Network Layer Support for Overlay Networks

John Jannotti <jj@lcs.mit.edu>

January 20, 2002

Abstract

This thesis will describe an approach to supporting overlay networks in the Internet's network layer. Two primitives, Packet Reflection and Path Painting, will be described and evaluated.

Overlay networks represent a flexible and deployable means for applications to obtain new network semantics, but they suffer from some efficiency concerns. To alleviate these concerns while maintaining flexibility, two new primitives are proposed for implementation in the network layer. *Packet Reflection* allows end hosts to request short-circuit packet routing and duplication in nearby routers. *Path Painting* allows multiple end hosts to determine where their disparate paths to a rendezvous point meet, in order to facilitate overlay topology building.

These primitives have been designed to allow their benefit to be gained locally, promoting incremental deployment. Numerous applications of these primitives are considered to demonstrate their utility - application level multicast systems with various semantics, an extended Resilient Overlay Network with greater latency benefits, and a hierarchy of web caches.

1 Introduction

Increasingly, distributed applications would be well served by richer semantics than the network layer supplied by the Internet. Today, and for the foreseeable future, distributed applications have only one primitive from which they may build services - best-effort, unicast delivery of datagrams. But many applications would benefit from additional services. Mission critical applications would like control over the way their packets are routed - perhaps trading off resource usage for reliability by using multipath routing. Teleconferencing applications, chat rooms, and Internet broadcasting systems would like to perform 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 cache data throughout the network.

One approach to addressing these needs is to build a new network service into routers across the Internet that addressing the problem at hand. Generally, this has three drawbacks. First, it may be inappropriate to add the necessary functionality to routers. For example, a content distribution network would likely hope to store large amounts of data on stable storage at the router. Second, it may require universal deployment before it is useful.

IP Multicast has suffered greatly from this problem. It is generally pointless to build widely distributed applications that require IP Multicast because the large IP network providers refuse to transit IP Multicast packets. Widespread demand *might* change their minds, but widespread demand is unlikely without applications that require it. Finally, even if all goes as planned, only one problem has been addressed. It is all too easy to tailor a solution to the demands of a single class of applications leaving others to hope for an entirely separate effort to support them. Again, IP Multicast provides an 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.

Overlay networks provide a way to completely sidestep these issues. Several research groups and service providers are avoiding the issue of "dumb" IP routers by performing packet routing and duplication in edges nodes. These include RON [2], RMX [6], Yoid [8], X-bone [11], and Overcast [9]. In these systems, cooperating servers are sprinkled throughout the Internet to be used as waypoints in a virtual network. Packets are transmitted between these virtual routers using the underlying unicast mechanism provided by IP, but the servers may be tuned arbitrarily 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.

Yet overlay networks face challenges of their own. First, it can be difficult to build virtual topologies that resemble the topology of the underlying network. For obvious reasons it tends to be beneficial for the virtual links of an overlay network to connect nodes that are well-connected on the underlying network. It is also common to prefer that virtual links share as few underlying links as possible with other virtual links. This leads to independent failures, and less duplicate traffic on underlying links. Second, overlay networks operate at a disadvantage to router-based systems because of the physical location of their computational elements. This is both a performance problem, packets go out of their way to reach the overlay nodes; and 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 group join mechanism relies on the ability of routers to observe passing messages.

A single set of 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 should be a top priority in wide area network design.

The contributions of this thesis will be:

- A proposed set of design criteria and implementation techniques for practical Internet extension stressing incremental deployment, coexistence with existing protocols, and versatility of application.

- *Packet Reflection*: a new primitive by which applications may request packet routing and duplication to occur in the routers where there are most efficient. Packet Reflection is incrementally deployable and flexible enough to suggest multiple uses.

- *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. Painting is also incrementally deployable and of general utility to distributed applications.

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

- A demonstration that the primitives are more generally useful, in the form of two examples outside of the realm of multicast. A Resilient Overlay Network (RON) could offer better latency improvements by using Packet Reflection and a web cache hierarchy could be automatically built using Path Painting.

Section 2 describes design requirements for network support of overlays. In Section 3, Packet Reflection and Path Painting are described in detail. Section 4 examines how these primitives may be used to implement various multicast semantics; including the semantics of IP Multicast, a system that copes well with heterogeneous receivers, and a novel hybrid of real-time and reliable multicast — *Fast Then Correct* multicast. Section 5 describes alternative uses for these new primitives, demonstrating their general utility. Section 6 covers the details of the proposed primitives by presenting a pseudocode sketch of the existing NS implementation. Section 7 describes the experiments that will conducted to evaluate the utility of the primitives. Section 10 describes related work. Section 8 mentions additional topics that have not yet been integrated into the body of the proposal.

2 Overlay Requirements

After considering a number of overlay network systems (see Section 10), a number of patterns emerge that indicate areas in which network support could simultaneously help many 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, but every overlay system would benefit if routing and duplication could be directed by the end hosts, but *performed* by the appropriate routers. Second, all of these systems have difficulties constructing overlay topologies. RMX uses hand configuration, End System Multicast lacks scalability due, in part, to it topology generation algorithm. X-Bone and Yoid need help pruning an initial quadratic number of possible links down to a manageable size. Overcast avoids scalability issues through the use of hierarchy, but consumes bandwidth to conduct constant network measurements.

A third, more subtle requirement, is that whatever an overlay network is created, and however much of the functionality is pushed into the IP network, the virtual nodes must retain complete control of forwarding when necessary. It would be unacceptable, for example, to simply allow overlay nodes to set up a forwarding mesh that operated in the core of the network, functioning much like IP Multicast. 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.

In addition to these needs, the problems that previous router modification proposals have faced should be considered. In particular, it is critical that new functionality be designed in such a way as to allow incremental deployment. A local portion of the network should derive benefits from deployment even if the Internet at large is unchanged. It is also important to keep proposed enhancements small so that future modifications to their behavior is unlikely to be required, and generally useful so that greater utility might someday be obtained by using then in novel combinations.

3 Primitives

This section proposes two primitives that meet the needs of overlay networks outlined in the previous section. These primitives provide end hosts with the same capabilities that routers obtain by being fortuitously located, yet allow more interesting applications because control is retained in end hosts which may evolve far more rapidly than the router infrastructure.

First, *packet reflection* exposes the ability of routers to route and duplicate packets at the appropriate location in the network. Consider a packet leaving host X, intended for host Y and Z. It is best to duplicate that packet at the final point shared by the two paths the packet would have taken if unicast separately from X to Y and Z. That point, of course, is a router. If an application-level system is running only on X, Y and, Z, then it hard to see how the duplication can occur in the appropriate router. If the packet is sent to Y, and Y relays it to Z, the packet will travel twice on the links near Y. Packet reflection allow Y avoid this inefficiency by requesting that the duplication occur inside the network. It is called *reflection* because, it is used to allow Y to "reflect" the duplication point back against the path of the packet, from Y itself, back to an appropriate router.

Second, *path painting* exposes the ability of routers to examine routed traffic. IP Multicast can build and maintain spanning trees by examining join messages as they work their way through the network. End host based systems have no such ability. There is no general way to determine the topology of the nearby network, so it is difficult to determine where a new host should attach to an existing overlay topology. Path painting allows end hosts to "paint" the path from themselves to a rendezvous point. When another end host paints the same path, a notification is sent and the nodes may coordinate to graft the new end host to the overlay topology.

3.1 Packet Reflection

In an overlay network each node carries out explicit unicast communication with its neighbors in the topology. When multicasting on an overlay network, this requires packet duplication to occur in end hosts, which, in turn, forces essentially equivalent packets to be retransmitted on the same link multiple times. For example, Figure 1 shows a simple application level multicasting tree in which two links, R_3R_4 and R_4E_1, are used three times and one link, R_3R_5, is used twice.

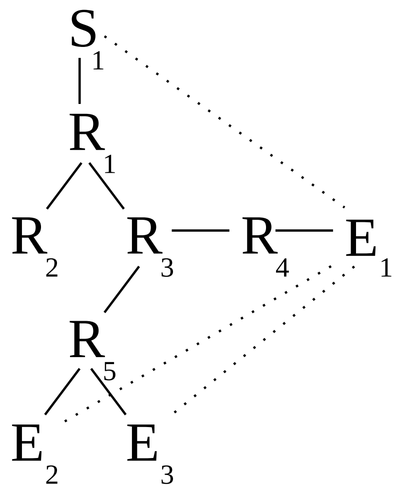

Figure 1: An example application-level multicast distribution tree. Packets are sent from the source S_1 to end host E_1. E_1 sends the packets on to E_2 and E_3.

Fortunately, this retransmission is stylized. When a packet arrives, the appropriate action can be driven by a simple table lookup using a demultiplexing identifier inside the packet. We will refer to that identifier as a *port*, by analogy with TCP and UDP, however its in-packet location could differ. Packets can be identified as belonging to a particular group by inspecting the source and destination IP address and port fields. Concatenating these four fields yields the inbound connection identifier. Looking up the identifier in a table built during the construction of the distribution tree yields a list of outbound connection identifiers. The source IP addresses in these identifiers will all be identical and correspond to the destination IP address of the inbound identifier. For each outbound connection identifier the packet is duplicated and emitted with the new source and destination identifiers.

The stylized nature of this operation suggests a simple optimization. An end host may ask its router to perform *packet reflection* on its behalf. A packet reflection request packages up a piece of the end host's forwarding table and asks the router to perform the forwarding on behalf of the end node. At its simplest, such a request consists of an inbound connection identifier and a list of outbound connection identifiers.

In Figure 2 end host E_1 has requested packet reflection by router R_4. This alleviates link R_4E_1 from carrying the same packet three times (once in, twice out). It does *not* however prevent E_1 from receiving packets addressed to it. In addition to performing requested reflections, routers continue to forward packets using their normal forwarding rules.

The packet reflection primitive is fine-grained for precise control. For example, each specification sets up packet flow from only a single source. To create complete connectivity among four neighbors, an end host would use four reflection requests. Each neighbor would appear once in an inbound connection identifier and three times in an outbound connection identifier. Later sections will demonstrate the use of such versatility.

Packet reflection is suited to incremental deployment because there is a 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

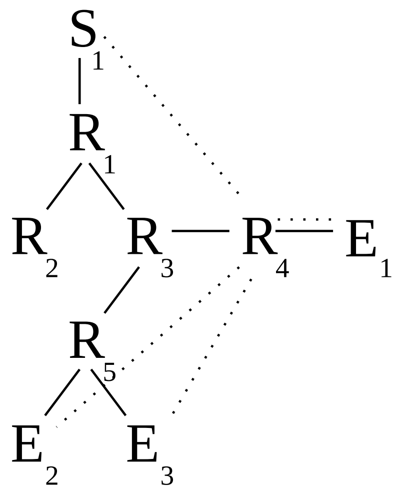

Figure 2: End host E_1 avoids overloading link R_4E_1 by sending reflection request $(S_1E_1, \{E_1E_2, E_1E_3\})$ to R_4. *(port numbers elided for clarity)*

reflection and save bandwidth on their LAN as well as trimming latency to their neighbors. In fact, the router itself is likely to experience less total load compared to a purely end host based multicast. Instead of receiving multiple packets, performing multiple route lookups, and transmitting multiple packets, it receives one packet, performs one lookup, and transmits multiple copies of the packet. Additionally, the lookups performed for packet splitting may be faster than a normal routing lookup, as they are exact matches rather than longest-prefix matches.

Furthermore, because packet reflection requests are normal IP datagrams, requests may 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.

As described thus far, packet reflection allows end hosts to avoid wasted packets on the link between themselves and the nearest router to them. Although this is a useful optimization, its utility increases when routers themselves make reflection requests. A router that has been asked to reflect a packet may find that it has been asked to emit a packet back out the interface on which it was received. In such cases, the router would do well to ask to make a similar reflection request, saving the return transmission. In Figure 3, router R_4 takes advantage of packet reflection by propagating its responsibility to reflect packets. By pushing E_1's original request on to R_3, R_4 avoids work and (more importantly) avoids overusing the R_3R_4 link. After R_3 accepts this responsibility from R_4, R_4 must not reflect packets for E_1.

Generally, at any one time, the Internet routing infrastructure provides only a single path from one host to another and that route is symmetric. The propagation of route reflection requests takes advantage of this fact when it propagates reflection request toward the source IP address of the inbound connection identifier. The assumption is that the request, wherever

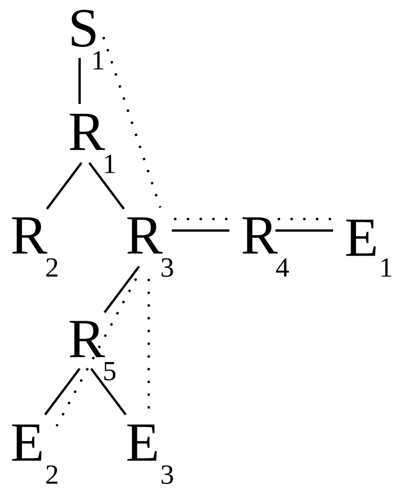

Figure 3: Router R_4 avoids overloading link R_3R4 by pushing E_1's original reflection request on to R_3.

it ends up, is guaranteed to lie on the path from the source to the destination of the inbound connection ID. However this may not be true is some situations. In rare cases, non-symmetric routing paths may exist between two hosts. A more common concern is that routes in the underlying network change as a result of broken links or configuration changes. A reflection request may have propagated to a reasonable router to intercept its quarry, but a route change may allow future packets to evade duplication.

[I no longer have much faith in the ubiquity of route symmetry. I will need to research its truth, and if it's not add a bit of evaluation showing how significant a problem it is.]

For correctness in the case of these challenges, packet reflectors must signal the original destination node when a packet has been successfully reflected. In the absence of such confirmation, the requesting node may perform the split on its own, as if packet reflection was never 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. Thus, a request takes the form: $reflect(A \rightarrow B, \{B \rightarrow C, B \rightarrow D\}, T)$ indicating that after emitting two copies of the packet traveling from A to B (sending them to C and D), the router should forward the original packet with T written to its tag field. 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 a reflection request (See Figure 4).

The tag associated with a request is only written to the original packet, not to any of the duplicates. This means that the "namespace" of the tag is local to the inbound connection identifier of the request. Each time that a router passes on a request that causes duplication by yet another router, it increments the tag associated with the request. Thus, if a portion of a request is forwarded, while the rest stays local (because the router represents a branch that is appropriate for splitting of some but not all of a request's duplications), there is no

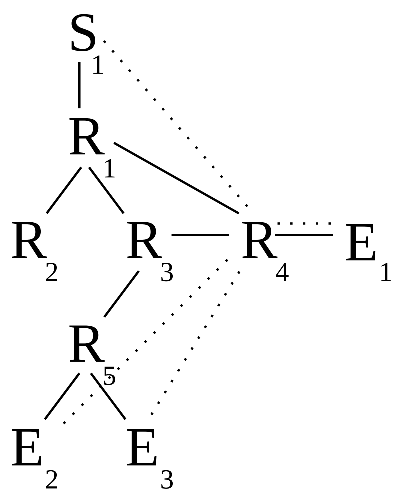

Figure 4: A new physical route is brought online between R_1 and R_4, bypassing the reflection request in R_3. R_4 notices that it receives untagged packets and performs the reflection on its own. E_1 need not be notified.

chance that the upstream router's tag could be confused for indicating the success of the entire reflection. The router passing on the request must note the correct success tag from the upstream router, perform its own duplications, and then write the success expected by the original requester.

In addition to the tag associated with a request as a whole, reflection requests can also contain a tag associated with each outbound connection identifier. This is necessitated by the interaction of multiple reflection requests. Suppose that a router has accepted a request: $reflect(A \rightarrow B, \{B \rightarrow C\}, 1)$. Now suppose that the router receives another request: $reflect(B \rightarrow C, \{C \rightarrow D\}, 5)$. The router can now deduce that when it receives a $A \rightarrow B$, it must send two packets in addition to the original - $B \rightarrow C$ and $C \rightarrow D$. The $B \rightarrow C$ packet must be tagged with a 5 so that it is clear that the second request was honored. Finally, suppose that the router emits $B \rightarrow C$ packets on the same interface that it receives $A \rightarrow B$ packets. In that case, it would like to make a request to its upstream router: $reflect(A \rightarrow B, \{B \rightarrow C\}, 2)$ relieving itself of the burden of forwarding to C. But part of the contract with the second requester is that the $B \rightarrow C$ would have its tag set to 5. So the reflect request interface must be widened to allow the desired tag to written on the router's behalf. Tags can be associated with each outbound connection identifier, and the router requests: $reflect(A \rightarrow B, \{(B \rightarrow C, 5)\}, 2)$. This is summarized in Figure 5.

3.2 Path Painting

A reflection capable router may find that it has been asked to emit copies of a packet multiple times through the same interface (but not the one on which it was received). Referring again to Figure 3, router R_3 must emit packets destined for E_1 to both E_2 and E_3 which requires using link R_3R_5 twice. Unfortunately, the router can not use packet reflection to alleviate this burden. Packet reflection only allows a node to avoid emitting packet duplicates on

Reflect Request
IP Source IP Destination IP Proto = REFLECT
Source Port Destination Port Tag `REQUEST` Copy Count Copy Source Address 1 Copy Destination Address 1 Copy Source Port 1 Copy Destination Port 1 Copy Tag 1 ...

Figure 5: Detailed packet contents of reflect request messages.

the interface that the packet originally arrived. If R_3 propagated E_1's original request, $(S_1E_1, \{E_1E_2, E_1E_3\})$, then the reflection would never occur because R_5 would never see packets intended for E_1, the inbound connection identifier.

Path painting enables nodes to learn enough about the topology of the underlying network to avoid this problem entirely. Using path painting, nodes can set up topologies that resemble the distribution trees built by IP Multicast, which reflect the underlying network. Overlay networks with these topologies can be optimized using packet reflection.

Fairly simple rules could prevent the kinds of topologies that can not be optimized by packet reflection. Those same topologies are beneficial to overlay networks in general because the minimize the number of links into any particular area of the network. For example, in Figure 3, if only E_2 were the child of E_1 and E_3 were a child of E_2, then packet reflection would be sufficient to avoid transmitting the same data twice on a single underlying link and there would be only one virtual link into the $R_5E_2E_3$ portion of the network. This is suggestive of an important rule: if the paths from multiple nodes to the rendezvous meet at router R, only one of the nodes should have a parent whose path to the rendezvous does *not* go through R. This rule ensures that there will be only one path into a region of the network gatewayed by a single router — the virtual topology will reflect the underlying topology.

To use path painting, an end host propagates a paint request toward its rendezvous point by addressing it to the rendezvous. A paint request is implicitly intended to observe similar packets that match the destination address and port of the request packet. Each path painting capable router along the path begins to watch for any further packets addressed to the same destination. Whenever the router observes such a packet, a notification is forwarded to all interested parties, including the latest requester. The notification contains information about all of the painters.

Like reflection requests, paint requests are normal IP packets and will be propagated by

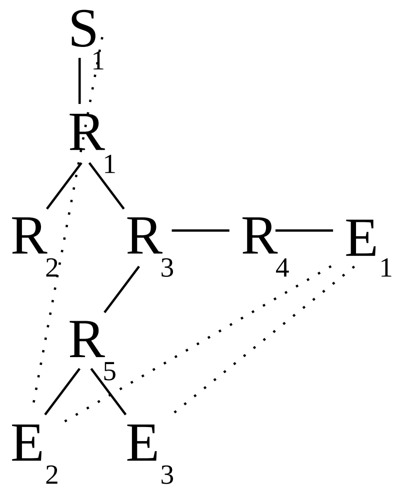

Figure 6: A poorly built distribution tree. If R_3R_5 is a slow link then there is an incentive to ensure that there is only one virtual link across it, allowing transcoding to a thinner stream before use.

normal IP routers. This allows path painting to be effective across regions of the network that do not implement painting. As long as a 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.

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. This avoids an implosion of interest at the rendezvous. Without application hints, the propagated request is arbitrarily (but deterministically) chosen by the router at which the paint requests meet. To allow applications to choose propagation of a particular request, paint requests may fill in a *concede* field, which indicates that the request should not proceed past a node with a request that originated from *concede*.

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 in the overlay network based there, a request may also contain any number of *ignore* addresses. These indicate that the paint request should continue even if it passes a node colored by one of the *ignore* nodes.

4 Multicasting With Primitives

The previous section proposed two useful primitives that overlay networks can take advantage of in order to increase efficiency. In this section three sample multicast protocols will be described to demonstrate the versatility of the proposals.

First, a system that mimics the semantics of IP Multicast is described. 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

Paint Request	Paint Notification
IP Source (color)	(router of collision)
IP Destination (key)	(request originator)
IP Proto = PAINT	IP Proto = PAINT
`REQUEST`	`NOTIFY`
Concede	Painter Count
Ignore Count	Painter 1
Ignore 1	Painter 2
Ignore 2	Painter 3
Ignore 3	...
...	

Figure 7: Detailed packet contents of paint request and paint notification messages.

will be described that has more functionality than similar systems built on IP Multicast (RLM [10], Thin Streams [13]). Finally, a novel multicast application will be described, demonstrating the flexibility provided by application level control. The versatility and simplicity of these protocols demonstrates the constructive power of the proposed primitives.

4.1 IPM Emulation

To demonstrate the effectiveness of the proposed primitives this section describes how they can be used to emulate the service model of IP Multicast. It is easiest to begin by describing a mapping between elements of IP Multicast to elements of the emulation that can be provided with the proposed primitives.

Element	IP Multicast	Emulation
Group address	Class E address	(address, port)
Rendezvous	Core router	End host
Join request	Graft message	Paint request

A multicast system requires a rendezvous so that various potential group members can come together and share packets. In IP Multicast, the rendezvous is somewhat hard to pin down. Various protocol (PIM [7], DVMRP [12], CBT [4]) have proposed different rendezvous points. In emulation, a simpler approach is taken. The rendezvous is explicitly named as part of the group. The group name will be the IP address of a suitable rendezvous. A port number is added to the IP address to provide a vastly greater namespace.

As in IP Multicast, a join message is sent to the rendezvous point by new group members. 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 a painted router is encountered, the joining node and the previous painter are notified.

One of these 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. This can be determined from the TTL field in the collision reports. In case of a tie, any deterministic tie breaker is sufficient, such as an ordering on the nodes' IP addresses. The parent/child relationship is cemented by establishing the port number over which the two nodes will converse. 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 (as defined by the application - perhaps it lacks a cryptographic key), the paint requests are modified to mention them in their *ignore* lists.

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. Each neighbor will appear once in an inbound connection identifier and in all other requests in an outbound connection identifier.

Emulating IP Multicast in this way has a number of benefits. First there is an expanded address space. 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.

4.2 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 setup 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. Each node also exchanges congestion information with each of its neighbors [3, 1]. If a 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 link and forwarded explicitly.

Allowing stream thinning to occur wherever appropriate creates a flexible 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.

4.3 Fast Then Correct

A running joke in computer science is that a system can be fast or correct, but not both. In multicast delivery this has been exactly the trade off. Normal IP Multicast provides best-effort, low-latency delivery — it's fast but unreliable. Reliable multicast offers guaranteed delivery, but suffers from higher latencies as retransmits correct for lost packets.

For the most part, this dichotomy is not a problem. Audio and video conferencing tools require interactive delays and can accept the occasional packet loss. Applications envisioned for reliable multicast, such as efficient netnews delivery or software updates, require bit for bit integrity but can be delayed for relatively long periods of time without sacrificing their utility.

Although most applications fall into one of these two categories, it is possible to think of applications that would best be served by a third model, Fast Then Correct (FTC). Consider a quarterly earnings report teleconference. Some investors that are closely following a company want to receive the report as it happens, and perhaps participate in an interactive question and answer period. Others are perfectly willing to wait until later, but they ought to receive a high-fidelity copy of the entire conference call.

Applications of this sort have two types of consumers — those that want data quickly, and those that want high quality data. Surprisingly, these are sometimes the same users. Consider "channel surfing" television stations. An immediate response when changing channels is needed to decide if the program is worth further consideration. To provide this, a low-fidelity version of the channels above and below the current station might be transmitted at all times. Switching stations could be quite fast. Once a viewer stays on a given channel for a few moments, a buffer could be built up by imperceptibly prolonging silent periods. Once a buffer is in place, the increased latency of a reliable protocol could be hidden, so a better picture could be delivered.

To provide FTC an overlay would begin by building an application-level distribution tree is exactly 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, just as for heterogeneous multicast, pairs of nodes may choose to avoid the reflection primitive so that they may take explicit control over packet dropping policy, avoiding congestion. When a downstream node notices that a packet was dropped, two responses are possible, depending on application needs.

If the application normally runs at near congestion speeds, then the node that missed a packet will wait until an off-peak time to "fill in holes" by asking its parent for the missing data. If the application runs at less than congestion speeds then the application might immediately ask for the lost data. The first scenario supports the "earnings report" scenario, in which the stream can not be buffered due to interactive needs. A perfect copy of the session can be created later. The second scenario supports the "channel surfing" scenario. During normal watching it is expected that enough bandwidth exists to transfer the data for a channel. The channel is buffered, perhaps by a few seconds, to support momentary glitches.

Reliable multicast systems built on IP Multicast are considerably more complex. 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 is to ask for rebroadcast from the *entire group* (or a TTL limited subset). 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 "hole filling" hours after the original transmission. The designers of IP Multicast could not ask routers to buffer gigabytes of data just in case someone might want it later.

5 Primitives in Other Applications

Although reflection and painting were most 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, two sample applications that could make use of them are considered.

RON A RON is a Resilient Overlay Network. A RON attempts to route packets as quickly as possible between any two of its nodes, even in the face of a failure on the direct IP route between those nodes. Most of the time, packets are unicast directly between the two communicating RON nodes. However, in the face of failures, routing pathologies, or simple congestion, a RON will often find a two hop route that is faster than the direct IP route.

A RON could take advantage of Packet Reflection to decrease latency in its two hop routes. If a RON makes the decision to route 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. More commonly, the request would be repeated, pushing back into the network. The deployment of RONs would be greatly simplified because the choice of location for RON 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.

A RON is a perfect example of a generic overlay network. It exists for no purpose except to be a better network on top of the Internet. It adds no new functionality, only improving upon the best-effort nature of the existing unicast delivery model. As such, the fact that a RON can be improved by using packet reflection is telling.

Web Caching Hierarchy Some projects [5] have suggested the hierarchies of web caches would be an efficient extension to the common practice of single, isolated caches. However, the automatic creation of these hierarchies is difficult. Yet static configuration is error-prone and tedious. By using path painting, caches could discover each other, and automatically configure themselves in appropriate hierarchies.

To participate, all caches would send paint requests to the same destination. Just as for IP Multicast emulation, at each paint collision, one cache would be determined to be the best parent (perhaps the largest cache), and all others would concede.

As for multicast rendezvous, there is no requirement that the address to which the caches are sending paint requests denotes a participating cache. As long as it is a routable IP address, the paint requests will aggregate as they approach its expected network location. Even in the face of network partitions, separate, efficient trees would be formed. For particularly popular sites, it might be worth while to create separate hierarchies using the popular site as the rendezvous.

6 Implementation

Reflection and painting have been implemented in *ns*, version 2.1b8a. The code is publicly available at [url to be determined].

This section provides more detail about both primitives by showing simplified portions of the *ns* code, including the data structures that a reflecting/painting router must maintain, request processing, and the forwarding path.

```
def end_pt { ipaddr a, port p};
def cid    { end_pt src, end_pt dst };
def action {
  int: tag;
  list<cid,tag,offloaded?>: outputs;
};
```

An action expresses what should be done when a packet match is found. For each element of the outputs list, a copy is created unless the arriving packet is tagged appropriately.

```
reflect(cid, list<cid,tag>: output, tag)
paint(end_pt: dst, end_pt: painter)
```

A reflect request asks for a copy of each output item, with the associated tag in the copy. The isolated tag is intended for the original packet, to indicate the copies were created. A paint request ask for painter to be notified of any other requests for the same dst.

```
table<cid, action> reflect_table;
table<end_pt, list<end_pt>> paint_table;
```

The tables contain mappings from matching criteria to the actions that should be taken when they are met. Reflections create copies, paints create notifications destined for the associated end points.

```
forward (pkt p) {
  if (p.ip_proto == REFLECT) {
      reflect_request(p);
      drop(p);
      return;
  }
  if (p.ip_proto == PAINT and
      p.type == REQUEST) {
      paint_request(p);
      drop(p);
      return;
  }
  action act = reflect_table{p.src, p.dst};
  for o in act.outputs {
    // Skip copying if already done
```

```
    if (o.offloaded? and p.tag == o.tag+1)
      continue;
    c = clone(p);
    c.src = o.cid.src;
    c.dst = o.cid.dst;
    ip_forward(c);
  }
  p.tag = o.tag; // Indicate success
  ip_forward(p);
};
```

The forwarding path handles reflect and paint requests, then drops those packets. All other packets are matched against the reflect_table, and copies made if necessary. Copies and the original packet are forwarded using the standard IP forwarding path.

```
reflect_request(pkt p) {
  int d_if = interface_of(p.src);
  int s_if = interface_of(p.dst);
  act = new action;
  act.tag = p.tag;
  for c in p.copies {
    int c_if = interface_of(c.dst);
    if (c_if != d_if)
      act.outputs.add(c.src, c.dst,
                      c.tag, c_if == s_if);
  }
  reflect_table.add(p.dst,p.src,act);
}
```

The requested copies of a reflection request are inserted into the appropriate action if doing so makes sense for that router (the copy will not have to be sent to the same interface as the original packet). If the router will be passing on the request to another router (the copy will be sent out the same interface that the original arrived on), then the copy is marked conditional. If packets arrive appropriately tagged, the copy will not be made.

```
paint_request(pkt p) {
  list<end_pt> l = paint_table{p.dst};
  l.add(p.src);
  // Send list of to every member
  for e in l
    notify(e, l);

  if (l.has(p.concede))
    return;

  // best() is arbitrary, but deterministic.
  if (p.src == l.best() or
      p.ignore.has(l.best()))
```

```
    ip_forward(p);
}
```

All interested parties are notified of the colors present at the router. The paint request then continues if it does not concede, and is either the “winning” color for the node or refuses to be stopped by that winning color.

```
bool flatten() {
  bool done = true;
  for a in reflect_table.vals()
    for o in a.outputs {
      sub_action = reflect_table{o.cid}
      if (sub_action) {
        for sub_o in sub_action.outputs
          if (! a.outputs.has(sub_o.cid)) {
            a.outputs.append(sub_o.cid,
                             sub_o.tag,
                             o.offloaded?);
            done = false;
          }
        o.tag = sub_action.tag;
      }
    }
  return done;
}
```

Multiple cascading reflection requests are flattened. If fulfilling a reflection request would produce a packet that would also be reflected, then the results of that reflection are inserted into the action for the original packet. This allows all forwarding to occur after one lookup.

7 Evaluation

The goal of network support for overlay networks is to enable greater efficiency in overlay networks, as well as simplify their construction. In general, these goals are addressed separately by the two proposed primitives. Reflection is used to decrease the network utilization of overlay networks, while painting is used to ease their construction. Although their effects are enhanced in combination, for evaluation purposes, separate experiments will attempt to distinguish their effects before a combined experiment demonstrates their total utility.

In addition, these goals should be approached as deployment increases - there should be no “critical mass” of adoption that is required before benefits can be realized (or that critical mass should be very small). Finally, the resource utilization of the primitives in routers that support them should be minimal and controllable.

7.1 Network Utilization

This section quantifies the gains that overlay networks can achieve using Packet Reflection. Packet Reflection should allow overlay networks to decrease network *stress*, a measure how often identical data is sent over a given link; as well as decrease latency among virtual nodes.

To demonstrate the reduction in stress possible with reflection, an Application Level Multicast system will be simulated. In the first set of runs, the ALM system functions as usual, duplicating packets as needed in edge servers. In the second set of runs, edge servers make use of Packet Reflection to request duplication inside the network.

The exact method of construct these topologies is yet to be determined. To avoid conflating these results with Path Painting, an independent construction technique should be used. An existing technique, such as Overcast's or End-System Multicast might be appropriate. Though an simpler approach would also demonstrate the effect - and would show that a system need not work so hard on topology if reflection is available.

Although stress is mainly a concern for overlay networks that duplicate packets, latency is a factor in all overlay networks. The added latency required for packets to reach an edge server and to be retransmitted can be reduced or eliminated with Packet Reflection. To demonstrate this reduction in latency, a RON-like overlay network is simulated. As above, in one set of runs, the overlay functions as it would on the Internet today. In the second set, Packet Reflection is used.

7.2 Overlay Construction

Providing network support for overlay construction can have two primary benefits. First, it can allow better topologies to be constructed, decreasing network overhead when packets are transmitted in the overlay. Second, it can eliminate the overhead of construction itself. Overlay networks are generally forced to conduct online measurements to aid in topology building. Allowing these systems to avoid that step can eliminate unnecessary network traffic. In addition, secondary benefits include faster topology building and decreased implementation effort.

To demonstrate the effect of Path Painting as an aid to topology building, multiple overlay networks are constructed. First, traditional techniques are used. Again, the exact techniques to be used are not yet determined, they will likely come from published works such as Overcast or ESM, The approach is compared to a simple Path Painting enhanced topology building algorithm. In the Path Painting algorithm, all nodes send paint messages to an agreed upon rendezvous. When the paths meet, a deterministic, but random, node is made to be the parent.

The results or this experiment will be analyzed in two contexts. First, the quality of the topologies will be considered. Second, network utilization during construction and maintenance will be compared.

7.3 Primitives in Combination

Reflection and Painting were designed to work well together. To demonstrate this, similar experiments will be conducted in which Reflection and Painting are both available. In complete deployment scenarios, no links should have a stress greater than one. Latency should be nearly minimal (comparable to PIM-SM).

7.4 Incremental Deployment

Incremental deployment is a key advantage of these primitives. To demonstrate that incremental deployment is reasonable, various partial deployment scenarios will be simulated.

1. Random routers support primitives. Consider efficiency gains as adoption levels change.
2. Only edge routers support primitives. Consider latency and bandwidth benefits, particularly for scenarios in which a bottleneck connects the node's network to a fast ISP network (a common scenario), and the ISP's edge router supports the primitives.
3. A single AS supports primitives. Demonstrate that the AS gains local efficiencies.

7.5 Router Resources

Particularly for comparisons to IP Multicast, measurements of resource utilization in routers (such as state table size) is important. An experiment comparing state table sizes will conducted comparing a simple ALM system using the primitives to a purely IPM system.

If the optimization that would allow routers to pass on requests without updating their own tables (when their only job would be to receive a packet and forward it once), the effectiveness of that optimization will also be demonstrated.

When an IPM system uses "too much" state, group members end up missing data. With the proposed primitives resource usage can be controlled in the routers without data loss. Experiments in which routers reject requests that overload their state tables will be conducted to demonstrate that an ALM system using the primitives will gracefully degrade under network overload.

In another vein of router resources, it may be fruitful to demonstrate that the speed of the normal forwarding path of an IP router is scarcely affected by supporting the primitives. An actual experiment may not be worth doing (it is probably better to explain why it will be true), but Click represents one way to proceed if it is worthwhile. It's nearly self evident though: most traffic is not using the primitives — there can be a simple check for fast path handling.

8 To Do

These are topics that haven't been addressed in the text yet, but will be in the thesis.

No description of how reflection and paint requests should be removed has been given. An explicit withdrawal protocol, combined with a timeout mechanism seems appropriate, but details have not been determined.

The security implications of the proposed primitives require more attention. Some details are already clear. For example, routers should reject reflection requests that appear on interfaces other than the expected interface of the requester. A handshake might also be important to ensure that the request is being made by the supposed requester (if an attacker can spoof the handshake, the requester has already lost anyway).

9 Thesis outline

1. **Introduction** People like overlay networks, the network should support them.
2. **Design Requirements** Discuss the needs of overlay networks. Describe requirements that proposed primitives must meet.
3. **Reflection** Design goals and specification of the reflection primitive. Demonstrate that reflection is simple, incrementally deployable, effective, and easy to implement.
4. **Painting** Design goals and specification of the painting primitive. Demonstrate that painting is simple, incrementally deployable, effective, and easy to implement. Bonus points for making painting seem like less of a hack - demonstrate other uses and/or focus on showing it to be simple and/or show that other ideas have important flaws.
5. **Applications** Multiple multicast semantics, and Overcast/RON improvements.
6. **Implementation** Describe ns and Click implementations.
7. **Evaluation** Experiments to show
   (a) Reflection relieves network stress and decreases latency
   (b) Painting helps build better trees with less network utilization
   (c) Incremental deployment
   (d) Localized benefit
   (e) State table growth
   (f) Effect of asymmetric routes
8. **Related work** Overlay networks, Multicast protocols.
9. **Discussion/Conclusions/Future work.**

10 Related Work

The seems to be little work intended to support overlay networks at the network layers. Currently, the focus of overlay networking research is focused mainly on what is possible without any support, rather than determining what small additions could make overlay networks function more efficiently. That said, the scope of overlay networks is quite broad, and examining a few has been helpful in determining what primitives could be generally useful to many such systems.

A number of research groups and service providers are investigating multicast systems based on overlay networks. All share the goal of providing the benefits of IP Multicast without requiring direct router support or the presence of a physical broadcast medium. The main way that these systems differentiate themselves is the way that they organize their distribution trees and in the exact semantics of the multicast service they provide.

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 their work focuses on *semantic* rather than *data* reliability. For instance, RMX can be used to change high resolution images into progressive JPEGs before transmittal to underprovisioned clients. RMX's reliability semantics allow the image to be transcoded, but not dropped. Transcoding is an excellent example of functionality that is best handled by application-level servers in an overlay.

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. End System Multicast could gain significantly from pushing packet processing into the network because no nodes are deployed toward the core of the network. Only the end nodes are used to replicate packets.

X-Bone and Yoid are general-purpose overlay network architectures that can support many different network services. They build distribution trees by first selecting a set of virtual links between components and then running traditional multicast routing protocols over those links. Unfortunately, information about the effectiveness of this technique for producing efficient distribution tree is unavailable. It is clear, however, that additional support in selecting the initial virtual links would be welcome.

Overcast is a specialized overlay network designed with the express purpose of supporting high-bandwidth, single-source multicast. Overcast demonstrates the benefits of an application-level multicast system by providing features that IP Multicast can not support, such as on-demand access to archived groups. Additionally, 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 can not occur in routers and distributions trees must be inferred from measurement, rather than through knowledge of the underlying network.

11 Schedule

The following is a proposed schedule that errs toward optimism. Jan 31 is a fairly firm deadline for the proposal to be signed though (MIT imposed). I would be pleased with finishing the thesis any time this summer.

1/25 . . . . . . Proposal signed and submitted
2/01 . . . . . . Sigcomm deadline
2/15 . . . . . . Click implementation
2/28 . . . . . . Openarch Camera-ready due
3/08 . . . . . . Organize cannibalization from OSDI (Overcast), Openarch, and Sigcomm papers.
3/22 . . . . . . Reflection chapter
4/05 . . . . . . Paint chapter
4/26 . . . . . . Overcast/RON improvements chapter
5/03 . . . . . . First complete draft
6/14 . . . . . . Defense

References

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

[2] 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.

[3] 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.

[4] 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)*, September 1993.

[5] Anawat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A hierarchical Internet object cache. In *Proc. USENIX 1996 Annual Technical Conference*, pages 153–164, January 1996.

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

[7] 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.

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

[9] 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. 4nd Symposium on Operating Systems Design and Implementation (OSDI '00)*, October 2000.

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

[11] J. Touch and S. Hotz. The X-bone (white paper). Technical report, SIS, May 1997. `www.isi.edu/x-bone`.

[12] David Waitzman, Craig Partridge, and Stephen E. Deering. Distance vector multicast routing protocol. Technical report, ARPANET Working Group, DDN Network Information Center, November 1988.

[13] 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*, May 1997.