Versatile Primitives for Application-Level Multicasting

John Jannotti

Abstract

A new multicast model is needed to support a wide range of powerful multimedia applications on the Internet. Today's model, IP Multicast, is admirable for its conceptual simplicity, but hides too much important behavior behind a veil of abstraction. Additional services have been built on top of IP Multicast, however each has been more complicated than necessary, indicating that IP Multicast is the wrong primitive for alternative multicasting applications.

Application-level multicast systems avoid the use IP Multicast to increase their flexibility, but suffer from efficiency concerns in the process. These system perform packet duplication in external servers rather than in routers at the network layer like IP Multicast. To alleviate this inefficiency while maintaining flexibility, two new primitives are proposed for implementation in the network layer. *Packet splitting* allows end-hosts to request packet duplication in nearby routers. *Packet snooping* allows end-host to observe certain traffic as it passes nearby routers.

1 Introduction

IP Multicast builds a low-level group communication mechanism into the network. Using IP Multicast, receivers can indicate interest in packets sent to a particular group address. Other nodes simply send a packet to the group address and the IP network attempts to delivery a copy of the packet to all group members.

IP Multicast adequately supports multicast applications that require low-latency, lossy communication within large homogeneous pools of Multicast aware hosts and routers. This is unsurprising, as it is exactly the application in mind when IP Multicast was designed. The true test of IP Multicast's viability as a network primitive is its ability to be used as a building block for applications with different needs.

Unfortunately, there are a wide range of applications that might wish to use efficient multicasting for which IP Multicast is inappropriate. Examples include software updates which require reliable transport, security minded applications that require admittance control, widely dispersed applications which require ways to automatically bridge across portions of the network that lack IP Multicast, and communication amongst group members with various levels of network connectivity. There are, in fact, an almost infinite number of styles of group communication.

Of course, IP Multicast's failure to support every group communication model possible is hardly proof that a real problem exists. For example, TCP is a very useful, flow-controlled protocol for providing reliable byte streams between two IP-based hosts. TCP does not work well for streaming audio or RPC, but there is no reason to decry TCP. TCP does not need to address these needs because TCP is not the lowest-level access available to applications. Applications with special needs can use UDP, a very basic primitive that provides nothing more than best-effort delivery of a single packet at a time. No ordering or reliability is enforced on such packets by the network.

Unfortunately, IP Multicast *is* the lowest level primitive available for group communication on IP networks. If its semantics do not match the application's there is little to be done. Efforts *have* been made to build on IP Multicast to provide semantics such as reliability and support for heterogeneity, but these efforts have been overly complicated and inefficient.

More recent systems built to support multicast ignore IP Multicast entirely in favor of application-level multicast. In these systems, all packet duplication is performed by normal end-hosts, not routers. These systems have been hampered by the inefficiency of duplicating packets wherever they can, in distributed servers spread around the network, instead of in the routers that are, inevitably, best situated for duplication.

The contributions of this paper are:

- A thorough explanation of the weaknesses of IP

Multicast as a building block for various multicast applications.

- A simple primitive by which applications may push a limited class of packet processing operations into the network, allowing packet duplications to occur where there are most efficient while being directed by conveniently placed servers.
- A primitive for allowing end-hosts to snoop a certain class of packets as they pass through nearby routers, allowing application-level multicast systems to build efficient distribution trees.
- A demonstration of the flexibility allowed by these new primitives in the form of example multicast systems for three group communication semantics.

Section 2 details prior work on IP Multicast and multicasting in general, making the case that much of this work has been needlessly complicated by the IP Multicast model. In Section 3, two primitives are proposed that allow applications running on end hosts to make use of the power of nearby routers. Finally, Section 4 examines how these primitives may be used to implement three different multicast semantics, including the semantics of IP Multicast, a system that copes well with heterogenous receivers, and a novel hybrid of real-time and reliable multicast — *Fast Then Correct* multicast.

2 Related Work

IP Multicast IP Multicast [5] is designed to provide efficient communication between multiple nodes as a low level network primitive. The basic unit for communication is that of the *group*, which corresponds to an IP address chosen from a particular reserved range of all IP addresses. In IP Multicast, nodes register their interest in a particular group by sending *graft* messages toward a designated rendezvous point. As graft messages propagate to the rendezvous, a spanning tree is built which includes all interested nodes and the routers that connect them. Graft messages may be suppressed when they reach the existing tree, they need not reach the rendezvous. When a packet is sent to an IP Multicast group it is forwarded by each router to all interfaces that correspond to edges in the spanning tree.

IP Multicast, taken as a whole, suffers from a number of drawbacks that have hampered its widespread acceptance. First, it requires router support. The spanning tree for a group can not cross regions of the network that are not running IP Multicast enabled routers. This creates little incentive to be "the first on the block" to roll out IP Multicast support.

Second, IP Multicast's admittance control and, by extension, its security, are weak. Any Internet node can send or receive packets for a given IP Multicast group. Proponents argue for end-to-end encryption and authentication, but such a solution requires greater computational resources at clients and, more importantly, would do little to avoid denial of service attacks. Even if unauthenticated packets were disposed of at end-hosts, IP Multicast's willingness to send copies of "junk" packets to all members of a group would present a ripe target for Internet trouble makers. In fact, recent experience with the Raman Worm shows that IP Multicast is so susceptible to bandwidth wasting attacks that it can be taken advantage of by mistake.

Third, and most subtly, IP Multicast is difficult to build on. For example, attempts have been made for years to describe a simple, scalable reliable multicast protocol on top of IP Multicast [6, 10, 11, 12, 14], yet there is no consensus that any such protocol has been described. One problem plaguing such attempts is that there is no agreed upon semantics for reliable multicast. One application might wish to use flow control to slow the data stream down to the bandwidth of the slowest link. Another might want to stream data as quickly as possible to nodes that have fast connections, while trickling in to those that do not. Yet another might wish to relax ordering constraints in favor of receiving most of the data in near real-time and the rest later.

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

Receiver-driven Layered Multicast [13] presents one solution to the problem and Thin Streams [17] refines that idea. In RLM senders encode transmissions in a number of layers. The lowest layer is a low-fidelity encoding of the transmission. Additional layers contain extra data which allows a more faithful reproduction of the stream at the receiver. Each layer is transmitted in a separate IP Multicast group, allowing receivers to independently subscribe to each layer.

RLM receivers determine the appropriate number of layers to subscribe to by conducting *join experiments*. In a join experiment a receiver subscribes to the next higher layer and determines whether its subscription creates congestion by monitoring loss rate over a short period called the *decision time*. If it does, it unsubscribes. Failed join experiments contribute to network congestion though, so RLM scales back the frequency of an individual node's experiments with the size of the multicast group. To allow nodes to learn the appropriate level of subscription rapidly in spite of less frequent experiments, nodes learn by observing the network during other nodes' experiments. If the network becomes congested during another nodes experiment on say, level 3, of a presentation, then the observing node can act as though it conducted a failed level 3 experiment.

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

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

RLM and Thin Streams have a few significant drawbacks. First, they implicitly limit the multi-source IP Multicast model to a single source. Second, they require encoder research to develop codecs that decompose well into a layered architecture. Third, they use extra IP Multicast groups. Using extra groups contributes to an existing problem area for IP Multicast, the large size of the IP Multicast routing tables that must be present in every router. In addition, IP Multicast's small address space is strained even more by using additional groups for each presentation.

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

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

Video Gateways address all of the problems of layered IP Multicast, and even allow some new capabilities that layered systems do not even attempt to provide (such as the ability to transcode data formats and bridge unicast networks). They have a downside, however, that is likely to prevent widespread use of Video Gateways for bridging large groups with many segregated IP Multicast sessions. Video Gateways are statically configured entities.

Each pair of gateways connect exactly two IP Multicast groups, in exactly the way they have been configured. This quickly becomes untenable as the desire to connect more groups, on an ad-hoc basis, is considered. The human effort involved in configuring tens of gateways in order to video conference would be considerable. Nonetheless, the fundamental architecture appears sound. A network of forwarding nodes, similar to Video Gateways, that organize themselves to bridge between segregated multicast groups would be a logical extension.

Application Level Multicast A number of research groups and service providers are investigating services that could be described as a network of Video Gateways. Often referred to as "Overlay networks", these services consists of many application level nodes using unicast to move data to multiple recpients. These include RMX [3], End System Multicast [8], Yoid [7], X-bone [15], and Overcast [9]. 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.

RMX focuses on real-time reliable multicast. As such, its focus is on reconciling the heterogenous 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. In that sense, RMX is the closest relative to Video Gateways. Transcoding is an excellent example of functionality that is best handled by application-level servers, providing an incentive for supporting application-level multicast.

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 by using the primitives proposed here because no nodes are deployed toward the core of the network. Only the receivers 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.

Overcast is a highly-specialized system designed with the express purpose of supporting high-bandwidth, single-source applications. Overcast's contribution is two-fold. First it demonstrates the benefits of an application-level multicast system by providing features that IP Multicast can not support, such as on-demand access to content. Second, simulated evaluations of Overcast's tree building technique provide evidence that reasonable trees can be built under application-level control. Inefficiencies remain, however, because packet duplication can not occur in routers.

3 Primitives

This section proposes two primitives that are intended to allow multicast applications built on end-hosts to perform as efficiently as systems that run in routers, such as IP Multicast. These primitives provide end-hosts with the advantages that routers obtain by being fortuitously located, yet allow more interesting multicast applications because control is retained in external servers.

First, *packet splitting* exposes the ability of routers to duplicate multicast packets at exactly the optimal point 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 splitting allows end-hosts to to request that the duplication occur inside the network.

Second, *packet snooping* exposes the ability of routers to examine routed traffic. Consider how IP Multicast maintains the spanning trees along which packets are transmitted for a particular group. Packets are sent from interested end-hosts toward a rendezvous point. As they are forwarded toward the rendezvous, each router notes the need to transmit multicast packets for the group in question along the same path (in either direction). When a new end-host joins the group, its graft request will also be forwarded toward the rendezvous point. When the graft request makes its way to a router that is already involved in the multicast, it need not be further forwarded and the router in question simply adds the interface on which the packet arrived to the spanning tree.

End-host based system have a much more difficult problem when trying to build efficient distribution trees. There is no general way to determine the topology of the nearby network, so there it is difficult to determine where a new host should attach to the existing distribution tree. Packet snooping allows interested end-hosts to request notification when graft requests pass nearby routers. New nodes simply send graft requests toward a rendezvous. If the packet passes through a router from which a member of the multicast group has requested notification, it can be inferred that the new end-host should receive data from the snooping node.

3.1 Packet Splitting

In an application-level multicast system each node carries out explicit unicast communication with its neighbors in the distribution tree. Inevitably, this requires packet duplication to occur in end-hosts, which, in turn, forces essentially equivalent packets to be transmitted 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.

Fortunately, this duplication is highly stylized. When a packet arrives, the appropriate action can be driven by a simple table lookup. A packet 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 identifier. 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.

The stylized nature of this operation suggests a simple optimization. An end host may ask its router to perform *packet splitting* on its behalf. A packet splitting request packages up a piece of the end

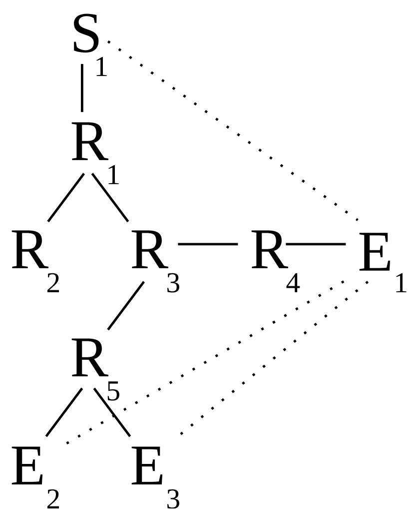

Figure 1: An example application-level multicast distribution tree. Packets are sent from the source S_1 to end-host E_1. E_1 sends the packets on to E_2 and E_3.

host's forwarding table and asks the router to perform the forwarding on behalf of the end node. 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 splitting from router R_4. This alleviates link R_4E_1 from transmitting the same packet three times (once in, twice out). It does *not* however prevent E_1 from receiving packets addressed to it. Whatever else a router is instructed to do, it is always expected to also forward packets using its normal forwarding rules.

The packet splitting primitive is extremely fine grained for maximal flexibility. Notice, for instance, that each specification sets up packet flow from only a single source. To create complete connectivity among four neighbors, an end-host would use four split requests. Each neighbor would appear once as an inbound connection identifier and three times as an outbound connection identifier. Later sections will demonstrate the use of such versatility.

Packet splitting is incrementally deployable 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 advatage of packet splitting and save bandiwdth on their LAN. In fact, the router itself is likely to experience less load compared to a purely end-host based multicast. Instead of receiving multiple packets, performing multiple

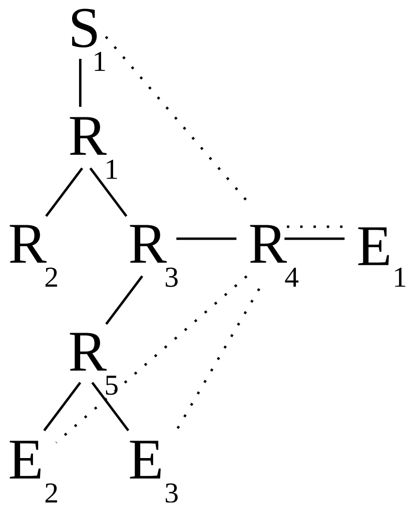

Figure 2: End-host E_1 avoids overloading link R_4E_1 by sending split request $(E_1, \{E_2, E_3\})$ to R_4. *(port numbers elided for clarity)*

route lookups, and transmitting multiple packets, it can receive the packet one time, perform one lookup, and transmit it multiple times. Additionally, the lookups performed for packet splitting are faster than a normal routing lookup. A single hash table lookup suffices as compared to the longest-prefix match that must be performed to route normal packets.

Packet splitting 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 also use it. A router that has been asked to split 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 its neighbor router on that interface to perform the split on its behalf, saving the return transmission. In Figure 3, router R_4 takes advantage of packet splitting by propagating its responsibility to split 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 *stop* splitting packets for E_1.

3.2 Packet Snooping

A router may find that it has been asked to emit the same packet multiple times through the same interface (but not the one on which it was received).

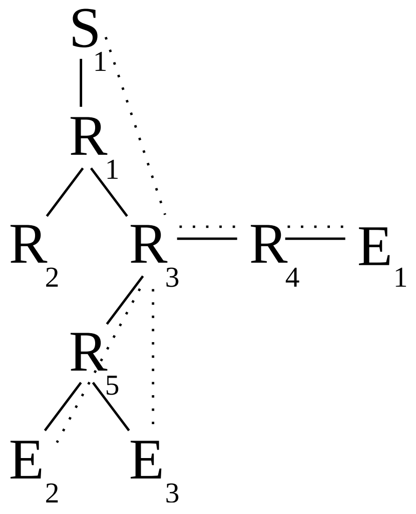

Figure 3: Router R_4 avoids overloading link R_3R4 by pushing E_1's original split request on to R_3.

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 splitting to alleviate this burden. Packet splitting only allows a node to avoid emitting packet duplicates on the interface that the packet originally arrived.

To understand why, consider exactly what splitting request R_3 might send to R_5. If it propagated E_1's original request, $(E_1, \{E_2, E_3\})$, then the splitting would never occur because R_5 would never see packets intended for E_1, the inbound connection identifier. It is tempting to force packet splitting into addressing this problem. Perhaps R_3 could install a packet splitter in R_5 that used E_2 as the inbound connection identifier and E_3 as the only outbound target. Such a splitter would have the intended effect, packets would be split at the appropriate point, but it "lies" to E_3. E_3 would perceive E_2 as the source of these packets despite their logical origin at E_1. For this reason, using packet splitting in the way will be called *deceptive*.

There is a concrete reason to look for a better solution than deceptive splitting. The original intent in decomposing the functionality of IP Multicast into its constituent primitives was to provide increased flexibility to application level multicast systems and to allow incremental deployment. Using the packet splitting primitive deceptively would run counter to those goals.

Consider the distribution tree in Figure 4. Poor decisions were made during the construction of this

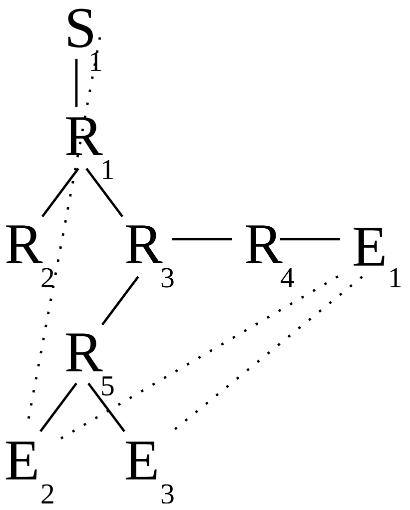

Figure 4: A poorly built distribution tree. If R_3R_5 is a slow link then there is an incentive to ensure that there is only one virtual link across it, allowing transcoding to a thinner stream first before use.

tree. As a result R_3R_5 is used for three virtual links. Using only packet splitting, the damage *can* be undone. E_2 can propagate its splitting duty up to R_3. Similarly, E_1 can propagate its job to R_3. With both splitting requests on hand, R_3 could determine that packets for E_1 should go to E_1, E_2, and E_3. Using a deceptive packet split to push part of that task down to R_5 leaves the network in an efficient state. No link is used more than once.

This complicated "unfolding" of the poor distribution tree is counter to the goal of building flexible multicast systems both because it "lies" about the source of packets and because it requires the concerted effort of all routers between two regions of tangled distribution tree to iron out the problem. This implies that upgrading new routers might not have any benefit in the region they are deployed because of a tangle in another region that can't be corrected.

Furthermore, even if every router in the Internet supported packet splitting, security concerns between distrustful domains may preclude the propagation of splitting requests between certain administrative domains (to ease billing, for example). Or application-level concerns (such as transcoding) might preclude the use of packet splitting at some point in the tree. In such cases a true unicast between end-hosts would be necessary to bridge the border. In Figure 4 if the R_3R_5, was slow link then the the nodes at each end of the virtual links that cross it might choose not to use packet splitting, preferring to handle packets for the link explicitly so that that they might transcode to thin the stream. If that is done, packet splitting can not be counted on to "unfold" the tree. It is important that the number of virtual links between various portions of the network be minimized.

For all these reasons, it would be best to ensure that distribution trees reflect the underlying topology rather than misuse packet splitting and hope everything sorts itself out. A new primitive is needed to help end-hosts build better trees, trees that aggregate nearby nodes.

The desires to avoid using deceptive packet splits and to aggregate nearby nodes are compatible. 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 nondeceptive packet splitting would be sufficient to avoid ever 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 suggests a useful 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 gatweayed by a single router — the virtual topology will reflect the underlying topology. This, in turn, ensures that deceptive packet splitting will be unnecessary in order to enter that portion of the network because deceptive packet splitting is only needed to push multiple packets down the same link.

To build better trees, end-hosts need more information. *Packet snooping* is a simple primitive designed to export a router's ability to observe packets as they are forwarded. The packet snooping primitive is designed to allow end-hosts to duplicate IP Multicast's ability to aggregate paths from multiple end-hosts at the first router shared by the paths. To do so, end-hosts must be able to snoop not only at the router closest to them but in as many routers as possible leading to the multicast rendezvous point.

To use packet snooping, an end-host propagates a snoop request toward its rendezvous point by addressing it to the rendezvous. A snoop request is implicitly intended to snoop packets that match the destination address and port of the request packet. Each packet snooping capable router along the path installs a tap to observe any further packets addressed to the same address (and port) *unless an equivalent tap already exists at the router.* Whenever the router observes such a packet, a copy is forwarded to the snooping host in addition to the normal forwarding performed on the packet. The source address and port are not modified in the copy. Only the destination is changed to reflect the address and port of the snoop request. Notice that no information is lost from the snooped packet. Although the original destination address and port must be overwritten in order to delivery the copy, the receiver can map the port on which it receives the copy back to the original destination.

There are two important ideas to understand about the propagation of snoop requests. First, snoop requests are *quashed* if they match an existing snoop request at a router with the same matching predicate. The snoop has already "served its purpose" at this point, because a copy of the request will be sent to the snooper that has already installed a snoop request. This will allow the two nodes to rendezvous and build a better tree. This is equivalent to the quashing of IP Multicast graft requests when they reach a portion of the distribution tree.

The second interesting fact about snoop request propagation is that they are normal IP packets and will be propagated by normal IP routers. This allows packet snooping to be effective across regions of the network that do not implement packet snooping. As long as a least one router on the shared portion of two nodes' paths to the rendezvous is "snoop capable", information will be gained that will allow the application-level distribution tree to more accurately reflect the underlying topology. When there is *no* support for snooping between a node and a rendezvous point, the node may fall back on previous work in building application-level distribution trees.

At least one thorny issue remains in the design of packet snooping. The question of exactly what a packet snooping request looks like is surprisingly important. Unlike packet splitting requests, packet snooping requests should be routable by unmodified Internet routers. The packet should certainly be IP then. If the requests are UDP, then a way of distinguishing them from normal UDP packets is necessary. One avenue to is to set aside some UDP port to mark snoop requests, but doing so would change the meaning of a previously normal UDP packet. To avoid that difficulty, another IP protocol number might be used, but that has traditionally been shunned in the Internet community. More study is necessary to choose the lesser of these evils.

4 Multicast

The previous section proposed two useful primitives that application level multicast systems can take advantage of in order to increase efficiency. In this section three sample usages of these primitives will be described to demonstrate the versatility of the proposals.

First, a system that mimics the semantics of IP Multicast is described. This is a minimum bar that such primitives must pass to be considered as a replacement to IP Multicast. In fact, the proposed system will provide *more* functionality than IP Multicast. Second, a heterogenous multicasting system will be described that has more functionality that RLM, Thin Streams, or Video Gateways. Finally, a novel multicast application will be described, demonstrating the flexibility provided by using application level control. Special attention will be paid to the complexity of the solution compared to reliable IP Multicast proposals. This is intended to demonstrate the greater constructive power of the proposed primitives compared to IP Multicast.

4.1 Best-effort datagrams

To demonstrate the effectiveness of the proposed primitives this section describes how they can be used to emulate IP Multicast. It is easiest to begin by describing a mapping between elements of IP Multicast to elements of the IP Multicast 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	Snoop 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 [4], DVMRP [16], CBT [2]) have proposed different rendezvous points. In emulation, a far simpler approach is taken. The rendezvous will be 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 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 snoop request. The snoop request will either reach the rendezvous, indicating that no other nodes have joined the tree yet, or it will be quashed when it arrives at a router that is already snooping for the same rendezvous. In the first case, no action is required since the new node is the only member of the group. In the second case, the joining node is notified that its snoop has been quashed, and the node that caused the quashing is notified that a matched packet has been observed.

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 quash occured to be the parent. This can be determined from the TTL field in the snooped packet when compared to the TTL of the quash notification. 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.

Whenever a node's 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 splitting requests. To allow complete connectivity, a node makes as many splitting requests is it has neighbors, as described in Section 3.1.

There is a final feature of IP Multicast that is difficult to emulate. When a packet is sent using IP Multicast, its final hop often takes place over a broadcast medium, such as Ethernet. When transmitting over such a medium, IP Multicast can avoid multiple transmissions because it uses the multicast properties of the medium. The difficulty is not in using the broadcast medium, but in avoiding the additional use of unicast transmission in the last hop as well.

For example, suppose three group members are on the same Ethernet segment. One of these nodes will be the parent of the other two because of the tree construction technique. The parent might ask its router to do the splitting for it. Furthermore the parent might be clever, and direct the router to send the packets to one of the Class E addresses used by IP Multicast. Doing so would cause the router to emit packets using the broadcast attribute of the local medium. The children could be told to listen for such transmission. The problem is that the router will *also* deliver packets, using unicast, to the parent. This inefficency is unfortunate, but appear inevitable, unless the restriction that forces packet split requests to implicitly include the split requester is relaxed.

On the upside, emulating IP Multicast in this way has a number of benefits. An obvious benefit is the much 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. Heterogeneous multicasting is also a easy addition, as described in the next section.

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. Of course, the point of proposing new primitives is to show that systems using them can either be built more simply or with more features than with IP Multicast. In this case a more featureful system can be built.

RLM (and Thin Streams) 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 node will be unable to get a high-quality stream from one source and a low-quality stream from another. If each node uses a separate set of groups then a node will need to subscribe to (at least) one multicast group for each node in the group. It will quickly become unscalable just to learn all the groups in question, let alone run join experiments on all the high layers.

A multi-source, heterogenous multicast system can be made by combining the transcoding of Video Gateways, with the tree building of application-level multicast systems, and the efficiency of the newly proposed primitives.

Using packet snooping, nodes can arrange themselves into an efficent distribution tree as in IP Multicast emulation. Each node makes splitting requests to forward all traffic amongst its neighbors. Each node also exchanges congestion information with each of its neighbors. If a link is found to be suffering from congestion then the use of split requests to forward along that path is suspended. Instead the stream is thinned at each end of the link.

Because the need to thin may develop at on any virtual link, it is important to make it as computationally simple as possible. Although Video Gateways demonstrate a novel technique to simplify transcoding, layered codecs appropriate for RLM would provide an even lower computational burden for the slow link's endpoints and be more finely adaptable to various levels of congestion.

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 loss of packets. Applications envisioned for reliable multicast, such as more 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's certainly 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 want to receive a high-fidelity copy of the entire conference call.

Applications of this sort require 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 multicast 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.

FTC is so simple compared to reliable IP Multicast protocols that it really drives home the fact that application-level multicast systems are incredibly flexible. The application-level distribution tree is setup as for IP Multicast emulation. Like heterogenous multicast, out of band communication between parents and children determine the level of transmission that is possible without creating congestion. When possible, splitting is used to increase the efficiency of the transmission, but as for heterogenous multicast, pairs of nodes may choose to avoid the splitting primitive so that packets may be dropped, avoiding congestion. When a 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 offpeak 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. The 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 single channel. The channel is buffered, perhaps by 10 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 to ask for rebroadcast is to ask the *entire group.* A node doesn't know who else to ask. Second, because an application-level system is being built for an express purpose, it is possible to design the application-level nodes for the task at hand. Nodes can contain large disks to support "hole filling" hours after the original transmission. The designers of IP Multicast could not ask routers to buffers gigabytes of data just in case someone might want it later.

5 Future Work

A number of issues related to the proposed primitives require further study. First, no description of how split and snoop requests should be removed has been given. An explicit withdrawal protocol, combined with a timeout mechanism would seem appropriate, but the details have not been worked out.

On another front, the security implication of these primitives is difficult to assess. On the one hand, packet snooping is not as dangerous as it sounds. Only packet snoop requests are susceptible to snooping, which is roughly requivalent to saying that only Multicast graft messages as susceptible to snooping. This is not a terrible imposition. On the other hand, packet split requests appear to offer intruders the ability to sniff network traffic of other nodes. One precaution is for routers to only accept splitting requests for an IP address X if the request arrives on the interface to which packets for X would be directed. Furthermore, the protocol for requesting splits should probably involved a handshake between the router and the requester. This would mean that a machine's traffic can only be sniffed if traffic to the machine has already been compromised.

6 Conclusions

Multicast is an important capability for an increasing number of modern applications, but IP Multicast supports only a single kind of multicast. Furthermore it has proven difficult to build different kinds of multicast on top of IP Multicast.

Instead of providing a complete multicast primitive as a network service, IP should provide capabilities that allow end-hosts to build application-level multicast systems. Packet splitting and packet snooping allow end-hosts to build efficent multicast systems by exporting the unique capabilities of routers. Packet splitting allows end-hosts to benefit from the advantageous position of routers for packet duplication. Packet snooping allows end-hosts to use routers to learn enough about the topology of the network to build efficent distribution trees.

Both of these primitives are incrementally deployable. Application-level multicast systems *can* be built without them, but in portions of the network in which they are deployed, these systems can be considerably more efficient.

Packet splitting and snooping are demonstrably effective, allowing the implementation of various multicast semantics. Further, it is hoped that the versatility of these primitives might lead them to be used in previously unconsidered ways.

References

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

[2] T. Ballardie, P. Francis, and J. Crowcroft. Core based trees (CBT) an architecture for scalable inter-domain multicast routing. In *Proc. ACM SIGCOMM Conference (SIGCOMM '93)*, September 1993.

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

[4] S. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended LANs. *IEEE/ACM Trans. Networking*, 8(2):85–110, May 1990.

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

[6] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *Proc. ACM SIGCOMM Conference (SIGCOMM '95)*, August 1995.

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

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

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

[10] B. N. Levine, D. B. Lavo, and J Garcia-Luna-Aceves. The case for reliable concurrent multicasting using shared ack trees. In *Proc. ACM Multimedia*, November 1996.

[11] J. C. Lin and S. Paul. Rmtp: A reliable multicast transport protocol. In *Proc. IEEE Infocom*, March 1996.

[12] C.-G. Liu. Local error recovery in SRM: Comparison of two approaches. Technical report, Univeristy of Southern California, 1997.

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

[14] Jörg Nonnenmacher, Ernst W. Biersack, and Don Towsley. Parity-based loss recovery for reliable

multicast transmission. In *Proc. ACM SIGCOMM Conference (SIGCOMM '97)*, pages 289–300, September 1997.

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

[16] D. Waitzman, C. Partridge, and S. Deering. Distance vector multicast routing protocol. Technical report, ARPANET Working Group, DDN Network Information Center, November 1988.

[17] L. Wu, R. Sharma, and B. 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.