Network Layer Support for Overlay Networks

John Jannotti <jj@lcs.mit.edu>
Massachusetts Institute of Technology

Abstract

Overlay networks represent a flexible and deployable approach for applications to obtain new network semantics, but they suffer from some efficiency concerns. To support overlay networks efficiently, 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. Both primitives are incrementally deployable.

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 efficiently distribute and store data throughout the network.

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

Overlay networks 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 throughout the Internet are 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 amongst participating nodes.

The overlay network approach faces challenges of its 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 in the underlying network. 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.

Second, overlay networks operate at a disadvantage to router-based systems because of the physical location of their computational elements. This drawback is both a performance problem, packets go out of their way to reach the overlay nodes; 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 group join mechanism relies on the ability of routers to observe passing messages.

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 should be a design goal in wide area network design.

The contributions of this paper are:

- *Packet Reflection*: a new primitive by which applications may request packet routing and duplication to occur in the routers where there are most efficient. In contrast to similar proposals, packet reflection is shown to be incrementally deployable, usable in the face of routing changes, and easier to assess from the standpoint of security.

- *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 for two group communication 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.

A common metric by which application level multicast systems distinguish themselves is *stress*. 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. With path painting and packet reflection, they can.

Section 2 discusses the design goals of the proposed primitives by examining existing overlay systems and the functionality required to efficiently support them. In Section 3, packet reflection and path painting are described in detail. Section 4 examines how these primitives may be used to implement two multicast semantics: the semantics of IP Multicast, and a system that copes well with heterogenous receivers. Section 5 describes alternative uses for these new primitives, demonstrating their general utility. Section 6 evaluates the effectiveness of the primitives in decreasing stress in a multicast system. Section 7 discusses related work and Section 8 details future work.

2 Design Goals

The motivation for this work comes from the increasing number of research groups and service providers that are investigating systems based on overlay networks.

A number of conclusions can be made based on the diverse need of these 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.

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 virtual nodes must retain complete control of forwarding when necessary. 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 needs, the problems of previous router modification proposals, particularly IP Multicast, should be considered. In particular, it is critical that new functions are incrementally deployable. 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. In addition, they are incrementally deployable—overlay networks can operate correctly even when few (or no) routers support the primitives. As additional routers support them, the overlays will become more efficient.

First, *packet reflection* exposes the ability of routers to route and duplicate packets at the appropriate location in the network. Consider a packet leaving a source, intended for two end hosts. It would be most efficient for a the single packet to travers the network until it reaches the router which would emit packets for the two intended destination through two different interfaces. If an application-level system is running only on the end-hosts, it is hard to see how the duplication can occur in the appropriate router. If the packet is sent to one destination which retransmits it to the next, then the packet will travel twice on the links near the first destination. Packet reflection allow the first destination to avoid this inefficiency by requesting that the duplication occur inside the network. It is called *reflection* because, it is used to allow the first destination to "reflect" the duplication point back against the path of the packet to an appropriate router. The "reflection" stops at the first router that would emit packets for the second destination through a different interface than it would emit packets for the first destination.

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 to all painters 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, end hosts duplicate packets, which forces semantically 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.

3.1.1 Reducing stress with reflection

Retransmissions in an overlay network are easily described: when a packet arrives, the node's action can be driven by a simple table lookup using a demultiplexing *flow identifier* inside the packet. The action to be taken is simple: duplicate the packet some number of times, substituting new source and destination addresses, and re-emit.

The stylized nature of this operation suggests a simple optimization. An end host may ask "the network" to perform *packet reflection* on its behalf. In Figure 2 end host E_1 directs a reflection request toward S_1,

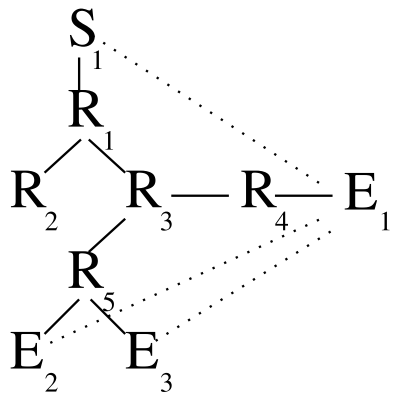

Figure 1: An application-level multicast distribution tree. Packets are sent from the source S_1 to end host E_1 through routers R_1, R_3, and R_4. E_1 sends the packets on to E_2 and E_3.

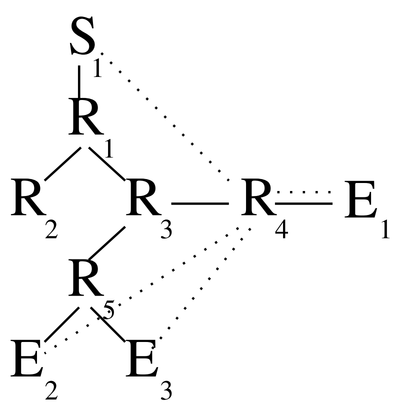

Figure 2: End host E_1 avoids overloading link R_4E_1 by sending $reflect(S_1 \rightarrow E_1, 1, \{(E_1 \rightarrow E_2, 0), (E_1 \rightarrow E_3\}, 0))$ to R_4. R_4 will now duplicate packets for E_1 from S_1, sending copies to E_2 and E_3. In both duplicates the source will be E_1.

which takes it to router R_4. This alleviates link R_4E_1 from carrying the same packet three times (once in, twice out). In addition to performing requested reflections, routers continue to forward packets using their normal forwarding rules. Thus, E_1 will receive all packets addressed to it.

The format of a reflection request is denoted $reflect(S \rightarrow D, T, \{(S_i \rightarrow D_i, t_i\})$. This should be read as, "When a 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 3.1.2.

As seen so far, packet reflection allows end hosts to avoid wasted packets on the link between them-

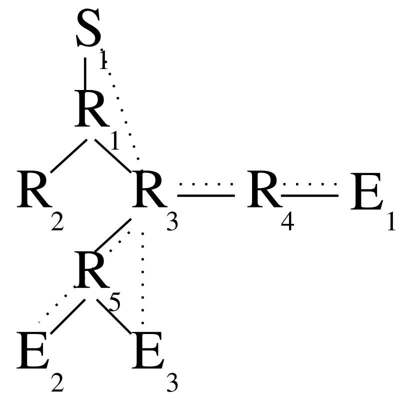

Figure 3: Router R_4 avoids overloading link R_3R_4 by sending $reflect(S_1 \rightarrow E_1, \mathbf{2}, \{(E_1 \rightarrow E_2, 0), (E_1 \rightarrow E_3\}, 0))$ to R_3. Note that the tag has been incremented. Router R_4 must continue to process $S_1 \rightarrow E_1$ packets. If the tag is 2, the only necessary action is decrementing the tag and re-emission. If the tag is not 2, R_3 must not have been able to perform the reflection. R_4 must perform the duplication and tag the orginal packet with 1.

selves and the nearest router to them. Although this is a useful optimization, 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, router R_4 takes advantage of packet reflection by propagating 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) avoids overusing the R_3R_4 link. Of course, if R_3 performs the reflection, R_4 should not. Tags allow R_4 to know whether a given packet has already been reflected by R_3.

3.1.2 Tags confirm reflection

Generally, at any one time, the Internet routing infrastructure provides 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 requests toward the source address of the inbound flow identifier. The assumption is that the request, wherever it ends up, will lie on the path from the source to the destination of the inbound flow identifier. However this may not be true in some situations. In some cases, asymmetric routing paths may exist between two hosts. This could allow a packet to arrive at an end-host without passing through the router that would be expected to perform reflection on it.

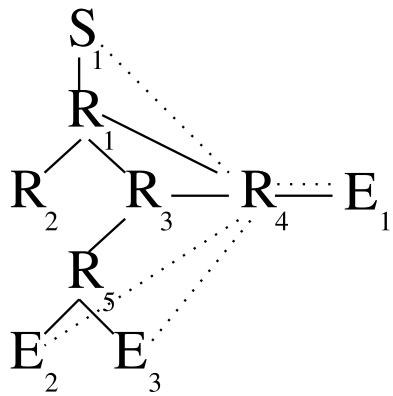

Figure 4: A new physical route is brought online between R_1 and R_4, bypassing the reflection request in R_3. R_4 notices that it receives untagged packets and performs the reflection on its own. E_1 need not be notified.

A similar 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 router that, after a route change, no longer sees the packets that are to be reflected. Figure 4 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 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 signalling 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 a reflection 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 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.

Although asymmetric routes present a persistent problem (as opposed to transient route changes), the tag mechanism addresses the missed reflection request inside the network, allowing at least a portion of the hoped for benefits. For example, assume that link S_1R_4 in Figure 4 is a one-way link. The routes between S_1 and E_1 are now asymmetric. Although this means that the reflect request in R_3 will never be used, R_4 will detect that it should continue to perform the reflection on E_1's behalf.

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: $reflect(A \rightarrow B, 1, \{(B \rightarrow C, 0)\})$. This means that when a packet going from $A \rightarrow B$ is observed, a copy will be sent from $B \rightarrow C$ and 1 will be written to the $A \rightarrow B$ packet. Now suppose that the router receives another request: $reflect(B \rightarrow C, 5, (\{C \rightarrow D, 0)\})$. 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 makes a request to its upstream router using the tag associated with the $B \rightarrow C$ flow identifier: $reflect(A \rightarrow B, 2, \{(B \rightarrow C, \mathbf{5})\})$. By doing so, it insures that it is upholding its contract to write a 5 into $B \rightarrow C$ packets.

3.1.3 Deployment details

The packet reflection primitive is fine-grained for precise control. For example, each request 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 flow identifier and three times in an outbound flow identifier.

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. 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.

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 greatly simplifies initial deployment.

The security implications of reflection seem, at first, difficult to predict. However, a simple rule allow routers to reject reflection requests that might be intended to "reflect" sensitive data to prying eyes. Routers should accept a reflection request on a given interface only if control of that interface would have been sufficent to implement the behavior requested by the reflection request. For example, if packets addressed to A would be admitted through a router's first interface, then requests to reflect packets intended for A should only be accepted on the first interface. Further, the outbound flow identifier's source for such requests must be A.

It would also be approriate to require a "handshake" between the requesting node and the reflecting router. The router would tentatively accept the request, responding to the requester with a large random number. Only after the requester echos the random number back to the router would the request go into effect. With this addition, a malicious node could reflect A's packets only if it could intercept A's packets. In such a case, the malicious node already has access to A's data.

3.2 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. In many applications nearby nodes conduct more traffic amongst themselves than distant nodes; this should be as reasonable in an overlay as in the physical network. Second, virtual links should be, as much as possible, independent of each other in the physical network. In addition to leading to independent failures, this property helps the overlay network deal with network characteristics more naturally. For example, when links are independent, a slow link can be routed around easily. But if the route "around" the first link actually shares physical links with the congested link, there may be no gain.

To build overlays that resemble the underlying network, nodes would like to aggregate locally into small clusters. Then, small clusters might further aggregate into larger clusters, and so on. To allow this, path painting takes advantage of the fact that, in general, the Internet is organized so that nearby nodes share most of their paths, when contacting far away nodes. If nodes could determine which other nodes it shares 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. All computers of the university also share most of their paths, though not necessarily the first few hops. Beyond that, all customer's of the university's ISP share paths, and so on.

To use path painting, all end hosts send paint requests toward an agreed upon rendezvous point. 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 watches for any further packets addressed to the same destination. Whenever the router observes such a packet, information about all painters is forwarded to all other painters. However, to avoid an implosion of painters at the rendezvous, only one painter's request is allowed to proceed past any given router. Figure 5 illustrates the interactions of three paint requests.

Under normal circumstances paint requests are *quashed* if they match a request previously made at

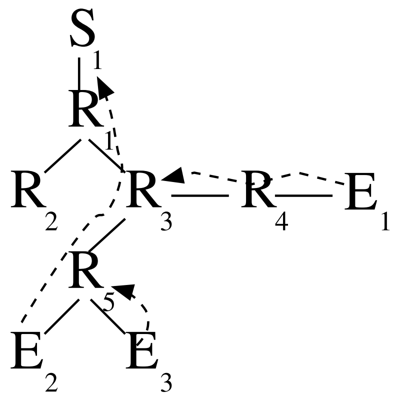

Figure 5: Three nodes, E_1,E_2, and E_3 send paint requests to a rendezvous, S_1. At both R_3 and R_5, E_2 paint request "wins", quashing the requests of E_1 and E_3. After notifications, E_2 knows about E_1 and E_3. E_1 and E_3 know only of E_2.

the same router. Only one "paint color" continues on from an intersection point. 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 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*.

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.

Like reflection requests, paint requests are normal IP packets and will be propagated by normal IP routers. This decision 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.

4 Multicasting With Primitives

The previous section proposed two useful primitives that overlay networks can take advantage of in order to increase efficiency. This section describes two sample multicast protocols to demonstrate the versatility of the proposals.

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 heterogenous multicasting system will be described that has more functionality than similar systems built on IP Multicast (RLM [10], Thin Streams [14]). 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 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 addr	Class E addr	(IP addr, 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 simple 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. 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 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 deterministic tie breaker is sufficient, such as an ordering on the nodes' IP addresses. 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 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 flow identifier and in all other requests in an outbound flow identifier.

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.

4.2 Heterogeneous Multicast

Having built an IP Multicast emulation layer in the previous section, one possible way to handle heterogenous 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 amongst 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 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.

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, we present two non-multicast applications that could take advantage of the primitives.

RON A RON is a Resilient Overlay Network, which 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 through the route selected by Internet routing prootocols between the two communicating RON nodes. However, in the face of failures, routing pathologies, or simple congestion, a RON will sometimes find a two hop route through another RON node that is faster than the direct IP route.

A RON could take advantage of packet reflection to decrease latency for 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 encouraging.

Web Caching Hierarchy Some projects [5] have suggested that 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 rendevous, 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 rendevous.

6 Evaluation

In acyclic topologies, the combination of painting and reflection is functionally identical to IP Multicast. Link stress will never exceed one. Only in the presence of more highly connected networks is it still possible for inefficencies to occur.

Therefore, to evaluate the proposed primitives, we used the Georgia Tech Internetwork Topology Models [15] (GT-ITM) to generate highly-connected "transit-stub" network topologies. Simulations were conducted over ten different 600 node graphs. Each graph is made up of three transit domains. Each transit domain consists of an average of eight stub networks. The stub networks contain edges amongst themselves with a probability of 0.5. Each stub network consists of an average of 25 nodes, in which nodes are once again connected with a probability of 0.5. These parameters are from the sample graphs in the GT-ITM distribution; we are unaware of any published work that describes parameters that might better model common Internet topologies. In each network, some number nodes form a multicast group using the algorithm described in Section 4.1. Average link stress as group size increases is shown in Figure 6.

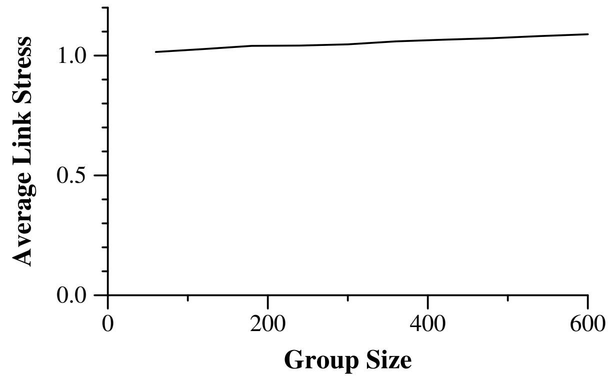

Figure 6: Average link stress as multicast group size increases in a 600 node network. All routers implement painting and reflection.

To understand why stress increases as group size increases, recall that painting plus reflection is sufficent, in an acyclic network, to eliminate all unnecessary stress. Although the networks used are highly connected, when only a small portion of the nodes are particpating in a group, those nodes (and the paths amongst them) form an acyclic graph. As the number of nodes increases, more cycles are formed among participating nodes.

Another interesting question is how closely an overlay network can approach these results when only a portion of the underlying network's routers are enabled for painting and reflection. In the following simulations, the same 600 node graphs are used, but group size is held to 60 members in all runs. The value of incremental deployment is assessed by varying the number of routers supporting the primitives. The routers are chosen randomly, though it would be more realistic to expect that intelligent overlay network designers would chose to locate their nodes in ISPs that support the primitives. Figure 7 demonstrates that link stress is decreased even when only a small percentage of routers are enabled. Although the network-wide effect on link stress may seem modest at low levels of

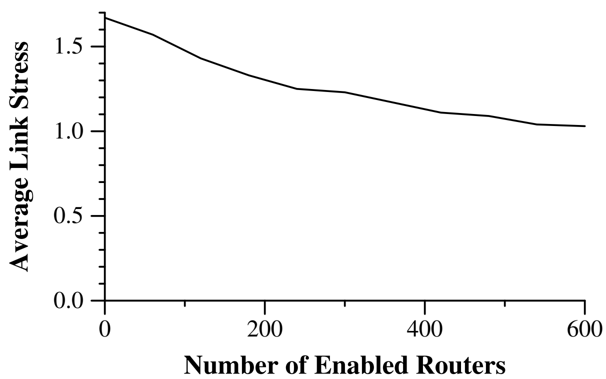

Figure 7: Average link stress decreases as more routers implement painting and reflection. In all runs 60 out of a total 600 nodes create a multicast group.

deployment, remember that all of the benefit comes in the area of the network around enabled routers. Therefore, it is reasonable to conclude that each ISP would see an economic advantage to local deployment, regardless of acceptance by other ISPs.

7 Related Work

Thus far, most overlay work has described the capabilities of such systems, rather than addressing their inefficiencies. Wen, Griffioen, and Calvert's "Building Multicast Services from Unicast Forwarding and Ephemeral State" [13], is closely related, however, and a detailed comparison is appropriate. In particular, it 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 their ability to be incrementally deployed, and their ability to handle route changes in the underlying network. It is not clear that *dup* is sufficient 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. 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 brings the promise of untold power, it carries the baggage of complexity and difficult to predict security implications. The primitives presented here are simple, have *enough* power, and the security implications can be well understood.

8 Future Work

A number of issues related to the proposed primitives require further study. First, no description of how reflection and paint requests should be removed has been given. A simple timeout mechanism, combined with an explicit withdrawal protocol seems plausible, but details have not been determined.

Second, complete applications using the proposed applications should be developed. Only by familiarity with the primitives in the context of large applications can their interface be fine tuned. There may be, for example, a better interface to control paint propagation than the proposed `concede` and `ignore` mechanisms.

Finally, it may be fairly easy to extend the proposed primitives to handle all symetrically routed networks as well as IP Multicast does. Currently, a reflection request is propagated only if packets intended for the destination address of the outbound flow identifier would be emitted through the same interface is the source address of the inbound flow identifier. This is the "reflection" aspect of the primitive. However, after using a paint request to determine that a given node is reachable from a shared "ancestor", it may make sense to indicate that a reflection request should be propagated to that ancestor, regardless of the apparent utility of doing so. This would allow an application to mimic IP Multicast behavior completely, forcing the reflection to occur at the specified router.

9 Acknowledgments

Many people contributed a great deal to the ideas here through conversations and close readings of earlier incarnations of this paper. I would like to thank Suchitra Raman and Timothy Roscoe for their early comments and criticisms. I would also like to thank Hari Balakrishnan, Frans Kaashoek, and Ion Stoica for their continued interest and input. Finally, Eddie Kohler and Joe Touch contributed greatly to the clarity of presentation.

This research was sponsored by the Defense Advanced Research Projects Agency (DARPA) the Space and Naval Warfare Systems Center, San Diego, under contract N66001-00-1-8933, and IBM.

10 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.

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.

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. 4th 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)*, pages 85–95, 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*, pages 795–804, 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, 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. 4th Symposium on Operating Systems Design and Implementation (OSDI '00)*, pages 197–212, 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), May 1997. `www.isi.edu/x-bone`.

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

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

[14] 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.

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