminating redundant draw-ops before sending data to the client, we further reduce the number of bytes that must be sent over the low-bandwidth link to the client. For example, if an object has been placed on the canvas and later deleted, the canvas will refrain from sending any information to the client about that object. Similarly, if an object has been moved multiple times, all move operations are combined into a single draw-op before sending it to the client.

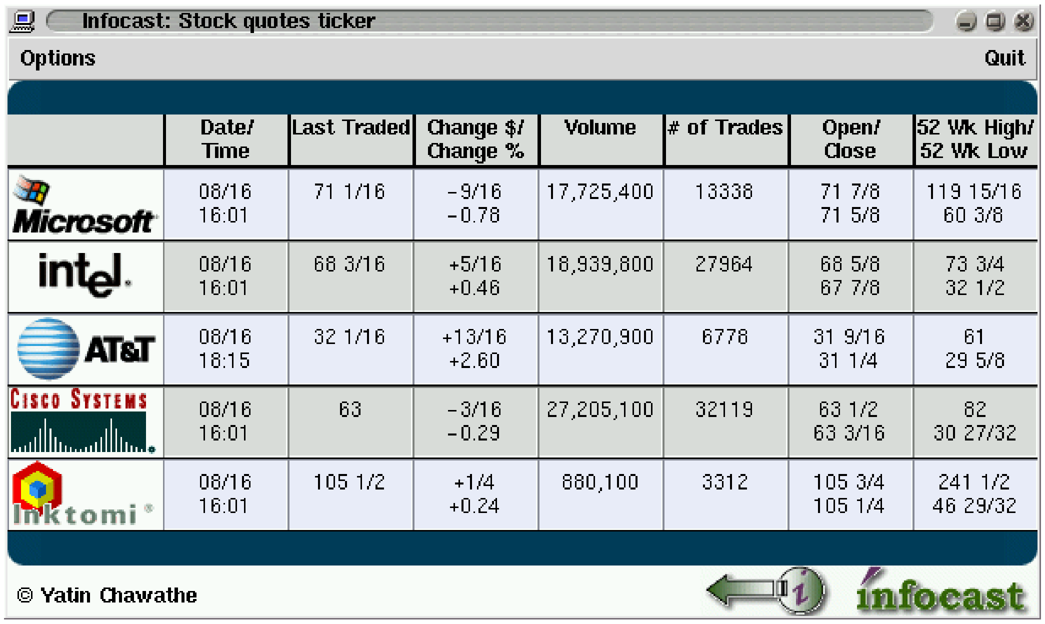
Lastly, the SCX keeps track of the current page that each client is viewing. Rather than blindly forwarding all ADUs generated in the session, the SCX transmits only the data associated with the client's current page. All other data is kept buffered in the pseudo-canvas until the client actually switches to a new page. At that time, the SCX collects all new data on that page, packages it into draw-ops, and sends them to the client.

6.2.6 Implementation Experience

We implemented the mediapad client using a PalmPilot emulation environment called *xcopilot* [31]. This allowed us to debug the entire application using the emulator instead of the actual hardware. Once the software was stable, the emulated application was uploaded to the PalmPilot. The mediapad SCX was implemented on top of the scattercast framework.

Our experience with the mediapad application demonstrates the ability of the scattercast framework to optimize the application for the PalmPilot PDA. Table 6.2 shows the bandwidth savings that are possible with such ALF-based adaptation. As expected, through lossy compression, the SCX reduces the number of bytes that need to be transmitted to the PDA by over a factor of 10 at the PDA's typical zoom level of 33%[1]. Freehand drawings show an interesting tradeoff between bandwidth utilization and interactivity. For maximum interactivity, the desktop mediaboard protocol sends each line segment of a free-hand sketch as a separate packet as soon as it is generated. To avoid this overhead, the

[1]The PalmPilot screen is approximately one-third the size of desktop mediaboards.



	Date/ Time	Last Traded	Change $/ Change %	Volume	# of Trades	Open/ Close	52 Wk High/ 52 Wk Low
Microsoft	08/16 16:01	71 1/16	-9/16 -0.78	17,725,400	13338	71 7/8 71 5/8	119 15/16 60 3/8
intel.	08/16 16:01	68 3/16	+5/16 +0.46	18,939,800	27964	68 5/8 67 7/8	73 3/4 32 1/2
AT&T	08/16 16:15	32 1/16	+13/16 +2.60	13,270,900	6778	31 9/16 31 1/4	61 29 5/8
Cisco Systems	08/16 16:01	63	-3/16 -0.29	27,205,100	32119	63 1/2 63 3/16	82 30 27/32
Inktomi	08/16 16:01	105 1/2	+1/4 +0.24	880,100	3312	105 3/4 105 1/4	241 1/2 46 29/32

Figure 6.3: The Infocast application: A stock ticker example (snapshot from August 2000).

SCX intelligently groups these individual line segments and sends a coalesced draw-op to the PalmPilot. The example in Table 6.2 consists of a total of 131 individual line segments. The SCX can once again achieve savings of up to a factor of 10, albeit at some loss of interactive drawing. A final data point is arrows, where the actual data transmitted to the PalmPilot is more than the original data in the mediaboard command—the SCX computes the arrowheads for the client and sends the coordinates as part of the draw-op. Yet, the packet size is smaller for the PDA protocol simply due to the elimination of costly scattercast headers.

As seen from the above discussion, the mediapad application showcases the ability of scattercast to adapt to extreme device heterogeneity. Even though our implementation focuses on a specific device—the 3COM PalmPilot—the framework is actually general enough to handle different kinds of devices. Only the last stage in the framework is customized for the specific device. In particular, the mediapad SCX can be extended to other devices simply by replacing the device-specific transformation algorithms for images and postscript data with a different transformation engine customized for the new device. When the mediapad client joins the session, it performs an initial handshake with the SCX informing the SCX of its capabilities and adaptation needs. The SCX can use the information from this handshake to determine the kinds of transformations that the client device will require.

6.3 InfoCast: Information Dissemination

Our next example application is a push-based information dissemination application called *InfoCast* [141]. This application periodically transmits updates of a live data feed, such as news headlines, stock quotes, weather updates, etc. to a group of receivers. The InfoCast source constructs a namespace hierarchy that groups related information into containers. For example, a stock quotes service creates a container for each symbol on the stock exchange; the latest stock quotes for each symbol are transmitted as data objects in the corresponding container. Similarly, a news service creates containers for each news section (headlines, world news, business, sports, etc.). Figure 6.3 shows a screenshot of the Infocast application for viewing stock quotes.

One of the earliest examples of an Infocast-like application was *PointCast* [111]. The PointCast design included an independent connection between each client and the PointCast server. Not surprisingly, this proved to be a crucial design flaw in terms of the ability of the PointCast service to scale to large numbers of clients. Although a later version of PointCast introduced the notion of caching proxies between clients and the PointCast server, this solution was specific to the application at hand and could not be generalized to other push or broadcast applications. Unlike PointCast, the scattercast approach provides such applications with a general-purpose infrastructure for scalable broadcasting while at the same time allows applications to embed custom policies within the infrastructure to optimize the broadcast for their needs.

Let us now look at how scattercast can be customized for this application. Like the mediaboard, InfoCast uses rich content such as a combination of images and text. Hence, transformations similar to those used for the mediaboard application can be used in the InfoCast *asmod*. The *asmod* converts images to low resolution versions or to progressive representations. In addition, since InfoCast data is periodic, new data replaces old data. For example, in a stockcast scenario, newer stock quotes obsolete older information, thus obviating the need to recover any missing old quotes. The *asmod* uses this feature of the data to its advantage and suppresses recovery for any missing data that might have been superseded by newer data.

In addition to controlling the loss recovery process, receivers drive the bandwidth management mechanisms by indicating their interest in certain categories of data. For example, in a weather-cast application, receivers in Boston may only care about weather information in Boston and its surroundings, while receivers in San Francisco would be more interested in weather for the San Francisco Bay region. The *asmod*s use this feedback from receivers to filter out unnecessary data objects.

6.4 MusiCast: Radio over the Internet

Our final application deviates from the previous examples in that it is composed of real-time audio content. This application, MusiCast, provides an Internet audio broad-

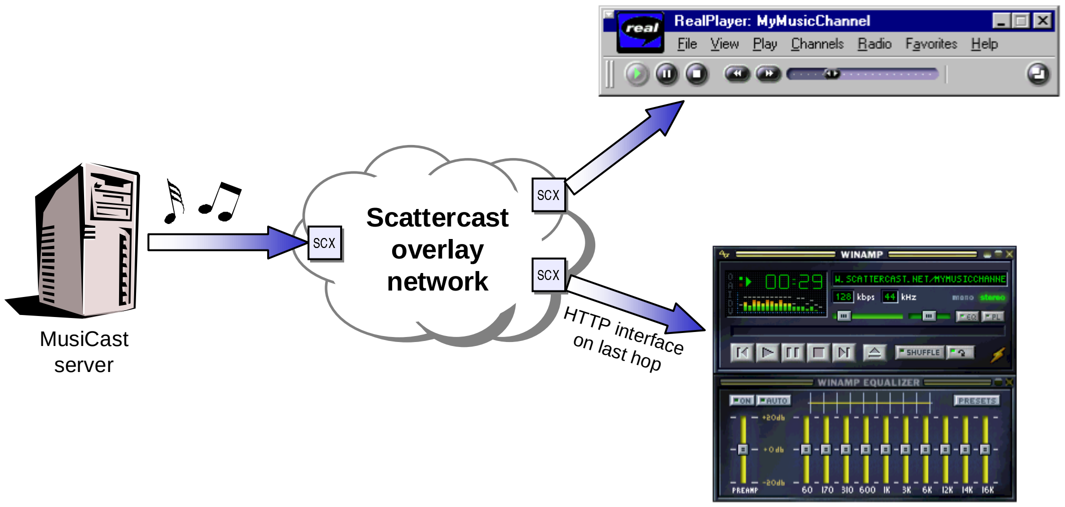

MusiCast server
SCX
Scattercast overlay network
SCX
SCX
HTTP interface on last hop
RealPlayer: MyMusicChannel
File View Play Channels Radio Favorites Help
WINAMP
WINAMP EQUALIZER

Figure 6.4: The MusiCast application: an illustration of the use of the scattercast architecture for online audio dissemination.

casting environment. As depicted in Figure 6.4, a MusiCast server injects audio content into the scattercast network via a local SCX. We use MP3 (MPEG 1 or 2 Layer III Audio) as the underlying audio format. The source broadcasts MP3 frames to the entire session through its SCX. Users wishing to "tune" into the broadcast use standard MP3 clients such as WinAmp [103], RealAudio [118] or mpg123 [97]. Traditional audio broadcasting systems rely either on a simply client-server model (e.g. Internet Radio websites) that cannot scale to large audiences or on ad hoc hand-configured distribution networks such as the Real Network. Unlike these approaches, MusiCast benefits from scattercast's ability to provide a dynamically configurable distribution channel that relies on an intelligent overlay network for efficiently broadcasting the audio content to a wide audience.

For this application, SCXs export an HTTP interface to the MP3 clients. A client tunes to a specific broadcast by sending an HTTP request to its local SCX cluster and including the address of the session announcement for the audio channel that the user wishes to listen to. For example, a request for a San Francisco-based music channel may point to the URL `http://www.musicast.net/SanFrancisco/KZQZ/`. The SCX cluster in turn issues an HTTP redirect to point the client towards the appropriate SCX for that audio channel. Once the client contacts the SCX, it streams the MP3 frames to the clients via an HTTP interface. If the client is capable of using multicast to receive MP3 broadcasts (e.g. WinAmp with a multicast plugin [84]), it may directly communicate with its SCX using a locally scoped multicast channel.

Within the scattercast network, MusiCast SCXs use the Real-time Transport Protocol (RTP) [125] for communication with each other. Based on RTP receiver reports providing reception quality feedback from downstream nodes, it is possible for an SCX to dynamically transcode the audio stream using lower bit-rates if the SCX notices packet loss due to congestion within the network. Our prototype scattercast implementation however does not include this feature.

6.5 Summary

This chapter has demonstrated the flexibility of the scattercast framework to support a wide range of client and application environments. The scattercast transport layer is capable of being customized to accommodate a variety of application data ranging from persistent reliable data for mediaboard applications to audio data with real-time constraints. The ability to separate the transport protocol into a collection of core mechanisms—TINS namespace for reliability, traffic classes for scheduling, and hooks for dynamic transformation—that can be customized via application-specific modules within the SCXs has proven to be an extremely powerful construct in the context of efficient wide-area broadcasting.

We conclude that our judicious use of Application Level Framing within the scattercast protocols has allowed us to build a general-purpose broadcasting framework that is at the same time customizable to the needs of a variety of applications.

Chapter 7

Related Work

"Consider the past and you shall know the future."

–Chinese proverb.

In this chapter, we survey other research that is related to our own. We seek to compare the various other sources of related work with our architecture and discuss how the related work has contributed to our design choices. We point out the limitations of some of the existing approaches to broadcast distribution and describe how scattercast attempts to ameliorate those issues.

The work in this thesis is influenced by research ideas from a number of different related areas. The scattercast architecture represents a synthesis of two independent areas of research that have grown in the past few years. One is the notion of intelligent network infrastructure such as proxies to assist in distributed applications. Proxies have been used in the past to provide value-added services to end-clients and servers. Scattercast builds upon this concept and incorporates explicit application-level intelligence into the network to simplify the broadcasting problem. The second research area that has influenced the scattercast architecture is a relatively new idea, that of using application-level components rather than network-level routers to distribute broadcast content. This notion of performing application-level multicast by building distribution trees using purely application-level entities has generated a lot of interest recently in the research community as well as within the commercial world. Scattercast combines this concept of application-level multicast with an intelligent application-aware proxy infrastructure to provide a customizable framework for efficient broadcasting.

In this chapter, we look at key technologies that have influenced the design of the scattercast architecture and contrast some of these technologies with our design choices for scattercast. We start with a survey of other approaches to embedding intelligent computation within the network. Then we present related work in the context of building overlay networks. We look at a selection of different protocols for performing application-level multicasting. Next, we present a survey of the range of reliable transport protocols designed for multi-point communication and compare how they relate to the scattercast transport framework. Finally we discuss other forms of customizable transport protocols, look at how other related applications adapt to heterogeneity within the network, and survey commercial systems that are addressing similar problems.

7.1 Intelligent Network Infrastructure

The notion of embedding application-aware infrastructure within the network has historically been used to provide various kinds of network and application adaptation. Internet firewalls [28], web proxy caches [67] and WAP (Wireless Access Protocol) servers [138] are a few examples of network infrastructure services that provide additional functionality such as network security, efficient web surfing, and mobile device access to end users and applications.

Numerous network infrastructure services have been proposed in the form of proxies for HTTP. The HTTP proxy mechanism was originally designed for implementing security firewalls. It has since been used in a number of creative applications, including Kanji transcoding [121], Kanji-to-GIF transformation [145], application-level stream transducing [18, 122], and personalized agent services for web browsing [13]. Proxies have been used as caching and pre-fetching agents [16, 99] to hide latencies in fetching data from across the network. In the context of multicast, *rtpgw* [9] is a proxy framework for real-time audio/video data. The InfoPad project [66] used an extreme approach with proxies for limited-capability PDAs: move all intelligence into the infrastructure and use the PDA simply as a dumb terminal.

The infrastructure service concept has been used to hide the effects of error-prone and low-bandwidth wireless links [45, 81]. Bruce Zenel [147] applies the proxy mechanism to the mobile environment: *filters* on an intermediary host drop, delay, or transform data moving between mobile and fixed hosts. However, the filters are part of the application, complicating their reuse and making it awkward to support legacy applications. Balakrishnan et al. proposed the use of a snooping infrastructure agent in their Snoop Protocol [11]. Snoop is a TCP-aware link layer protocol designed to improve the performance of TCP over networks of wired and single-hop wireless links that experience packet losses due to bit-errors. TCP mis-classifies such losses as being due to congestion in the network and results in dropping its transmission window thereby causing degraded throughput. The Snoop protocol works by deploying a Snoop agent at the wireless base station and performing retransmissions of lost segments based on duplicate TCP acknowledgments and locally estimated last-hop round-trip times.

More recently, infrastructure-based architectures have been proposed in the context of the Wireless Access Protocol (WAP) [138], which allows wireless devices such as cell phones and PDAs to connect to the rich content of the world wide web. WAP sends a request from the client device to a special WAP server. Upon interpretation of the address or URL, the content is requested via the Internet and returned to the client in simplified markup language such as Hand-held Device Markup Language (HDML) or Wireless Markup Language (WML).

Researchers have proposed the use of intelligent computing in the network to assist in the design of reliable multicast protocols. Active Reliable Multicast (ARM) [79] uses the concept of "active routers" that can perform customized computation on behalf of the end-points. They provide best-effort soft-state storage and perform data caching for local retransmission, NACK fusion/suppression, and partial multicasting for scoped retransmission.

Scattercast builds upon the lessons learned from the range of intelligent infrastructure services described above to provide an application-aware infrastructure-based solution for Internet broadcasting. Like HTTP proxies, the scattercast infrastructure sits between the source of data and the receivers to intelligently adapt the broadcast distribution to efficiently transmit content across the network. However, unlike traditional applications of proxies where a single proxy (or possibly a small chain of proxies) acts as an intermediary between the client and the server, scattercast extends the proxy concept into a highly configurable distribution network composed of many such proxies collaboratively providing a single function.

An extreme form of embedding application awareness within the network infrastructure is the Active Networks initiative [129]. In an active network, the routers within the network infrastructure themselves perform customized computations on the packets that flow through them. In an active network environment, new protocols and application-specific code can be downloaded dynamically into the routers thus allowing each user or application to customize the behavior of the routing infrastructure for their packets. The ANTS toolkit [139] is a Java-based implementation of the Active Networks idea that uses a "capsule"-based design to embed the customization code within IP packets. However, this model of incorporating application-specific computation into IP routers at a per-packet level introduces a number of difficult problems in terms of resource sharing within routers, downloading application layer code into network layer entities, and router deployment. For a large class of applications, we question the usefulness of this approach of moving application-level complexity into the network layer. There is a lot of merit in the original Internet design of leaving the core network layer technology simple, robust, and easy to understand, and migrating complex services to higher layers instead. As was originally proposed by the Active Services framework [8], the scattercast infrastructure instead relies on a programmable network infrastructure that restricts the programmability to application-level services instead of routing entities. In doing so, scattercast inherits the benefits of programmability without the complexities associated with Active Networks.

7.2 Application Level Multicasting and Overlay Networks

In the past eighteen months, the idea of providing an Internet broadcasting solution via the use of application-level routing components rather than tradition network-layer multicast routers has generated a lot of interest both within the research community and in the commercial world. A number of researchers as well as startup ventures have proposed a range of application-level multicast schemes which effectively build an overlay network on top of the traditional IP Internet that is used to efficiently distribute broadcast content. Scattercast couples this notion of application-level multicast with an intelligent and application-aware network infrastructure to provide a flexible and customizable solution for efficient Internet broadcasting. In this section, we look at a variety of research proposals for application-level multicast. We leave the discussion of commercial ventures that are tackling this problem to Section 7.3.

The End System Multicast [65] and YOID (Your Own Internet Distribution)[1] [49] research projects have proposed similar multi-point data distribution frameworks that build distribution trees purely on an end-host basis. Like scattercast, both End System Multicast and Yoid address the ineffectiveness of IP multicast for content distribution. They rely on self-organizing protocols for constructing distribution trees out of unicast tunnels across end-hosts participating in the multicast session. Like scattercast, End System Multicast builds a mesh structure across participating end-hosts and then constructs source-rooted trees by running a routing protocol. On the other hand, Yoid directly builds a spanning tree structure across the end-hosts without any intermediate mesh structure. Although this approach avoids the redundant edges that a mesh structure incurs, it requires expensive loop detection and avoidance mechanisms, and is also extremely fragile and susceptible to partitions.

The main difference between our approach and that of Yoid and End System Multicast is the explicit use of infrastructure service agents—SCXs—in our architecture. Although it is possible to incorporate proxies into End System Multicast and Yoid, SCXs are an integral aspect of the scattercast architecture. We believe that for such a framework to scale well beyond a few hundred clients, infrastructure support will be absolutely crucial. A fully decentralized end-host-only self-organization protocol will not scale beyond a few hundred or a few thousand participants. On the other hand, since scattercast proxies can simultaneously serve many clients, we believe that one or a small number of proxies per ISP will be sufficient to serve a large client population.

The Banana Tree Protocol (BTP) [62] is another end-host-based multicast protocol that constructs a virtual network across the hosts participating in a multicast session and routes multicast data through the graph themselves without router cooperation. Similar to the Yoid framework, BTP is based on a tree topology. Each host in the session is a node in the tree. The host that creates the tree is the root node. BTP is designed mostly for

[1]Formerly known as *yallcast*.

one-to-many file transfer applications. Unlike scattercast, it does not account for network and client heterogeneity, and does not have any mechanisms to adapt the transport protocol to changing conditions.

Overcast [74] is yet another application-level multicasting system that has been proposed recently. Overcast builds data distribution trees out of a collection of nodes placed at strategic locations within the network fabric. Distribution trees are constructed to provide bandwidth efficiency and are restricted to single-source multicast. One of the main applications for overcast appears to be high-bandwidth on-demand video services. To support such applications, overcast nodes explicitly provide disk storage capabilities in addition to application-level multicasting for the data distribution services.

The scattercast architecture itself germinated from prior work by Ratnasamy et al. [116]. They defined a *delivery-based model* for reliable multicast communication where receivers organize themselves into a multilevel hierarchy of disjoint multicast delivery groups. Data transmission is achieved by unicasting data between group representatives which in turn multicast data to their delivery group. Such a delivery model enables the data delivery process to be tailored to match the homogeneous network characteristics within individual delivery groups. In [117], Ratnasamy et al. use a multicast-tree-inference algorithm to build a protocol building block—a distributed Group Formation Protocol (GFP)—that allows receivers to self-organize into a source-rooted hierarchy of disjoint multicast groups where the hierarchy is congruent with the native multicast tree topology. However, this protocol relies on the existence of a global multicast control channel, which scattercast explicitly intends to avoid.

A lot of work has been done recently to build localized clusters of multicast receivers and construct overlay structures on top of them by dynamically discovering the multicast topology. Levine et al. [80] proposed the use of IGMP MTRACE packets to allow receivers to obtain their path to the source of a multicast group; receivers use the multicast path information to determine how to achieve local error recovery and effective congestion control. Self-organizing Transcoders (SOT) [77] are a scheme for dynamic adaptation of continuous-media applications to varying network conditions by allowing groups of co-located receivers that experience losses due to a bottleneck link to elect a representative transcoder for local repair.

The Adhoc Multicast Routing Protocol, AMRoute [83], is an approach for multicast in mobile adhoc networks that creates bidirectional shared trees for data distribution using only group senders and receivers as tree nodes. Unicast tunnels are used as tree links to connect neighbors on the user multicast tree. Thus, AMRoute does not need to be supported by network nodes that are not interested in or capable of multicast and group state cost is incurred only by group senders and receivers. However, unlike scattercast, AMRoute does not attempt to optimize the distribution tree in any form. Scattercast, on the other hand, explicitly relies on Gossamer to build an efficient overlay network for data transmission.

In [14], Bauer et al. compare a number of heuristics to find efficient degree-restricted multicast trees in the presence of constraints on the copying ability of the individual switch nodes in the network. Although some of these heuristics may be applied to construct distribution trees in scattercast, we believe that the mesh-first approach used by Gossamer is superior to building spanning trees directly.

In addition to the above approaches for efficient broadcast distribution, the concept of overlay networks has been applied to a number of other scenarios as well. The multicast backbone or MBone [40] itself is an overlay network composed of clouds of native IP multicast connectivity interconnected by unicast tunnels. Scattercast too provides similar functionality by interconnecting locally scoped IP multicast groups via an overlay network of SCXs. Unlike the MBone, however, scattercast explicitly incorporates application-level intelligence into the design of its forwarding and transport algorithms. Moreover, the Gossamer algorithms used by scattercast build a self-configurable dynamic overlay structure as opposed to the static hand-configured structure of the MBone.

The X-Bone [130] is a system for rapid, automated deployment and management of overlay networks to support many different network services. By providing a coordinated framework for deploying and managing IP-level overlays, the X-Bone simplifies resource management, including address allocation, bandwidth utilization, and router capacity, and allows rapid, user-level deployment of overlays. Like the MBone and unlike scattercast, X-Bone produces statically configured overlays.

Systems such as Napster [98] and Gnutella [52] allow users to download files from each other in a peer-to-peer fashion over application-driven overlay networks. Napster relies on a single centralized server to search for specific documents and to locate peers that host those documents. Gnutella, on the other hand, uses a fully-decentralized approach that floods search messages across its entire distribution network to locate peers that can satisfy the search request. Clearly this model is insufficient to scale beyond a small number of participants. A richer more structured overlay such as scattercast's may be able to provide efficient search propagation to the Gnutella network.

7.3 Commercial Development

In addition to research interest, the commercial world too has demonstrates a great deal of momentum in developing products based on the concept of application-level multicast. Companies such as FastForward Networks [41] (recently acquired as the Inktomi Media Division), SightPath (now part of Cisco) [29], Akamai [3], and Digital Island [37] have developed or are in the process of developing proprietary systems for streaming content distribution on a large scale. Unfortunately, the architectures of most of these systems are not public knowledge, and hence evaluating their mechanisms in comparison to scattercast is difficult.

FastForward Networks is one of the few commercial ventures that has a publicly available white-paper detailing their Broadcast Overlay Architecture (BOA) [41].[2] Although the FastForward BOA and scattercast were designed and implemented independently, there are a number of similarities and cross-influences between the two architectures. Discussions with researchers at FastForward Networks influenced some of the architectural decisions in scattercast. Yet, the scattercast and FastForward architectures differ in two fundamental ways. Like scattercast, the FastForward BOA constructs a media distribution network (MDN), an overlay network that is composed of MediaBridges (a.k.a. SCXs). The MDN is customizable to individual application environments via the MediaBridge Adapter (a.k.a. the SCX *asmod*). However, unlike scattercast, the MDN is constructed out of a statically configured distribution topology. Within the static overlay structure, the MDN can dynamically route packets over the best possible data paths. The FastForward architecture makes this explicit choice in order to allow expert network administrators to hand-configure the overlay topology for efficient performance. Scattercast's Gossamer protocol, on the other hand, allows SCXs to dynamically discover the best possible topology and to adapt the topology to changing network conditions. This dynamism allows scattercast to optimize the overlay structure without excessive human interaction.

A second key difference between the scattercast and FastForward architectures is the extent of explicit application-aware customization that is possible with the scattercast transport framework. The FastForward BOA transport is mostly limited to real-time streaming applications with limited support for on-the-fly transcoding of audio and video streams to adjust to available bandwidth. Scattercast, on the other hand, provides a general-purpose framework that applications can use to optimize the transport protocol for their own needs ranging from real-time best-effort requirements to applications that require total reliability.

7.4 Multi-point Reliable Transport

Scattercast employs a customizable transport framework that applications can use to provide various flavors of reliable distribution to their users. In this section, we present a survey of the various forms of multi-point reliable transport protocols that have influenced the scattercast transport framework. Most of these protocols were developed in the context of providing reliability for IP multicast applications, but the research ideas embodied in the protocols are applicable to the scattercast environment as well.

The Scalable Reliable Multicast (SRM) protocol [43] uses the concepts of negative acknowledgments, multicast damping, and shared loss recovery to avoid the typical problems such as traffic implosion associated with reliable multicast protocols. Receivers in SRM multicast loss reports to the entire session and any member of the session can respond with a retransmission of the lost data. The retransmitted data is multicast as well to allow

[2] In addition, the Cisco SightPath architecture is known to be based on the Overcast system [74] described in Section 7.2.

other receivers that might have lost the same data to recover it at the same time. This form of global recovery is non-scalable and there have been proposals for localizing the recovery process to only a small group of receivers. Like SRM, scattercast uses negative acknowledgments or loss reports to recover lost data. However, instead of broadcasting the lost report to the entire session, scattercast propagates the report along a structured path upstream from the receiver towards the source.

The Reliable Multicast Transport Protocol (RMTP) [82] organizes group members into a hierarchical tree structure for aggregating acknowledgments at midpoints in the network. Each branch in the tree has a designated receiver (DR) to receive acknowledgments from its children and aggregate them upwards to the sender. The scattercast architecture is similar to RMTP in that it groups clients around an SCX just as RMTP groups receivers around DRs. But RMTP uses its tree structure only for acknowledgments and recovery of lost data; all initial data transmission happens over a global multicast group. Scattercast, on the other hand, relies on tunneled distribution trees for data transmission as well.

Pragmatic General Multicast (PGM) [127] is a reliable multicast transport protocol for applications that require ordered or unordered, duplicate-free, multicast data delivery from multiple sources to multiple receivers. Rather than guarantee acknowledged delivery to a known group of recipients, PGM simply provides reliable multicast data delivery within a transmit window advanced by a source according to a purely local strategy. PGM guarantees that a receiver in the group either receives all data packets from transmissions and repairs, or is able to detect unrecoverable data packet loss. Reliable delivery is provided within a source's transmit window from the time a receiver joins the group until it departs. PGM is thus best suited to those applications in which members may join and leave at any time, and that are either insensitive to unrecoverable data packet loss or are prepared to resort to application recovery in the event.

The Breadcrumb Forwarding Service (BCFS) [144] is a generalization of the network-layer components of PGM into an explicit-source-group based forwarding service. Instead of introducing transport-aware mechanisms into the network layer as suggested by PGM, BCFS provides the essence of these mechanisms in the form of a generic forwarding service. The BCFS service model provides a single-source request-based multicast service where clients send an explicit request for data upstream towards a source and sources respond to these requests. Routers along the path towards the source aggregate incoming requests for the same data and provide efficient transmission and replication of the responses from the source. Figure 7.1 depicts an example of this service model. Scattercast bases its end-to-end reliability mechanisms on the BCFS approach by applying these techniques at the application layer across SCXs. This is evident by comparing the BCFS service model with scattercast's end-to-end loss recovery scheme shown in Figure 5.7.

Finally, Xu et al. propose a resilient multicast delivery model [143] in which each receiver in a multicast session is allowed to decide its own tradeoff between reliability and real-time requirements. They have developed a protocol called STORM (STructure-

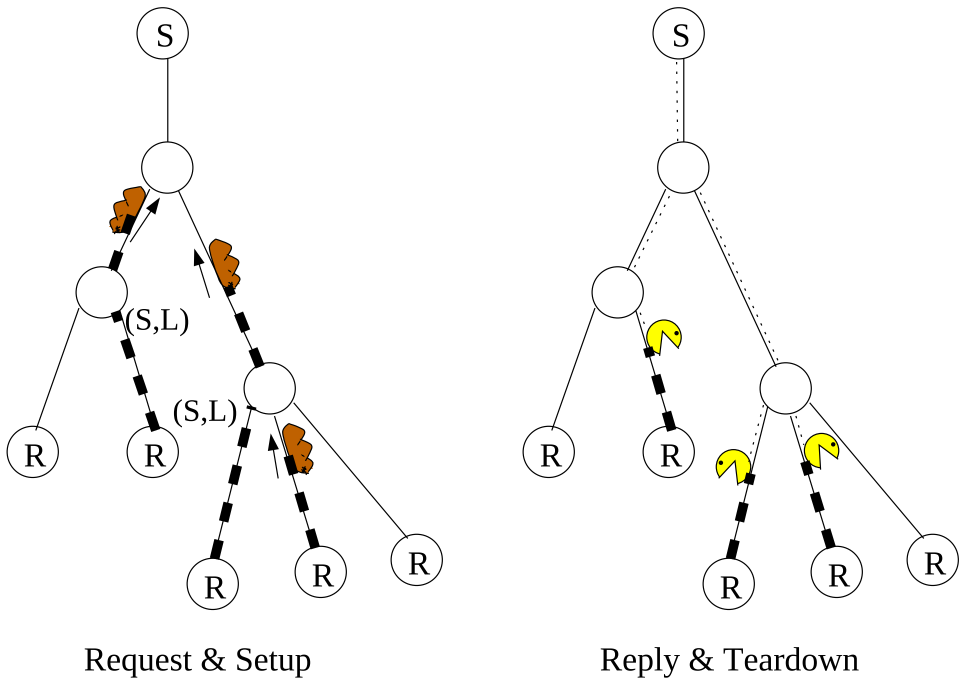


Figure 7.1: The BCFS Service Model. Reproduced from [144]

Oriented Resilient Multicast) in which senders and receivers collaborate to recover lost packets by self organizing themselves into a distribution structure for the recovery phase. STORM is specialized to adapt to efficient delivery for continuous media applications by trading off between the desired playback quality and the desired degree of interactivity. Scattercast, on the other hand, provides a more generic transport framework where application-level intelligence is embedded within the SCXs to tradeoff between not only varying degrees of reliability but also different forms of bandwidth management, congestion control, and application-specific adaptation.

7.5 Customizable Transport

Our experience with the Scalable Reliable Multicast (SRM) toolkit [113] was one of the earliest efforts towards building a generic transport framework that is customizable for various forms of applications. The SRM toolkit provides a flexible reliability framework in the context of IP multicast in which applications can customize the extent and the form of reliability that they desire. Early versions of the scattercast transport extended the SRM toolkit to the scattercast environment.

The Image Transport Protocol (ITP) [112] is a customized protocol for image transmission on the web over loss-prone congested or wireless networks. ITP attempts to improve user-perceived latency using application level framing and out-of-order data delivery thereby achieving significantly better interactive performance. Another attempt to improve web performance via a receiver-driven transport protocol is WebTP [55]. WebTP is designed to be completely receiver-based in terms of transport initiation, flow- and congestion-control. It optimizes web transport by using the concepts of application level framing to create a request-response protocol that is customized to include the user into the transfer loop.

Finally, the Congestion Manager (CM) architecture [10] proposes a novel framework for managing network congestion from an end-to-end perspective using application-specific information. The architecture centers around a per-host Congestion Manager that ensures proper congestion behavior and allows applications to easily adapt to network congestion. The CM maintains congestion parameters and exposes an API to enable applications to learn about network characteristics, pass information to the CM, and schedule data transmissions. Applications can customize how they react to congestion using this API. Internally, the CM uses a window-based control algorithm, a scheduler to regulate transmissions, and a lightweight protocol to elicit feedback from receivers.

7.6 Adapting to Heterogeneity

The scattercast framework uses on-the-fly adaptation of data to adjust the transport protocol to the vagaries of Internet heterogeneity. SCXs act as application-level proxies that can potentially transform data dynamically before forwarding it to downstream receivers. This notion of transcoding proxies is not new or unique to scattercast. It has been used in the past by the Transend system [47, 48] to provide web acceleration services for browsing the world wide web. The MeGa system [5] provides transcoding services for real-time audio/video data to deal with heterogeneity in multicast environments.

Content layering is another scheme that has been widely used to tackle heterogeneity in multi-point delivery environments [126, 20, 90]. Layering typically involves encoding the source data into multiple layers; the base layer provides an approximate representation of the original data, and each additional layer provides more information about the original data.

Rizzo et al. [119] describe a Reliable Multicast data Distribution Protocol (RMDP) that relies on Forward Error Correction techniques to adapt to client and network heterogeneity. Digital Fountain [19] is a scheme that encodes data in a form that allows it to be reconstructed from any subset of the encoding packets that is equal in length to the source data. In combination with a clever layering strategy, this scheme can service a heterogeneous set of receivers, where each participant receives the encoded fountain at whatever rate best suits its network capacity.

Most of this work is complementary to the scattercast architecture and the layering and FEC techniques described in the works above can be incorporated on a per-application basis into the scattercast transport framework.

Another avenue of dealing with extreme heterogeneity that is used by scattercast is that of partitioning application complexity between the client and infrastructure. A related project, TopGun Wingman [44], uses similar infrastructure support to enable a simplified web-browser on the PalmPilot. In [133], the authors have proposed the use of a simplified document format (HDML) to reduce the complexity of PDA application. The Rover system [75] provides a distributed object model that presents a queued RPC mechanism for disconnected operation and object migration. For example, simple UI code can be migrated to a mobile client, where it uses queued RPC to communicate with the rest of the application running on the server.

7.7 Summary of Related Work

In this chapter, we surveyed related research in a number of different fields. The scattercast architecture represents a synthesis of two independent research ideas: the use of intelligent network infrastructure to provide application awareness within the network and the concept of application-level multicast that moves multi-point distribution away from network layer routers into application-level entities either at the end-points or at strategic locations within the network.

The notion of infrastructure services has been used in a number of different systems to assist in application protocols. However, most such systems are targeted for specific applications and are not generalizable beyond those applications. The scattercast architecture, on the other hand, is a general-purpose framework for Internet broadcasting, yet it is specifically designed to be customizable by individual applications and end-user environments.

A number of different research projects have recently proposed the use of application-level components to provide multi-point communication without the use of router support. Most of these proposals deal with end-host-based protocols unlike scattercast which explicitly involves infrastructure support. We believe that for a real broadcasting infrastructure to take off, pure end-host based systems will not be sufficient to scale to large audiences. That said, it remains to be seen how these end-host protocols would compare to scattercast's own application-level multicasting protocol, Gossamer.

A number of the customizable transport ideas for scattercast are influenced by previous work in similar fields, particularly the SRM toolkit for reliable multicast communication. Similarly, scattercast adapts the BCFS work on a multi-point forwarding service for reliable protocols to build its own end-to-end reliability mechanisms. Finally, scattercast builds upon lessons learned from other dynamic adaptation schemes in the context of web and real-time applications to adapt to network and device heterogeneity.

The commercial world has recently demonstrated a great deal of interest in scattercast-like architectures for broadcast communication. Most of these companies however are building proprietary closed infrastructures over which they allow content providers to efficiently distribute information to their clients. In the spirit of the global shared Internet, we believe that such a crucial infrastructure service should have an open architecture that a number of different service providers can plug into and cooperate with each other. Scattercast is an attempt in that direction.

Chapter 8

Conclusions and Future Directions

In this chapter, we conclude the dissertation by revisiting the lessons learned from this work and presenting directions for future research in this area.

8.1 Conclusions

In this dissertation, we have presented an architecture for Internet broadcast distribution that relies on application-level intelligence embedded within the network infrastructure rather than on network-layer multicast primitives to provide efficient multi-point data distribution. Our architecture, which we call scattercast, makes use of a collection of intelligent network agents (ScatterCast proXies or SCXs) that collaboratively provide the multicast service for a session. SCXs organize themselves into an overlay network of unicast interconnections to provide an application-level distribution service as opposed to the traditional network-layer distribution model. Clients participate in the session via a nearby SCX by using either locally scoped IP multicast groups or direct unicast connections to the local SCX. SCXs are typically hosted at ISP points-of-presence. To provide a robust, scalable and available service, the scattercast components are materialized out of clusters of machines. The clusters provide the properties of scalability, load balancing and fault tolerance to the scattercast service. In addition, robustly designed protocols on top the clustered implementation provide resilience to network failures and partitions.

The scattercast architecture builds upon three key mechanisms to alleviate the problem of wide-area broadcast distribution:

1. An infrastructure-service-oriented approach that moves complexity out of the network layer into application-level service components embedded within the network infrastructure.

2. An application-level multicast protocol that builds a distribution tree out of unicast interconnections across SCXs rather than relying on IP level multicast support.

3. A flexible transport framework that incorporates application semantics to optimize the transport protocol.

By migrating the multi-point delivery functionality out of the network layer to a higher infrastructure service layer, scattercast maintains the simplicity of the underlying network model. Moreover, scattercast simplifies the design of complex reliability and congestion control protocols by localizing the use of IP multicast and by allowing for application-specific adaptation to deal with the heterogeneity that typically cripples traditional multicast protocols.

Scattercast relies on a protocol called Gossamer to build an efficient overlay structure and provide application-level multicast distribution. Gossamer constructs an overlay network composed of unicast connections across SCXs through a three-step process: gossip-style discovery to locate SCXs participating in a given session, a randomized mesh construction process with local optimization to build a strongly connected graph structure, and a final routing step that runs a distance-vector routing protocol to build source-rooted reverse-shortest-path distribution trees across the network. Our simulation results for the Gossamer protocol show that the latencies incurred by transmitting data using the overlay mesh are typically within 1.6 to 1.9 times those associated with directly multicasting or unicasting the data from the source to the various destinations. At the same time, the mesh generated by Gossamer substantially limits the bandwidth usage of the physical Internet links in comparison to naïve n-way unicast.

SCXs also act as application-aware intermediaries between the source and the consumers of data to mitigate network heterogeneity by dynamically adapting the content and/or the rate of the data to best suit the clients' needs and interests. Scattercast builds a flexible application-aware transport framework on top of the Gossamer distribution layer. It incorporates the principles of Application Level Framing (ALF) to adapt to network and client heterogeneity and to allow applications to determine their own semantics for the transport protocol. The scattercast transport framework differs from traditional transport protocols in two key ways. First, unlike traditional transport protocols such as TCP, which provide a single well-defined but inflexible set of semantics, scattercast purposely stops short of defining the exact semantics of the transport protocols and instead only provides a core set of mechanisms that applications can use to customize the transport to their own needs. Second, it explicitly involves the distribution infrastructure in the transport protocol by exposing the topology of the overlay distribution tree to the transport protocol. By moving transport complexity away from end clients into infrastructure SCXs and by requiring SCXs to explicitly participate in the transport protocol, scattercast effectively simplifies the implementation of the transport protocol at the clients. Moreover, by incorporating application-specific adaptation into the SCXs, scattercast addresses the heterogeneity that arises across the entire session.

The flexibility of the scattercast transport framework is demonstrated by the range of applications that we have developed on top of this architecture. Scattercast applications can range from electronic whiteboards which require persistent reliable state to audio broadcasting where the data is transient in nature but has real-time constraints. The transport framework flexibly accommodates this variety of requirements by allowing applications to embed their own semantics into the transport library via a well-defined general purpose API. Our suite of applications demonstrates the programmability of this API and the ability of the architecture to adapt to different flavors of applications and client environments.

8.2 Lessons Learned

Our experience with the scattercast architecture has driven home (or reinforced) a number of key lessons:

- Maintaining the simplicity of the network layer is key to sustaining a robust distribution architecture.
- Using IP multicast only in the local area where it is both feasible and efficient is an important mechanism that allows scattercast to eliminate or mitigate many of the traditional problems with network layer multicast.
- Migrating application-specific intelligence into the network protocols is crucial to the design of an adaptable Internet system.
- No system can be designed perfectly in one shot.

We discuss each of these lessons in turn.

8.2.1 Keep the network simple

Maintaining the simplicity of the IP network architecture has obvious appeal: a simple suite of network protocols is easy to implement and reason about, and ensures the robustness of the system. This principle was apparent in the original design of the Internet architecture where the routing infrastructure at the network layer provided a basic packet forwarding functionality and end clients provided all other richer end-to-end transport functionality.

Scattercast strives to retain this simple and robust nature of the Internet by advocating that the network layer is not the appropriate place to implement global multi-point distribution. We instead simplify the network layer problem by moving complexity into a new infrastructure service layer composed of scattercast proxies that are spread across the Internet. This infrastructure service layer simply assumes a point-to-point service model at the network layer that may be potentially enhanced by locally scoped multicast. Network layer multicast that is restricted to a well-defined local scope is much simpler to implement without complicating the routers. Thus rather than assume the existence of a global IP multicast “dial-tone” for large-scale broadcasting, scattercast leverages multicast as an efficient protocol building block wherever it is available without requiring multicast support to be a necessary component. For efficient wide-area communication, it builds upon the robust and congestion-friendly behavior of wide-area unicast protocols such as TCP.

A commonly heard argument against scattercast-style infrastructure service approaches is that they introduce additional complexity and points of failure into the network. But, we argue that by delegating multicast functionality away from the network layer into such infrastructure services, we actually end up simplifying the network layer and reducing complexity within the network routers. This complexity is instead shifted into application-level entities within the infrastructure service where it can be more easily managed. In particular, any state management and scalability concerns can be solved by distributing the service implementation across a cluster of machines. Clustered implementation also ensures the fault tolerance and availability of the service thus mitigating concerns regarding service failure. A well-administered service cluster can be robust enough to provide 24×7 operation.

8.2.2 Limit IP multicast to domains where it is feasible and efficient

Rather than provide a single global network layer multicast service, scattercast partitions the large wide-area heterogeneous session into many smaller and simpler homogeneous data groups, each serviced by a local scattercast proxy. This divide-and-conquer approach effectively decouples each data group into independent locally scoped IP multicast channels thereby shielding them from the vagaries associated with the rest of the session participants. It thus localizes the hard multicast problems of scalable loss recovery, congestion control and bandwidth allocation.

In keeping with the Internet’s philosophy of layering transport functionality on top of a basic packet distribution framework, scattercast isolates the multi-point distribution service within a well-defined application-level multicast protocol, Gossamer. On top of Gossamer, it layers an application-aware transport framework that relies on hints from the underlying distribution protocol as well as from the application to direct the multi-point transport decisions.

8.2.3 Application-level intelligence is crucial

The scattercast architecture seamlessly weaves in application semantics into its transport framework. The transport protocols are designed to expose a structured yet flexible API to the application, which can then control transport-level mechanisms such as reliability, bandwidth management, and adaptive data forwarding.

Although it is straightforward to conceptualize a set of transport protocols that can be customized by individual applications, realizing such protocols in a general-purpose framework without requiring each application to rewrite large parts of the protocols is difficult. The two key building blocks that scattercast relies on towards a flexible transport framework are (1) the structured naming system, which allows applications to reflect data semantics into the transport layer, and (2) the interaction between the underlying Gossamer layer and the transport framework, which exposes the local Gossamer topology to the transport-level services.

This notion of adaptive applications and protocols that are customizable to a wide range of requirements will prove to be a valuable architectural component in building new infrastructures and applications that can adapt gracefully to accommodate the extent of heterogeneity across the Internet. We described in Section 7.5 some recent progress in other application environments that embed application-defined semantics into the underlying protocol architectures.

8.2.4 Plan on throwing away at least one implementation

We went through several iterations of designing and implementing various components of the scattercast architecture. Each iteration step was a learning process that contributed to the final design of the architecture. For example, an early version of the system was tailored for a very specific application—the PalmPilot MediaPad. That prototype allowed us to explore the design space for building a distribution framework for electronic whiteboards. The lessons that we learned from the prototype were extremely valuable in generalizing the distribution framework into the scattercast architecture described in this dissertation. In particular, our experience with building custom modules for the PalmPilot application gave us an insight into the set of mechanisms that would be required to provide flexible reliability, adaptation and scheduling constraints, and led to the design of the scattercast transport API described in Section 5.6.

A good systems research project has to undergo multiple iterations before it is can mature to a well-understood and robust system. By throwing away our initial implementations and starting over but remembering the lessons from those early prototypes, we were able to develop the scattercast architecture into a more robust and well-designed system.

8.3 Directions for Future Research

The work in this dissertation has opened up some interesting avenues for future research. Some of these ideas directly extend the scattercast architecture while others touch upon the application of our work to other research problems.

8.3.1 Multi-level scattercast

The scattercast architecture as described in this dissertation constructs a flat single-level overlay network structure. As discussed in Section 4.3, this has clear scalability implications. As the number of SCXs within a session increases, the time for a session to stabilize to a quiescent overlay structure becomes longer and longer. A successfully deployed scattercast infrastructure can be composed of a large number of SCXs. With over 7000 ISPs world-wide, the number of SCXs will be on the order of at least a few thousand nodes. To be able to support such a large globally deployed infrastructure, we expect the scattercast overlay structure to evolve into a multi-level hierarchy composed of a core distribution network within the Internet backbone that peers with smaller localized distribution networks at the edges of the backbone. This evolution introduces a number of interesting questions in terms of how the peering relationships across these various scattercast overlays work and how data flows from overlay to overlay across the entire Internet.

At the same time, this evolution of scattercast also provides an incremental expansion path for the infrastructure. Rather than requiring all ISPs around the world to start supporting the scattercast architecture, we envisage a deployment scenario where the scattercast infrastructure is initially installed in the core of the network, and gradually pushed towards the edges as the system becomes popular and begins to scale.

8.3.2 Source-rooted Trees vs. Shared Trees

The Gossamer distribution protocol constructs source-rooted trees à la DVMRP [32]. Although source-rooted trees result in the most optimal distribution paths on top of the overlay network, with many sources within a session, we end up constructing multiple trees. A single shared tree may be a solution if an application wishes to avoid the overhead associated with multiple source-based trees. Although it would be fairly straightforward to extend the Gossamer routing protocols to build a single shared tree akin to PIM [33] or CBT [12] in the IP multicast world, such an extension begs the questions "where is the core of the shared tree located, how is the core advertised, who determines the location of the core, and so on." One potential solution is to allow the session creator to determine whether or not to use shared distribution trees, and to announce the location of the session core in the context of shared trees as part of the session announcement. The session announcement document described in Figure 3.4 could include a section similar to the <RENDEZVOUS> section to announce the session core.

8.3.3 Infrastructure Administration

An important aspect of an actively deployed scattercast architecture is the administration and monitoring of the various infrastructure components associated with the architecture. There are two levels of administration required for this distributed system: local administration to monitor SCXs running within a single cluster, and distributed administration to manage interaction across SCX clusters. The local administrator must take care of issues such as availability of the SCX cluster, monitoring for excessive load, and managing the day-to-day operation of the service cluster. The distributed administrator deals with the interaction across service clusters. Although scattercast's distribution protocols are engineered to route around failures and deal with faulty or run-away processes, automated failure management cannot entirely replace a trained human administrator who can take emergency measures to overcome unforeseeable catastrophic problems that may sometimes afflict the scattercast infrastructure.

The scattercast infrastructure should include enough monitoring and management hooks to allow a human administrator to "poke around" a running installation and detect faults or debug problems that may occur during the execution cycle of the infrastructure. In the IP world, protocols such as SNMP (Simple Network Management Protocol) and tools such as *ping* and *traceroute* allow network administrators to manage and debug their IP networks. Correspondingly, scattercast should include a suite of infrastructure monitoring applications and protocols that make it easy for a system administrator to investigate problems associated with the installation and to rectify those problems with help from the infrastructure.

8.3.4 An Improved Application-level Multicast Protocol

In this dissertation, we presented our protocol, Gossamer, for Application Level Multicasting. Gossamer is one of a small number of research protocols that have recently been proposed for constructing overlay networks for multi-point distribution. We have barely begun to scratch the surface of this rich new problem domain and have simply opened the door for new research to investigate in depth the various problems associated with application level multicasting such as node discovery, overlay optimization, and tree formation. We expect to see a lot of research and commercial ideas develop in this field over the next few years that will build upon our initial research ideas. Moreover, although we have experimented with our protocol using an extensive set of simulations, it would be instructional to evaluate the behavior of the protocol via detailed analysis of the node discovery and mesh optimization algorithms.

8.3.5 Long-term Research Plan: A Pervasive Application-aware Internet Middleware

The scattercast architecture represents a new direction in networking research that explicitly introduces application-level infrastructure components into the network architecture to assist in data distribution. As the Internet grows, we expect more and more applications that will have to tackle issues of scale and network heterogeneity. To address these issues via a general-purpose infrastructure, we envisage a new model for Internet applications that, like scattercast, will rely not just on a simple IP dial-tone service from the network, but will actually demand more complex and application-aware computation from services embedded within the network. The scattercast architecture is one instantiation of such a rich application-aware network infrastructure. In the long term, we would like to generalize this vision of intelligent application components embedded within the network into a

Pervasive Application-aware Internet Middleware that, instead of coexisting with IP at the network layer, is a middleware layer that sits on top of IP and provides application-specific intelligent computation within the network. Such a middleware would consist of computing clusters that are located not within the fast path of IP routers (as in Active Networks), but at IP end-points that are potentially co-located with or very close to existing IP routers. A pervasive middleware architecture would enable a new class of applications that instead of relying simply on a very basic IP dial-tone service from the network, would leverage more complex and application-aware services from a pervasive middleware built on top of the IP Internet.

Just as the network-layer Internet architecture provides a well-defined structure for IP routing and for peering of IP networks, so also this new pervasive middleware layer will require an infrastructure architecture that imposes structure on the peering model for the pervasive middleware clusters, the services that such a middleware offers, and the interaction between the various middleware components. A lot of research effort in the past has focused on Internet service platforms, their programming model, and their scalability and availability strategies. However, our vision for a pervasive Internet middleware moves this to a new level where instead of an ad-hoc collection of service components distributed across the Internet, we will have a well-structured pervasive protocol architecture that takes into account both the needs of applications and the effect of applications and services on the underlying network performance. Such a protocol architecture should leverage lessons learned from the design of the network-layer architecture to apply them to this higher middleware layer. This dissertation is an initial foray into such a rich architecture. Although it primarily focuses on a specific middleware service for broadcast distribution, it provides an insight into a number of issues such as cluster peering relationships, service location, and adaptive interaction across clusters that the middleware protocol architecture will have to address. Understanding the architectural and network protocol issues that underlie the middleware model will be key to the successful transition from the simplistic IP connectivity model of the Internet today to a richer, more structured, application-aware middleware-based Internet.

8.4 Summary

In summary, the contributions of this thesis can be viewed at two levels. At one level, this thesis presents a comprehensive architecture for broadcast distribution over the Internet. It describes a set of protocols for building an overlay distribution network for high-bandwidth broadcast data and a framework for enabling application customization of this overlay network. We presented a novel approach for Internet content distribution that leverages support for application-level infrastructure agents rather than embedding the broadcast distribution protocol entirely within the network layer. We described a set of real applications that can adapt to a range of heterogeneous environments by involving the scattercast infrastructure in a dynamic adaptation framework that relies upon application-specific semantics to drive the adaptation process.

At a higher level, the scattercast architecture is a first step towards a new approach for Internet applications that explicitly moves application intelligence into the network infrastructure, while at the same time maintaining compatibility with the existing IP architecture. We believe that as the Internet evolves, architectures similar to scattercast based on intelligent application-aware network components will become increasingly prevalent. Our experience with scattercast can provide valuable input for the design of such next-generation Internet architectures.

Appendix A

Hierarchical Start-time Fair Queuing (H-SFQ) Algorithm

We present a brief summary of the Hierarchical Start-time Fair Queuing algorithm. This algorithm is used by the traffic scheduler in the scattercast transport framework as described in Section 5.5.2. The details of the SFQ algorithm and its proof of correctness can be obtained from the original SFQ proposal by Goyal et al [53].

The input to the H-SFQ algorithm is a hierarchy of traffic classes starting with a root class. ADUs may be scheduled for transmission in any of the leaf classes in the hierarchy. Each class is assigned a numeric weight between 0 and 1, which determines the proportion of available bandwidth that the class may use. To achieve fair scheduling across the hierarchy, we run an instance of the SFQ algorithm at each level of the hierarchy. The scheduling of packets occurs recursively: the scheduler for the root traffic class schedules its subclasses; the schedulers of the subclasses in turn schedule their subclasses, and so on.

Within each scheduler, SFQ assigns a start tag to each subclass and schedules classes in increasing order of start tags. Let $A^j(c)$ denote the time at which the j^{th} packet is requested to be scheduled for traffic class c. If the class is idle when the request is made, then $A^j(c)$ is the time at which the request is made, otherwise it is the time at which the previous packet finishes transmission. The SFQ algorithm is then defined as follows:

1. When class c is requested to be scheduled, it is stamped with a start tag S_c which is computed as follows:

   $$S_c = max\{v(A^j(c)), F_c\}$$

   where $v(t)$ is the virtual time at time t and F_c is the finish tag of class c. F_c is initially 0, and when the j^{th} packet finishes transmission it is incremented as:

   $$F_c = S_c + \frac{l_c^j}{w_c}$$

where l_c^j is the length of the j^{th} packet and w_c is the weight associated with the class c.

2. The virtual time $v(t)$ is initially set to 0. When the scheduler is busy transmitting data, the virtual time at time t is defined to be equal to the start tag of the class being serviced at time t. When the scheduler is idle, $v(t)$ is set to the maximum of the finish tags associated with any class.

3. Finally, traffic classes are serviced in increasing order of their start tags, and ties are broken arbitrarily.

As is clear from the above algorithm, SFQ is computationally inexpensive. The main complexity is in scheduling classes in increasing order of start tags, which can be performed efficiently by using a priority queue.

Bibliography

[1] 3COM Corporation. 3COM PalmPilot. `http://www.3com.com/palm/index.html`.

[2] D. Agrawal, A. Abbadi, and R. Steinke. Epidemic Algorithms in Replicated Databases. In *Proceedings of the 16th ACM Principles of Database Systems*, May 1997.

[3] Akamai Technologies Inc. FreeFlow: High Performance Internet Content Delivery, 1999. `http://www.akamai.com/`.

[4] Elan Amir. *An Agent-based Approach to Real-time Multimedia Transmission over Heterogeneous Environments*. PhD thesis, University of California at Berkeley, May 1998.

[5] Elan Amir. MeGa: The Media Gateway Architecture, 1998. Service deployed at UC Berkeley.

[6] Elan Amir and Hari Balakrishnan. An Evaluation of the Metricom Ricochet Wireless Network. Class report, UC Berkeley, May 1996.

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

[8] Elan Amir, Steven McCanne, and Randy Katz. An Active Service Framework and its Application to Real-time Multimedia Transcoding. In *Proceedings of ACM SIGCOMM '98*, Vancouver, British Columbia, Canada, September 1998.

[9] Elan Amir, Steven McCanne, and H. Zhang. An Application-level Video Gateway. In *Proceedings of ACM Multimedia '95*, pages 255–265, San Francisco, CA, November 1995.

[10] Hari Balakrishnan, Hariharan S. Rahul, and Srinivasan Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proceedings of SIGCOMM '99*, Cambridge, MA, September 1999.

[11] Hari Balakrishnan, Srinivasan Seshan, and Randy H. Katz. Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks. *ACM Wireless Networks*, 1(4), December 1995.

[12] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core Based Trees (CBT). In *Proceedings of ACM SIGCOMM '93*, pages 85–95, San Francisco, CA, September 1993.

[13] Rob Barrett, Paul Maglio, and Daniel Kellem. How to Personalize the Web. In *Proceedings of CHI '97*, Atlanta, GA, March 1997.

[14] Fred Bauer and Anujan Varma. Degree-constrained Multicasting in Point-to-point Networks. In *Proceedings of IEEE Infocom '95*, March 1995.

[15] Tim Berners-Lee, Robert Cailliau, Ari Loutonen, Henrik Frystyk Nielsen, and Arthur Secret. The World Wide Web. *Communications of the Association for Computing Machinery*, 37(8):76–82, August 1998.

[16] C. Mic Borman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. The Harvest Information Discovery and Access System. *Computer Networks and ISDN Systems*, 28:119–125, 1995.

[17] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. *XML: Extensible Markup Language*, December 1997. W3C Proposed Recommendation. `http://www.w3.org/TR/PR-xml-971208`.

[18] C. Brooks, M. S. Mazer, S. Meeks, and J. Miller. Application-specific Proxy Servers as HTTP Stream Transducers. In *Proceedings of WWW-4*, Boston, MA, December 1995. `http://www.w3.org/pub/Conferences/WWW4/` Papers/56.

[19] John Byers, Michael Luby, Michael Mitzenmacher, and Ashu Rege. A Digital Fountain Approach to Reliable Distribution of Bulk Data. In *Proceedings of ACM SIGCOMM '98*, Vancouver, British Columbia, Canada, September 1998.

[20] Navin Chaddha, Gerard A. Wall, and Brian Schmidt. An End to End Software Only Scalable Video Delivery System. In *Proceedings of the Fifth International Workshop on Network and OS Support for Digital Audio and Video*, Durham, NH, April 1995. Association for Computing Machinery.

[21] Yatin Chawathe. MediaboardV2: A *libsrm*-based Distributed Whiteboard for the MBone. Class project, UC Berkeley, December 1998.

[22] Yatin Chawathe and Eric Brewer. System Support for Scalable and Fault Tolerant Internet Services. In *Proceedings of Middleware '98*, Lake District, U.K., September 1998.

[23] Yatin Chawathe, Steve Fink, Steven McCanne, and Eric Brewer. A Proxy Architecture for Reliable Multicast in Heterogeneous Environments. In *Proceedings of ACM Multimedia '98*, Bristol, U.K., September 1998.

[24] Yatin Chawathe, Steven McCanne, and Eric A. Brewer. RMX: Reliable Multicast for Heterogeneous Networks. In *Proceedings of INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[25] Yatin Chawathe, Angela Schuett, Elan Amir, et al. An Active Service Infrastructure for Real-time Conferencing. In *Proceedings of ACM Multimedia '98 (Demonstration)*, Bristol, U.K., September 1998.

[26] C. Cheng, R. Riley, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves. A Loop-Free Extended Bellman-Ford Routing Protocol Without Bouncing Effect. In *Proceedings of SIGCOMM '89*, 1989.

[27] David R. Cheriton and Stephen Deering. Host Groups: A Multicast Extension for Datagram Internetworks. In *Proceedings of the Ninth Data Communications Symposium. ACM/IEEE*, September 1985.

[28] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison Wesley, Reading, MA, 1994.

[29] Cisco SightPath. SightPath: Intelligent Content Delivery, 2000. `http://www.sightpath.com/`.

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

[31] Ivan A. Curtis et al. *XCopilot PalmPilot Emulator*. Software available at `http://xcopilot.cuspy.com/`.

[32] Stephen Deering and David Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems*, 8(2):85–110, May 1990.

[33] Stephen Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. An Architecture for Wide-area Multicast Routing. *IEEE/ACM Transactions on Networking*, 4(2), April 1996.

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

[35] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic Algorithms for Replicated Database Maintenance. *ACM Operating Systems Review*, 22:8–32, January 1988.

[36] Digital Island Inc. The Digital Island Networks Footprint Service. `http://www.digisle.net/services/cd/footprint.shtml`.

[37] Digital Island Inc. Footprint Streaming Solution, 2000. `http://www.digisle.com/services/cd/ftprntlive.shtml`.

[38] Christophe Diot, Brian Neil Levine, Bryan Lyles, Hassan Kassan, and Doug Balensiefen. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Networks special issue on Multicasting*, 2000.

[39] R. Droms. *Dynamic Host Configuration Protocol*, March 1997. RFC-2131.

[40] H. Eriksson. MBone: The Multicast Backbone. *Communications of the Association for Computing Machinery*, pages 54–60, 1994.

[41] FastForward Networks/Inktomi Media Division. Broadcast Overlay Architecture, 2000. `http://www.ffnet.com/`.

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

[43] Sally Floyd, Van Jacobson, C. Liu, Steven McCanne, and L. Zhang. A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing. In *Proceedings of ACM SIGCOMM '95*, pages 342–356, Boston, MA, August 1995. A more complete version of this paper appeared in IEEE/ACM Transactions on Networking, December 1997, 5(6) pp. 784–803.

[44] Armando Fox, Ian Goldberg, Steven D. Gribble, David C. Lee, Anthony Polito, and Eric A. Brewer. Experience With Top Gun Wingman: A Proxy-Based Graphical Web Browser for the 3Com PalmPilo. In *Proceedings of Middleware '98*, Lake District, U.K., September 1998.

[45] Armando Fox, Steven Gribble, Eric Brewer, and Elan Amir. Adapting to Network and Client Variability via On-demand Dynamic Distillation. In *Proceedings of ASPLOS-VII*, Cambridge, MA, October 1996.

[46] Armando Fox, Steven Gribble, Yatin Chawathe, Eric Brewer, and Paul Gauthier. Cluster-based Scalable Network Services. In *Proceedings of SOSP '97*, pages 78–91, St. Malo, France, October 1997.

[47] Armando Fox, Steven D. Gribble, Yatin Chawathe, and Eric A Brewer. TranSend Web Accelerator Proxy, 1997. Free service deployed at UC Berkeley.

[48] Armando Fox, Steven D. Gribble, Yatin Chawathe, and Eric A. Brewer. Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives. *Special issue of IEEE Personal Communications on Adaptation*, August 1998.

[49] Paul Francis. Yoid: Extending the Internet Multicast Architecture. `http://www.aciri.org/yoid/`.

[50] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman and Company, San Francisco, 1979.

[51] Paul Gauthier, Josh Cohen, Martin Dunsmuir, and Charles Perkins. *Web Proxy Auto-Discovery Protocol*, December 1999. Internet Draft.

[52] gnutella.wego.com. Gnutella: Distributed Information Sharing, 2000. `http://gnutella.wego.com/`.

[53] Pawan Goyal, Harrick M. Vin, and Haichen Cheng. Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. In *Proceedings of ACM SIGCOMM '96*, Stanford, CA, August 1996.

[54] A. Gulbranden and P. Vixie. *A DNS RR for specifying the location of services (DNS SRV)*, October 1996. RFC-2052.

[55] Rajarshi Gupta. WebTP: A User-Centric Receiver-Driven Web Transport Protocol. Master's thesis, University of California, Berkeley, December 1998.

[56] Paul Haeberli. An Image Convertor for 2 bits/pixel Grayscale Images. Private communication, 1997.

[57] M. Hamilton and R. Wright. *Use of DNS Aliases for Network Services*, October 1997. RFC-2219.

[58] Mark Handley. *Session DiRectory.* University College London. Software available at `ftp://cs.ucl.ac.uk/mice/sdr/`.

[59] Mark Handley. *SAP: Session Announcement Protocol*, March 1997. Internet Draft.

[60] Mark Handley and Jon Crowcroft. Network Text Editor (NTE): A scalable shared text editor for the MBone . In *Proceedings of SIGCOMM '97*, Cannes, France, September 1997. Association for Computing Machinery.

[61] Mor Harchol-Balter, Tom Leighton, and Daniel Lewin. Resource Discovery in Distributed Networks. In *Proceedings of the 18th Annual ACM-SIGACT/SIGOPS Symposium on Principles of Distributed Computing*, Atlanta, GA, May 1999.

[62] David A. Helder and Sugih Jamin. Banana Tree Protocol, an End-host Multicast Protocol. Unpublished Report, July 2000.

[63] C. Hendrick. *Routing Information Protocol (RIP)*, June 1988. RFC-1058.

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

[65] Yang hua Chu, Sanjay Rao, and Hui Zhang. A Case For End System Multicast. In *Proceedings of ACM Sigmetrics '00*, Santa Clara, CA, June 2000.

[66] InfoPad. UC Berkeley, `http://infopad.eecs.berkeley.edu/`.

[67] Inktomi Corporation. Inktomi Traffic Server: Network Cache for Service Providers, 1998. `http://www.inktomi.com/products/network/products/tscclass.html`.

[68] Internet Engineering Task Force. *HyperText Transfer Protocol – HTTP 1.1*, March 1997. RFC-2068.

[69] Internet Software Consortium. Internet Domain Survey, January 2000. See `http://www.isc.org/`.

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

[71] Van Jacobson and Stephen Deering. Administratively Scoped IP Multicast. In *Proceedings of the 30th Internet Engineering Task Force*, Toronto, Canada, July 1994.

[72] Van Jacobson and Steven McCanne. *LBL Whiteboard*. Lawrence Berkeley Laboratory. Software available at `ftp://ftp.ee.lbl.gov/conferencing/wb`.

[73] Van Jacobson and Steven McCanne. *Visual Audio Tool*. Lawrence Berkeley Laboratory. Software available at `ftp://ftp.ee.lbl.gov/conferencing/vat`.

[74] John Jannotti, David K. Gifford, and Kirk L. Johnson. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the 4th Symposium on Operating System Design and Implementation (OSDI)*, San Diego, CA, October 2000. USENIX.

[75] Anthony Joseph et al. A Toolkit for Mobile Information Access. In *Proceedings of 15th ACM Symposium on Operating Principles*, Copper Mountain Resort, CO, December 1995.

[76] C. R. Kalmanek, H. Kanakia, and S. Keshav. Rate-controlled Servers for Very High-Speed Networks. In *Proceedings of the IEEE Conference on Global Communications*, December 1990.

[77] Isidor Kouvelas, Vicky Hardman, and Jon Crowcroft. Network Adaptive Continuous-Media Applications through Self Organised Transcoding. In *Proceedings of the Network and Operating Systems Support for Digital Audio and Video*, Cambridge, U.K., July 1998.

[78] Satish Kumar, Pavlin Radoslavov, David Thaler, Cengiz Alaettinoglu, Deborah Estrin, and Mark Handley. The MASC/BGMP Architecture for Inter-domain Multicast Routing. In *Proceedings of SIGCOMM '98*, Vancouver, British Columbia, Canada, September 1998.

[79] Li-Wei H. Lehman, Stephen J. Garland, and David L. Tennenhouse. Active Reliable Multicast. In *Proceedings of INFOCOM '98*, March 1998.

[80] B. N. Levine, S. Paul, and J.J. Garcia-Luna-Aceves. Organizing Multicast Receivers Deterministically According to Packet-Loss Correlation. In *Proceedings of ACM Multimedia '98*, Bristol, U.K., September 1998.

[81] M. Liljeberg et al. Enhanced Services for World Wide Web in Mobile WAN Environments. Technical Report C-1996-28, University of Helsinki CS, April 1996.

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

[83] Mingyan Liu, Rajesh Talpade, Anthony McAuley, and Ethendranath Bommaiah. AM-Route: Adhoc Multicast Routing Protocol. Technical Report CSHCN T.R. 99-1/ISR T.R. 99-8, Institute for Systems Research, University of Maryland, College Park, MD, 1999.

[84] live.com. A Multicast Plugin for WinAmp. `http://www.live.com/multikit/winamp-plugin.html`.

[85] K. Lougheed and Y. Rekhter. *A Border Gateway Protocol (BGP)*, June 1989. RFC-1247.

[86] A. Luotonen. Proxy Auto Configuration. Netscape Corporation. `http://home.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html`.

[87] Steven McCanne. A Distributed Whiteboard for Network Conferencing. Class report, UC Berkeley, May 1992.

[88] Steven McCanne et al. Toward a Common Infrastructure for Multimedia-Networking Middleware. In *Proceedings of the Seventh International Workshop on Network and OS Support for Digital Audio and Video*, St. Louis, Missouri, May 1997. Association for Computing Machinery.

[89] Steven McCanne and Van Jacobson. *vic*: A Flexible Framework for Packet Video. In *Proceedings of ACM Multimedia '95*, pages 511–522, San Francisco, CA, November 1995.

[90] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven Layered Multicast. In *Proceedings of ACM SIGCOMM '96*, pages 117–130, Stanford, CA, August 1996.

[91] J. M. McQuillan, I. Richer, and E. C. Rosen. New Routing Algorithm for the ARPANET. *IEEE Transactions on Communications*, 28(5), 1980.

[92] D. Meyer. *Administratively Scoped IP Multicast*, July 1998. RFC-2365.

[93] D. Meyer and P. Lothberg. *Static Allocations in 233/8*. Internet Engineering Task Force, draft-ietf-mboned-static-allocation-*.txt, May 1999. Internet Draft.

[94] Microsoft Corporation. *Windows NetMeeting*. Information available at `http://www.microsoft.com/windows/netmeeting/`.

[95] P. Mockapetris and K. Dunlap. Development of the Domain Name System. In *Proceedings of the SIGCOMM '88 Symposium*, pages 123–133, August 1988.

[96] J. Moy. *Open Shortest Path First (OSPF) Version 2*, July 1991. RFC-1247.

[97] mpg123. A Real-time MPEG Audio Player for Layers 1, 2 and 3. `http://www.mpg123.de/`.

[98] Naspter. The Napster Online Music Community, 2000. `http://www.naspter.com/`.

[99] National Laboratory for Applied Network Research. The Squid Internet Object Cache. `http://squid.nlanr.net/`.

[100] NetRadio.com. NetRadio Home Page. `http://www.netradio.com/`.

[101] Netscape Corporation. Netscape NetCaster. `http://developer.netscape.com/software/netcaster.html`.

[102] Nokia Systems. Communicator 9000 Press Release. Available at `http://www.forum.nokia.com/nf/products/communicators/9000/index.html`.

[103] Nullsoft Inc. The WinAmp MP3 Player. `http://www.winamp.com/`.

[104] Christos Papadimitriou. Personal communication.

[105] Joseph C. Pasquale, George C. Polyzos, Eric W. Anderson, and Vachspathi P. Kompella. Filter Propagation in Dissemenation Trees: Trading Off Bandwidth and Processing in Continuous Media Networks. In *Proceedings of the Fourth International*

Workshop on Network and OS Support for Digital Audio and Video, pages 269–278, Lancaster, U.K., November 1993. Association for Computing Machinery.

[106] A. Pelc. Fault-tolerant Broadcasting and Gossiping in Communication Networks. *Networks*, 28(5), October 1996.

[107] R. Perlman, C. Lee, T. Ballardie, J. Crowcroft, Z. Wang, T. Maufer, C. Diot, J. Thoo, and M. Green. *Simple Multicast: A Design for Simple, Low-overhead Multicast.* Internet Engineering Task Force, March 1999. Internet Draft (work in progress).

[108] Radia Perlman. *Interconnections: Bridges and Routers.* Addison Wesley, Reading, MA, 1992.

[109] Sridhar Pingali, Don Towlsey, and Jim F. Kurose. A Comparison of Sender-initiated and Receiver-initiated Reliable Multicast Protocols. In *Proceedings of the SIGMETRICS '94 Annual Conference on Measurement and Modeling of Computer Systems*, Nashville, Tennessee, May 1994. Association for Computing Machinery.

[110] Pioneer Consulting. Global Broadband Access Markets: xDSL, Cable Modems and the Threat from Broadband Satellite, Wireless and All-Optical Solutions, 1998. See `http://www.pioneerconsulting.com/`.

[111] PointCast Inc. PointCast Home Page. `http://www.pointcast.com/`.

[112] Suchitra Raman, Hari Balakrishnan, and Murari Srinivasan. ITP: An Image Transport Protocol for the Internet. In *Proceedings of the 8th International Conference on Network Protocols (ICNP) 2000*, Osaka, Japan, November 2000.

[113] Suchitra Raman, Yatin Chawathe, and Steven McCanne. libsrm*: The Berkeley SRM toolkit.* University of California, Berkeley. Software available at `http://www-mash.cs.berkeley.edu/mash/software/srm2.0/`.

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

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

[116] Sylvia Ratnasamy, Yatin Chawathe, and Steven McCanne. A Delivery-based Model for Multicast Protocols using Scattercast. *Unpublished*, July 1999.

[117] Sylvia Ratnasamy and Steven McCanne. Scaling End-to-end Multicast Transports with a Topologically-sensitive Group Formation Protocol. In *Proceedings of the 7th International Conference on Network Protocols*, Toronto, Canada, October 1999.

[118] real.com. RealPlayer. `http://www.real.com/`.

[119] Luigi Rizzo and Lorenzo Vicisano. A Reliable Multicast Data Dristribution Protocol Based on Software FEC Techniques. In *Proceedings of HPCS '97*, Greece, June 1997.

[120] J. Saltzer, D. Reed, and D. Clark. End-to-end Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):195–206, 1984.

[121] Y. Sato. DeleGate Server. Documentation available at `http://www.aubg.edu:8080/cii/src/delegate3.0.17/doc/Manual.txt`.

[122] M. A. Schickler, M. S. Mazer, and C. Brooks. Pan-browser Support for Annotations and Other Meta-information on the World Wide Web. In *Proceedings of WWW-5*, Paris, France, May 1996. `http://www5conf.inria.fr/fich\_html/papers/P15/Overview.html`.

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

[124] Angela Schuett, Suchitra Raman, Yatin Chawathe, Steven McCanne, and Randy Katz. A soft-state Protocol for Accessing Multimedia Archives. In *Proceedings of the Eight International Workshop on Network and OS Support for Digital Audio and Video*, Cambridge, UK, July 1998. Association for Computing Machinery.

[125] Henning Schulzrinne, Steve Casner, R. Frederick, and Van Jacobson. *RTP: A Transport Protocol for Real-Time Applications*, January 1996. RFC-1889.

[126] Nachum Shacham. Multipoint Communication by Hierarchically Encoded Data. In *Proceedings of IEEE Infocom '92*, pages 2107–2114, 1992.

[127] Tony Speakman, Dino Farinacci, Steven Lin, and Alex Tweedly. *Pragmatic General Multicast (PGM) Reliable Transport Protocol.* CISCO Systems, 1998. Internet Draft.

[128] Ion Stoica, Hui Zhang, and T. S. Eugene Ng. A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service. In *Proceedings ACM SIGCOMM '97*, Cannes, France, September 1997.

[129] David L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine 35*, pages 80–86, January 1997.

[130] Joe Touch and Steve Hotz. The X-Bone. In *Proceedings of the Third Global Internet Mini-Conference in conjunction with Globecom '98*, Sydney, Australia, November 1998.

[131] Teck-Lee Tung. Mediaboard: A Shared Whiteboard Application for the MBone. Master's thesis, University of California, Berkeley, December 1997.

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

[133] Unwired Planet. Handheld Device Markup Language. `http://www.uplanet.com/tech/products/hdml.html`.

[134] Robert van Renesse, Yaron Minsky, and Mark Hayden. A Gossip-style Failure Detection Service. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware '98)*, pages 55–70, Lake District, U.K., September 1998.

[135] J. Veizades, E. Guttman, C. Perkins, and M. Day. *Service Location Protocol*, October 1997. Internet Draft.

[136] Lorenzo Vicisano, Luigi Rizzo, and Jon Crowcroft. TCP-like Congestion Control for Layered Multicast Data Transfer. In *Proceedings of INFOCOM '98*, March 1998.

[137] David Waitzman, Craig Partridge, and Stephen Deering. *Distance Vector Multicast Routing Protocol (DVMRP)*, November 1988. RFC-1075.

[138] WAP Forum. Wireless Application Protocol Architecture Specification, April 1998. `http://www.wapforum.org/`.

[139] David Wetherall, John Guttag, and David Tennenhouse. ANTS: Network Services without the Red Tape. *IEEE Computer*, 32(4):42–49, April 1999.

[140] Brian Whetten and Jim Conlan. A Rate-based Congestion Control Scheme for Reliable Multicast. GlobalCast Communications, October 1998.

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

[142] Xerox LiveWorks. The LiveBoard Interactive Meeting System. `http://www.liveworks.com/liveboard/index.html`,.

[143] X. Rex Xu, Andrew C. Myers, Hui Zhang, and Raj Yavatkar. Reliable Multicast Support for Continuous-Media Applications. In *Proceedings of the Seventh International Workshop on Network and OS Support for Digital Audio and Video*, St. Louis, MO, May 1997. Association for Computing Machinery.

[144] Koichi Yano and Steven McCanne. The Breadcrumb Forwarding Service: A Synthesis of PGM and EXPRESS to Improve and Simplify Global IP Multicast. *ACM Computer Communication Review*, 30(2), April 2000.

[145] K. P. Yee. Shoduoka Mediator Service. `http://www.shoduoka.com/`.

[146] Ellen W. Zegura, Kenneth L. Calvert, and Samrat Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE Infocom '96*, San Francisco, CA, March 1996.

[147] Bruce Zenel and Dan Duchamp. A General-purpose Proxy Filtering Mechanism Applied to the Mobile Environment. In *Proceedings of MobiCom '97*, Budapest, Hungary, October 1997.