# Network Layer Support for Overlay Networks

by

John Jannotti

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 2002

Author . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Department of Electrical Engineering and Computer Science
August 30, 2002

Certified by . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
M. Frans Kaashoek
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

# Network Layer Support for Overlay Networks

by

John Jannotti

Submitted to the Department of Electrical Engineering and Computer Science
on August 30, 2002, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

## Abstract

Overlay networks are virtual networks formed by cooperating nodes that share an underlying physical network. They represent a flexible and deployable approach for applications to obtain new network semantics without modification of the underlying network, but they suffer from efficiency concerns. This thesis presents two new primitives for implementation in the network layer (i.e., the routers of the physical network). These primitives support the efficient operation and construction of overlay networks. *Packet Reflection* allows end hosts to request that routers perform specialized routing and duplication for certain packets. *Path Painting* allows multiple end hosts to determine where their disparate paths to a rendezvous point meet, in order to facilitate overlay topology building that reflects the topology of the underlying network. Both primitives can be incrementally deployed for incremental benefit.

This thesis describes a variety applications of these primitives: application level multicast systems with various semantics, an extended Internet Indirect Infrastructure with latency benefits over the original proposal, and an extension to Chord which would allows faster lookups.

Experimental results on simulated topologies indicate that when all routers support the proposed primitives, less that 5% overhead (in terms of link usage and latency) remains in two common overlay network usage scenarios. In addition, the benefits gained from deployment are significant even at low deployment levels. At approximately 25% deployment, the primitives have reduced overhead by over 50%. When intelligent deployment strategies are used, link usage overhead is less than 30% at less than 10% deployment. Finally, the results indicate that these benefits affect the area local to the deployed routers, providing a deployment incentive to independent networks.

Thesis Supervisor: M. Frans Kaashoek
Title: Professor of Computer Science and Engineering

## Acknowledgments

Many people have contributed substantially to this thesis — technically, emotionally, or both.

I would like to thank my thesis committee, Hari Balakrishnan, Ion Stoica, and my advisor, Frans Kaashoek. Their interest and insight has proven invaluable. I would especially like to thank Frans for his encouragement throughout my years at MIT. His faith in me may be misplaced, but it has encouraged me nonetheless.

The PDOS group have shaped my views considerably, and I am richer for it. I would like to thank Robert Morris, Dawson Engler, Eddie Kohler, and David Mazières in particular. I've learned more from them than I can ever repay.

I would like to thank Cisco Systems. As a company, Cisco has been extremely understanding of my commitment to my work at MIT while I have also worked for there. As individuals, they have been be friends, and technically helpful. I would especially like to thank Jim O'Toole and Suchitra Raman for discussions related to this thesis.

I would like to thank Ann Hintzman. Although she served as proofreader for this thesis, to describe her contribution in such limited terms would be a profound understatment. She has supported me emotionally more than I have deserved, and I remain eternally grateful.

I would like to thank my family. They have loved me, supported me, and encouraged me. I hope I have made them proud.

Finally, I would like to thank all of the people that have distracted me, making this thesis take longer to write, but making my life brighter for their presence.

# Contents

# List of Figures

# Chapter 1

# Introduction

Distributed applications would be well served by richer semantics than the network layer supplied by the Internet. Today's distributed applications have only one primitive from which they may build services: Internet Protocol (IP) Unicast, the best-effort, single source and destination delivery of datagrams. Unicast delivery allows a single network node to ask for a single packet of data to routed through the Internet to a single destination node. Additional semantics have been built on top of this single primitive. For example, Transmission Control Protocol (TCP) provides reliability and flow control.

Many applications, however, would benefit from additional services that are more difficult to build on top of IP Unicast delivery. Mission critical applications would like control over the way their packets are routed — perhaps trading off resource usage for reliability by using multi-path routing [35]. Teleconferencing applications [20], chat rooms, and Internet broadcasting systems would benefit from efficient group communication. A stock ticker application might like to perform latency measurements over many paths to find a low latency path undetected by normal IP routing. A content distribution network would like to distribute and store data throughout the network [21, 16].

One approach to addressing these needs is to build new network services into routers across the Internet. Generally this approach has two drawbacks. First, it may be inappropriate to add the necessary functions to routers that should remain

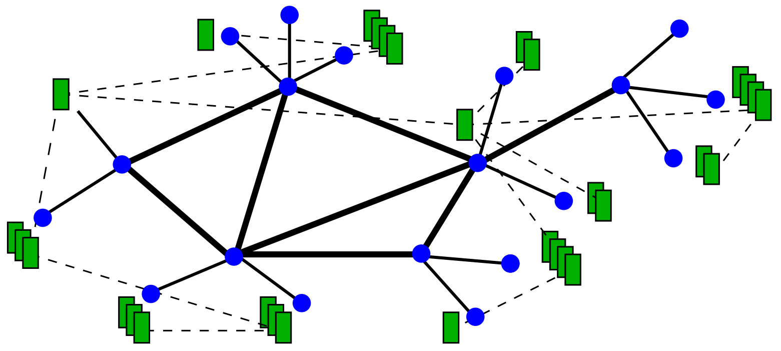

Figure 1-1: An overlay network. Rectangular end-hosts and dashed links form an overlay network over the physical network of round routers and solid links.

fast and simple to ensure their continued availability as an important shared resource. Second, adding an important network service to routers is likely to support only those applications envisioned during the design of the service. For example, IP Multicast provides a single service model that is inappropriate for a number of multicasting applications. Efforts to revamp IP Multicast for reliability or for secure admission control require yet more modifications to routers.

## 1.1 Overlay networks

Overlay networks completely sidestep these two drawbacks. Overlay networks avoid the issue of "dumb" IP routers by performing packet routing and duplication in edge nodes. Many examples are described in Section 2.2. In these systems, cooperating servers throughout the Internet act as routers in an overlay network.

Figure 1-1 demonstrates an overlay network. Just as a physical network has a topology consisting of the nodes of the network and the links between them, an overlay network has a virtual topology, which exists by the agreement of the overlay nodes. Packets are transmitted only along the virtual links between the overlay nodes using the underlying unicast mechanism provided by IP.

In contrast to the Internet, in which routers are a shared resource that cannot be specialized for a particular purpose, the members of an overlay network may provide specialized services specific to the application at hand. An overlay-based multicast system can duplicate packets in the servers, a content distribution network can cache gigabytes of data, RON provides resilient routing by constant performance measurements among participating nodes.

We will use three metrics to evaluate how efficiently an overlay network is operating. *Stress* indicates the number of times that a semantically identical packet traverses a given link. In IP Multicast, stress never exceeds one. On the other hand, overlay networks could not hope to achieve such efficiencies because packets being forwarded by an edge node will traverse (at least) the node's local link twice. *Stretch* indicates the ratio of latency in an overlay network compared to some baseline, generally IP unicast or multicast. *Misfires* measures the number of times that duplicate packets are mistakenly sent to the same destination. These metrics are used to evaluate overlay performance with the primitives proposed in this thesis in Chapter 7.

## 1.2 Problem

The overlay network approach faces two important challenges. First, overlay networks operate at a disadvantage to router-based systems because of the physical location of their computational elements. The components of an overlay network are servers located at the edge of the network. This drawback is both a performance problem, packets going in and out of external servers increases stress and stretch; as well as a functional problem, overlay nodes are not in a position to observe network traffic that is not explicitly directed to them. For example, IP Multicast's mechanism for joining groups relies on the ability of routers to observe passing messages.

Second, it can be difficult to build virtual topologies that resemble the topology of the underlying network. It is beneficial for the virtual links of an overlay network to connect nodes that are well-connected in the underlying network. Choosing well-connected virtual links is akin to supplying a physical network with a higher bandwidth link-layer. It is also common to prefer virtual links that share as few underlying links as possible with other virtual links. This property leads to independent failures, and less duplicate traffic on underlying links. Unfortunately, it is very difficult to determine these characteristics today. Overlays have fallen back on wasteful and error-prone techniques such as continual bandwidth probes to learn about the underlying network.

## 1.3 Design goals

A single set of simple extensions to IP that support overlay networks would, at once, address the needs of a large variety of applications that would otherwise each require separate network support. Just as the ability to support multiple processes through virtual memory and supervisor mode is considered a first priority in processor design, the ability to support multiple virtual networks as overlays should be a design goal in wide-area network design.

The goal of this thesis is to propose simple extensions to the network layer that would allow the construction of simple, efficient overlay networks. The extensions should be beneficial to the large variety of research projects and service providers that are investigating systems based on overlay networks.

A number of design criteria can be drawn from the diverse needs of existing overlay systems. First, and most directly, overlay networks would benefit from pushing work toward the core of the network. End System Multicast exhibits this need most acutely because its participants are expected to be desktop machines — no nodes are expected to be located at strategic network points. Yet every overlay system would benefit if routing and duplication could be *directed* by the end hosts, but *performed* by the appropriate routers, thereby allowing their packets to follow a more optimal path through the underlying network.

Second, all of these systems have difficulties constructing overlay topologies. RMX uses hand configuration. End System Multicast lacks scalability due, in part, to its topology generation algorithm. X-Bone and Yoid need help pruning an initially quadratic number of possible links down to a manageable size. Overcast consumes bandwidth to conduct constant network measurements.

A third, more subtle conclusion, is that the overlay nodes must retain complete control of forwarding when necessary. While it is important to provide a mechanism that allows overlay networks to obtain efficient forwarding, that mechanism must be fine-grained enough to allow the overlay to handle a particular link in a specialized way when necessary. For example, systems such as RMX would like to take advantage of automatic forwarding when possible, but when a link is congested they must retain the ability to perform their own forwarding so that they may transcode to a thinner format. Similarly, Overcast must be able to interpose its nodes in the forwarding mesh so that they may cache the forwarded data and replay transmission for new nodes.

In addition to these criteria, there are lessons to be drawn from the challenges that have faced other proposals for router modification, particularly IP Multicast. It is critical that new functions are incrementally deployable — there must be benefits to deployment even when only a small portion of routers have been modified to support the new functionality. In addition, the benefits from deployment should be concentrated around the portion of the network in which deployment occurred. These benefits offer an economic incentive to early adopters.

In order to provide for incremental deployment, this thesis proposes mechanisms that are for optimization only. Overlay networks must be prepared to operate as if the primitives do not exist. When the primitives *are* available, the network will provide explicit signaling to the application, allowing it to avoid work that has been performed in the network.

Finally, it is important to keep proposed enhancements small so that future modifications to their behavior are unlikely to be required, and generally useful so that greater utility might someday be obtained by using then in novel combinations.

## 1.4 Contributions

The contributions of this dissertation are:

- *Packet Reflection*: a new primitive by which applications may request packet routing and duplication to occur in the routers in order to decrease stress and stretch. In contrast to similar proposals, packet reflection is incrementally deployable and provides feedback allowing applications to perform correctly in the face of routing changes.

- *Path Painting*: a new primitive for allowing end hosts to coordinate and learn where their individual paths to a rendezvous point converge, allowing efficient overlay topologies to be built.

- A demonstration of the flexibility allowed by these new primitives in the form of example multicast systems with varying communication models.

- A demonstration that the primitives are more generally useful, in the form of an example outside of the realm of multicast. *i3* [36] could use reflection to offer better performance with less setup work.

## 1.5 Outline

The remainder of this thesis is divided into three parts. The first part consists of Chapters 3, 4, and 5. These chapters describe the Packet Reflection and Path Painting primitives, with Chapter 5 serving to provide detailed implementation information. The second part is Chapter 6, which presents realistic applications of those primitives in various overlay networks. Finally, Chapter 7 examines the performance of the primitives in many scenarios, including limited deployment and asymmetric networks, evaluating the ability of the primitives to reduce stress and stretch while creating few misfires and consuming limited resources on the participating routers.

# Chapter 2

# Related Work

Related work can broadly be divided into three areas. First, IP Multicast and systems built on IP Multicast. IP Multicast provides a group communication primitive for IP, and a number of systems have attempted to build additional semantics on top of IP Multicast. The efforts have seen limited success, in part because of IP Multicast's limited deployment, and in part because IP Multicast provides a difficult base upon which to build — IP Multicast is a monolithic primitive combining all needed mechanisms to provide a single high-level abstraction.

A second set of related projects are overlay networks. These systems seek to avoid the deployment issues of IP Multicast by providing network abstractions purely in end hosts. This thesis complements these systems by providing protocols to allow these systems to approach the efficiency of router based network services.

The most closely related work is an approach to network-layer extension that is quite similar to the approach advocated in this thesis. In Section 2.3 we examine a active networks based system that builds multicast from unicast forwarding and ephemeral state. That approach yields a primitive that resembles reflection, as well as a few other primitives that may also be of use in general overlay networks.

## 2.1 Multicast specific

Multicast, with various semantics, is a fertile area of research in the networking community. Numerous proposals for enhancing the network layer to support multicast in one form or another have been made over the years. This section examines the most important of these proposals. In each case, the single greatest difference between the goal of these ideas and the purpose of this thesis is breadth. The primitives presented here are intended to allow overlay networks to accomplish much more than a single form of multicast communication.

### 2.1.1 IP Multicast

IP Multicast [12] (IPM) is a low-level network primitive that provides efficient communication between multiple nodes. The basic unit for communication is that of the *group*, which corresponds to an IP address chosen from a particular reserved range of all IP addresses. In IP Multicast, nodes register their interest in a particular group by sending *graft* messages toward a designated rendezvous point. As graft messages propagate to the rendezvous, the protocol builds a spanning tree which includes all interested nodes and the routers that connect those nodes. Graft messages may be suppressed when they reach the existing tree, they need not reach the rendezvous. When a node sends a packet to an IP Multicast group, routers forward it to all interfaces that correspond to edges in the spanning tree.

IP Multicast, taken as a whole, suffers from a number of drawbacks that have hampered its widespread acceptance. First, it requires router support. The spanning tree for a group cannot cross regions of the network that are not running IPM enabled routers. This hurdle creates little incentive to be "the first on the block" to roll out IPM support because IPM's utility is greatly diminished without widespread adoption.

Second, IP Multicast's admittance control and, by extension, its security, are weak. Any Internet node (assuming deployment of IPM routers) can send or receive packets for any IP Multicast group. Proponents argue for end-to-end encryption and authentication, but such a solution would do little to avoid denial of service attacks that rely only on the volume of data, not its authenticity. Even if unauthenticated packets were disposed of at end-hosts, IP Multicast's willingness to send copies of "junk" packets to all members of a group presents a ripe target for Internet trouble makers. In fact, recent experience with the Ramen Worm [41] shows that IP Multicast is so susceptible to bandwidth wasting attacks that it can be taken advantage of by mistake. The Raman Worm probed IP addresses at random, some of which were IP Multicast addresses. The Worm was first detected by the enormous growth in Multicast traffic it accidentally consumed as it spread.

Third, and most subtly, IP Multicast is an awkward primitive on which to build other network services. For example, attempts have been made for years to describe a simple, scalable reliable multicast protocol on top of IP Multicast [15, 24, 25, 26, 29], yet there is no consensus that any single protocol is appropriate for deployment in the world's routers. One problem plaguing such attempts is that there is no single agreed upon semantics for reliable multicast. One application might wish to use flow control to slow the data stream down to the bandwidth of the slowest link. Another might want to stream data as quickly as possible to nodes that have fast connections, while trickling in to those that do not. Yet another might wish to relax ordering constraints in favor of receiving most of the data in near real-time and the rest later. Attempting to support all of these semantics in router based software calls for a consensus that each is important enough to standardize upon, and risks destabilizing core routing functionality as increasingly complex software is added. Instead, more complex semantics should be provided by end-hosts, just as TCP provides flow-control and reliability.

### 2.1.2 SSM and Express

Source-specific Multicast [18] (SSM) and Express [19] avoid some of the problems of standard IP Multicast by offering a simplified, single-source service model. In both systems, end-hosts subscribe to a group that is, in part, named by its *source*. This approach greatly simplifies the join protocol because it implicitly names a rendezvous point for join messages — the source.

Besides this simplification, these single-source approaches have additional benefits compared to standard IP Multicast. First, the namespace of possible group names is vastly expanded *and* managed more easily. In standard IP Multicast, 28 bits of address space have been set aside to be used as Multicast group addresses. Management of that space, however, is a difficult problem. In single-source approaches, the IP address of the source is a portion of group identifier, which allows each Internet host to manage its own namespace.

In many applications, the limited power of a single-source approach is also a benefit. Applications that are fundamentally one-to-many benefit from the simpler admission control and security of a hard-wired single source. Only the source of the group may send packets to the group, so the flooding attacks of standard IP Multicast are impossible. Additionally, with only one source, simple cryptographic schemes for admission control are possible. The sender encrypts and signs all packets. Only those nodes possessing the group key may decrypt and authenticate the packets.

Finally, as we will see in the next section, a single-source models lends itself more easily to reliability extensions.

### 2.1.3 Repliers, PGM, and BCFS

Papadopoulos' Replier scheme [31], PraGmatic Multicast [14], and the BreadCrumb Forwarding Service [47] all offer support for reliable multicast applications. As each system slightly extends or modifies the previous, they work in very similar ways.

All three schemes support reliability by allowing "downstream" receivers to contact "upstream" receivers for lost packets. It remains the responsibility of the application residing at the receivers to buffer and retransmit the lost packets. At each router, a particular interface is specially designated to receive request packets when they are sent "upstream". This interface is the *replier link*. In a router that is not acting as a branch point, the replier link is the upstream link. In a router that is a branch point, one of the downstream interfaces is chosen. When an end-host sends a request packet upstream, it will be forwarded along these replier links (a packet coming from a replier link is sent upstream). As a result, all upstream packets will flow through a hierarchy of repliers.

In this way, multiple acknowledgments, sent upstream, may be coalesced when they arrive at a designated *replier*. That replier may respond to the entire set of receivers using a directed multicast down the distribution tree.

These schemes have the usual drawbacks of network-layer multicast schemes in comparison to the primitives presented in this thesis — they are not incrementally deployable nor may they be used outside of a multicast context.

### 2.1.4 REUNITE

REUNITE [38] is a multicast protocol that multicasts using recursive unicast distribution trees. In other words, as in an overlay network, packets are transmitted from point to point using traditional IP unicast. Unlike overlay networks, REUNITE uses point to point unicast transmissions mainly between routers, only involving end-hosts at the edges of the tree.

REUNITE accomplishes a number of things that the primitives proposed in this thesis also hope to achieve. It is a fairly simple protocol, it is incrementally deployable, and state requirements can be manages explicitly by overloaded routers. REUNITE, however, is aimed strictly at supporting multicast. Path Painting and Packet Reflection are intended to support a much broader range of applications.

### 2.1.5 Heterogeneous multicast

Many researchers have attempted to extend basic IP Multicast to support heterogeneous receivers in a multicast group. In a large IP Multicast group there is likely to be a wide distribution of available bandwidth between any two members of the group. This diversity presents an unenviable tradeoff for senders. On the one hand, a source might decrease its transmission bandwidth to the level of the most poorly connected receiver. This choice would unnecessarily decrease the quality of transmission for nearby receivers, but would result in less packet loss and congestion in the rest of the network. Alternatively, senders might ignore the problem, and transmit at full speed. Nearby receivers would receive high-quality feeds, but the volume of traffic would cause congestion at slow links. Furthermore, though poorly connected receivers might receive the same number of bytes in either scenario, the reconstituted signal is likely to be of higher quality in the first because the source carefully selected the best information to send to bandwidth-starved hosts rather than allowing the network to drop packets at random.

Receiver-driven Layered Multicast (RLM) [27] presents one solution to the problem and Thin Streams [46] refines that idea. In RLM, senders encode transmissions in a number of layers. The lowest layer is a low-fidelity encoding of the transmission. Higher level layers contain extra data which allows a more faithful reproduction of the stream at the receiver. Sends transmit each layer in a separate IP Multicast group, allowing receivers to independently subscribe to each layer.

RLM receivers determine the appropriate number of layers to subscribe to by conducting *join experiments*. In a join experiment a receiver subscribes to the next higher layer and determines whether its subscription creates congestion by monitoring loss rate over a short period called the *decision time*. If it does, it unsubscribes. Failed join experiments contribute to network congestion though, so RLM scales back the frequency of an individual node's experiments with the size of the multicast group. To allow nodes to learn the appropriate level of subscription rapidly in spite of less frequent experiments, nodes learn by observing the network during other nodes' experiments. If the network becomes congested during another nodes experiment on say, level three, of a presentation, then the observing node can act as though it conducted a failed level three experiment itself.

Thin Streams offers a number of improvements over RLM while retaining the same basic idea of allowing receivers to determine the appropriate level of traffic for a given presentation by subscribing to the correct number of streams in a presentation. First, presentations of Thin Streams consist of many more layers than those advocated by RLM. In an effort to prevent packet loss during join experiments, Thin Streams pares down both the bandwidth of individual layers and the length of the decision time. This design allows the network to buffer an entire join experiment if it causes congestion. Join experiments can now be run without causing packet loss in unrelated traffic.

Obviously, packet loss can no longer be used as metric to measure the success of join experiments. Instead, Thin Streams uses a technique developed in TCP Vegas [8]. A node compares the expected bandwidth of a new stream to the actual bandwidth being observed. If the new layer is arriving slowly it reflects growing buffering in the network. The join experiment can be judged a failure before packet loss occurs.

RLM and Thin Streams have a few significant drawbacks. First, they implicitly limit the multi-source IP Multicast model to a single source. They would not allow two groups of researchers, at MIT and Berkeley, to teleconference in a way that allows MIT researchers to see each other in high-fidelity and Berkeley researchers to see each other in high-fidelity. Such scenarios would require a set of IP Multicast Groups for each source, which would greatly compound the remaining problems.

Second, they require encoder research to develop codecs that decompose well into a layered architecture. Third, they use extra IP Multicast groups. Using extra groups contributes to an existing problem area for IP Multicast, the large size of the IP Multicast routing tables that must be present in every router. Large tables require large amount of expensive fast memory and slow all lookups when using a hash-based lookup scheme. In addition, IP Multicast's small address space is strained even more by using additional groups for each presentation.

Video Gateways [1] are designed to solve the same problems as layered multicast, while avoiding the problems of the layered approaches. What makes Video Gateways even more interesting though, is that they address an additional problem. On the Internet today, and for the foreseeable future, IP Multicast cannot be assumed to reach all hosts. Some networks have deployed IP Multicast, some have not. This situation has existed for some time and it is difficult to predict when it might change.

Video Gateways are application-level proxies that connect two separate IP Multicast groups using unicast. Each pool of nodes contains a Video Gateway that subscribes to the local IP Multicast group. The Video Gateways then communicate using IP unicast to bridge the content of the two groups. Because such bridging is under application control it may perform additional functions if desired. For example, a Video Gateway connecting a group using Motion-JPEG with a group using H.261 by transforming data between the two formats has been demonstrated. The Video Gateways perform the necessary transcoding in each direction so that all members may source and sink data with all other nodes in both groups.

Video Gateways address all of the problems of layered IP Multicast, and even allow some new capabilities that layered systems do not even attempt to provide (such as the ability to transcode data formats and bridge unicast networks). They have a downside, however, that is likely to prevent widespread use of Video Gateways for bridging large groups with many segregated IP Multicast sessions. Video Gateways are statically configured entities. Each pair of gateways connects exactly two IP Multicast groups, in exactly the way that a human operator has configured the two gateways. This manual setup becomes untenable as the desire to connect more groups, on an ad-hoc basis, is considered; the human effort involved in configuring tens of gateways in order to video conference would be considerable. Nonetheless, the fundamental architecture is appealing. A network of forwarding nodes, similar to Video Gateways, that organize themselves to bridge between segregated multicast groups would be a logical extension.

## 2.2 Overlay networks

A number of research groups and service providers are investigating services that could be described as a network of Video Gateways. Often referred to as "Overlay networks", these services consists of many application-level nodes using unicast to move data to multiple recipients.

Some overlays are extremely generic, such as RON [3], the Dynabone [40], and Yoid [16]. These systems exists solely to provide an overlay network with "better" properties than the underlying network, such as lower latency or greater resistance to DOS attacks. In the following sections, more specialized overlays are examined.

### 2.2.1 Application-level multicast

Application-level multicast (ALM) systems include RMX [10], End System Multicast [20], and Overcast [21]. All share the goal of providing the benefits of IP Multicast without requiring direct router support or the presence of a physical broadcast medium.

RMX focuses on real-time reliable multicast. As such, its focus is on reconciling the heterogeneous capabilities and network connections of various clients with the need for reliability. Therefore RMX focuses on *semantic* rather than *data* reliability. For instance, RMX can change high resolution images into progressive JPEGs before transmittance to underprovisioned clients. In that sense, RMX is the closest relative to Video Gateways. Transcoding is an excellent example of function that is best performed by application-level servers, providing direct evidence that an architecture that involves external servers may be preferable to enhancing the network layer.

End System Multicast provides small-scale multicast groups for teleconferencing applications using only the group members to duplicate packets. End System Multicast's chief drawback is its limitation to small scale groups. The primitives proposed here could provide a significant benefit to ESM because reflection can duplicate packets in the core of the network, rather than relying on the receivers alone.

Overcast is an ALM system designed to support high-bandwidth, single-source applications. Overcast's contribution is two-fold. First, it demonstrates the benefits of an ALM system by providing features that IP Multicast cannot support, such as on-demand access to content. Second, simulated evaluations of Overcast's tree building technique provide evidence that reasonable trees can be built under application-level control. Inefficiencies remain, however, because packet duplication cannot occur in routers and online measurements must be used extensively and continuously in order build and maintain the distribution tree.

### 2.2.2 Peer-to-peer

Many Peer-to-peer (P2P) systems can also be considered overlay networks. In these systems, large numbers of end-hosts cooperate toward some end. For many such systems, such as Gnutella [17], and Freenet [11], that goal is the widespread availability of content. For others, such as CAN [32], Chord [37], Pastry [33], and Tapestry [22], the goal is a more generic lookup service. Still others, such as Mix Nets [9], attempt to provide an overlay network in which cryptographic techniques provide anonymity for participants (a secondary goal in many of the previously mentioned systems as well).

Painting and Reflection are complementary to these systems, though some P2P systems will be unable to derive benefits from both primitives. For example, in systems such as Chord, important properties of the system are based on the random path a lookup request takes before reaching its destination. Further, once the lookup is completed, no more packets will be expected to take that path. In such cases, Packet Reflection cannot be used to optimize the lookup.

On the other hand, Gnutella might take advantage of Packet Reflection when sending responses to requests. These responses may be quite large, and they are pipelined through the set of nodes through which the request came. The nodes along that path could use Packet Reflection to speed the delivery of the responses.

## 2.3 Active networks

In active networks [39, 45, 34, 44], applications can download new protocols and code into routers, allowing for rapid innovation of network services. This thesis avoids many of the hard problems of active networks by focusing on specific functionality; it does not need to address the problems created by dynamic downloading of code, sharing resources among multiple competing applications, or standardizing a programming platform. Despite this major philosophical difference, some researchers have used an active networks to describe a set of primitives [43] with similar power to the primitives proposed here.

Therefore, a detailed comparison to this approach is appropriate. The Kentucky approach presents a primitive *dup* that closely resembles packet reflection, and a number of primitives (in an active networks framework) that accomplish goals similar to packet reflection.

A strength of their approach is that it handles asymmetric routes better than painting and reflection. However, this appears to come at the cost of some scalability – part of their join mechanism involves an *echo* packet reaching a central rendezvous point before being returned to the sender. Very large groups could overwhelm the rendezvous point's network.

A strength of the primitives presented here is that they have been designed to operate correctly in the face of route changes in the underlying network. *dup* is insufficient to handle these cases. For example, if a route change causes the router maintaining node A's *dup* request to stop receiving the group's packets (because they are now taking a different path), A will lose the packets. With reflection, duplications are explicitly confirmed to a node's parent, using success tags. If the duplication point is bypassed, the parent knows it. It can then perform the duplication on its own and make a new reflection request.

A final difference is largely philosophical. *Dup* and friends are presented as a few of the infinity of possible primitives that could be constructed with an active networks approach. While this approach brings the promise of arbitrary programmability, it carries the baggage of complexity and difficult to address security problems. The primitives presented in this thesis are simple, have *enough* power, and the security implications can be well understood.

# Chapter 3

# Packet Reflection

In an overlay network each node carries out explicit unicast communication with its neighbors in the topology. When one overlay node forwards packets between two other nodes, that packet is transmitted on the same link multiple times as it reaches the overlay router and is re-emitted toward the final destination. The links near the overlay router will have a stress of two, and the stretch of the packet will exceed one as time is wasted while the packet approaches and then leaves the overlay router.

When multicasting on an overlay network, the stress problem is exacerbated. The forwarding node duplicates packets, forcing semantically equivalent packets to be transmitted on the same link, in the same direction, multiple times. In such cases, some links will have a stress equal to the number of packet duplications plus one (one packet arrives, each duplicate leaves). For example, Figure 3-1 shows a simple application-level multicasting tree in one link, $R_4E_1$, has a stress of three another link, $R_3R_4$, has a stress of two.

Stretch is also a problem in Figure 3-1. $E_2$ receives packets only after they have traversed eight links, rather than the four of a direct unicast. $E_3$ must wait for six traversals instead of four. Assuming unit latencies, they would experience a stretch of 2.0 and 1.5 respectively.

In the remainder of this chapter, we will see how packet reflection can reduce stress and stretch in these situations. Most examples will describe a multicasting overlay system both because multicast is a common application of overlay networks and

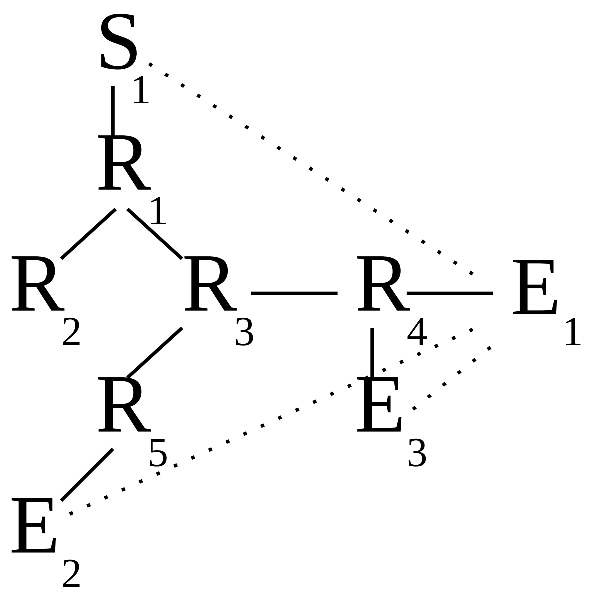

Figure 3-1: An application-level multicast distribution tree. Packets are sent from the source $S_1$ to end host $E_1$ through routers $R_1$, $R_3$, and $R_4$. $E_1$ sends the packets on to $E_2$ and $E_3$.

because unicasting overlay networks can be seen as a degenerate case of a multicasting system.

## 3.1 Reducing stress with reflection

IP routers perform a simple operation on most packets: when a packet arrives, lookup the destination IP address in a routing table, and use the resulting entry to choose an interface on which to emit the packet. An overlay node, acting as an overlay router, performs a similar operation. Overlay routers determine the overlay address and forward the packet, using IP unicast, to the next overlay node. The next node is, again, determined by consulting a routing table. In order to perform multicasts, these routing tables may contain multiple next hops for a single overlay destination address.

The stylized nature of this operation suggests that it could be succinctly described so that another node could perform the operation instead. An end host may ask "the network" to perform *packet reflection* on its behalf. In Figure 3-2 end host $E_1$ directs a reflection request toward $S_1$, which takes it to router $R_4$. This optimization allevi-

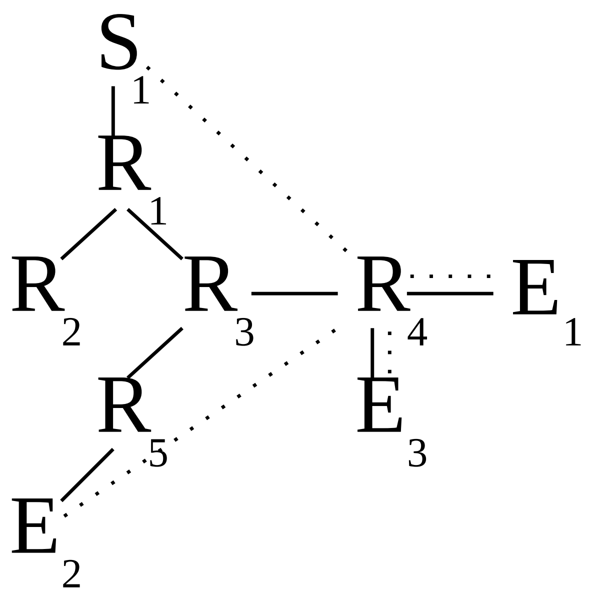

Figure 3-2: End host $E_1$ avoids overloading link $R_4E_1$ by sending $reflect(S_1 \rightarrow E_1, 1, \{(E_1 \rightarrow E_2, 0), (E_1 \rightarrow E_3\}, 0))$ to $R_4$. $R_4$ will now duplicate packets for $E_1$ from $S_1$, sending copies to $E_2$ and $E_3$. In both duplicates the source will be $E_1$.

ates the excess stress on link $R_4E_1$. In addition to performing requested reflections, routers continue to forward packets using their normal forwarding rules. Thus, $E_1$ will continue to receive all packets addressed to it.

The format of a reflection request is denoted $reflect(S \rightarrow D, T, \{(S_i \rightarrow D_i, t_i\})$. Such a request will be addressed to $S$, and rely on routing symmetry to direct it to routers that can fulfill the request. This notation should be read as, "When a reflectable (IP Protocol = REFLECT) packet arrives matching the inbound flow identifier $S \rightarrow D$, duplicate it once for each outbound flow identifier $S_i \rightarrow D_i$. Rewrite the source and destination in each duplicate and emit each, tagged with the associated $t_i$. Emit the original packet tagged with $T$." Tags are used to ensure that nodes know when their reflection requests have been honored and are described in detail in Section 3.3. The operation of a router receiving a reflection request and handling packets that match the request are formalized in Rules 1 and 2.

**Rule 1** *Upon agreeing to a reflection request, the router shall install a reflection table entry as per the request. An entry is keyed by source and destination addresses and ports. A table entry contains a* success *tag, and a number of copy entries. Each copy*

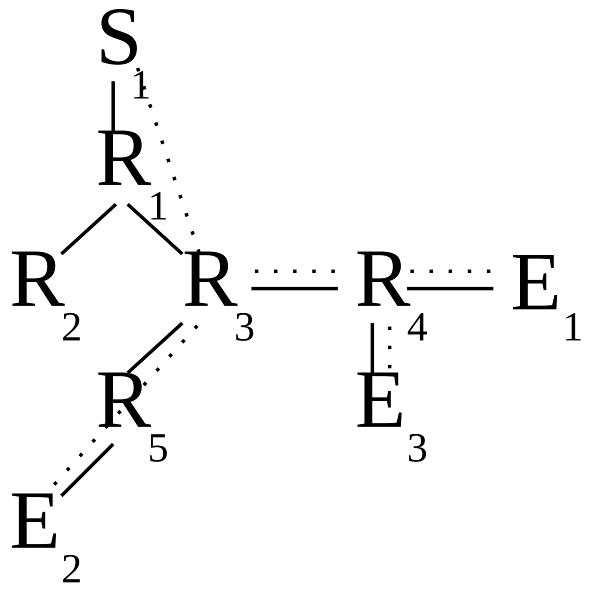

Figure 3-3: Router $R_4$ avoids overloading link $R_3R_4$ by sending $reflect(S_1 \rightarrow E_1, \mathbf{2}, \{(E_1 \rightarrow E_2, 0)\})$ to $R_3$. Note that the tag has been incremented, and one copy has been eliminated from the original request from $E_1$.

*entry contains new source and destination addresses and ports, and a tag for each copy. (In future rules, "address" will be shorthand for "address and port".)*

**Rule 2** *When forwarding an IP packet, if a reflectable packet matches a reflection table key, make one copy for each copy entry in the table entry. Each copy receives new source and destination addresses and a tag. The original packet is tagged with the success tag of the reflection table entry. All packets, including the original, are forwarded by normal unicast rules.*

As seen so far, packet reflection allows end hosts to avoid wasted packets on the link between themselves and the nearest router to them. Although this optimization is useful, greater utility is achieved when routers themselves make reflection requests. A router that has been asked to reflect a packet out the same interface on which it is received may pass on a similar reflection request. In Figure 3-3, router $R_4$ takes advantage of packet reflection by propagating part of its responsibility to reflect packets. By pushing a request similar to $E_1$'s original request on to $R_3$, $R_4$ avoids work and (more importantly) alleviates the stress on link $R_3R_4$.

**Rule 3** *Upon receiving a reflection request, a router shall mark each copy entry* `NORMAL`*.* `NORMAL` *entries shall be treated as in Rule 2. The router may make a new reflection request that asks for some copies to made on its behalf. Requested copies are marked* `DEMANDED`*. The success tag associated with the new request is recorded in the reflection table entry for the request. The tag is the entry's* expected *tag, which is distinct from the* success *tag to be written into the original packet before forwarding. When a packet arrives with the* expected *tag,* `DEMANDED` *copies are not made.*

Of course, if $R_3$ performs the reflection to $E_2$, $R_4$ should not. Tags allow $R_4$ to know whether a given packet has already been reflected by $R_3$. This mechanism is formalized in Rule 3 and described in more detail in Section 3.3.

## 3.2 Anatomy of a reflection request

Outside of this section, this thesis refers to reflection "requests". A request, however, actually consists of a three packet handshake. An `ASK` packet initiates the request. It contains a list of copies that the requester would like made on its behalf. Each copy represents a packet that should be generated in response to the observation of a copy meeting the reflection's match criteria. When the `ASK` reaches a router that supports reflection, it responds with an `OFFER`. An `OFFER` contains a subset of the copies requested in the `ASK`, and a cryptographically generated nonce. The subset of copies are those copies that the router is actually willing to make on the requester's behalf. Finally, the original requester finishes the request with a `DEMAND` packet which contains some subset of the copies from the `OFFER`, and echoes the nonce back to the router that made the `OFFER`.

All reflection packets have the structure depicted in Figure 3-4. Many fields have the same meaning in all packets. For example, the Source Address/Port and Destination Address/Port always refer to IP address and ports of the packets that will match the reflection request. The Success Tag is always the value to write into packets that match the reflection request after fulfilling the request. Tags are described in detail in Section 3.3. The Copy Count is the number of five-tuples to follow. Those

| Reflect Request |
|---|
| IP Source<br>IP Destination<br>IP Protocol = REFLECT |
| Opcode = `ASK|OFFER|DEMAND`<br>Source Address<br>Source Port<br>Destination Address<br>Destination Port<br>Success Tag<br>Nonce<br>Copy Count<br>Copy Source Address 1<br>Copy Destination Address 1<br>Copy Source Port 1<br>Copy Destination Port 1<br>Copy Tag 1<br>... |

Figure 3-4: Detailed contents of reflect packets.

tuples describe the copies that should be created in response to observing a packet that matches the Source and Destination fields. The tuples contain new source and destinations for the copies as well as a tag to write into those copies.

### 3.2.1 Ask

A reflection request begins with an `ASK` packet. The IP Destination of an `ASK` will always match the Source Address, because `ASK` packets are sent toward the source of packets they are intended to match. The `ASK` is generally acted upon by a node other than the node named by the IP Destination. Instead, the first reflection-capable router along the way will intercept it.

The Nonce is unused in an `ASK`. As seen in the following sections, the Nonce is a challenge that is created by the router forming an `OFFER`.

In an `ASK` the copy information is speculative. It contains all copies that the asker would like handled for it. The offerer will decide which of those requests it will handle.

### 3.2.2 Offer

When the `ASK` arrives at a router capable of reflection, an `OFFER` packet is calculated and returned. The `OFFER` packet differs from the `ASK` in three places: IP Source and Destination, Nonce, and Copy Information.

The IP Source and Destination in an `OFFER` indicate the IP address of the router making the offer, and the IP address from the Destination Address field. An `OFFER` packet is *not* necessarily addressed (at the IP level) to the asker. Instead, the asker must intercept the `OFFER` on its return path. This complication is introduced in order to increase the security value of the Nonce.

The Nonce field contains a cryptographically generated integer. The nonce will be echoed back in the `DEMAND` packet, confirming that the demander controls the IP address in question. The nonce should be the result of a one-way hash function run on the match criteria of the request and a router secret. This technique, based on the idea behind SYN cookies [7], will allow the router to confirm a nonce without storing it, preventing a denial of service attack. The nonce prevents the use of reflection to intercept communications that could not be intercepted by other means. That is, an accurate nonce in a forged `DEMAND` implies that the attacker can *already* intercept packets addressed to the victim.

The copy information in an `OFFER` lists the subset of copies from the `ASK` that the router is willing to service. The router might determine this subset in any way it chooses, though Rule 4 is a guideline. Generally, a router should be willing to make a copy if, when consulting its own IP routing table, it determines that the copy would not be emitted on the same interface as a packet that meets the matching criteria of the `ASK`. Rule 4 means that a router would make a copy if doing so would decrease stress on one of its own links.

**Rule 4** *A router shall offer to perform all copies in a reflection request which will require that the copy and the original packet be emitted on the same interface.*

Alternatively, Rule 5 is a *recursive* approach to determining what copies a router should offer. In this formulation, all `ASK` packets would recursively propagate to the source, then a series of `OFFER` packets would propagate back to original asker. Finally, `DEMAND` packets would proceed toward the sender. The recursive approach will push requests further into the network at the cost of more network traffic during setup.

**Rule 5** *After receiving an* `ASK`*, a router shall begin to pass along the reflection request (as per Rule 3)* before *offering a response. The router shall then offer to perform all copies implied by Rule 4 or offered by the next router.*

### 3.2.3 Demand

A `DEMAND` is the final phase of a reflection request, and is made by the same node that sent the `ASK`. A `DEMAND` is sent in response to an `OFFER`, and contains the nonce of the `OFFER`. It will also usually contain the same copy information as the `OFFER`. However, this property is not a hard and fast rule. The demander may choose to eliminate some copy requests and, in some cases, must do so in order to maintain the correctness of success tags. A rule for constructing `DEMAND` packets is deferred until tags are described.

## 3.3 Tags confirm reflection

Tags are a network feedback mechanism that allows end-hosts to determine when their reflection requests have been honored. They are crucial for correctness, as they allow end-hosts to perform necessary duplications when the network does not.

The propagation of route reflection requests toward the source of the match criteria assumes symmetry in IP routing. Under this assumption, the request, wherever it ends up, will lie on the path from the source to the destination of the inbound flow identifier. This assumption may not hold, however, in some situations. In those cases, asymmetric routing paths exist between two hosts. This asymmetry could allow a packet to arrive at an end-host without passing through the router that would be expected to perform reflection on it.

A similar concern is that routes in the underlying network change as a result of

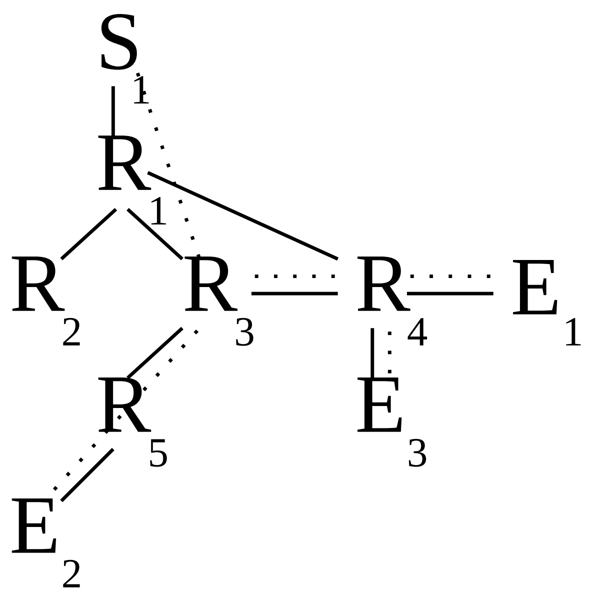

Figure 3-5: A new physical route is brought online between $R_1$ and $R_4$, bypassing the reflection request in $R_3$. $R_4$ notices that it receives untagged packets and performs both copies on its own. $E_1$ is unaffected.

broken links or configuration changes. A reflection request may have propagated to a router that, after a route change, no longer sees the packets that are to be reflected. Figure 3-5 demonstrates this problem.

For correctness in these scenarios, packet reflectors must signal the original destination node when a packet has been successfully reflected. In the absence of such confirmation, the requesting node would perform the reflection on its own, as if packet reflection had not been requested. In addition, the requester might try to reestablish the reflection request. If the problem was caused by a new route, a router on the new path might accept the request.

To implement this signaling mechanism, packet reflection requests contain a *tag*, as do all packets forwarded by the reflection mechanism. When a router performs a reflection, it writes the value of the tag for that reflection request to the original packet, which continues on its way to the original destination. If a packet is received without the appropriate tag, it is clear that duplication did not occur, so the receiver performs the duplication as if it had never made the reflection request.

There are two important phases in the use of success tags in reflection requests.

In the first phase, a requester must choose appropriate success tags for reflection requests. We will see that as a request propagates from router to router the success tag must be changed to avoid ambiguity. The second phase begins once reflection requests have been established. When a router receives a packet for which it has agreed to perform reflection, it must either perform the entire reflection itself or determine that some part of it has already been done, and perform the rest.

### 3.3.1 Establishing tags

The most important goal to keep in mind with respect to the choice of success tags is that the meaning of particular tag must be unambiguous. A secondary goal is that the success tags in successive requests remain unchanged. As we will soon see, certain situations allow the same success tag to be used in successive requests without creating ambiguity. The advantage of doing so is that the effects of route asymmetry and route changes are mitigated.

We begin by assuming that edge hosts use a success tag of one when initiating a request, codified in Rule 6. For example, in Figure 3-2, the first request, $reflect(S_1 \rightarrow E_1, \mathbf{1}, \{(E_1 \rightarrow E_2, 0), (E_1 \rightarrow E_3\}, 0))$, requests that success be indicated by writing a 1 to the original $S_1 \rightarrow E_1$ packet. Zero is reserved to indicate that no reflection has occurred yet. Reflectable packets begin with their success tag set to zero. The important question is how routers should choose success tags when propagating their reflecting responsibility.

**Rule 6** *An end-host shall use a* success *tag of one in its reflection requests.*

Changing success tags is sometimes a necessity. In Figure 3-3, $E_1$ has requested that two copies be made whenever it receives a packet from $S_1$. $R_4$ agreed to perform those copies, but went on to request that $R_3$, in fact, should make one of the copies. The two requests must be:

$$reflect(S_1 \rightarrow E_1, 1, \{(E_1 \rightarrow E_2, 0), (E_1 \rightarrow E_3, 0)\})$$

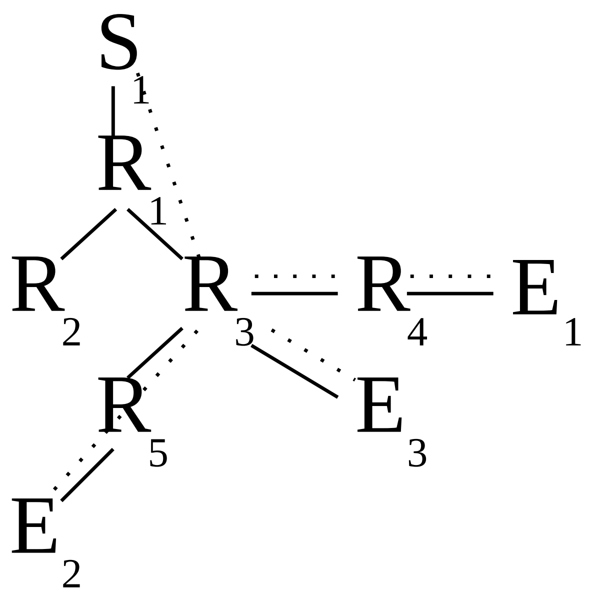

Figure 3-6: A slightly different underlying topology allows $R_4$ to propagate its entire responsibility to $R_3$. In this situation, $R_3$ can make make its request using the same success tag that it has been asked to use.

$$reflect(S_1 \to E_1, \mathbf{2}, \{(E_1 \to E_2, 0)\})$$

Following this request, the $E_1 \to E_2$ copy entry in $R_4$ will be marked `DEMANDED` by Rule 3.

The second request must change the success tag in case the $S_1 \to E_1$ packet emitted by $R_3$ ever makes it to $E_1$ without passing through $R_4$ (due to a route change or a new asymmetric path). $R_4$ is ensuring that $R_3$ will not confuse $E_1$ with a claim that is not true. A success tag of 1 would indicate that both copies have been sent, so $R_3$ must use a different success tag after making only one copy.

Changing success tags, however, is not always a requirement. Compare Figure 3-6 to Figure 3-3. In Figure 3-6 $R_4$ has agreed to the same request from $E_1$, but was able to go on to request that $R_3$ perform both copies on its behalf. In this case, there is no chance of confusion. If the $S_1 \to E_1$ packet emitted by $R_3$ ever makes it to $E_1$ without passing through $R_4$ with a success tag of 1, $E_1$ will not be confused. The claim is accurate: both copies requested by $E_1$ have been made.

The difference between these two cases is that in Figure 3-6, $R_3$ has agreed to perform the exact same reflection request that $R_4$ had previously been assigned.

Under these circumstances, it is reasonable for $R_4$ to go a step further and ask that $R_3$ perform $R_4$'s tagging operation as well. If $R_4$ had used a new tag, then when such a packet arrived, its only required operations would have been to rewrite the success tag to its own value. By reusing its own value in the new request, $R_4$ can simply forward the packet. $R_4$ has arranged matters so that its operation under Rule 3 has led to a degenerate case: the incoming packet's tag is the same as the expected tag, and no copy entries are `NORMAL`, so IP forwarding of the original packet is all that remains to be done.

**Rule 7** *When propagating the request associated with a reflection table entry, a router shall use a new tag unless all copy entries of the reflection table entry are offered by the next router. The new tag shall be chosen by incrementing the* success *tag of the reflection table entry. In such a case, the* expected *tag will be one greater than the* success *tag for that entry.*

There is a significant advantage to avoiding unnecessary tag changes, beyond the ease with which $R_4$ may now forward packets. If a new link were brought up connecting $R_3$ directly to $E_1$, reflection would proceed without any difficulty. The packet would be tagged at $R_3$ in exactly the way that $E_1$ expects, so $E_1$ would correctly detect that its request had been fulfilled. The fact that $R_3$, rather than $R_4$, performed the duplications is irrelevant. If, instead, routers always changed success tags when propagating requests, then $R_4$'s presence would be required between $R_3$ and $E_1$ so that the tag could be transformed.

By the same reasoning that allows packets to skip $R_4$, we can also conclude that $R_4$ may safely throw out the reflection table entry associated with the request. This optional space optimization is codified in Rule 8. Routers should not eliminate this state without cause, however. If, for example, a new route should be added to the network that skips $R_3$ but not $R_4$, it would be beneficial for $R_4$ to still have enough information to perform the necessary duplications. If the state has been eliminated, the packet will not be duplicated until it reaches the application in $E_1$.

**Rule 8** *A router that has successfully passed on an entire reflection request, thereby avoiding the creation of a new success tag by Rule 7, may throw away the state associated with the request without decreasing the efficiency of the overlay under normal circumstances.*

Route asymmetry, however, presents the same difficulty in a more persistent form. For example, if the previously described new link were a one-way link toward $E_1$, then $E_1$'s next request would still go through $R_4$. Only by keeping the success tag unchanged can the network avoid an extra reflection. Of course, sometimes route asymmetry will occur around a router that was forced to change the success tag in its request because it was unable to forward the entire reflection. In these cases, missing the router in question causes a misfire. Although the same duplication may already have been performed, when the packet arrives at the next router the tag will not be correct so the entire reflection must be performed. This problem is described in more depth in Section 3.5.

### 3.3.2 Using tags during reflection

Once a router has agreed to service a reflection request, it is expected to make the appropriate copies or ensure that another router has done so, and then write its success tag to the original packet. In the simplest case the router simply performs the reflection and writes the success tag. The router will always perform the reflection when it has not made a reflection request to any other router. The packet will arrive with no success tag (the field will be set to zero), the router will make the copies, and then forward the original packet to the next hop with the success tag filled as previously requested.

If a router has made a reflection request, then it might face a second case. A packet may arrive with its success tag correctly set to the value requested by the router in its reflection request. In this case, the local router knows that its reflection request has been honored, and certain copies (those labeled `DEMANDED` by Rule 3) have already been made by another router. In some cases, this knowledge will allow the local router to avoid making any copies. No copies will be required when the router was able to make a reflection request to offload its entire responsibility. Other times, the next router will have offered to perform only a portion of the local router's work. In those cases, the local router will still need to make some number of copies (those marked `NORMAL`). Once the router makes any remaining copies, it tags and forwards the original packet.

A final case is possible. A non-zero success tag appears in a packet, but the success tag does not correspond to the value of the router's previous reflection request. For example, the router has requested that a reflection be performed and that 11 be written to the forwarded packet, but the packet arrives with a 13. As we will see later, this mismatch can happen when further routers have made requests to yet more routers, but some router that is expected to perform a reflection has been skipped by the path of the original packet. The value 13 indicates that *some* reflection request was performed on the packet, but the local router cannot know what has occurred, so the entire reflection must be performed, followed by writing the success tag. The local router *always* writes the same success tag because it is confirming that its own reflection request has been completed.

In summary, Rule 9 refines Rule 2 to account for tags.

**Rule 9** *To reflect a packet that does not have a success tag corresponding to the* success *of its reflection table entry, a router shall make all copies, then tag and forward the original. To reflect a packet that does have a matching* success *tag, make all copies that are marked* `NORMAL`*, then tag and forward the original.*

### 3.3.3 Tags in reflection copies

In addition to the tag associated with the request as a whole, reflection requests also contain a tag associated with each outbound flow identifier. This feature is necessitated by the interaction of multiple reflection requests. Suppose that a router has accepted a request: $reflect(A \rightarrow B, 1, \{(B \rightarrow C, 0)\})$. Now, when a router observes a packet going from $A \rightarrow B$, it sends a copy from $B \rightarrow C$ and 1 will be written to the $A \to B$ packet. Now suppose that the router receives another request: $reflect(B \to C, 5, (\{C \to D, 0)\})$. The router now determines that when it receives a packet from $A \to B$, it must send two packets in addition to the original: $B \to C$ and $C \to D$. The router tags the $B \to C$ packet with a 5 so that it is clear to the originator of the second request that its request was honored.

Finally, suppose that the router emits $B \to C$ packets on the same interface that it receives $A \to B$ packets. In that case, it makes a request to its upstream router using the tag associated with the $B \to C$ flow identifier: $reflect(A \to B, 2, \{(B \to C, \mathbf{5})\})$. By doing so, it insures that it is upholding its contract to write a 5 into $B \to C$ packets. The need to associate success tags with copy entries is codified in Rule 10.

**Rule 10** *If a router contains a reflection table entry, $R_1$, in which a copy entry, $C_1$, matches another reflection table entry $R_2$, the copies associated with $R_2$ should be added to the copy entries of $R_1$. The* success *tag of $R_2$ should be written to $C_1$. The newly added copy entries of $R_1$ should be marked* `NORMAL`*, although a subsequent reflection request could change them to* `DEMANDED`*.*

### 3.3.4 Tags complicate DEMAND

Rule 10 is the first rule that places non-zero tags in the copy entries of a reflection table entry. Those non-zero tags complicate matters because they must be used only when the copies they signify are also made. A node must be careful not to make a reflection request that moves the copy entry (and non-zero success tag) to a new router unless it also moves the copies associated with that tag. In order to maintain the correctness of tags, a router may be forced to drop some copies during the demand phase of a request if dependent copies were dropped during the offer phase.

Suppose that a router, R, has agreed to two reflection requests:

$$reflect(A \to B, 1, \{(B \to C, 0)\})$$

$$reflect(B \to C, 1, \{(C \to D, 0), (C \to E, 0)\})$$

From these two requests, R uses Rule 10 to determine that it must make *three* copies when it sees an $A \rightarrow B$ packet. First, it must send a $B \rightarrow C$ copy. Next, it must send $C \rightarrow D$ and $C \rightarrow E$ copies in response to the $B \rightarrow C$ packet it just generated. Furthermore, the $B \rightarrow C$ packet must be tagged with a 1 to confirm that the two extra copies were sent (as per the second reflect request).

Now R decides to save itself work by propagating a reflection request toward B. It intends to request:

$$reflect(A \rightarrow B, 2, \{(B \rightarrow C, 1), (C \rightarrow D, 0), (C \rightarrow E, 0)\})$$

R builds an `ASK` for this request. If the `OFFER` comes back indicating that the offerer is unwilling to make the $C \rightarrow D$ copy, R cannot go ahead with the obvious `DEMAND` that includes the other two copies. If $(B \rightarrow C, 1)$ is demanded, those packets will go out with a success tag equal to 1. That tag would erroneously indicate that $(C \rightarrow D, 0)$ was generated.

R has two choices to assure correctness. It may demand only the $B \rightarrow C$ copy, but ask that it be sent out tagged with a 0, not 1. In this case R should remove the $C \rightarrow E$ copy from its request. If the $C \rightarrow E$ is not removed, it will only cause a misfire when a later router creates the same copy because no success tag has indicated that it has already been done. Alternatively, R may simply drop the request for $B \rightarrow C$ completely. In this case, that would mean dropping the request completely, but a larger request could still have valid copies to be made. The key requirement is to drop the copy whose tag would erroneously indicate that additional copies were also made. This requirement is summarized as Rule 11.

**Rule 11** *A router shall only demand a tagged copy if it also demands the copies that are implied by that tag.*

## 3.4 Multiple reflection requests

It is relatively easy to understand the effects of a single reflection handshake in isolation. The router that services the handshake will begin duplicating some subset of the copies that the end-host originally asked for, and tagging the original packet when it does so. Complications arise as we begin to consider the effects of multiple reflection requests.

### 3.4.1 Chain of requests

As each router passes a reflection request toward the source using Rule 3, a smaller and smaller subset of the copies will be serviced by the accepting routers. As per Rule 4, the subset chosen will relate to branch points in the underlying topology. A router will refuse to perform copies that require emitting the copy on the same interface as the original, so as the request moves toward the source it will contain fewer and fewer copies.

Under these circumstance, one might expect a chain of reflection requests that cause copies to be made as close as possible to the source. This expectation is not always correct, however, due to “local maxima”. A copy may stop at router because the next router in the chain refuses to perform the copy, even though yet another router, closer to the source, *would* be willing to make the copy.

Imagine that a router, $R$, has been asked to reflect a packet to end-host, $E$. The router has three interfaces, and $E$ lies on a separate interface from both the source and destination of the original packet. $R$ will attempt to pass the reflection request further up toward the source, but the next router may consider $R$ to be the next hop for $E$. In such a case, the next router would refuse the request, and $R$ would be left performing the copy. This situation is a reasonable because stress is reduced, but it is certainly possible that there is a “higher” branchpoint, closer to the source that would be preferable in some way, because, for example, it decreases stretch. In that case, the recursive Rule 5 may be preferable to Rule 4.

### 3.4.2 Multiple, separate requests

The interactions of multiple, separate reflection requests can be subtle. If the requests contain common endpoints, they may interact under Rule 10 when their request chains meet at a single router. For example, consider these requests:

$$reflect(A \rightarrow B, 1, \{(B \rightarrow C, 0)\})$$

$$reflect(B \rightarrow C, 1, \{(C \rightarrow D, 0)\})$$

When these requests meet at a single router, the router should set up its reflection tables to act as though it received the following two requests:

$$reflect(A \rightarrow B, 1, \{(B \rightarrow C, 1)\}, \{(C \rightarrow D, 0)\})$$

$$reflect(B \rightarrow C, 1, \{(C \rightarrow D, 0)\})$$

The important rule to keep in mind is that reflection is always an optimization of an existing overlay relay — two unicast transmissions. The interactions of separate requests will never change the character of those transmissions. This property simplifies predictions about the final result of multiple reflection requests. For example, if there are no cycles in the overlay network, no cycles can be induced by using reflection.

Consider that an acyclic overlay network can be seen as a tree, rooted at the origin, $O$, of a packet. In each reflection request made by a node, $X$, in the tree, the match criteria will name a source node that is closer to the origin than the destination of the copies. For example, consider:

$$reflect(S \rightarrow X, \{(X \rightarrow D_1)\}, \{(X \rightarrow D_2), \ldots\})$$

Each destination ($D_1$, $D_2$, ...), is guaranteed to be further from $O$ in the overlay topology than $S$ or $X$. Every time that a copy is made, its new destination is further from the origin in the overlay topology. The tree must have finite depth, so the number of copies is bounded. No cycles are possible.

On the other hand, malicious overlays can also be "optimized". For example, three nodes might make the following requests:

$$reflect(A \to B, 1, \{(B \to C, 0)\})$$

$$reflect(B \to C, 1, \{(C \to A, 0)\})$$

$$reflect(C \to A, 1, \{(A \to B, 0)\})$$

These requests would create a cycle among reflecting routers, just as there is a cycle among the three nodes in the overlay. The duplicate suppression techniques described in Section 3.6 would be expected to mitigate the effects of cycles, however.

## 3.5 Misfires

Misfires are caused when the "success tag" system is unable to convince a node that a packet has already been duplicated. In such cases, the node must perform the duplication again, on its own. One common reason for such a failure is network asymmetry, as seen in Figure 3-7. Others causes include route changes and multipath routing.

In the case of network asymmetry changing success tags in the core of the network, the tag mechanism cannot prevent misfires, but it can address the missed reflection request inside the network. Stress and stretch will still be reduced at the point the asymmetry "comes back together". For example, in Figure 3-7 $R_3R_6$ is a one-way link toward $R_6$. The routes between $S_1$ and $E_1$ are now asymmetric. Although the reflect request in $R_4$ will never be used in this scenario, $R_6$ will detect that it should continue to perform both copies on $E_1$'s behalf because packets will arrive without the success tag that $R_6$ asked $R_4$ to use. Once $R_6$ performs the reflection, it will write its success tag as usual. From the standpoint of $E_1$, it does not matter who performed the reflection. $R_6$'s success tag assures $E_1$ that no more work is required.

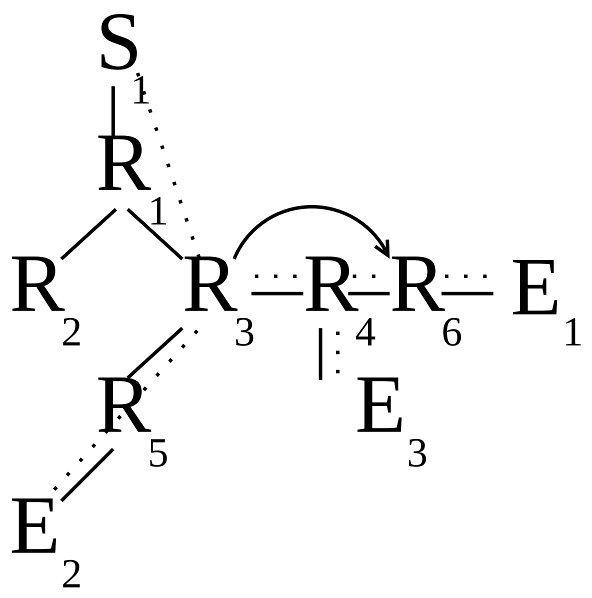

Figure 3-7: Network asymmetry causes a misfire. A one-way link and a new router have been added to the previous topology. $S_1 \rightarrow E_1$ packets now skip $R_4$, arriving at $R_5$ tagged only by $R_3$. $R_5$ recognizes that its request was not properly fulfilled and makes the copies on its own. $E_2$ will receive packets created at $R_3$ and $R_5$.

## 3.6 Preventing repeated reflection

Due to misfires, routers and edge nodes must expect that, in some situations, they will receive duplicate packets. These nodes might be expected to reflect all matching packets that they receive, but it is pointless to reflect the same packet multiple times. A node receiving a packet that it has already reflected should not perform reflection for the packet again.

Two possible designs would allow duplicate detection. In the first, packets contain a sequence number. When a packet is reflected, its sequence number is associated with its reflection table entry. Further packets with equal of lower sequence numbers are duplicates. As usual in designs with sequence numbers of finite size, “greater than” is defined to be circular comparison. This technique faces difficulty in the face of network reorderings, however. If a higher numbered packet arrives before lower numbered packet, the lower numbered packet would not be reflected. Although this is not a problem for correctness because the packet would not receive its success tag and the end-host could perform the reflection, it would result in a loss of efficiency.

For the sake of foiling denial of service attacks, a different approach is possible. Instead of using sequence numbers, each packet could be tagged with a large random number. A router would associate a small cache of recently seen packets with each entry in its reflection table. Again, duplicates would not be reflected. This scheme also faces problems in the face of reorderings, if the number of packets reordered exceeds the size of the cache. Again, this is not a correctness issue, but for a different reason. In this scheme, when there is a cache miss on a duplicate packet, it will be reflected a second time. This addition duplication is unnecessary, but not a problem for correctness.

The random number scheme has an additional benefit compared to the sequence number scheme. It permits a strengthening of design choice that states that duplicates should not be reflected. Instead, packets that have been detected as duplicates may be dropped. The random number scheme produces false negatives, which would allow safe dropping. The sequence number scheme permits occasional false positives which should not be dropped.

## 3.7 Soft state

Routers should maintain reflection table entries for a finite period of time. It is the responsibility of end hosts to repeat reflection requests on a periodic basis in order to maintain the state in each router. When routers receive a refreshing request, they should repeat their own attempt to pass on the request by Rule 3. This thesis does not explore appropriate timeout intervals, though it is expected that timeouts on the scale of minutes would be appropriate.

## 3.8 Security

The security implications of reflection seem, at first, difficult to predict. One wonders if malicious users might use reflection to snoop packets from afar. However, a Rule 12 allows routers to reject reflection requests that might be intended to "reflect" sensitive data to prying eyes.

**Rule 12** *Routers should accept a reflection request on a given interface only if control of that interface would have been sufficient to implement the behavior requested by the reflection request. For example, if packets addressed to A would be emitted through a router's first interface, then requests to reflect packets intended for A should only be accepted on the first interface. Further, the source to be written in the copies for such reflections must be A.*

In addition, the Nonce field ensures that a request is being made by a node that, at the least, already has access to packets destined for the destination. A malicious node can reflect $A$'s packets only if it can intercept $A$'s packets. In such a case, the malicious node already has access to $A$'s data, so reflection would merely optimize the theft.

## 3.9 Deployment

Packet reflection is suited to incremental deployment because there is an immediate gain wherever it is deployed. Even if only a single router implements the primitive, application-level multicast nodes attached to that router can immediately take advantage of packet reflection and save bandwidth on their LAN as well as trimming latency to their neighbors. For example, in the previous example of Figure 3-1, even if only $R_4$ had implemented reflection, stress would still have been reduced to one on link $R_4E_1$. Interestingly, if only $R_3$ had implemented reflection, the reflection would have been exactly as successful as it was with all routers implementing the primitive.

Further encouraging deployment, a router is likely to experience less total load compared to a purely end host based multicast system. If overlay networks become more common, network operators will want to support reflection for their *own* benefit (cheaper provisioning), not just for their customers'. Instead of receiving multiple packets, performing multiple route lookups, and transmitting multiple packets, a reflecting router receives one packet, performs one lookup, and transmits multiple copies of the packet. Additionally, the lookups performed for packet reflection may be faster than a normal routing lookup, as they are exact matches rather than longest-prefix matches.

Packet reflection requests are normal IP datagrams, so requests pass through legacy routers unchanged. If, for example, only the border router of a large organization's network is capable of packet reflection, then all reflection requests for flows originating outside of the organization would make their way to the border router. The effect is that all such flows are short-circuited at the border router, saving the organization from internal resource usage. This design decision simplifies initial deployment.

# Chapter 4

# Path Painting

Path painting enables nodes to set up efficient overlay topologies that resemble the underlying network. Overlay networks generally seek to optimize two attributes of their topologies. First, nodes that are near each other in the physical network should be near each other in the overlay. The alternative is clearly unacceptable. If nearby nodes are *not* connected in the virtual topology then when they communicate packets will need to follow a route through another, distant node in the physical network. Second, virtual links should be, as much as possible, independent of each other in the physical network. Physical independence leads to independent failures of virtual links, and allows the overlay network deal with network characteristics more naturally. For example, when links are independent, a clever overlay network can route around a slow link. But if the route "around" the first link actually shares physical links with the congested link, there may be no gain.

## 4.1 Approach

To build overlays that resemble the underlying network, nodes would like to aggregate locally into small clusters. Then small clusters might further aggregate with nearby small clusters to form larger clusters, and so on. In this way, nodes that are nearby in the underlying network will be nearby in the overlay network. To allow for this aggregation, path painting takes advantage of the fact that, in general, the Internet is organized so that nearby nodes share most of their paths to far away nodes. If nodes could determine which other nodes they share paths with, they could determine which nodes are near them.

This property works at many scales. Locally, the computers of a single university dorm share almost all of their paths, beginning with their LAN. All computers of the university also share most of their paths, though not necessarily the first few hops. Beyond that, all customers of the university's ISP share paths once they reach the ISP, and so on.

To use path painting, all end hosts send paint requests toward an agreed upon rendezvous point. As the requests moves toward the rendezvous point, there are two basic possibilities at each router. First, the request may arrive at a router that has no "color" associated with the requests's rendezvous. In that case, the router becomes colored by the request — the source address and port are noted — and the packet is forwarded as usual. Alternatively, a request may arrive at a router that is already "colored" for the rendezvous of the request. In that case, a notification is sent to two nodes — the source of the current request and the source of the router's current color. The notification contains information about each node, allowing them to make an application level decision about aggregation. In addition, the request is dropped, or "quashed", allowing many nodes to paint toward a given rendezvous without overwhelming it.

Figure 4-1 illustrates the interactions of three paint requests. The first painter emits a paint request which paints all routers on the way to its destination. After that two more painters emit paint requests. Those requests proceed only until they reach a router that has already been painted.

## 4.2 Anatomy of a paint request

A paint request is sent to elicit information about other nodes that share an interest in a given rendezvous. Responses take the form of `NOTIFY` packets containing the addresses of those nodes.

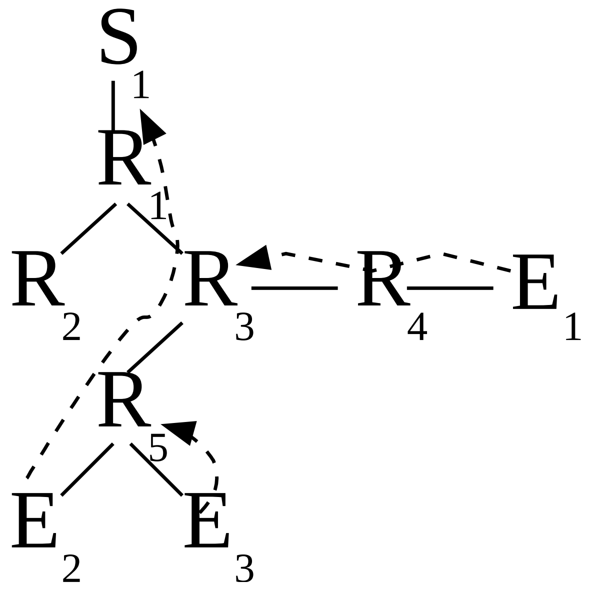

Figure 4-1: Three nodes, $E_1$,$E_2$, and $E_3$ send paint requests to a rendezvous, $S_1$. $E_2$ has sent its request first, so the later requests from $E_1$ and $E_3$ are quashed. After notifications, $E_2$ knows about $E_1$ and $E_3$. $E_1$ and $E_3$ know only of $E_2$.

| Request | Notify |
|---|---|
| IP Source (color)<br>IP Destination (key)<br>IP Protocol = PAINT | (router of collision)<br>(a painter)<br>IP Protocol = PAINT |
| `REQUEST`<br>Concede<br>Ignore Count<br>Ignore 1<br>Ignore 2<br>Ignore 3<br>... | `NOTIFY`<br>Painter Count<br>Painter 1 (router color)<br>Painter 2<br>Painter 3<br>... |

Figure 4-2: Detailed packet contents of paint request and paint notification messages.

### 4.2.1 Request

A simple `REQUEST` packet contains no information in the IP payload except an Opcode, `REQUEST`. The Concede and Ignore lists are optional. They are useful for directing the paint process in greater detail. See Sections 4.3 and 4.4.

### 4.2.2 Notify

A `NOTIFY` packet contains the addresses of nodes that have sent `REQUEST` packets to the same destination. Except as described in Section 4.7, Painter Count will always be set to one or two. Painter 1 will always be the current color of the router sending the `NOTIFY`. When responding to a `REQUEST` from the node that is currently coloring the node, that will be the only Painter in the response. When responding to another node's `REQUEST`, that node's address will be Painter 2.

Once a router has determined the painters that should appear in the `NOTIFY`, it sends a separate copy of the `NOTIFY` to each of them. Rule 13 summarizes.

**Rule 13** *Upon receiving a paint request, a router consults its paint table for the packet's IP destination. If the router is not painted, the request is forwarded normally, the router becomes painted by the requester, and a notification is sent to the requester containing only its own address. If the router was already painted, the request is dropped, and the router sends two notifications, each containing the color of the router and the address of the requester. The router's color is listed first in the packet. The notifications are addressed to the requester and the current color of the router.*

## 4.3 Concede

Under normal circumstances paint requests are *quashed* if they match a request previously made at the same router. Only one "paint color" continues on from an intersection point. Without application hints, the propagated request is arbitrarily chosen to be the first observed color at the router at which the paint requests meet.

To allow applications to choose propagation of a particular request, paint requests may contain an IP address and port in a *concede* field, which indicates that the request should not proceed past a node with a request that originated from *concede*. Rules 14 and 15 provide details.

**Rule 14** *When a router receives a paint request with a filled concede field, the request is first treated normally, as in Rule 13, including forwarding if the paint color is the router's current paint color. In addition, if the request's color matches the current color of the router, the router notes the value of the concession color.*

**Rule 15** *When a router receives a paint request from a color that matches a previously noted concession color, the router changes its color to the concession color before following Rule 13.*

In Figure 4-3, $E_2$ painted first and therefore would be expected to color all routers on the path to $S_1$. Instead, after $R_3$ sent notifications to $E_2$ and $E_1$, they agreed that $E_1$'s paint should continue. $E_2$'s subsequent paint requests contained $E_1$ in their *concede* fields. Such a paint request is forwarded until it arrives at a node that is *not* colored by the requester. In this case, the first such request would proceed to $S_1$, allowing $E_3$ to "win" at $R_3$ and $R_1$ as well. Further such requests would stop at $R_3$.

## 4.4 Ignore

On the other hand, to avoid a denial of service attack from a node that might paint to a rendezvous point but refuse to participate with other nodes painting to the same rendezvous, a request may also contain any number of *ignore* addresses. The paint request should continue even if it encounters a router colored by one of the *ignore* nodes. An overlay node would use an *ignore* list to allow its paint request to continue past an uncooperative node that has colored a router on the path to the rendezvous. In Figure 4-4, $E_2$ is a malicious node that was the first to paint toward $S_1$. Without *ignore*, $E_1$ and $E_3$ would be unable to rendezvous. Their paint requests would be quashed, leaving them without knowledge of any non-malicious nodes.

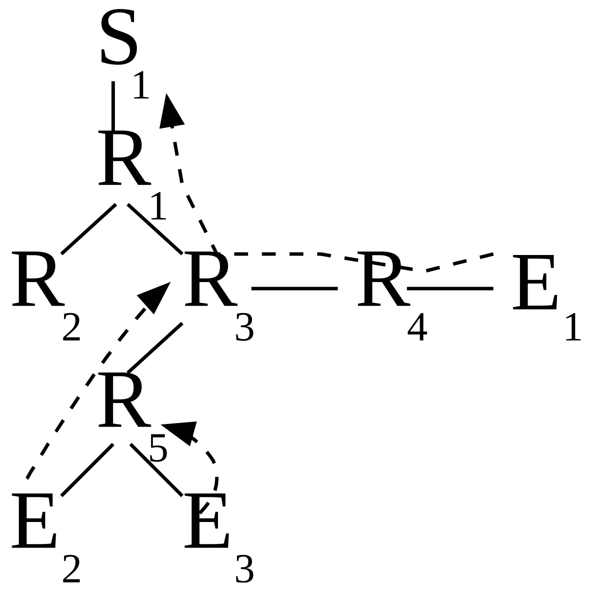

Figure 4-3: As in Figure 4-1, $E_2$ paints $R_3$ before $E_1$ does. However, following notification, $E_1$ and $E_2$ agree (for some application-level reason), that $E_1$'s paint should continue to the rest of the network. $E_2$ will make subsequent paint requests with *concede* set to $E_1$.

When a node is uncooperative, other participants should use *ignore* to allow their paint requests to skip the uncooperative node. In the previous situation, $E_1$ and $E_3$ are expected to detect the difficulty (perhaps because $E_2$ is unable to participate in a cryptographic join mechanism) and add $E_2$ to their paint requests as an ignored color. Subsequent paint requests would operate independently of the malicious node's paint.

Routers must maintain a list of colors, rather than a single color, in order to support *ignore*. For example, in Figure 4-4, $R_3$ must maintain two colors: $E_2 and E_1$. Normally, $R_3$ acts as though it is colored by $E_2$. When a paint request arrives that ignores $E_2$ (as $E_3$'s request does), the router acts as though it is colored by $E_1$. Rule 16 details this operation.

**Rule 16** *A router shall maintain an ordered list of colors. Previous rules shall be followed as if the current color of the router is the first color which is not specified in a request's ignore list. If there is no such color, the request shall be treated as described in Rule 13 when the router is uncolored. In this case, the router shall add*

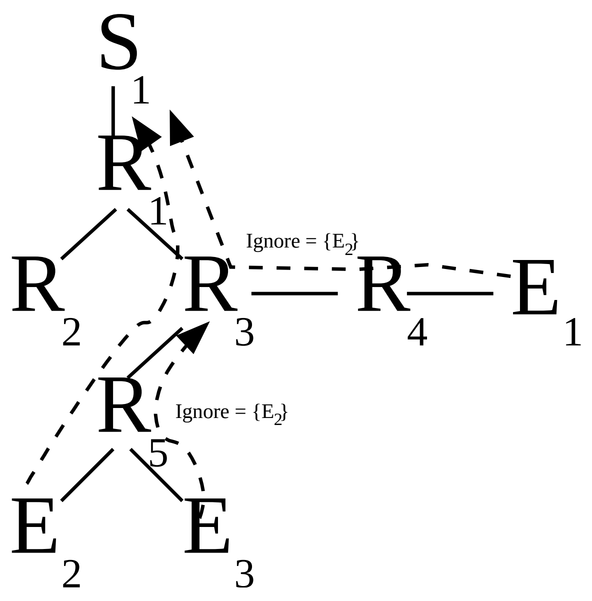

Figure 4-4: As in Figure 4-1, $E_2$ paints first. $E_2$ malicious. When $E_1$ and $E_3$ are unable to establish an application level connection to $E_2$, they begin to include $E_2$ in their paint requests' ignore lists.

*the request's color to the end of its list of colors.*

## 4.5 Soft state

As for reflection, paint state is maintained as soft state. End hosts repeat their paint requests periodically. All paint requests return `NOTIFY` packets as they encounter enabled routers. These notifications act as acknowledgments so that paint requests may be retransmitted in case of loss. This reliability allows end hosts to know when a router's state was last refreshed, so that the time of the next refresh can be determined. This thesis does not explore appropriate timeout intervals, though it is expected that timeouts on the scale of minutes would be appropriate.

## 4.6 Deployment

Like reflection requests, paint requests are normal IP packets and will be propagated by normal IP routers. This decision allows path painting to bridge across regions of the network that have not deployed routers that implement painting. As long as at least one router on the shared portion of two nodes' paths to the rendezvous is "paint capable", information will be gained that will allow the overlay topology to more accurately reflect the underlying topology.

Only when there is no support for painting between a node and a rendezvous point must the node fall back on previous work in building overlay topologies. Alternatively, the rendezvous itself could act like a painting router in this degenerate case. Of course, when there are few paint-capable routers, it may be beneficial to employ some traditional overlay construction techniques. For example, if 300 nodes all come together at a single router, it would be a good idea to perform some traditional topology building techniques to avoid a single node with 300 children [21, 20, 6].

## 4.7 Design options

The format of the `NOTIFY` packet has been left deliberately flexible in order to accommodate a number of slight design variations. In particular, a `NOTIFY` packet may contain information about an arbitrary number of painters even though no more than two painters ever need mentioning in the current proposal.

Two design variations that take advantage of that flexibility are considered below. These variations are not explored in depth, however.

### 4.7.1 Batch notify

When many painters' paths meet at a router, the painter that has colored the router receives many `NOTIFY` packets — one for each painter. Because path painting relies on periodic refreshes, these messages are repeated periodically.

As an optimization, a router could keep track of a set of all painters and send a single `NOTIFY` containing all of their addresses periodically. This choice would require more local state at the router, but that state can be managed if resources are limited.

The router could eliminate the extra state requirement entirely by falling back on unbatched notifications, or it could set a finite limit to the number of addresses stored, sending a batched `NOTIFY` when the limit is reached and flushing its store.

### 4.7.2 Notify all

The previous section outlined the benefits of sending information about all painters when notifying the router's current colorer. There is also a benefit to sending that extra information to all painters.

A malicious node that has painted a router can be bypassed by adding the node to the ignore list of the next paint request. However, when multiple malicious nodes paint a router, this can be a tedious process. It may take several tries, each involving a paint request, a failed attempt to connect to the return node, and a new paint request with an expanded ignore list.

If, instead, a `NOTIFY` message mentioned all painters that have made requests to the router, then an intelligent painter could attempt to contact each painter until an agreeable node is found. That node could report the address of a node that the true group participants have agreed upon as the color of the router. Finally, the new painter would send a second paint request with the appropriate *concede* and *ignore* fields.

# Chapter 5

# Implementation

This chapter explores the implementation of the proposed primitives. In the first section, the implementation used for evaluation is described. This implementation is for the ns [30] network simulator, but also represents the approach that might be taken in a generic, software router. The second section examines how the primitives might be implemented in hardware for very fast routers, comparing closely to the approach used to implement IP Multicast in similar circumstances.

## 5.1 Ns

For simulation purposes, reflection and painting have been implemented in ns, a network simulator.

Ns is an object oriented simulator, written in C++ and OTcl. OTcl is an interpreted language that acts as a front end for the underlying compiled code. New features can be added in C++, OTcl, or a combination of the two languages. After creating a new object type in C++, it may be manipulated using OTcl. The implementation of reflection and paint consists of three new OTcl objects:

**ReflectAgent** Accepts reflection requests from applications

**PaintAgent** Accepts paint requests from applications and allows the application to inspect the results of paint notifies.

**PrimConnector** Inspects all packets entering a node. Acts upon packets that meet reflection or paint matching criteria.

### 5.1.1 ReflectAgent

*ReflectAgent* derives from the ns standard *Agent* class. The Agent class is used as an end-point for communications. For example, other Agents include RenoTcpAgent, VegasTcpAgent, and PingAgent. An agent can be *attached* to an ns node, from which it may send and receive packets.

The ReflectAgent responds to three commands at the OTcl level: `request`, `send`, and `clear`. Scripts use `request` separately for each copy in a reflection request. For example, to prepare to send:

$$reflect(23:1 \rightarrow 36:3, 1, \{(36:5 \rightarrow 42:5, 0), (36:6 \rightarrow 47:6, 0)\})$$

two OTcl commands would be issued:

```
request 23:1 36:3 36:5 42:5
request 23:1 36:3 36:6 47:6
```

Following this preparation, the request may be sent by issuing: `send` with no additional arguments.

`clear` may be used to clear the set of copies that will be requested.

### 5.1.2 PaintAgent

*PaintAgent* also derives from *Agent*. A PaintAgent, like ReflectAgent, supports a `send` command. All ns Agents support the `connect` command to set the end-point to which a packet from that Agent will be sent. In the common case, `connect`ing to a rendezvous is the only preparation necessary before issuing a `send`. However, PaintAgents also support `concede`, `ignore`, and `clear` to manage those attributes in the paint requests that will be sent. The `concede` command takes a single color (IP address and port number), which will be written to the *concede* field of subsequent paint requests made by the PaintAgent. `ignore` takes a list of colors that are written to the *ignore* lists of subsequent requests. `clear` takes no arguments. It removes the colors associated with the *concede* field and *ignore* list.

In addition, PaintAgents support a set of commands for extracting information from the `NOTIFY` packets that are returned to the Agent. OTcl scripts may use `parent` to determine the color of the router that has blocked the further progression of node's paint request. They may also call `children` to obtain a list of the nodes whose paint requests have been blocked by the Agent's own request. These names derive from the common use of this information: a node will treat the node whose color blocked its own color as its parent in a multicast distribution tree.

### 5.1.3 PrimConnector

*PrimConnector* contains the largest fraction of code implementing the proposed primitives. PrimConnector derives from the ns standard *Connector* class, which, at its simplest acts as a a conduit for packets. A Connector is attached to two end-points. Packets arrive from one, and are sent to the other. PrimConnectors are attached to the "front" of every ns node in each simulation. When enabled, PrimConnectors conduct the reflection handshake, maintain paint colors, and operate on any packets that fit the match criteria of existing reflection requests according to rules presented in Chapters 3 and 4.

## 5.2 Hardware

The ns implementation of reflection and painting could easily be ported to software routers, such as Click [23] or Scout [28]. It would also be an appropriate codebase for the addition of the primitives to standard operating systems. High speed routers, however, are likely to have very different requirements.

The data-path of a high speed router should consist of steps that can be implemented in hardware. For the purposes of reflection and painting, this concerns only reflection processing. Path painting and the reflection handshake are control-path operations, and can be assumed to be handled by the routers slow path without

| | Step | IP Multicast | Reflection |
|---|---|---|---|
| 1. | Fast/Slow Path | 4 bits of IP Destination | 8 bit IP Protocol |
| 2. | Exact Match | 32 bit IP Destination | 128 bit match criteria |
| 3. | Duplicate | On output | Must buffer |
| 4. | Rewrites | *n/a* | New source and destination |
| 5. | Unicast Lookups | *n/a* | Longest prefix match |
| 6. | Emit | Up to one copy / interface | Unlimited copies / interface |

Figure 5-1: The steps in IP Multicast forwarding compared to the steps in Reflection processing.

degrading overall performance.

For the most part, the steps involved in processing reflections are quite similar to the steps needed to perform IP Multicast. However, some details differ, allowing IP Multicast to be implemented somewhat more efficiently in hardware. A complete comparison is summarized in Figure 5-1.

The first two steps differ only in the size of the fields that must be checked. Step one must match against the 8 bit IP Protocol field in order to determine that a packet may be reflected, instead of the first 4 bits of the IP Destination Address that determine that a packet is an IP Multicast packet. In step two, reflection must perform a lookup over the 128 bits comprised of 32 bits for each of the IP source and destination addresses combined with 32 bit source and destination ports. IP Multicast uses only the 28 bits remaining in the IP destination to perform its lookups. When this exact match fails in IP Multicast, the packet is dropped. When a reflectable packet does not match a reflection table entry, it is forwarded as a normal IP packet.

Step three is the most significant difference between IP Multicast and packet reflection. In IP Multicast, the number of copies is upper-bounded by the number of output interfaces (minus one). No IP Multicast packet will be emitted on the same interface twice. In contrast, a packet may be copied any number of times on a given output interface during reflection. This difference is quite significant to hardware implementations.

A fast IP Multicast router may perform steps 3-6 at line speed on its output crossbar switch. No packet will be duplicated on the same output interface, so there will be no need for buffering. Furthermore, an IP Multicast router does not actually need to perform steps 4 and 5. All emitted packets are identical, as addressing information is contained solely in the IP Destination.

A reflecting router, on the other hand, must allow for the possibility that packets will be duplicated, but destined for the same output interface. To mitigate this effect, a router may choose to bound the number of copies that will be emitted through the same interface. During the reflection handshake, such a router will perform route lookups on each copy and refuse to offer to reflect more than a bounded number of copies to the same interface. At the same time, the reflecting router may choose to cache the results of the route lookups (the output interfaces obtained) in order to avoid step five when forwarding packets.

We conclude that a fully functional reflecting router can approach, but not meet, the speed of an IP Multicast router. However, by compromising some functionality (rejecting certain requests), a reflecting router can more closely compete with an IP Multicast router. In the case of extremely fast routers, however, it may be difficult to support reflection *or* IP Multicast. If that is the case, reflection has an important advantage — it continues to function. Again, because reflection is incrementally deployable, some routers may ignore reflection requests without severely affecting performance. Section 7.2.2 explores this situation in depth, concluding that support in routers just outside the core of the network (border routers) is as effective as support in the core (transit routers).

# Chapter 6

# Applications

The previous chapters proposed two primitives that overlay networks can take advantage of in order to increase efficiency. These primitives are intended to be flexible, supporting overlay networks of all kinds. To demonstrate that this is the case, this chapter presents various uses of the primitives in real world applications. The first sections explore their use in various ALM systems beginning with a system that emulates the characteristics of IP Multicast and continuing through numerous extensions and modifications.

Packet reflection and path painting were originally conceived with application-level multicast in mind. It is no surprise then, that they are suitable for such applications. Nonetheless, their flexibility and suitability as primitives upon which interesting systems can be built can be assessed by looking at ALM. We will find that, when using the primitives, it is easy to extend a simple IPM-like system to handle heterogeneity and reliability. This flexibility is in stark contrast to IP Multicast, in which support for heterogeneity and reliability represent significant design efforts.

First, we describe a system that mimics the semantics of IP Multicast. In fact, the proposed system will provide a number of improvements over IP Multicast while maintaining the ability to provide the same service model. Second, a heterogeneous multicasting system will be described that has more functionality than similar systems built on IP Multicast (RLM [27], Thin Streams [46]). The versatility and simplicity of these protocols demonstrates the constructive power of the proposed primitives.

Finally, we present a simple, reliable multicast protocol. Its virtue, in fact, is its simplicity. By decomposing and exposing the constituent parts of IP Multicast, we have created two abstractions that, taken together, are more powerful than the monolithic "primitive" of IP Multicast.

After exploring application-level multicast, the final sections demonstrate that the primitives are useful outside of multicast by showing their utility in two other common overlay tasks: two-hop routing and locating nearby nodes.

## 6.1 IP Multicast emulation

We describe first a mapping between features of IP Multicast to elements of the emulation that can be provided with the proposed primitives.

| Feature | IP Multicast | Emulation |
|---|---|---|
| Group address | Class E IP address | (IP address, port) |
| Rendezvous | Core router | End host |
| Join request | Graft message | Paint request |
| Data Path | Routing table | Reflection state |

### 6.1.1 Group joins

A multicast system requires a rendezvous so that various potential group members can come together and share packets. In IP Multicast, the rendezvous point is somewhat hard to pin down. Various protocols (PIM [13], DVMRP [42], CBT [5]) have proposed different rendezvous points. In emulation, a simple approach is taken. The rendezvous is explicitly named as part of the group, much as in SSM or Express. The group name will be the IP address of a suitable rendezvous. A port number is added to the IP address to provide a larger, independently managed namespace.

As in IP Multicast, a join message is sent to the rendezvous point by new group members. In emulation, the join message is a paint request. If the paint request encounters no router that is already painted on its way to the rendezvous, then no action is required; the new node is the only member of the group. If the paint request

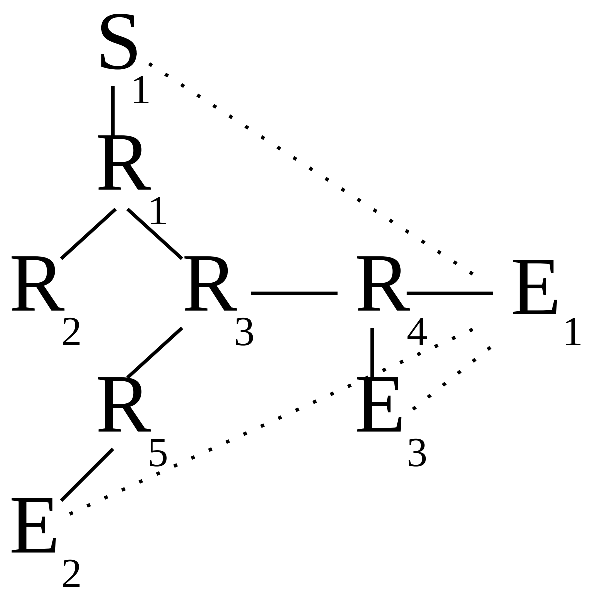

Figure 6-1: An application-level multicast distribution tree.

encounters an already painted router, that router notifies the joining node and the previous painter.

One of these two nodes must become the parent of the other. Various rules are possible, but one rule that appears promising is to set the node nearer to the router at which the collision occurred to be the parent. The nodes can determine their nearness from the TTL field in the collision reports. In case of a tie, any tie breaker is sufficient, such as an ordering on the nodes' IP addresses. More simply, they can rely on the router to select a paint color, which will be the first painter.

Finally, determining the port numbers over which the nodes will converse cements the parent/child relationship. Once the nodes have decided who will be the parent, the child begins sending paint request with *concede* set to the address of the parent. On the other hand, if the other node is uncooperative, the emulation adds the node to the paint request's *ignore* list.

In Figure 6-1, three nodes have joined an emulated multicast group using $S_1$ as a rendezvous. Here we assume that the end-hosts sent paint requests in their natural order ($E_1$, $E_2$, then $E_3$), and that the topology induced by that ordering was acceptable. $E_2$'s paint would have reached $R_3$, which would have notified $E_1$ and $E_2$. The nodes would then set up communication by an application-level protocol, which would include choosing a pair of ports over which they would communicate. $E_3$'s paint would reach $R_4$, leading to a similar exchange between $E_1$ and $E_3$.

In Figure 6-1, the rendezvous, $S_1$, is an active participant in the multicast group. Thus $S_1$ itself acts as a painting router. That is, when $E_1$'s first paint arrives at $S_1$, a `NOTIFY` is sent to $E_1$, informing it that it has reached a router that is colored by $S_1$. $S_1$ and $E_1$ carry out the same protocol to establish a pair of ports to communicate over as $E_1$ and its children did.

The rendezvous does not need to be an active member of the multicast group, however. Suppose $S_1$ were eliminated for Figure 6-1 (but IP routing entries still exist for it in its current location). In such a case, $E_1$ paint would have elicited only the "empty" `NOTIFY` packets from $R_1$, $R_3$, and $R_4$ that tell $E_1$ that it has colored each of those routers. When $E_2$ and $E_3$ sent their paint requests, the group would be formed in exactly the same way as before. The group would consist only of $E_1$, with its children $E_2$ and $E_3$.

### 6.1.2 Forwarding

Whenever a node's overlay neighbors change, whether because the node itself is new to the tree, or because another new node has situated itself as a neighbor, the node sends out new reflection requests. To allow complete connectivity, a node makes as many reflection requests as it has neighbors in the distribution tree. Each neighbor will appear once as a source address on which to match and in all other requests as the source of the copies to be made. For example, in Figure 6-1, after the final node ($E_3$) has joined the tree, $E_1$'s neighbors have changed. $E_1$ would send the following reflection requests:

$$reflect(S_1 \rightarrow E_1, 1, \{(E_1 \rightarrow E_2, 0), (E_1 \rightarrow E_3, 0)\})$$

$$reflect(E_2 \rightarrow E_1, 1, \{(E_1 \rightarrow S_1, 0), (E_1 \rightarrow E_3, 0)\})$$

$$reflect(E_3 \rightarrow E_1, 1, \{(E_1 \rightarrow E_2, 0), (E_1 \rightarrow S_1, 0)\})$$

Any member of the multicast group can send packets to the group. To do so, it sends packets to each of its neighbors. For example, $E_2$ would need to send only one packet, addressed to $E_1$ (on the port they have chosen). $E_1$ would need to send three packets, one to each of its children and to its parent, $S_1$.

### 6.1.3 Distribution trees

The goal of this section is to understand the topology of the distribution trees created by IPM Emulation. In this analysis we will be assuming that all routers implement the primitives and that each router accepts all reflection requests that make sense from a topological perspective. That is, they obey Rule 4 or Rule 5. They do not reject requests due to space concerns, administrative decree, or for any other reason. We also assume that IP routing is symmetric, single-pathed, and stable. This assumption means that the same, single route is always used from $A$ to $B$ and back again.

The first important observation is that only one group member in a stub network will have a neighbor in the distribution tree outside of the stub network. A stub network is a subnetwork that is connected to the portion of the network containing the rendezvous by a single link. By definition, a portion of the network that contains the rendezvous is not a stub network.

As the paint requests of the members in the stub network travel toward the rendezvous, they must all traverse their shared, single link, and thus the same router. We will refer to the router on the stub end of the link as a border router. Only one group member may color the router, and only that group member's paint color will be seen outside of the stub network.

Our second observation is that tags will accurately inform requesters when their reflection requests have been performed. We know that tags has been designed to work if the path of the packet being reflected follows the reverse path of the reflection request propagation. This property exactly describes a symmetric network.

Our next observation is that stress will never exceed one on the link connecting a stub network to the rest of the network. The only possible reasons for a packet to traverse such links are to get *to* the node, X, that has colored the border router or *from* X to one of its (possibly many) children. However, reflection will always be able to eliminate the return packets by pushing the copies to the external router connected to the border router. Consider the propagation of the reflection request. The request must be of the form (tags elided):

$$reflect(A \rightarrow X, \{X \rightarrow B, X \rightarrow C, \ldots\})$$

As the reflection propagates toward the border router, every router will offer to perform the copies that name destinations outside of the stub network (*foreign* nodes). A router would only refuse to perform a copy if its destination is in the same direction a X, which is never the case for foreign nodes as the reflection request propagates away from X toward the border router. Their output interface is always toward the border, away from X.

The fact that a stub network's link has a stress of one is surprisingly powerful because our definition of a stub network is quite broad. For example, if the entire network is a tree (it contains no cycles), then every subtree of the network which excludes the rendezvous is a stub network. This property implies that in such cases, there will be no link with a stress greater than one. Similarly, if a stub network is a tree, stress will never exceed one anywhere in the stub (because each subtree of the stub is a stub). This result is particularly nice when one considers that trees are a common network architecture for networks of small to medium scale. All such networks can expect stress equal to one.

We now explain why a portion of the network that is *not* a tree can have stress greater than one. We first observe that if the copy entries of a reflection request propagated exactly to the router in which the paint color of the destination of the copy collided with the paint color of the requester, the network would, again, see a stress of exactly one. The distribution tree created would consist of all paths from the receivers to the rendezvous, superimposed on one another. Viewed another way, packets would follow the paint trails from the rendezvous back to the group members. Whenever paints collided, packets would be copied.

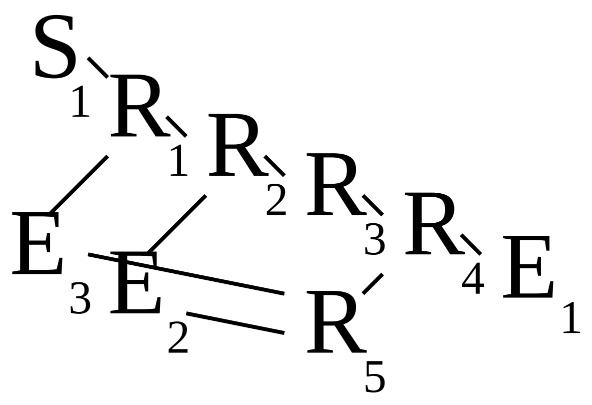

Figure 6-2: $E_1$ has become the parent of $E_2$ and $E_3$. However, $E_1$'s reflection request stops at $R_4$ rather than propagating to $R_1$ and $R_2$ where its paint collided with the paint of $E_3$ and $E_2$.

Figure 6-2 illustrates why copy entries do not always propagate exactly to the location of the paint collision of the requester and the copy destination. Assume that routing is by shortest path and that each link has unit weight. After being the first to paint to $S_1$, $E_1$ finds itself the parent of $E_2$ and $E_3$. It formulates the following reflection request (tags elided):

$$reflect(S_1 \rightarrow E_1, \{E_1 \rightarrow E_2, E_1 \rightarrow E_3\})$$

Following Rule 4 $R_4$ offers to perform both copies. However, when $R_4$ attempts to pass the request further, $R_3$ will not offer to perform either copy. This choice is because $R_3$ would emit the copies on the $R_3R_4$ link, which is the same link on which it will emit the original ($S_1 \rightarrow E_1$) packet. The request has stalled. Furthermore, because it has stalled, the $R_4R_5$ link has a stress of two because it must carry a packet for each of $E_3$ and $E_2$.

We now consider briefly how relaxing our assumptions affects the conclusions we have drawn. First, we discover that the assumption that the primitives have been implemented at every router is stronger than necessary. Only a router that is located at the collision of two paint requests needs to implement the primitives. All other routers will forward all requests unchanged, which is equivalent to a legacy IP router. Of course, in actual deployment it will be impossible to deploy enabled routers at precisely the locations that they might someday be needed, so it is also interesting to ask how well the emulated multicast system will behave in those cases. This question is examined throughout Chapter 7 with simulations on generated topologies.

If we relax our assumption of asymmetry, we lose our observation that the tag system always informs the requester that a reflection has occurred. As such, we lose the observation that the link connecting a stub network to the rest of the network will have a stress equal to one. However, we may continue to conclude that stress will not exceed one on a stub network's link if the stub contains no asymmetry, even if there is asymmetry elsewhere in the network. This observation is justified because we know that the border router will perform any needed reflections, and we know that the requester will receive accurate notification of that fact through the tags system because the notification will be passed through a symmetric subnetwork.

### 6.1.4 Simple extension

Emulating IP Multicast in this way has a number of benefits. First, there is an expanded address space: groups are named by an IP address plus a port. Emulated IP Multicast is also incrementally deployable for the reasons described above. More subtle are the benefits possible through simple extension. Admission control can be determined by the application. For example, a single-source system can be produced simply by having nodes only set up splitting from parents to children, not vice versa.

## 6.2 Heterogeneous multicast

Having built an IP Multicast emulation layer in the previous section, one possible way to handle heterogeneous receivers is to build RLM on top of the IP Multicast emulation. However, a simpler and more featureful system can be built directly by using the proposed primitives.

RLM is implicitly single source. To see why, imagine each source broadcasting over a number of multicast groups. If the groups are the same for each node, then a receiver node will be unable to subscribe to a high-quality stream from one source and a low-quality stream from another. Yet if each node uses a separate set of groups then a receiver node will need to subscribe to (at least) one multicast group for each node in the group. The number of join experiments in the network would grow as O($n^2$), an untenable situation for large groups.

Using the proposed primitives, an overlay can be set up that uses the IP routing infrastructure to do most of its work, but, when necessary, can fall back to explicit forwarding with transcoding over slow links. Using path painting, nodes arrange themselves into an efficient distribution tree as in IP Multicast emulation. Each node makes reflection requests to forward all traffic among its neighbors in the overlay network. Each node also exchanges congestion information with each of its neighbors [4, 2]. If an overlay link is found to be suffering from congestion, then the use of reflection requests to forward along that path is suspended. Instead the stream is thinned at each end of the overlay link and forwarded explicitly. The thinning may take the form of a transcoding to an entirely different lower-fidelity format (as in Video Gateways) or by dropping selected packets that are known to be less important to the reconstruction of the data stream (as in layered multicast approaches).

Allowing stream thinning to occur wherever appropriate creates a system that provides all participants with as much bandwidth as possible to all other participants. No single node or set of nodes has been singled out to receive the best connectivity. When two separate pools of well-connected users are joined by a low-bandwidth connection, the users on each pool will experience high-fidelity contact with the users in their own pool.

## 6.3 Reliable multicast

This section presents a reliable multicast protocol as an extension to the the previous heterogeneous multicast protocol. The protocol presented is simple compared to reliable IP Multicast protocols. This simplicity drives home the fact that application-level multicast systems are flexible, and that they retain their flexibility when using reflection and painting.

The application-level distribution tree is set up as for IP Multicast emulation. Like heterogeneous multicast, out of band communication between parents and children determine the level of transmission that is possible without creating congestion. When possible, reflection is used to increase the efficiency of the transmission, but, as for heterogeneous multicast, pairs of nodes may choose to avoid reflection so that packets may be dropped by the application before the congested link. Unlike heterogeneous multicast, the only possible “thinning” strategy is to drop packets. The intent is to transmit a bitwise correct data stream, so transcoding is not an option.

Those dropped packets will have to be retransmitted, and there are a number of possible strategies, depending on the application. First, if links are not congested, but a packet is lost anyway, it is appropriate for the node that detects the loss to immediately request a retransmission from its parent.

On the other hand, when a link is congested and the packet has been dropped explicitly by the parent, such a strategy is self-defeating. The best strategy is to wait until the congestion has subsided before asking for retransmissions.

Upon receiving a retransmission request, a parent has the opportunity to apply an optimization. If the parent receives multiple retransmission requests from its children, it may be most efficient to request the packet from its own parent, even if it possesses the packet itself. Such a request will cause the packet to be efficiently reflected to all children. The alternative is for the parent to employ iterative unicast to each child.

Reliable multicast systems built on IP Multicast are considerably more complex than the protocol sketched here. Two factors contribute to this complexity. First, IP Multicast hides details, so nodes don’t know who their parent is. Reliable IP Multicast protocols are greatly complicated by this fact. Because receivers know only what group they in, their only option to ask for rebroadcast is to ask the *entire group.* A node doesn’t know who else to ask. Second, because an application-level system is being built for an express purpose, it is possible to design the application-level nodes for the task at hand. Nodes can contain large disks to support retransmissions long after the original transmission. The designers of IP Multicast could not ask routers to buffers large amounts of data just in case someone might want it later.

## 6.4 Primitives in other applications

Although reflection and painting were directly influenced by the needs of application level multicast systems, they were designed with the intention of being useful for other distributed applications. To demonstrate that generality, we present two non-multicast applications that could take advantage of the primitives. These applications are not examined in detail, the intention is only to demonstrate the primitives general utility.

## 6.5 Two-hop routing

The Internet Indirection Infrastructure, i3, provides a flexible network service that allows applications to use logical addresses that can later be associated with one or more physical destinations. It provides this service with a layer of indirection. The logical address resolves to overlay network node that is willing to forward packets to interested parties.

i3 could take advantage of packet reflection to decrease latency for its two hop routes. If i3 is routing packets from X to Y by going through Z, Z would emit a reflect request: $reflect(X \rightarrow Z, Z \rightarrow Y, 1)$. At the very least, this would remove last-hop latency and resource use. Often the request would be pushed well into the network, eliminating much of the overhead associated with the indirection. The deployment of i3 would be greatly simplified because the choice of location for i3 nodes would become far less important. A node behind a 56k line would become a perfectly suitable candidate if routers at the node's ISP supported packet reflection. Results in Section 7.4.2 indicate that using reflection in this way would allow i3 to eliminate nearly all of the associated stretch of a two-hop route in networks that support reflection. Even at low deployment levels, overhead drops considerably.

Two-hop routing is a common feature of overlay networks. RON uses it to route around delays and broken links. Gnutella uses multi-hop routing to return requested data pseudo-anonymously.

## 6.6 Finding nearby nodes

Finding a set of nearby nodes is a common problem in many distributed systems. Content distribution networks would like to select a nearby server to push content to end users. Anycast is usually intended to send packets to a nearby node (not really *any* node). In this section, we describe how one such system, Chord, might use paint to find nearby nodes in order to accelerate its operation.

Chord is a distributed hash lookup primitive upon which many peer-to-peer services can be constructed (for example, i3 is built on top of Chord). Chord's lookup algorithm involves the cooperation of numerous nodes in an overlay network. At each node the lookup algorithm picks a new node to forward the lookup request to. After each hop, the request arrives at a node that is "closer" to the final destination in a logical address space. Eventually, the request arrives at the intended destination and the lookup completes.

Each overlay node that handles a request may know of several potential next hops: nodes whose logical addresses are closer to the intended target than the current node's. However, each node will only know about a limited number of other nodes. For efficient operation, it is preferred that nodes know about nodes that are physically nearby. In this way, lookups proceed quickly through the logical address space because they travel from node to node among physically well-connected nodes.

Chord then, has the following requirement: given a set of nodes, potentially numbers in the thousands, find a subset of those nodes that are physically nearby. Path painting can provide this functionality. Each node in the network will paint toward a small set of agreed upon addresses. Each resulting notification will contain information about a nearby node. In addition, nodes that appear multiple times are expected to be closer than those that appear only once. Once a few nearby nodes are located, those nodes may share information about other nodes that have been located in the same way, until a sufficiently large subset of nearby nodes is discovered. If necessary, that subset may be pruned by conducting measurements to each member of the pool, but path painting has allowed the nodes to find a useful set to begin those measurements upon.

# Chapter 7

# Evaluation

This chapter illustrates the effectiveness of the proposed primitives. The most important measures of effectiveness are decreased stress and stretch, as defined in Chapter 1. The proposed primitives should decrease stress and stretch in all situations, though they can be expected to be most effective when widely deployed.

Resource utilization is an important concern when proposing a network-layer enhancement. The primitives are intended to have modest space requirements at routers. In addition, the space required should scale slowly, if at all, with the size of the group. Again, we can expect full deployments to meet these goals more easily than sparse deployments.

A final metric, "misfires" is of use mainly to assess how well the primitives deal with route asymmetry. Misfires are packets that are reflected more than necessary, leading an edge host to receive duplicates. Misfires, like stress, indicate wasted network resources. However, stress measurements may not account for this waste because of the different paths taken by the extraneous packets.

## 7.1 Simulation methodology

The simulations presented here use highly-connected transit-stub topologies generated by The Georgia Tech Internetwork Topology Models (GT-ITM) [48] and run on the ns-2 [30] network simulator. All simulations were conducted over ten different 100

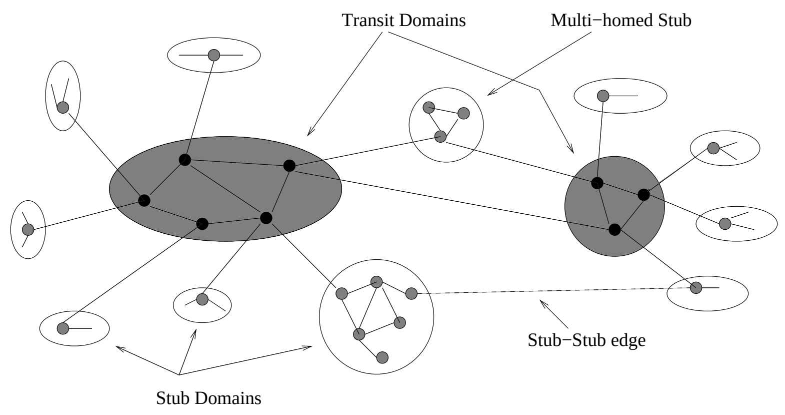

Figure 7-1: A transit-stub topology. In the experiments that follow, the topologies have been extended with an edge node at each router.

router graphs. The parameters used to generate these topologies are from the sample graphs in the GT-ITM distribution; no published work describes parameters that might better model common Internet topologies. Figure 7-1 demonstrates the look of a small transit-stub topology.

Each of the nodes in the GT-ITM graphs models an Internet router. To model end hosts in the simulation, an extra node is added at each router.

## 7.2 Link Stress

The most common metric by which overlay networks are judged, particularly in the context of application-level multicast is stress. In the following experiments stress is measured under a number of deployment and group membership scenarios.

In the first experiment, the number of participating edge nodes is varied while the number of routers supporting the primitives is held constant. Next, to model incremental deployment, group size is held constant as the number of enabled routers is increased. In the incremental deployment tests, a second set of tests compares the performance of random incremental deployment to an intelligent deployment model.

Finally, a separate facet of incremental deployment is tested. In these tests, the issue of local benefit of deployment is examined. In order to encourage deployment, it is important not only that benefits grow with deployment, but that the benefit is conveyed disproportionately to the areas of the network in which deployment occurs

### 7.2.1 Random deployment of enabled routers

The primitives should accomplish two things with respect to stress in an ALM system. First, they should reduce stress. Second, and more subtly, they should allow the application to scale better. Larger group sizes should see slower stress growth.

To test these hypotheses, a simple single-source ALM system, as described in Section 6.1 is simulated. Simulations are conducted at increasing levels of deployment and with multiple group sizes. The routers that support the primitives are chosen at random, as are the members of the multicast groups.

Our first hypothesis is that increased deployment of the primitives can reduce stress in ALM systems. Figure 7-2 demonstrates that link stress is decreased as deployment is increased. Furthermore, it shows that stress reduction is most rapid at the lowest deployment levels. This fact is encouraging as it indicates that early adopters will be well rewarded. Finally, it shows that stress reduction occurs for various group sizes. Large group sizes are affected the most, as their stress levels have the most room to improve, but even small groups show approximately a 30% reduction in stress after 20% deployment.

Our second hypothesis is that the primitives will allow an ALM system to scale more gracefully. As more routers support the primitives, stress should grow more slowly with group size. Figure 7-3 demonstrates this claim. Average link stress is plotted against group size for various levels of deployment. The slope of these plots is clearly decreasing as more routers support reflection and paint, indicating that stress growth is slower as the primitives become more widespread.

### 7.2.2 Intelligent deployment of enabled routers

In the previous experiments enabled routers were chosen randomly. It would be more realistic to expect that network designers would choose to enable routers that would have the greatest effect on overlay optimization.

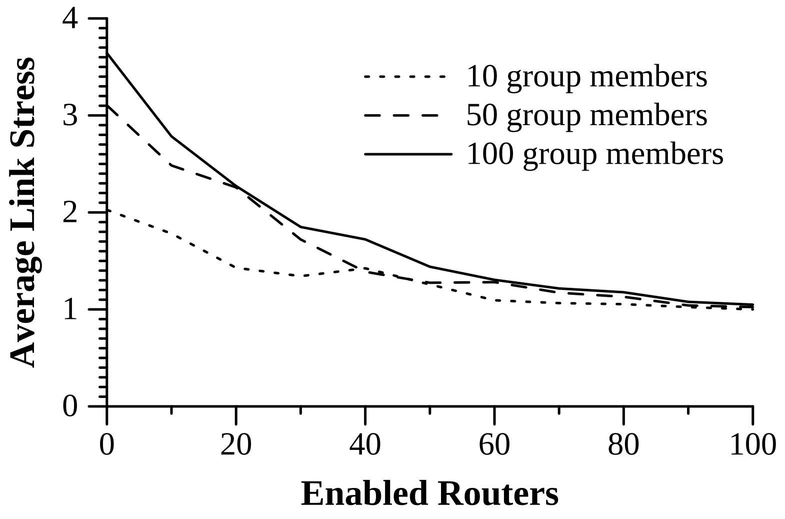

Figure 7-2: Average link stress decreases as more routers implement painting and reflection. The effect of increased deployment is most dramatic for large group sizes. As the network approaches 50% deployment all group sizes approach similar efficiencies.

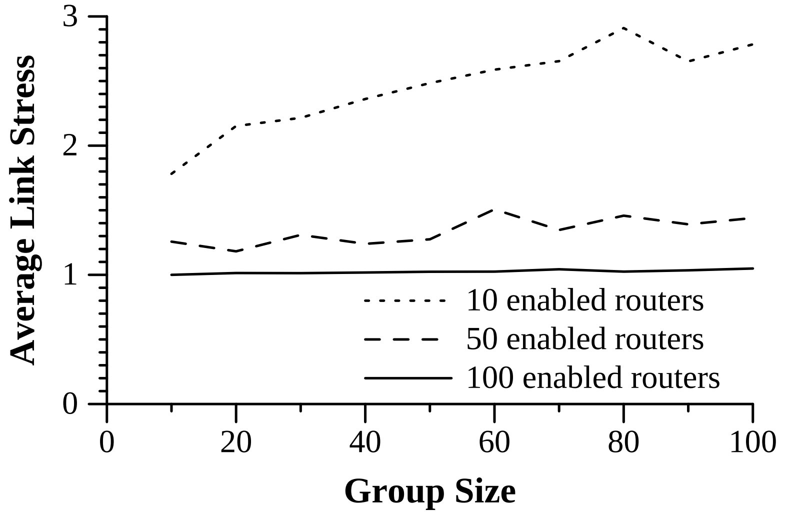

Figure 7-3: Average link stress as multicast group size increases in a 100 node network at various levels of deployment. In well-deployed scenarios, stress is fairly constant (and quite low) as group size grows. With few enabled routers, stress increases with group size.

To understand the effects of intelligent deployment, we compare network stress under four different deployment scenarios. Figure 7-4 shows the results. It is clear that the enabling of certain routers is far more effective than others. Enabling transit routers, the "core" of the simulated network is extremely effective at reducing stress even though they represent only 4% of routers. They are in an excellent location to be effective as rendezvous points for painting, and then to duplicate packets with reflection.

Border routers are nearly as effective as transit routers. When all are enabled, they lie on all of the same paths as the transit routers, thus they can eliminate inefficiencies in nearly all of the same cases. They are slightly less effective for small group sizes because of stresses among the transit routers that cannot be relieved without enabled routers in the core. With larger groups, the border routers become more effective. To understand why, consider the "zone of responsibility" of a transit or border router. In each case, the router effectively isolates a portion of the network allowing a single packet arriving at the router to be duplicated to service all members in its subtree. From that point on, however, there is, essentially, iterated unicast to those members from the router. Small groups have few members in each "zone of responsibility", so stress remains low. As groups size increases, so does the size of each isolated subnetwork, leading to higher stress. Border routers split the network into smaller subnetworks and avoid stressing the border-transit link, thus lowering stress more effectively for larger groups.

The other two deployment scenarios are equally interesting. These scenarios compare two deployment scenarios that involve some randomness, and do not turn on all routers of the given type. In one, half of all border routers are enabled. In the other, 20 stub routers out of a possible 84, are enabled. Despite the considerably larger number of stub routers, the border router strategy is somewhat more effective (though less consistent as shown by the fact that averaging over 10 runs was insufficient to smooth the performance of the strategy). There are two reasons that the stub router strategy is ineffective. First, nothing can be done to ease stress in the core, including the "expanded core" consisting of transit routers and border routers.

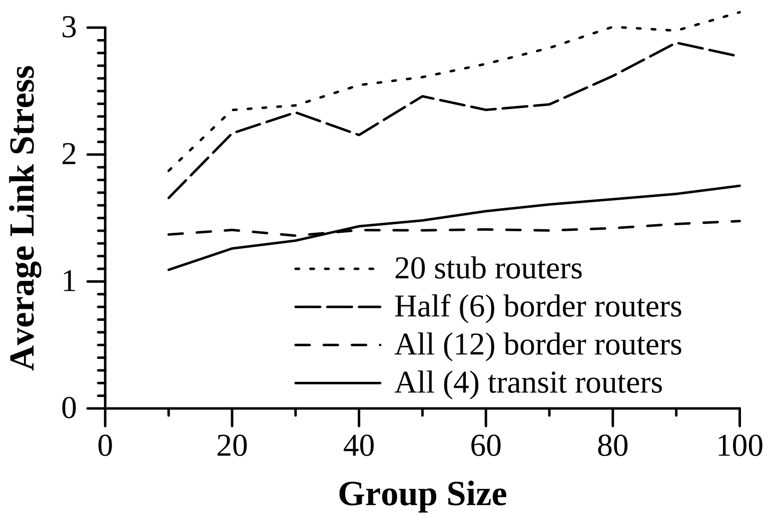

Figure 7-4: Average link stress is lower in more intelligent deployment scenarios. When all border routers or all core routers are enabled, stress is quite low, yet this requires very few enabled routers (4 or 12). Even enabling only half of the border routers allows overlays to create less stress than a random, stub-only enabling that contains over 3 times more enabled routers.

Second, the stub routers will often be ineffective because no group members happen to be located behind them.

### 7.2.3 Local deployment benefits

For organizations that deploy painting and reflection it may be more important to examine how deployment effects their local area of the network rather than the network as a whole. One promise of incremental deployment is that when a network operator deploys the primitives, the portion of network administered by that operator increases in efficiency. Previous experiments have focused on the efficiency of the entire network. To focus on this question, we gather more data from one of the previous experiments. In the "half-border" scenario, half of the border routers were chosen at random for enabling. We now examine, as separate functions, the stress levels in networks with an enabled border router and networks with a legacy border router. For comparison we also replot the data for the network as a whole.

Figure 7-5 shows that the advantages of deployment are gained in the areas of deployment. While this result is hardly surprising, it is important nonetheless. Local

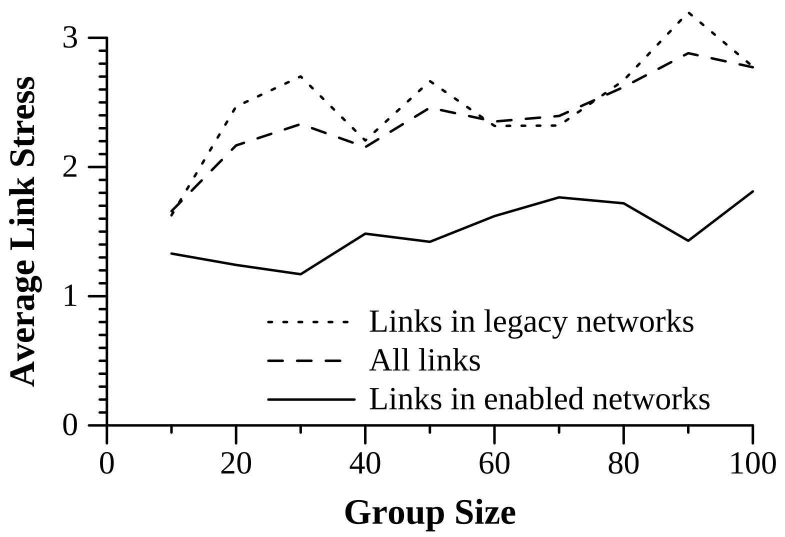

Figure 7-5: Link stress is shown separately for networks that enable their border router, those that do not, and the network as a whole. Networks that enable the proposed primitives see a local decrease in network stress.

advantage is a critical condition for incremental deployment to make economic sense. Investment by network operators in upgrades will be returned to those investors in the form of decreased local waste.

## 7.3 Primitives in isolation

Although reflection and painting are expected to be used together when appropriate, it is interesting to see how effective they are in isolation. In previous ALM experiments, nodes built distribution trees by using paint to learn about nearby nodes that would become parents or children of the painting node. Then reflection was used to iron out the remaining inefficiencies. In the following experiments, we explore the effectiveness ALM systems that use only one of the primitives.

### 7.3.1 Reflection

Figure 7-6 shows the effects of deploying routers that support reflection, but not paint, on the same ALM system described earlier. The only difference from previous experiments is that the overlay topology was built at random. Members were added to the tree sequentially. At the time of the join, the members selected a random

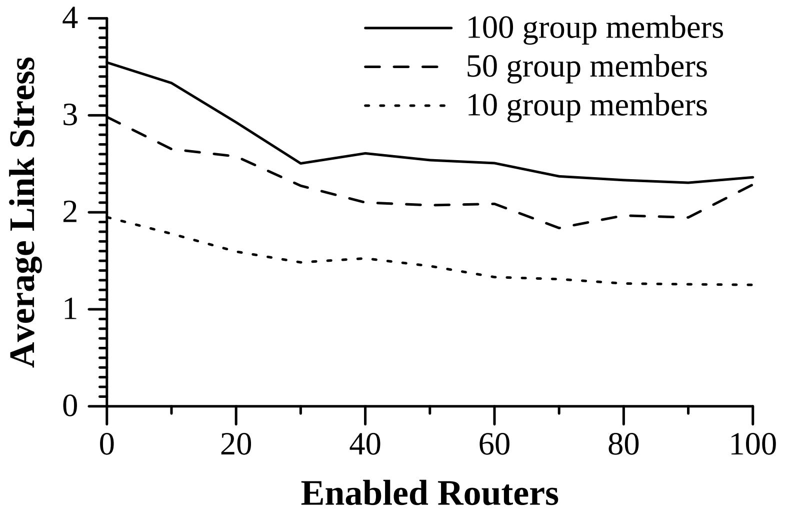

Figure 7-6: Stress as a function of deployment level when performing ALM using reflection to alleviate stress on random distribution trees. (Paint is *not* used.)

existing member of the tree to be its parent.

Reflection is clearly beneficial even in the absence of paint. As more routers are enabled stress is lessened. For small groups, stress can be nearly eliminated using only reflection. For larger groups, however, reflection alone is unable to decrease average stress below approximately 2. Reflection can only alleviate stress when it is caused by a packet traveling into, and then back out, of a portion of the network. When stress is cause by the need to transmit a packet multiple times into a stub network, nothing can be done.

### 7.3.2 Paint

Figure 7-6 shows the effects of deploying routers that support paint, but not reflection. Again, the same basic ALM experiment was conducted. Like reflection, paint appears useful in isolation. In interesting artifact of overlay topologies built with paint is a tendency toward a stress of two. This stress level can be attributed to the fact that building distribution trees with paint makes no effort to eliminate the "in and out" packet paths that reflection would normally be used to eliminate.

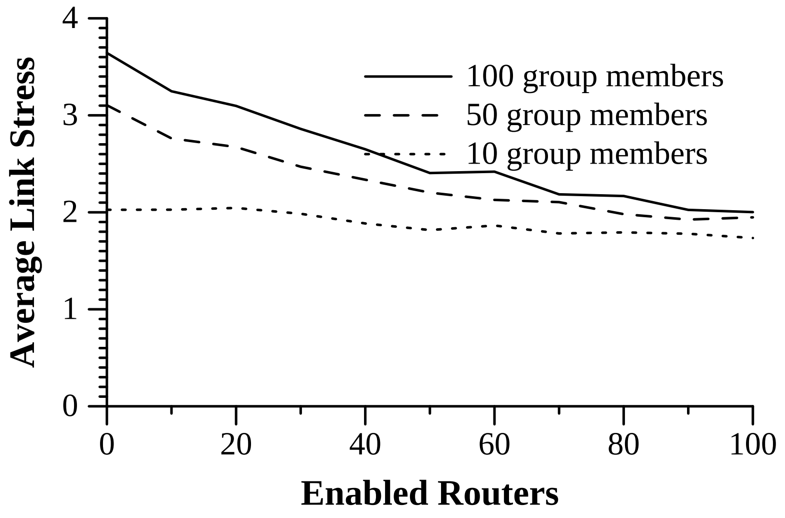

Figure 7-7: Stress as a function of deployment level when performing ALM on distribution trees built using paint. (Reflection is *not* used.)

## 7.4 Stretch

The primitives are also intended to reduce stretch in an overlay network. To evaluate this claim, we look at stretch in two scenarios, ALM and simple two-hop routing as used by RON and i3.

### 7.4.1 Multicast

Stress measures how much an ALM system strains the network, but does not explain how well it is performing its intended purpose: an ALM system should move packets to the members of the group quickly.

We examine the effect of the primitives on latency in a repeat of the experiment of Section 7.2.1. All end-hosts participate in an application-level multicast. A single packet is emitted from the source and arrival times are noted for all group members. For each member, these latencies are compared to the time for IP unicast to transmit from the source directly to the same member. Average stretch is calculated as by computing a stretch for each receiver (latency in ALM divided by latency with unicast) and averaging. The results are graphed in Figure 7-8.

Figure 7-8 shows that, as expected, stretch decreases with increased levels of deployment. At 30% deployment, the primitives have eliminated approximately 60%

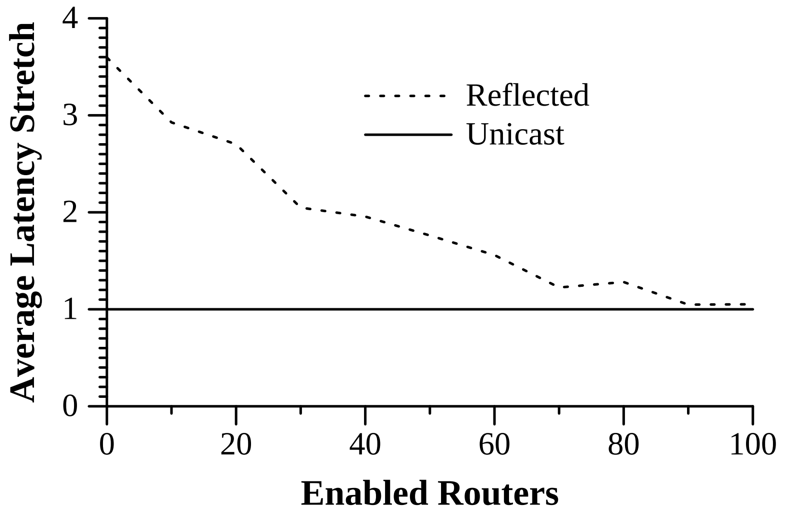

Figure 7-8: Latency stretch compared to iterated IP unicast in the base experiment. As more routers are enabled, latency nears that of IP unicast.

of the latency overhead. At complete deployment, they have eliminated nearly all added latency.

### 7.4.2 RON and i3

RON and i3 share the property that they route packets from A to B using a single intermediate waypoint. In the case of RON, these two-hop routes are used to route around difficulties in the normal IP unicast path, and occasionally to lower latency by finding a fast route that IP routing did not detect. i3 performs the exact same operation, but does so to provide a point of indirection. Overlay nodes rendezvous at the waypoint, thereby reducing the need for knowledge about the other endpoint in the communication.

Reflection can reduce the latency of the two-hop route. The waypoint sends a reflection request that causes the network to route the packet more efficiently:

$$reflect(A \rightarrow waypoint, 1, \{(waypoint \rightarrow B, 0)\})$$

In the following experiment, three nodes are chosen at random to act as the endpoints of communication and the waypoint. Two packet transmissions are timed to

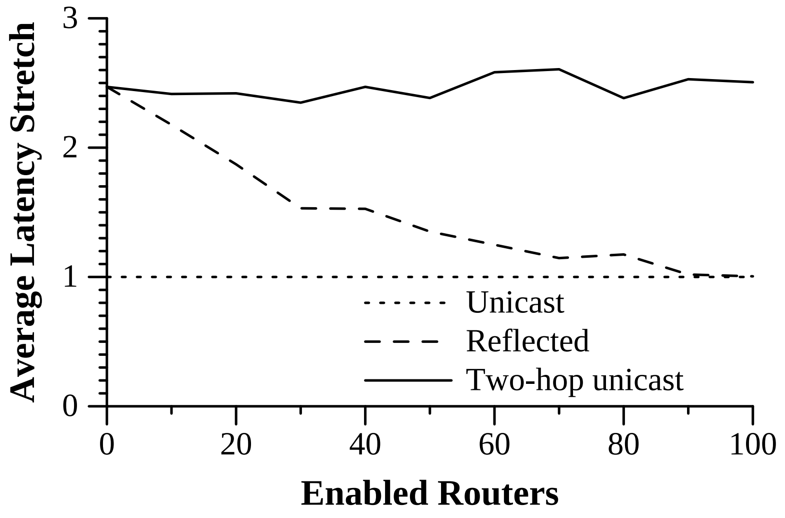

Figure 7-9: Latency comparison for three ways of sending packets from A to B. Unicast is a simple IP unicast, normalized to 1.0. Two-hop unicast consists of two IP unicasts, using a random waypoint. Reflect is the same as the two-hop unicast case, except that the random waypoint uses packet reflection to efficiently forward the first hop unicast to the final destination. As more routers are enabled, using reflection moves from approximating the two-hop case to approximating a direct unicast.

set baselines before testing reflection. First, normal IP unicast between the source and destination; then, a two-hop unicast route that uses the waypoint to route packets. These represent the two extremes of possible performance. Finally, after the waypoint sends its reflection request, a third time is measured that represents how well reflection has reduced the two-hop situation to IP unicast.

As usual, the experiment is conducted over the 10 transit-stub topologies at various levels of deployment. In each experiment, 100 triples are randomly selected for measurement. Two stretches are calculated for each triple using the IP unicast time as a baseline. Figure 7-9 graphs the result.

As deployment increases the performance of the system moves smoothly from that of two-hop unicast case toward that of single unicast. Again, it is encouraging that improvement is fastest at the early stages of deployment. At 30% deployment, the primitives have eliminated well over half of the overhead.

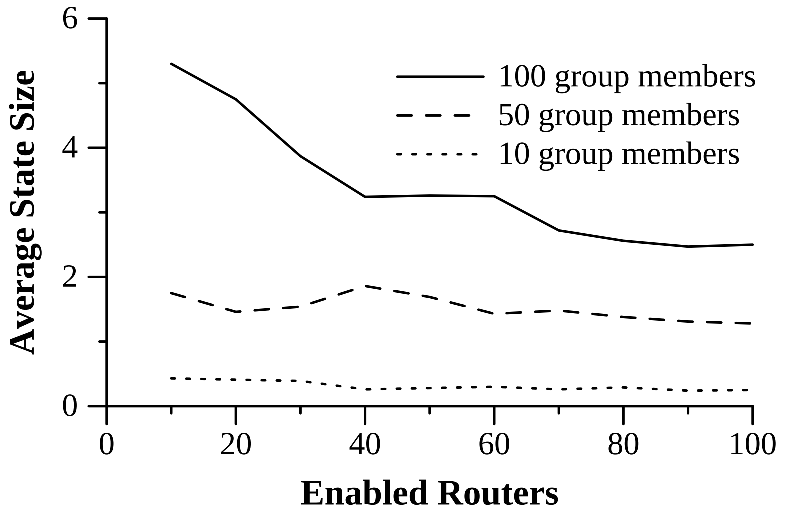

Figure 7-10: Average router state as deployment increases in a 100 node network with various group sizes. Greater deployment decreases the average state required in enabled routers. Greater deployment allows state to be shared among more routers.

## 7.5 Router state

A common concern for IP Multicast is the size of multicast routing tables. Large tables increase cache misses and degrade performance. This section explores the state requirements of the primitives, particularly reflection. Paint has limited state requirements (unless extended as described in Section 4.7.2). Only the current “color” of the router must be stored.

### 7.5.1 Average requirements

To evaluate the state requirements of the primitives, we examine the size of the reflection table in the experiments of Section 7.2.1. Randomly selected groups perform ALM in networks with a random selection of enabled routers.

Figure 7-10 shows that the average state requirements decrease as deployment increases. This result is unsurprising. Consider the difference between two networks, one of which has one extra enabled router. In general, that router will receive some amount of state from its downstream neighbors. The amount of state associated with a single reflection request decreases as it is propagated (because some routers see that they cannot fulfill portions of the request). Therefore the new node is likely

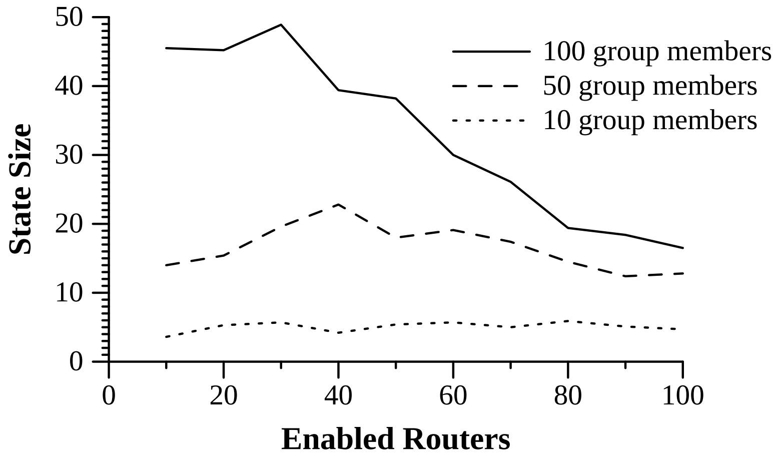

Figure 7-11: Maximum router state as deployment increases in a 100 node network with various group sizes.

to receive less state that its downstream neighbors had, bringing the average down. Furthermore, in small groups, it will be common for added routers to find themselves completely unused, lowering the average state requirements.

### 7.5.2 Maximum requirements

The maximum state required in any router can be as important as average state requirements. If the state requirements are extremely unbalanced, one router may be forced to carry too much information and refuse requests or experience degraded performance. Although the primitives are designed to degrade gracefully under such circumstances, they will surely perform better if they avoid it. In the same experiment as the previous section, we now examine the sizes of the largest reflection table in the network, rather than the average.

Figure 7-11 show two interesting facts. First, for smaller group sizes, the maximum state held in any one router is fairly constant, regardless of deployment levels. Second, large groups disproportionately load single routers at low deployment levels, but the maximum state held in any one router decreases as deployment increases. This decrease indicates that the work is more smoothly shared when more routers are enabled.

By comparing Figure 7-11 to Figure 7-10, we can draw one more conclusion. For a

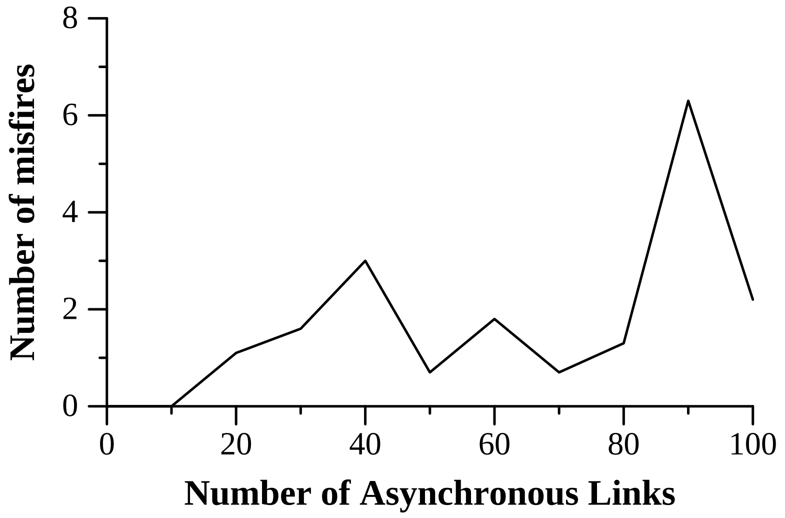

Figure 7-12: The number of misfires in increasingly asymmetric networks. Half of the routers are enabled, and half of the end-hosts are participating in an ALM group. As asymmetry increases, more misfires occur.

given group size, the shapes of the state requirements are similar in each graph. This property indicates that maximum state size requirements tend to move with average state size requirements. This fact gives some encouragement to the notion that in very large networks, such as the Internet, maximum state requirements will remain manageable if deployment is sufficiently high to keep average state size requirements down. This conclusion is speculative, however. The next section explores what can be done when a router finds that its state requirements are excessive.

## 7.6 Reflection misfires

Section 3.5 described the occurrence of "misfires", instances in which packets are duplicated extraneously. These misfires occur when route asymmetry occurs on the portion of a path following duplication, causing the original packet to travel to the reflect requester in a way that prevents success tags from being written correctly.

To study the likelihood of misfires, networks are constructed with intentional asymmetry. Beginning with the same topologies as used previous experiments, links are chosen at random to have their latency doubled in one direction only. These asymmetric links cause asymmetric paths.

Figure 7-12 demonstrates that misfires occur more frequently in asymmetric networks. However, the high variability of the results (despite averaging over ten runs for each data point), implies that this experiment may not be perfectly designed to demonstrate the conditions under which misfires will occur more frequently. The characterization of the asymmetry in networks is difficult, however, and as such, a subject of further study.

# Chapter 8

# Conclusions

Overlay networks are an important way for applications to obtain network behavior that would otherwise require widespread router modifications. By their very nature, it is possible to deploy overlay networks with no additional support. Yet doing so creates inefficiencies. Path painting and packet reflection address those inefficiencies with simple router extensions that can be used in creative ways to perform packet routing and duplication at appropriate locations in the network.

## 8.1 The Good

Packet reflecting and path painting allow end hosts to build efficient overlay networks by exporting the unique capabilities of routers. Packet reflection allows end hosts to benefit from the advantageous position of routers for moving and duplicating packets. Path painting allows end hosts to use routers to learn enough about the network to build efficient overlay topologies.

Both primitives are incrementally deployable. Overlay networks *can* be built without them, but in portions of the network in which they are deployed, these systems will be considerably more efficient.

Furthermore, the focus on incremental deployment has created numerous subsidiary benefits. Routers may choose to ignore requests for any reason, ranging from administrative policies, security concerns, or resource exhaustion. All of these cases are handled gracefully because they are functionally identical to routers that do not support the primitives.

## 8.2 The Bad

It has been difficult to determine exactly how much power the proposed primitives should have. Although keeping them simple has been an important goal, it has been a constant struggle to determine what features would make them "too complex". For example, a simpler version of reflection would not allow tags to be specified in copy entries. This simplification would preclude certain requests, and was therefore deemed worthy of its complexity. On the other hand, other features, described in the next section, have not been added. This tension is an inevitable consequence of refusing the complete generality of active networks in favor of a more manageable approach in hopes of easier adoption.

In depth analysis of the behavior of the primitives in large configurations has been difficult. A number of factors make analysis and understanding more difficult than the equivalent analysis of IP Multicast protocols.

First, routers have much greater flexibility in reacting to reflection requests. For example, Rule 4 and Rule 5 are both reasonable, but they have the opposite effect on the distance that reflection requests propagate.

Second, the flexibility introduced by the decoupling of paint (IPM's join) and reflection (IPM's forwarding) makes it difficult to create trees exactly like those of IP Multicast. In IP Multicast, branchpoints occur where join messages graft to the distribution tree. Using reflection, they occur where the combination of IP routing and individual router decisions push them. The next section outlines an approach that could give reflection requests enough power to control their eventual location based on information gathered during painting.

Finally, incremental deployment (and similar effects, such as resource based rejection of requests) vastly complicates the possible outcomes compared to IP Multicast. Not only must the analysis now account for two types of network nodes, it must also provide a reasonable model of deployment in order to choose deployed nodes. Deployment in the core of the network, for example, has different effects from deployment at the edges. Yet the deployment models we have considered are entirely guesswork.

## 8.3 The Unknown

A number of interesting questions remain to be answered, some of which were implied by the difficulties described in the previous section.

### 8.3.1 Power versus simplicity

Two ideas in merit further consideration to determine whether their value exceeds their complexity. First, a node may wish to exert some control over the rules that routers use to propagate its reflection request. For example, a RON would prefer the iterative Rule 4 to the recursive Rule 5, as the recursive rule would tend to force the two-hop route to match the original one-hop route that the RON is trying to avoid. The iterative rule would eliminate wasted transmissions near the waypoint, without radically altering the path chosen by the RON.

More generally, a node might wish to scope its reflection requests, providing some minimum and maximum distance for them to be propagated. This feature could be used to ensure that a reflection request propagates exactly to the router that responded to the node's paint request thereby reducing the uncertainty involved in the current scheme.

Second, both primitives might benefit from a mechanism which would allow a node to ask for requests to be sent *to* it rather than *from* it. The basic idea behind this notion is to eliminate difficulties due to asymmetric routes by causing requests (reflection and paint) to follow the same paths as the packets they are intended to affect.

### 8.3.2 Security

Both primitives have features designed expressly for added security. Reflection requests contain nonces, designed to prevent the reflection of a node's packets by an untrusted third party. Paint packets contain the ignore list, to avoid denial of service attacks by malicious painters. It is not clear, however, that these mechanisms are sufficient. Further thought and study is required.

In addition, reflection may be used to amplify existing denial of service attacks. Although reflection is acting "properly" by optimizing an operation that the end-hosts could legitimately perform (duplicating and sending packets), ISPs will no doubt wish to limit such activity. Some ISPs may wish to restrict the use of reflection to trusted end-hosts, others may wish to use a dynamic approach that limits the activity of a single reflection entry, or entries submitted from a single end-host. These strategies have not yet been explored.

### 8.3.3 Asymmetry

The final area of future work lies in the study of the primitive's behavior under network asymmetry. Section 7.6 attempts to quantify the effects of asymmetry, but its methodology is crude. Only recently have Internet-like topology generators, such as GT-ITM, come to exist. As such, their ability to create detailed topologies, including latency and routing information is limited. Is is customary to guess at latency figures for the various link types in a topology, and base routing on those figures. Further complicating the picture by adding random asymmetric links gives one little reason to believe that the generated topologies truly resemble Internet topologies with respect to properties like latency and the location of asymmetries. Further research detailing the sources and locations of asymmetry in the Internet is needed before any simulation can be trusted.

# Bibliography

[1] Elan Amir, Steven McCanne, and Hui Zhang. An application level video gateway. In *Proc. ACM Multimedia*, pages 255–266, November 1995.

[2] D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan. System support for bandwidth management and content adaptation in Internet applications. In *Proc. 4th Symposium on Operating Systems Design and Implementation (OSDI '00)*, pages 213–225, October 2000.

[3] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 131–145, October 2001.

[4] Hari Balakrishnan, Hariharan S. Rahul, and Srinivasan Seshan. An integrated congestion management architecture for Internet hosts. In *Proc. ACM SIGCOMM Conference (SIGCOMM '99)*, pages 175–188, September 1999.

[5] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core based trees (CBT) an architecture for scalable inter-domain multicast routing. In *Proc. ACM SIGCOMM Conference (SIGCOMM '93)*, pages 85–95, September 1993.

[6] Suman Banerjee, Bobby Bhattacharjee, and Christopher Kommareddy. Scalable application layer multicast. In *Proc. ACM SIGCOMM Conference (SIGCOMM '02)*, pages 205–220, August 2002.

[7] D. J. Bernstein. SYN cookies. `http://cr.yp.to/syncookies.html`.

[8] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In *Proc. ACM SIGCOMM Conference (SIGCOMM '94)*, pages 24–35, August 1994.

[9] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), 1981.

[10] Yatin Chawathe, Steven McCanne, and Eric Brewer. RMX: Reliable multicast for heterogeneous networks. In *Proc. IEEE Infocom*, pages 795–804, March 2000.

[11] Ian Clarke, Oskar Sandberg, Theodore W. Hong, and Brandon Wiley. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability*, July 2000.

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

[13] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *IEEE/ACM Trans. Networking*, 8(2):85–110, May 1990.

[14] T. Speakman et al. PGM reliable transport protocol specification. RFC 3208, Internet Engineering Task Force, December 2001. `http://www.ietf.org/rfc/rfc3208.txt`.

[15] Sally Floyd, Van Jacobson, Steven McCanne, Chin-Gung Liu, and Lixia Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *Proc. ACM SIGCOMM Conference (SIGCOMM '95)*, August 1995.

[16] Paul Francis. Yoid: Your Own Internet Distribution, April 2000. `www.aciri.org/yoid`.

[17] Gnutella. Gnutella, 2001. `http://gnutella.wego.com/`.

[18] H. Holbrook and B. Cain. Source-specific multicast. Internet draft (work in progress), Internet Engineering Task Force, November 2001. `http://www.ietf.org/internet-drafts/draft-ietf-ssm-arch-00.txt`.

[19] Hugh W. Holbrook and David R. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. In *Proc. ACM SIGCOMM Conference (SIGCOMM '99)*, pages 65–78, September 1999.

[20] Yang hu Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Proc. ACM SIGMETRICS Conference (SIGMETRICS '00)*, June 2000.

[21] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. In *Proc. 4th Symposium on Operating Systems Design and Implementation (OSDI '00)*, pages 197–212, October 2000.

[22] Satish Rao Kirsten Hildrum, John D. Kubiatowicz and Ben Y. Zhao. Distributed object location in a dynamic network. In *Proc. of the 14th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, August 2002.

[23] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Trans. Computer Systems*, 18(3):263–297, August 2000.

[24] Brian Neil Levine, David B. Lavo, and J. J. Garcia-Luna-Aceves. The case for reliable concurrent multicasting using shared ack trees. In *Proc. ACM Multimedia*, pages 365–376, November 1996.

[25] John C. Lin and Sanjoy Paul. Rmtp: A reliable multicast transport protocol. In *Proc. IEEE Infocom*, pages 1414–1424, March 1996.

[26] Chin-Gung Liu, Deborah Estrin, Scott Shenker, and Lixia Zhang. Local error recovery in SRM: Comparison of two approaches. *IEEE/ACM Trans. Networking*, 6(6):686–699, 1998.

[27] Steven McCanne and Van Jacobson. Receiver-driven layered multicast. In *Proc. ACM SIGCOMM Conference (SIGCOMM '96)*, pages 117–130, August 1996.

[28] David Mosberger and Larry L. Peterson. Making paths explicit in the Scout operating system. In *Proc. 2nd Symposium on Operating Systems Design and Implementation (OSDI '96)*, pages 153–167, October 1996.

[29] Jörg Nonnenmacher, Ernst W. Biersack, and Don Towsley. Parity-based loss recovery for reliable multicast transmission. In *Proc. ACM SIGCOMM Conference (SIGCOMM '97)*, pages 289–300, September 1997.

[30] Ns. `http://www.isi.edu/nsnam/ns/`.

[31] Christos Papadopoulos, Guru Parulkar, and George Varghese. An error control scheme for large-scale multicast applications. In *Proc. IEEE Infocom*, pages 1188–1196, March 1998.

[32] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM Conference (SIGCOMM '01)*, pages 161–172, August 2001.

[33] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, pages 329–350, November 2001.

[34] Jonathan M. Smith, Kenneth L. Calvert, Sandra L. Murphy, Hilarie K. Orman, and Larry L. Peterson. Activating networks: a progress report. *IEEE Computer*, 32(4):32–41, April 1999.

[35] Alex C. Snoeren, Kenneth Conley, , and David K. Gifford. Mesh-based content routing using XML. In *Proc. of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 160–173, October 2001.

[36] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. In *Proc. ACM SIGCOMM Conference (SIGCOMM '02)*, pages 73–88, August 2002.

[37] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. ACM SIGCOMM Conference (SIGCOMM '01)*, August 2001.

[38] Ion Stoica, T. S. Eugene Ng, and Hui Zhang. REUNITE: A recursive unicast approach to multicast. In *Proc. IEEE Infocom*, pages 1644–1653, March 2000.

[39] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, January 1997.

[40] Joe Touch and Greg Finn. The Dynabone (white paper). `www.isi.edu/dynabone/`.

[41] Max Vision. Ramen Internet worm analysis. `http://www.whitehats.com/library/worms/ramen/`.

[42] D. Waitzman, C. Partridge, and S. E. Deering. RFC 1075: Distance vector multicast routing protocol, November 1988. Status: EXPERIMENTAL.

[43] Su Wen, James Griffioen, and Kenneth Calvert. Building multicast services from unicast forwarding and ephemeral state. In *OpenArch 01*, March 2001.

[44] David Wetherall. Active network vision and reality: Lessons from a capsule-based system. In *Proc. of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, December 1999.

[45] David Wetherall, John Guttag, and David Tennenhouse. ANTS: Network services without the red tape. *IEEE Computer*, 32(4):42–48, April 1999.

[46] Linda Wu, Rosen Sharma, and Brian Smith. Thin streams: An architecture for multicasting layered video. In *Proc. IEEE International Workshop on Network*

*and Operating System Support for Digital Audio and Video*, pages 173–182, May 1997.

[47] K. Yano and S. McCanne. The breadcrumb forwarding service: A synthesis of PGM and EXPRESS to improve and simplify global IP Multicast. *ACM SIGCOMM Computer Communication Review*, 30(20), 2000.

[48] Ellen W. Zegura, Kenneth L. Calvert, and Samrat Bhattacharjee. How to model an internetwork. In *Proc. IEEE Infocom*, pages 40–52, March 1996.