The Design and Implementation of Free Riding Multicast

by Andrey Ermolinskiy

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Scott Shenker
Research Advisor

(Date)

* * * * * * *

Professor Ion Stoica
Second Reader

(Date)

Abstract

In this report, we revisit a much explored topic in networking - the search for a simple yet fully general and efficient multicast design. The many years of research into multicast routing have led to a generally pessimistic view that the complexity of multicast routing - and inter-domain multicast in particular - can only be overcome by restricting the service model, as in the case of single-source multicast. This report proposes a new approach to implementing IP multicast that, we hope, might lead to a reevaluation of this commonly held view.

This report proposes Free Riding Multicast (FRM) - a new approach to network-level multicast routing and our primary contribution is a new protocol aimed at simplifying cross-domain deployment of IP multicast. By replacing distributed tree computation with a multicast variant of source routing, FRM sidesteps much of the complexity associated with traditional solutions, while retaining efficiency and full generality of the service model.

Contents

List of Figures

List of Tables

1 Introduction

In 1990, Steve Deering proposed IP multicast [1] - an extension to the traditional IP unicast service model for efficient multipoint communication. This service model offers endhosts a very simple abstraction: a host can *join* and *leave* a multicast group G at any time and any host can *send* packets to a group G. As with unicast, the internals of the network are expected to provide the foundational best-effort delivery service for multicast packets, atop which richer application-level functionality may be implemented in the endhosts. The multicast service model provides two key benefits:

1. The efficient use of network bandwidth for multipoint communication.
2. The indirection facility of a group address, which allows for network-level rendezvous and service discovery.

Deering's seminal work triggered an era of research on the implementation and applications of IP multicast, but despite many years of progress, the practical impact of this research in terms of actual deployment in the Internet remains somewhat unclear. On the one hand, support for multicast is built into virtually every IP router and every end-system network stack. The multicast service is often deployed within enterprise networks, as well as within the boundaries of a single ISP. However, we have seen very little cross-provider deployment of multicast in the Internet and today, more than fifteen years after Deering's proposal, the vision of a ubiquitous multicast "dialtone" remains an elusive, if not altogether abandoned, goal.

Theories abound for why this vision never materialized and broadly speaking, most of them can be viewed as questioning the viability of IP multicast on two separate fronts:

1. The *practical feasibility* given the apparent complexity of deploying and managing multicast at the network layer, as well as the unresolved technical challenges associated with multicast routing (e.g., scaling of multicast forwarding state in routers).
2. The *desirability* of supporting the multicast function at the network layer. In particular, many have questioned whether the demand for multicast applications justifies the complexity of its deployment and whether ISPs have meaningful economic incentives to deploy IP multicast. Furthermore, network-level multicast requires routers to maintain and manage per-group (and, for some protocols, per-source) state, which can be seen as a violation of the end-to-end principle [2] - an architectural cornerstone of the early Internet that argues for simplicity and statelessness in the network core.

In this report, we attempt to tackle the abovementioned issue of practical feasibility and propose a simpler approach to implementing IP multicast that we call *Free Riding Multicast (FRM)*. In broad terms, FRM offers two key advantages over existing solutions:

1. By leveraging knowledge of existing unicast routes, FRM virtually eliminates the need for a distributed multicast route computation mechanism, thereby side-stepping much of the complexity associated with traditional approaches to multicast routing.
2. A domain's participation and use of multicast is signaled via the same channel as in the unicast case, namely the Border Gateway Protocol (BGP). This, in turn, offers net-

work administrators a familiar framework within which to tackle the management of a multicast service, including the issues of *access control* and *accounting*.

These advantages, however, come at a cost and the principal tradeoff in our approach is the avoidance of distributed route computation at the expense of optimal efficiency. Unlike traditional approaches that tend to rely on distributed protocol mechanism, FRM tilts the burden of multicast route construction to the *internals* of a router. As a consequence, FRM requires more storage and algorithmic sophistication at routers and is somewhat less efficient in bandwidth consumptions that traditional solutions. We believe, however, that given current technology trends that can endow routers with substantial memory and processing resources on the one hand and our continued difficulties taming wide-area routing algorithms on the other, this tradeoff may be worth exploring.

Our contribution is a new approach to IP multicast routing that we hope would lower the technical barriers to its deployment. While the primary focus of our work is on inter-domain multicast, for which deployment challenges proved particularly acute, the basic FRM scheme can be extended to the intra-domain scenario as well. And while we make no claims to understand the "market" for IP multicast and will not try to tackle the desirability aspect of the discussion, it is interesting to note that many of the applications that originally motivated the research on multicast have (finally) arrived.

One example is massively multiplayer online games (MMORPGs) with reports of annual subscription growth in the range $30 - 100\%$ and up to 5 million active subscriptions in a year [3]. In these games, a player's actions must be propagated to other players in the "virtual" vicinity and currently, game operators achieve this by deploying multiple servers, each responsible for a certain region of the virtual world, that relay communication between players. Hence, for n players in a virtual region, the bandwidth requirements at the corresponding server vary between $O(n)$ to $O(n^2)$ depending on the extent to which latency constraints allow multiple updates to be aggregated. [4]. In a simple scenario, game operators might use multicast to reduce server-side bandwidth consumption to between $O(1)$ and $O(n)$. In a slightly more sophisticated scenario, players might multicast updates directly to other players in the same region, thereby fully offloading data forwarding from servers.

Another promising example is the adoption of the IP-TV technology [5] with several providers already in customer trials. Multimedia conferencing, RSS dissemination, software updates, peer-to-peer file sharing, and grid computing are additional examples that could potentially benefit from the presence of network-level multicast support.

The primary focus of this report is the design and evaluation of FRM and the remainder of this document is structured as follows: We begin with a brief review of prior work on multicast routing in Section 2. We discuss our high-level approach and present the detailed design of FRM in Sections 3 and 4, respectively. Section 5 evaluates the performance and router resource consumption and Section 6 presents our prototype implementation of FRM. Finally, we conclude in Section 7.

2 Related Work

Multicast routing has been the topic of substantial research over the years and we have witnessed the emergence of a large body of competing designs and approaches. We begin our survey of prior work by drawing a fundamental distinction between the techniques that disseminate packets along traffic along a set of shortest-path trees (one for each sender) and those that make use of a single tree shared across all senders. We also briefly touch upon the restricted service model of single-source multicast and the overlay-based approach to multicast routing.

2.1 Shortest-Path Multicast Delivery

In this scheme, a multicast packet from source S is delivered to each member of the group along the shortest routing path from S. This approach minimizes the end-to-end transmission latency, as perceived by a receiver, but requires routers to maintain per-source state. Hence, this scheme can be seen as an attractive choice for supporting delay-sensitive and bandwidth-intensive applications with a relatively small number of simultaneous senders; examples of applications that fall into this category include multimedia conferencing, the various tools for online collaboration, and the delivery of streaming media.

In his pioneering work [1], Steve Deering proposed a number of extensions to the existing protocols for unicast routing with the goal of enabling shortest-path multicast delivery and this work has laid the foundation for DVMRP and MOSPF.

2.1.1 Distance Vector Multicast Routing Protocol

DVMRP provides a multicast tree maintenance mechanism on top of a conventional distance vector protocol for unicast routing. It uses a broadcast-and-prune scheme: a multicast packet sent by a source S is initially forwarded toward all endhosts in the topology (regardless of their group membership status) via a shortest-path broadcast tree rooted at S. Leaf routers that receive unwanted packets send periodic non-membership reports to their parents on the tree, which in turn aggregate and propagate these reports further toward the source and the initial broadcast tree is eventually transformed into a multicast tree that properly reflects the group membership status.; Packet forwarding in DVMRP makes use of the *reverse shortest path* technique, which works as follows: When a multicast packet originated by a source S and addressed to group G is received by a router R, it forwards the packet further only if the interface on which it arrived is the outgoing interface for the next hop on the shortest path from R to S. The packet is forwarded on all interfaces for which R serves as a parent in the broadcast tree rooted at S, except those with active non-membership reports.

DVMRP provides efficient shortest-path delivery trees from any potential source, but exhibits poor scaling properties because it requires routers to maintain per-source state. Specifically, for each potential source S in the topology, a router must maintain the set of its child interfaces with respect to S. Furthermore, the tree construction mechanism is sender-driven and penalizes non-members (i.e., routers off the dissemination path) by requiring them to participate in the generation and propagation of non-membership messages.

2.1.2 Multicast Open Shortest Path First

The MOSPF protocol is a fairly incremental extension to Open Shortest Path First - a widely-used unicast routing scheme from the link state family of protocols. In MOSPF, routers augment their link state advertisements (LSAs) with descriptions of groups that have members on that link and each router maintains the global group membership state as part of its link state database. When a new group member appears, its designated router notifies all other protocol participants of the membership change by re-broadcasting the respective SLA. In this scheme, forwarding a multicast packet originated by a source S requires computing the shortest path tree from S to the current set of receivers and forwarding copies of the packet toward the immediate children on the tree. For efficiency reasons, the results of tree computation may be cached and reused when processing subsequent packets addressed to the same group.

MOSPF constructs efficient dissemination trees, but is limited to networks that run link state protocols. One must also note that a tree for a given group must be recomputed after each membership change and the computational overhead grows with the number of links in the topology.

In summary, while DVMRP and MOSPF both provide efficient shortest-path multicast delivery, path efficiency comes at a substantial cost and presents a scalability challenge for the following reasons:

1. Setting up the forwarding state requires flooding certain information throughout the entire topology. In DVMRP, establishing the non-membership state in intermediate routers requires broadcasting the initial data packet. MOSPF restricts the dissemination of data packets to the actual delivery path from the source to the set of receivers, but requires periodic flooding of group membership updates.
2. The fast-path forwarding state at transit routers grows linearly with the number of active multicast groups. Furthermore, since IP multicast group addresses are, in effect, flat identifiers and do not easily lend themselves to topological aggregation, the forwarding state requirements can also be expected to grow linearly with the number of senders.

For the above reasons, these protocols are typically used to support multicast within a single domain and they are not well suited for large inter-domain environments characterized by sparse membership and high rates of membership churn.

2.2 Shared-Tree Multicast Delivery

A number of shared-tree multicast protocols have been proposed to address the scaling limitations of shortest-path delivery techniques in wide-area environments. In this approach, a multicast group is associated with a small number of dissemination trees (often one) shared among all senders. A sender unicasts its outgoing packets to the root of the shared tree, whose address is made well known, which in turn forwards the packet toward the set of group members along the shortest path.

2.2.1 Core-Based Trees

The CBT protocol proposed by Ballardie *et al.* in [6] associates each multicast group with a well-known *core* router that functions as the root of a single dissemination tree shared across all senders. Senders unicast their outbound packets to the group's core and when a new member joins a group G, its first-hop router sends a $Join(G)$ message along the reverse path to the core. This message, in turn, allows intermediate routers on the path between the core and the new receiver to set up the necessary forwarding state. Specifically, the interface on which the $Join$ message arrived is added to the list of children for group G and the outgoing interface becomes the router's parent for that group.

2.2.2 Protocol-Independent Multicast

PIM-SM [7] is another scheme for supporting multicast in wide-area environments characterized by sparse group membership. While similar in spirit to CBT, PIM-SM offers additional flexibility by allowing leaf routers to switch between shortest-path and shared-tree delivery on per-source basis and choose the most appropriate delivery method based on current network conditions and traffic characteristics. Each multicast group G is associated with a well-known *rendezvous point* (RP_G) router that plays a dual role:

1. It serves as the root of the group's shared tree.
2. For shortest-path delivery, it serves as a point of contact between the group's senders and receivers, allowing them to discover each other and establish the requisite forwarding state in intermediate routers without a network-wide broadcast.

Shared-tree multicast delivery schemes such as CBT and PIM-SM avoid the blowup of router state inherent to shortest-path protocols and thus offer a substantial improvement in scalability, but forfeit path optimality. For this reason, multicast architectures based on the shared tree approach are well suited for delay-insensitive applications that must support a large number of simultaneous senders and examples might include the various mechanisms for service discovery and network-level rendezvous.

The above two protocols have another desirable property: receiver-driven tree formation. These protocols succeed in constraining the flow of their control and data traffic to the distribution tree, which makes them an appealing choice for implementing inter-domain multicast[1].

Unfortunately, shared-tree protocols give rise to a number of non-trivial issues pertaining to the placement and ownership of RPs when deployed in wide-area environments that span multiple administrative domains. The RP can be seen as a single point of failure, whose availability in effect determines the availability of the respective multicast group and historically, ISPs proved reluctant to depend on RPs operated by other providers. In addition, the use of shared-tree mechanisms leads to reduced service availability in the face of network partitions: a sender partitioned away from the rendezvous point loses its ability to communicate with the entire group. Conversely, a group member that lacks bidirectional connectivity with the RP

[1]The caveat is that shared-tree protocols typically require flooding the mappings between group addresses and the corresponding RPs via some out-of-band mechanism

cannot receive any of the data sent to the group by any source S even if this member has a valid communication path to S.

2.3 Single-Source Multicast

It has been suggested that many of the target applications for IP multicast require delivery from only a single, typically well-known source, as is clearly the case with various streaming media delivery services, and that much of the complexity associated with traditional multicast routing can be eliminated by restricting the service model to allow only a single endhost to act as a sender for a given group.

The IP Multicast Channels architecture and the associated EXPRESS protocol proposed by Holbrook *et al.* in [8] exemplify this approach. In their scheme, endhosts subscribe to multicast *channels* that are identified by the source address ($SrcAddr$) and the channel number ($ChanNum$). To join an EXPRESS channel, an endhost sends a $Join(SrcAddr, ChanNum)$ request directly to the source, which causes intermediate routers on the path to establish forwarding state. Since the sender's address is assumed to be well-known, this mechanism does not require indirection via the use of a rendezvous point, thereby eliminating the need for an additional RP address discovery mechanism, as well as the abovementioned non-technical issues pertaining to their placement and ownership.

Although only one source may transmit to a given channel, multi-source applications can be accommodated in EXPRESS either by defining multiple channels (one for each source) or via the use of a Session Relay (SR) - a designated node that consolidates traffic from multiple senders and retransmits it on a single well-known channel, in effect acting as the root of a single tree shared across all senders. While the EXPRESS protocol is similar to PIM-SM in providing the ability to dynamically adjust between shortest-path and shared-tree delivery, the key point of distinction is that Session Relays in EXPRESS are application-level entities and control over the placement SRs and the policy for switching between the shared and the shortest-path delivery methods is delegated to the application and/or its users.

2.4 Overlay Multicast

While advocates of IP multicast have argued that the performance benefits of supporting multicast at the network layer justify the burdens of increased protocol complexity and router resource requirements, achieving widespread deployment of IP Multicast across the Internet has proven to be an increasingly uphill battle, which has led others to adopt precisely the opposite view and seek alternative approaches. In particular, there has been considerable interest in peer-to-peer overlay-based schemes, where participating endhosts organize themselves into a overlay mesh and cooperate on providing the multicast routing functionality.

In the Narada protocol for end-system multicast proposed by Chu *et al.* in [9], members of a group arrange themselves into a self-organizing overlay topology and neighboring nodes periodically exchange their lists of known members, so that every node eventually learns about all other members of the group. The overlay topology is maintained using a dynamic mechanism that allows any node to adjust its neighbor set by adding and droppings links with the

goal of improving the overall efficiency of the overlay with respect to latency and bandwidth. A path vector protocol deployed on top of the overlay allows each potential source to discover a routing path to every group member and construct a multicast delivery tree.

The packet forwarding mechanism in Narada follows the conventional reverse path forwarding technique: A member M that receives a multicast packet via a neighbor N forwards the packet iff N is the next hop on the shortest path from M to the packet's source.

Unlike network-level multicast, applications that utilize an overlay-based scheme are easily deployable in the current Internet, as they do not require ubiquitous router support for multicast routing, but this useful property comes at the expense of efficiency. An overlay dissemination tree, in which only endhosts can serve as replication points, is nearly always less efficient than what could be achieved with network-level multicast support, as some of the links will inevitably see redundant packet transmissions. Absence of router support also means that endhosts are responsible for the maintenance of the global group membership state, which presents a scalability challenge. For this reason, endhost-based systems such as Narada are limited to supporting only small to medium-sized groups.

To summarize, while both network-level and overlay solutions offer the benefits of multicast, namely efficient group communication and address indirection, they come with highly different tradeoffs, offering efficiency on the one hand and ease of deployment on the other.

Having provided a brief overview of prior research on multicast routing to help place our work in context, we now proceed to the discussion of Free Riding Multicast. In the following two sections, we describe our high-level approach, summarize the key design features that differentiate FRM from earlier work, and examine the tradeoffs in our design.

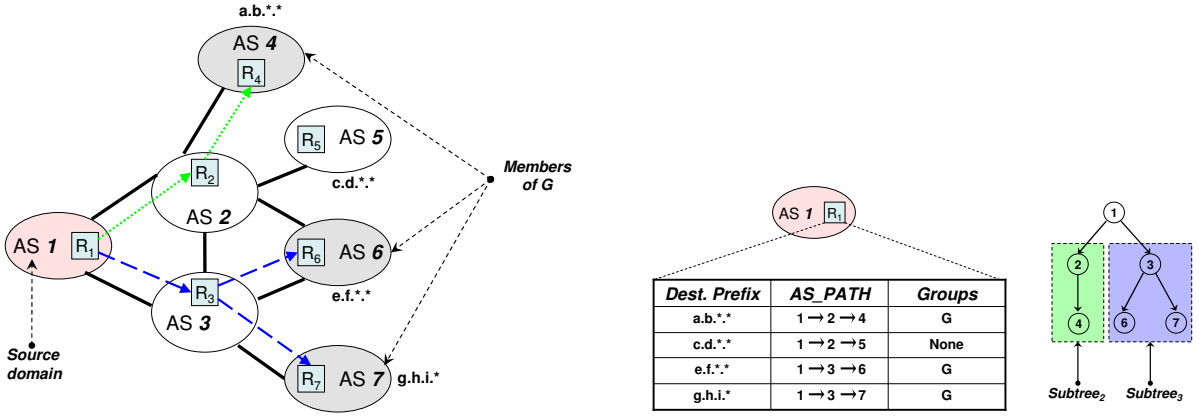
3 FRM: Approach Overview

In very broad terms, multicast packet delivery requires:

1. A *group membership discovery mechanism*: given a destination group G, identify the set of current members in G.
2. A *route discovery mechanism*: Given a group member M, identify a valid routing path from the packet source to M.

While many of the existing solutions combine these two components into a single monolithic mechanism, we take a slightly different approach by explicitly separating group membership maintenance from route discovery. This separation is crucial to our design and offers the following advantage: once group members are known, any source can locally compute the entire dissemination tree for an outgoing multicast packet from the union of its unicast routes to each member of the group. As we demonstrate below, centralizing the tree computation procedure allows us to sidestep many of the complexities associated with the traditional approaches to multicast routing, many of which rely on a distributed protocol for the construction and maintenance of dissemination trees.

At a very high level, the FRM scheme operates as follows: A domain's border router augments its BGP prefix advertisements with an encoding of the multicast group addresses



(a) A sample AS-level topology. $AS1$ originates a packet destined for group G with 3 member domains ($AS4$, $AS6$, and $AS7$).

(b) Content of the BGP Forwarding Information Base (FIB) at $AS1$ and the resulting dissemination tree.

Figure 1: **FRM forwarding in a sample AS-level topology.**

with active members in its domain [2] (we discuss the specifics of this encoding in Section 4) and disseminates this information via the standard BGP route advertisement mechanism. As a result, every border router learns which multicast groups are present in each destination prefix and this information is maintained as part of per-prefix state in the BGP Routing Information Base (RIB).

To forward a multicast packet addresed to G, the source domain's border router (denoted R_s) scans its BGP RIB to identify the set of prefixes with members in G and computes the dissemination tree from the union of the AS-level unicast paths (as specified by the value of the *AS-PATH* attribute) to all member domains. Having constructed the delivery tree, R_s forwards a single copy of the packet to each immediate child domain on this tree along with an encoding of the subtree rooted at that child. Upon receipt of a packet from R_s, the child domain's border router examines the encoding supplied in the packet and, in turn, forwards a copy of the packet toward its children on the tree, and so forth.

We illustrate this process in Figure 1(a). A packet originated by a host in *AS 1* and destined for a multicast group G arrives at the domain's border router (R_1). This router examines its BGP RIB(shown in Figure 1(b) on the left) and determines that prefixes *a.b.*.*.*, *e.f.*.*.*, and *g.h.i.** (which correspond to domains *AS 4*, *AS 6*, and *AS 7*, respectively) have active members in this group. Using this information, R_1 computes the dissemination tree from the union of the AS-level unicast paths from the local domain (*AS 1*) to each of the member domains. The resulting tree is shown in Figure 1(b) on the right. Finally, R_1 transmits one copy of the packet to R_2 along with an encoding of $Subtree_1$ and, accordingly, another copy to R_3 along with an encoding of $Subtree_3$.

[2] In the remainder of this report with use the terms *domain* and *autonomous system (AS)* interchangeably.

While our approach can be viewed as extending MOSPF to the inter-domain arena, it is important to note that unlike link-state protocols, BGP does not provide its participants with a global view of the network topology. A domain's border can leverage its unicast routes obtained from BGP to compute a valid multicast delivery path from itself to a given set of receivers. However, a BGP speaker has no easy way of determining (in the general case) whether its domain lies on the path from another source to a given receiver, which complicates multicast forwarding at transit routers. Returning to the example of Figure 1(a), R_2 and R_3 may both have BGP entries for the destination prefix *e.f.*.** and can therefore infer the presence of group members in the respective domain (*AS 6*). However, when R_2 receives a packet from R_1, it has no easy way of knowing not to forward the packet toward *AS 6* because the local BGP information at R_2 does not reveal that its domain is *not* on the unicast path used by *AS 1* to reach *AS 6*. In fact, the choice to route via $1 \rightarrow 2 \rightarrow 6$ rather than $1 \rightarrow 3 \rightarrow 6$ may have been the result of a local policy decision at *AS 1* and BGP does not currently provide a facility that would allow a participating domain to explicitly externalize its routing policies to its peers.

FRM addresses this problem by requiring the source border router to augment each outgoing packet with an encoding of the dissemination tree and the forwarding decision logic at intermediate routers makes use of this additional information, but this is by no means the only feasible design choice. As an alternate option, one could envision employing a limited form of DVMRP-style broadcast-and-prune, although this approach would raise concerns over scalability and increased bandwidth overhead for non-members.

At a slightly more speculative level, the per-packet overhead associated with the tree encoding can be altogether eliminated by modifying the underlying unicast routing infrastructure and replacing BGP with a policy-compliant link-state protocol. While this approach would face a daunting barrier to deployment in the current Internet, it might be interesting to explore this design in the context of a clean-slate perspective on the Internet architecture - a research area that is rapidly gaining momentum thanks to initiatives such as the NewArch project [10], NSF's GENI [11], FIND [12], and [13]. As an intermediate step in this direction, we propose an incremental extension to BGP aimed at providing routers with a broader view of the global topology, which in turn allows us to reduce the amount of per-packet bandwidth overhead incurred by the FRM forwarding scheme. We discuss and evaluate this extension in Section 5.2.2.

We now proceed to a detailed description of the design, followed by a summary of the core tradeoffs.

4 FRM: Design

The high-level design of FRM can be separated into two components: *group membership discovery* and the *packet forwarding mechanism* and we now present our solutions for each along with a qualitative examination of router resource requirements, including the storage overhead due to the global group membership state and the computational cost of packet processing.

4.1 Group Membership Maintenance

As mentioned in the previous section, our design attempts to leverage existing unicast routes for multicast packet delivery and toward this end, we extend BGP to include per-prefix group membership information.

We assume that a domain's border router discovers which group are present (i.e., have active members) in its local domain through interaction with the intra-domain multicast routing protocol. For instance, if PIM-SM is used for intra-domain multicast delivery, a group's internal Rendezvous Point could be configured to notify the border router of domain-wide membership changes (when the first member joins or the last member leaves). Upon receiving a packet addressed to a group with members in the local domain, the border router could tunnel the packet to the respective RP, which would in turn initiate local dissemination and vice versa.

A domain's BGP speaker augments its route advertisement with an encoding of the group addresses with active members in the advertised prefix. While one can consider several distinct techniques for encoding this membership information, simple enumeration would leave limited opportunities for scaling to large numbers of groups. Instead, we use an alternate method that achieves space efficiency by encoding the list of locally active groups into a bloom filter (denoted GRP_BF) - a probabilistic data structure that compresses a set of elements into a bitfield of predetermined length [14]. The use of bloom filters introduces the possibility of false positives, meaning that a domain may on occasion receive traffic for an unwanted multicast group. It is important to note, however, that our scheme is not susceptible to false negatives and therefore never results in service being denied to legitimate group members.

To deal with unwanted traffic resulting from false positives, the receiving domain's border router R can simply drop all such traffic or recode its advertisement into multiple bloom filters such that the offending false positive is eliminated. Yet another option would be to let the receiver inform the upstream domain U (i.e., the previous hop on the dissemination path) by the means of a non-membership report similar to DVMRP. In response, U could install an explicit filter rule to cease forwarding the offending group's traffic to R and we assume the use of this method in the remainder of the report.

Clearly, the length of a group bloom filter represents a tradeoff between the router memory requirements due to GRP_BF state on the one hand and the amount of unsolicited traffic a domain may receive (and hence the number of filter entries needed to handle it) on the other. In fact, knowing the number of filter entries a domain is permitted to install at its upstream provider(s) provides a way to reason about the choice of the filter length (denoted L_{grp}).

If we assume that each false positive results in a filter being installed in the upstream provider's domain, we can calculate the maximum false positive rate that a receiver can tolerate from the number of available filter entries (f), the number of group addresses to be encoded (G), and the total size of the multicast address space (A):

$$Rate_{fp} = Min(1, \frac{f}{A - G}).$$

The above false positive rate can then be used to compute the appropriate filter size L_{grp}.

We have:

$$Rate_{fp} = \left(1 - \left(1 - \frac{1}{L_{grp}}\right)^{HG}\right)^{H} \approx \left(1 - e^{-HG/L_{grp}}\right)^{H},$$

where H is the number of hash functions used by the bloom filter. Solving for L_{grp} yields:

$$L_{grp} = \frac{-HG}{\ln(1 - (Rate_{fp})^{1/H})}.$$

As an example, if we assume the standard IPv4 multicast address space (224.0.0.0 - 240.0.0.0) and 100 upstream filters, encoding 100,000 groups would require maintaining the false positive rate below $6 \cdot 10^{-7}$. Using the optimal number of hash functions ($H = 21$), we can encode these groups into a bloom filter of length 2,982,200 bits or approximately 364KB.

To simplify manipulation (compression, expansion, and aggregation) we require L_{grp} to be a power of two. Note that if a provider domain employs prefix aggregation, the aggregate group filter can be easily computed as the bitwise OR of the corresponding customer GRP_BFs.

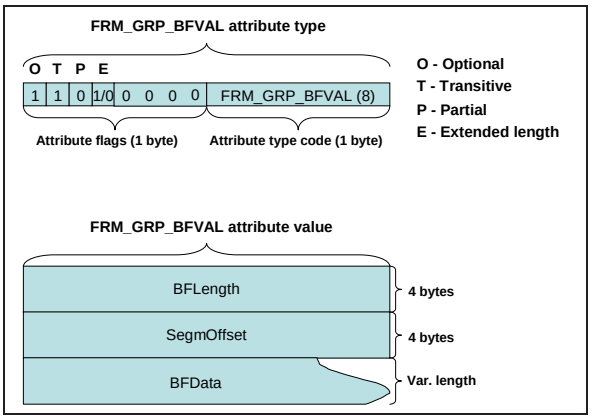
It is also important to note that unlike conventional BGP route advertisements, processing an FRM membership update imposes only a modest processing cost and does not require invoking the BGP decision process. Upon receipt of a GRP_BF update, a router must simply apply the update to the current copy of the originating domain's group bloom filter in the BGP RIB and, in certain rare cases (which we discuss in Section 4.2), update the multicast forwarding tables on line cards.

4.1.1 New BGP Path Attributes

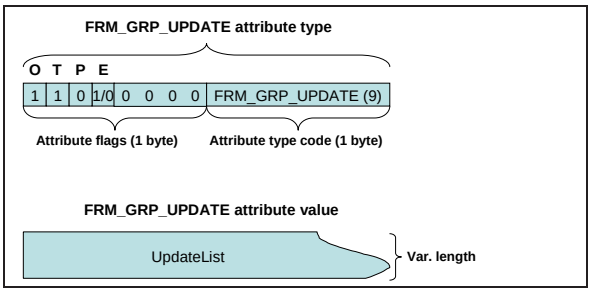
As we discussed above, our scheme requires border routers to piggyback the local membership information on their BGP prefix advertisements and toward this end, we introduce three new BGP path attributes:

FRM_GRP_BFVAL (Figure 2(a)) - An optional transitive path attribute of variable length used for the initial transfer of the group membership state at the start of a peering session between a pair of FRM-speaking routers. The attribute value consists of an 8-byte header followed by a variable-length bitfield ($BFData$) carrying the actual content of the group bloom filter in its entirety or a segment thereof. The header specifies the total length of the GRP_BF encoding in bytes ($BFLength$) and the offset of the supplied segment ($SegmOffset$).

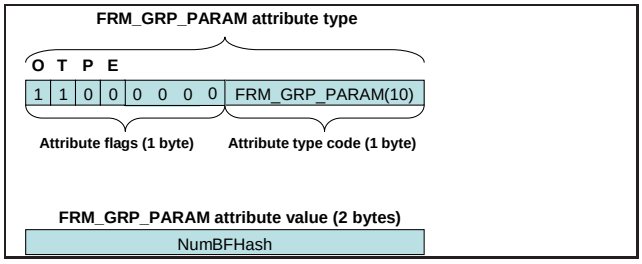
The need for segmentation and reassembly in our current design stems from the 4-KB limit on the maximum size of a BGP UPDATE message imposed by the current BGP protocol specification [15], which complicates the task of transmitting large membership encodings that exceed this size limit. While the obvious workaround would be to increase the maximum message length, this would require modifying the message header format, and hence breaking compatibility with legacy routers. Instead, we chose to work within the constraints of the current BGP specification and provided a mechanism that allows FRM-enabled border routers to transfer large GRP_BF encodings in a piecewise manner via multiple UPDATE messages. Fortunately, BGP is designed to operate



(a) Format of the FRM_GRP_BFVAL path attribute.



(b) Format of the FRM_GRP_UPDATE path attribute.



(c) Format of the FRM_GRP_PARAM path attribute.

Figure 2: **Format of new BGP path attributes for communicating group membership.**

on top a reliable transport protocol, which simplifies the handling of fragmentation and reassembly at the FRM level and eliminates the need for explicit mechanisms for dealing with segment loss, duplication, and reordering.

FRM_GRP_UPDATE (Figure 2(b)) - An optional transitive path attribute of variable length used to communicate incremental updates to the current GRP_BF state, allowing a domain to signal changes to its membership status without retransmitting the group bloom filter in its entirety. The attribute value contains a single variable-length field $UpdateList$ holding an array of bit positions, each of length $log_2(L_{grp})$ bits, whose corresponding values in the group bloom filter need to be flipped. If necessary, the $UpdateList$ field is padded with trailing 0-bits to an integer number of bytes.

FRM_GRP_PARAM (Figure 2(c)) - An optional transitive path attribute of fixed length used to communicate the parameters of the GRP_BF encoding. At present, its value contains only a single field specifying the number of hash function used in the encoding ($NumBFHash$). We assume that all implementation of FRM will share a single globally agreed-upon and uniformly ordered set of hash functions.

4.2 Multicast Packet Forwarding

We now turn to discussing the packet forwarding process. In the FRM scheme, the border router in the packet's source domain plays a special role and handles outgoing packets in a manner that differs from processing at transit routers and we discuss each in turn.

4.2.1 Forwarding at the Origin Border Router

A multicast packet sent by source S to a group G is delivered by the means of the intra-domain multicast protocol to the border router in the source's domain (denoted R_s). This router scans its BGP RIB and performs a lookup in each GRP_BF entry to identify the set of destination prefixes with active members in G. Using this information, R_s then constructs the domain-level multicast dissemination tree $T(G)$ from the union of the individual unicast paths to each member prefix (as given by the value of the AS_PATH attribute).

The source domain's immediate children on the tree constitute the set of next hop domains for R_s. However, as we described in Section 3, the source border router cannot simply forward the packet to each such domain, since the target may not necessarily have all the information it would need to make a correct a forwarding decision and propagate the packet further along the tree. Specifically, a transit router R_t may not know whether it lies on the unicast path used by R_s to reach a particular member prefix.

FRM addresses this issue by requiring R_s to communicate the relevant fragments of $T(G)$ to intermediate routers on the dissemination path. Specifically, for each child domain C, the source border router constructs a space-efficient encoding of the edges of the dissemination subtree rooted at C, inserts this encoding into the outgoing packet in the form of a "shim" layer above the IP header, and forwards the resulting packet to C. In the example of Figure 1(a), R_1 would encode the edge $(2 \rightarrow 4)$ in its packets to R_2 and $\{(3 \rightarrow 6), (3 \rightarrow 7)\}$ in those to R_3.

For scalability reasons, we encode the tree into the shim header using a bloom filter (denoted $TREE\_BF$) but unlike group membership encodings discussed in the previous section, we require the shim header to be of fixed and well-known length so as to be amenable to fast hardware-assisted processing in transit routers. Our reliance on bloom filters makes the encoding scheme susceptible to false positives which, in this particular context, manifest themselves as packet transmissions along inter-domain edges that do not belong to the original delivery tree (e.g., a domain A may forward a copy of the packet to its neighbor B even if $(A \rightarrow B) \notin T(G)$.

Figure 3 illustrates the format of the FRM shim header. It holds a bloom filter encoding of the subtree, preceded by a 32-bit control structure that carries additional information about the encoding. Specifically, the first 4 bits of the header hold the number of hash functions used to generate the encoding, followed by a 4-bit field denoting the length of the encoding in 32-bit words. The next 16 bits carry a checksum, which is computed by considering the entire header (including the control structure) as a sequence of 16-bit words. The last 8 bits of the control structure are currently unused and are reserved for future protocol extensions.

The choice of the shim header length represents another important tradeoff in our design, namely, a tradeoff between the number of unsolicited packet transmission resulting from false positives in the $TREE\_BF$ and the constant per-packet bandwidth penalty due to the shim

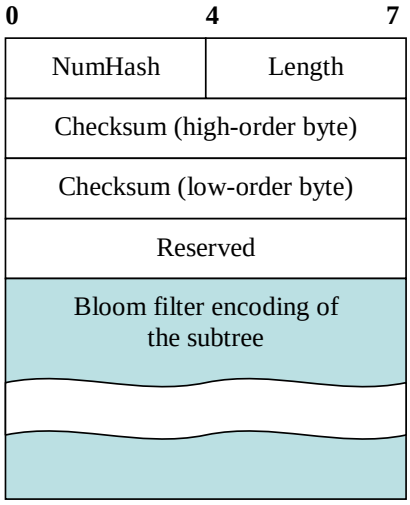


Figure 3: **Format of the FRM shim header.**

header. An insufficiently large header size can result in high false positive rates for large groups, whereas choosing a large enough value to accommodate even the largest groups would result in a needless waste of bandwidth due to the shim header. Our design tries to attain a compromise between these competing concerns: we pick a fixed $TREE\_BF$ size of L_{TREE_BF} bits and a target false positive rate $Rate_{fp}$ and compute E - the number of edges that can be encoded into L_{TREE_BF} bits while maintaining the rate of false positives below the chosen threshold. We have:

$$Rate_{fp} = 2^{-\ln(2)\frac{L_{TREE\_BF}}{E}}$$

or, equivalently:

$$E = L_{TREE\_BF} \cdot \frac{-\ln(2)^2}{\ln(Rate_{fp})}.$$

The optimal number of bloom filter hash functions H that yields the above false positive rate is given by:

$$H = \ln(2) \cdot \frac{L_{TREE\_BF}}{E}.$$

As an example, if we assume a 1% target false positive rate, 83 AS-level edges can be encoded into a single 100-byte shim header and achieving the optimal encoding efficiency would require 7 hash functions.

Having determined the number of edges that can be transferred in a single shim header, we use a standard bin-packing algorithm to decompose the subtree rooted at a child C of R_s into groups of smaller subtrees, ensuring that the number of edges in each group does not exceed E. We then encode each such group into a single shim header and forward it toward C along with a separate copy of the packet. Note that as a result, certain links on the dissemination

tree may see multiple redundant copies of a single multicast transmission and this unfortunate consequence stems from our decision to maintain a fixed-size shim header.

Let us now briefly consider the computational cost of the centralized tree construction procedure invoked at the source border router. This cost grows linearly with the number of prefixes in the router's RIB and the average AS path length. Although clearly expensive, the crucial factor that renders our approach feasible is that results of the initial computation can be cached and reused to process subsequent packets addressed to the same group and hence, the large cost is incurred only on the first packet sent to each group. We also note that the initial operation involving the scan of the BGP RIB is highly amenable to parallelization and can be handled efficiently on appropriately provisioned hardware.

The cached forwarding entry for a given destination group G consists of a k-element array

$$\{(ASNum_1, TREE\_BF_1), (ASNum_2, TREE\_BF_2), ..., (ASNum_k, TREE\_BF_k)\},$$

where k is the number of top-level children in $T(G)$ and each element holds the forwarding information for the corresponding child, namely: its AS number and a bloom filter encoding of the respective subtree. The cache of forwarding entries is indexed by the destination group address and any of the well-known techniques for efficient exact-match lookups can be employed to retrieve the forwarding entry for a a given destination address. For example, CAMs and direct memory data structures offer constant-time exact-match lookups, while more compact data structures achieve lookups in logarithmic time [16, 17].

We note that our scheme requires R_s to maintain per-group forwarding state that grows linearly with the size of a single forwarding entry (which, in turn, depends on the size of the tree $T(G)$), as well as the number of groups with active local sources. We consider this scaling reasonable because we expect that in any realistic scenario, only a small fraction of groups will have active sources in the local domain and the intra-domain multicast protocol is likely to exhibit similar scaling properties. We evaluate the memory requirements for the forwarding cache at R_s in detail in Section 5.1.2 and our results suggest that the cached state could be mostly accommodated in line card memory. Consequently, we expect that once the delivery tree has been computed, R_s will process its outgoing packets entirely via the fast-path forwarding logic. On the other hand, if line card memory buffers cannot accommodate the entire cache, one might consider caching only the forwarding entries for groups with high volume of traffic and leave the forwarding of low data volume groups to the route processor.

4.2.2 Forwarding at Intermediate Routers

As we saw in the previous section, the packet forwarding operation at the source border router incurs some nontrivial storage and computational costs. The payoff for complexity of forwarding at the source is highly scalable, simple, and efficient forwarding at intermediate routers. To forward a multicast packet, a border router R_t in a transit domain T simply inspects the $TREE\_BF$ encoding supplied in the packet's shim header and checks which of its AS neighbor edges are on the encoded subtree. Specifically, for each neighbor domain N, R_t performs

a bloom filter lookup in $TREE\_BF$ and forwards a copy of the packet toward the next hop for domain N if the edge $(T \rightarrow N)$ is present in the encoding.

To ensure efficient processing at R_t, we store neighbor edges in their encoded representation. That is, each neighbor edge is encoded into (and stored as) a separate bloom filter and standard filter matching techniques can be used to implement the forwarding lookup operation. Specifically, for each neighbor edge $(T \rightarrow N)$, R_t must simply check whether the corresponding bits are set in the packet's $TREE\_BF$. There are a variety of mechanisms for implementing filter matching and one simple and efficient option would be to use TCAM with the bloom filter for each neighbor edge stored in one TCAM row and all zero bits set to the "don't care" value [18]. This method would allow all neighbor edges to be matched in parallel with a single TCAM access. Alternatively, edge encodings can be stored in RAM, which would allow logarithmic-time matching.

Finally, we note that the content of the shim header requires no updating at R_t and remains unmodified along the entire path between the source domain and the set of receivers.

4.3 Summary of Design Tradeoffs

Having presented our solutions to the two core aspects of the multicast routing problem, namely, group membership discovery and packet forwarding, we are now ready to summarize the principal tradeoffs in our design. Our primary goal with FRM was to provide a "leaner" multicast solution that seeks to minimize the amount of distributed protocol mechanism at the expense of optimal efficiency. As we discuss below, this offers both advantages and challenges.

4.3.1 Advantages

Sparseness in protocol mechanism. In terms of protocol complexity, the basic FRM design requires:

1. Extending the inter-domain unicast routing protocol (BGP) to carry group membership information as part of route announcements.
2. Providing a mechanism that allows a transit domain to filter multicast traffic (on a per-group and per-link basis) upon request from a downstream customer domain.

Scalable forwarding at transit routers. As we saw in Section 4.2.2, our source-encoded forwarding scheme enables simple and efficient packet processing at intermediate routers. R_t's "forwarding" state is essentially a list of its neighbor AS edge encodings and hence, the number of forwarding entries depends only on the domain's AS degree. Measurement studies of the Internet topology report per-domain AS degrees that range from 1 to under 10,000 and follow a power-law distribution [19]. We can thus expect the number of forwarding entries at most transit routers to be low, possibly several orders of magnitude lower than the size of the multicast group address space, and thus easily accommodated on line cards. Crucially, the amount of transit forwarding state does *not* depend on the number of active groups and the number of sources in a group. Furthermore, since the forwarding state at R_t depends only on

its (largely static) set of AS neighbors, our design does not require a distributed protocol to construct and maintain this state.

The simplicity of transit forwarding and the efficient scaling of forwarding state at R_t are the key distinguishing features of our design that differentiate FRM from many of the existing approaches. To the best of our knowledge, FRM is the only multicast routing scheme that offers shortest-path delivery without requiring intermediate path elements to maintain per-source state. However, this attractive property comes at the cost of some additional bandwidth, memory, and computational overhead in the source domain.

Centralized route computation. In FRM, the dissemination tree is constructed in its entirety by the border router in the source domain using the knowledge of existing unicast routes. This not only eliminates the need for a separate multicast route discovery mechanism, but also protects us from the effects of certain routing anomalies, such as those reported in [20, 21].

General service model. In contrast to the Multicast Channels framework, FRM supports the general multi-source service model of IP Multicast and achieves efficient packet delivery with source-rooted trees.

Ease of configuration and management. Unlike shared-tree schemes such as PIM-SM and CBT, FRM avoids the contentious issues concerning the selection and placement of RPs. Furthermore, since our design requires only a few incremental extensions to BGP, it does not impose the burden of configuring a new inter-domain protocol and instead offers management within the familiar BGP framework.

Accountability and ISP control. Lack of accountability has been frequently cited as one of the primary impediments to the adoption of IP multicast in the wide area. In FRM, the border router in the source domain has full knowledge of (and control over) the set of destinations included in the dissemination tree. This allows ISPs to infer the degree of traffic "amplification" due to a multicast transmission originated by a customer and, in turn, provides a basis for a meaningful charging model. In addition, since group membership is explicitly advertised through BGP, an ISP also has complete control over its customers' group subscription status: blocking access to an undesired group is simply a matter of excluding it from the GRP_BF encoding.

Finally, we note that unlike the IP source routing paradigm, our source-encoded forwarding scheme selects paths compliant with the local policy choices of intermediate ISPs.

4.3.2 Challenges

State requirements. FRM incurs the overhead of advertising and maintaining group membership state. While true for all multicast protocols, our design disseminates membership information more widely than most traditional schemes and hence incurs greater overhead. Table 1 summarizes the state requirements and processing costs in FRM and we note that while our

State	*Scaling*	*Location*	*Lookup*
Membership state at R_s	$O(P \cdot G)$	Route processor	Linear scan
Cached forwarding entries at R_s	$O(G_S \cdot \|T(G_S)\|)$	Line card	Exact match
Neighbor link encodings at R_t	AS degree	Line card	Filter match

Table 1: **Summary of the state requirements and processing overhead in FRM. P denotes the total number of destination prefixes in the BGP RIB and G is the average number of groups per prefix. G_S denotes the number of groups with active senders in domain S and $|T(G_S)|$ is the average number of edges in dissemination trees for groups with senders in S.**

design tilts the burden of forwarding state and complexity onto senders' access domains, this is not an entirely displeasing arrangement, since the bandwidth-conserving benefit of multicasting is greatest at the sender.

Suboptimal bandwidth utilization. FRM's reliance on what is effectively a form of multicast source routing incurs additional bandwidth costs:

1. The FRM shim header incurs a fixed per-packet transmission overhead.
2. False positives in the $TREE\_BF$ encoding may on occasion trigger unnecessary packet transmissions to non-participants.
3. The use of a constant-size encoding necessitates duplicate packets transmissions (on a subset of links) for groups too large to be encoded into a single $TREE\_BF$.

Unconventional packet forwarding techniques. Traditional packet forwarding mechanisms require a longest prefix lookup on the destination address to identify the next hop along which to send the packet. By contrast, obtaining the set of next hops in FRM requires a full scan of the BGP table at the origin border router and a series of bloom filter lookup at intermediate nodes. Our design faces the challenge of achieving this in a manner that is both scalable and amenable to high-speed processing in hardware. We note, however, that the (relatively high) cost of tree computation is incurred only by the source border router and, even there, only once for each group with active sources in its domain.

In the following section, we address these concerns by presenting a detailed evaluation of our design.

5 FRM: Evaluation

In this section, we estimate the storage and bandwidth overhead incurred by FRM's group membership and packet forwarding mechanisms using trace-driven simulation. Below, we present only key results intended to demonstrate the practical feasibility of our scheme under likely usage scenarios, while a more detailed exploration of the parameter space can be found in an extended technical report [22].

The following simulation setup is used throughout this section: We associate a single multicast user with each routable unicast address and a domain represented by a prefix $p = x.x.x.x/L$ can thus be assumed to have $U(p) = 2^{32-L}$ users. Each user, in turn, selects and joins k distinct multicast groups from the address space of size A according to some group popularity distribution. Unless otherwise noted, we model group popularity via a zipfian distribution [23] and pessimistically assume no topological locality in group membership. For inter-AS connectivity data, we use Subramanian *et al.*'s snapshots of BGP routing tables from Oct'04 and their AS-level topologies annotated with inferred inter-AS peering relationships [24].

We begin by quantifying the router storage requirements due to the GRP_BF state and the cached forwarding entries and then proceed to an examination of bandwidth overhead.

5.1 Router Storage Overhead

5.1.1 Group Membership State Requirements

As we explained in the preceding section, our scheme requires border routers to maintain a group membership bloom filter for each destination prefix in the BGP table. For each prefix p of length $L(p)$, we estimate the expected size of its membership bloom filter GRP_BF_p. This is done by first computing the expected number of distinct groups $G(p)$ advertised by p given that $U(p)$ users each select k groups from A according to the chosen popularity distribution. Using the equations from Section 4.1, when then determine the minimum GRP_BF_p length needed to encode $G(p)$ items for a target false positive rate of $f/(A - G(p))$ (recall that f is the permitted number of filters per prefix). Finally, we compute the aggregate storage cost due to GRP_BF state at a border router by summing over all prefix entries in the BGP table.

Figure 4 illustrates the total group membership storage requirements at a BGP router as a function of the address space size A for $f = 10$ and $k = 1, 10$, and 100 groups per user. We see from the figure that GRP_BF state for 1 million simultaneously active groups and 10 groups per user requires approximately 3GB of router memory - an amount found today even on users' endhost machines. Overall, while the memory overhead due to group membership state is nontrivial in our scheme, it is certainly manageable given today's storage technology and costs. Furthermore, we expect that the current trend in memory costs should allow FRM to accommodate the relatively slower growth in BGP table size and incur only a modest increase to overall router costs.

5.1.2 Forwarding State Requirements

Forwarding State at a Transit Router. Recall from Section 4.2.2 that the forwarding state at a transit router R_t consists of bloom filter encodings of its AS neighbor edges and the number of such encodings is given by the AS degree of R_t's domain. Given the power-law AS degree distribution in the Internet topology [19], we can expect the number of forwarding entries to be remarkably small for the vast majority of domains under present conditions. More specifically, it has been observed that 90% of all domains in the Internet currently have fewer

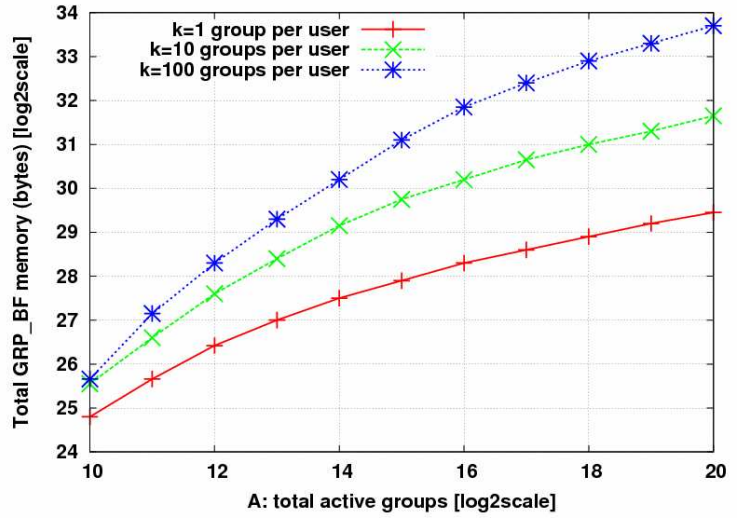


Figure 4: **Total group membership storage requirements at a border router.**

than 10 AS neighbor edges and 99% have fewer than 100. The domain with the highest AS degree has no more than 2,400 edges.

We can compute the corresponding memory requirements at R_t as the number of forwarding entries times the size of a single bloom filter edge encoding. Assuming the worst-case scenario of 2,400 neighbor edges and 128-byte bloom filters, the total forwarding memory requirements at R_t would not exceed 307KB and this amount can be comfortably accommodated on line cards with current TCAM usage [25].

Forwarding State at the Origin Border Router. The forwarding state at the source border router R_s is made up of the cached shim headers for multicast groups with active sources within the router's local domain. To estimate the memory requirements for the cached forwarding state at R_s, we consider a domain with active senders in A distinct groups and, as before, use a zipfian group popularity distribution to model receivers' behavior. In this experiment, we also enforce a minimum group size of 8 domains to avoid populating the cache with uncharacteristically small trees and 25 of the groups are multicasting to the entire Internet (i.e., have members in every domain). For each resultant group, we construct its dissemination tree, generate the appropriate $TREE\_BF$ encodings, and compute the amount of cache memory consumed by these encodings.

Figure 5 plots the total cache memory requirements for increasing values of A. We see that if we assume on the order of several hundred megabytes of RAM on line cards, our scheme would permit fast-path forwarding for up to several hundred thousand active groups. The initial sub-linear scaling trend is likely due to the fact that cache requirements for highly popular groups dominate the initial cache size, whereas the subsequent linear trend reflects our limit on the minimum group size.

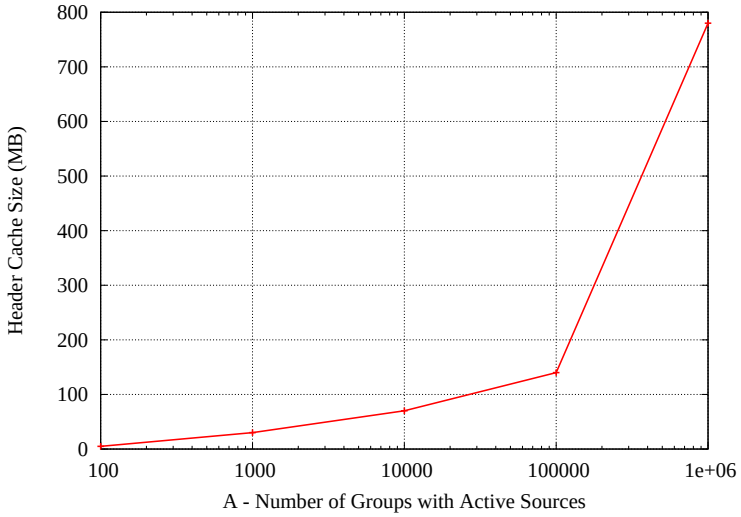


Figure 5: **Shim header cache size as a function of A (the number of groups with active sources in the local domain).**

5.2 Bandwidth Overhead

5.2.1 Overhead of Group Membership Update Traffic

The bandwidth cost of group membership update traffic is determined primarily by the rate at which groups are added to, or removed from, a domain's GRP_BF encoding and we use back-of-the-envelope calculations to demonstrate that the cost of membership update propagation is tractable.

Recall from Section 4.1 that a domain updates its membership state for a given group G only when the number of local members in G rises above zero or drops below one - a relatively rare event, particularly if withdrawals are damped, as suggested in [26].

Consider a (fairly stressful) scenario where every domain sees one such event (appearance of a new group or departure of an existing group) every second. Changes in the membership state are communicated as deltas (the set of GRP_BF bit positions to be modified) and if we assume that 5 hash functions are used to generate the GRP_BF encoding and that bit positions are represented as 24-bit values then conveying membership updates for a single prefix requires approximately 15 bytes of traffic per second[3].

If we assume the presence of 300,000 active prefix entries in the full BGP FIB (current reports indicate approximately 270,000 entries [27]) then the total bandwidth consumed by incoming membership updates would be approximately 4.5MBps - a small fraction of the total capacity at a core border router.

[3]This estimate takes into account only the length of the FRM_GRP_UPDATE attribute and does not include other fields of a BGP UPDATE message or its header.

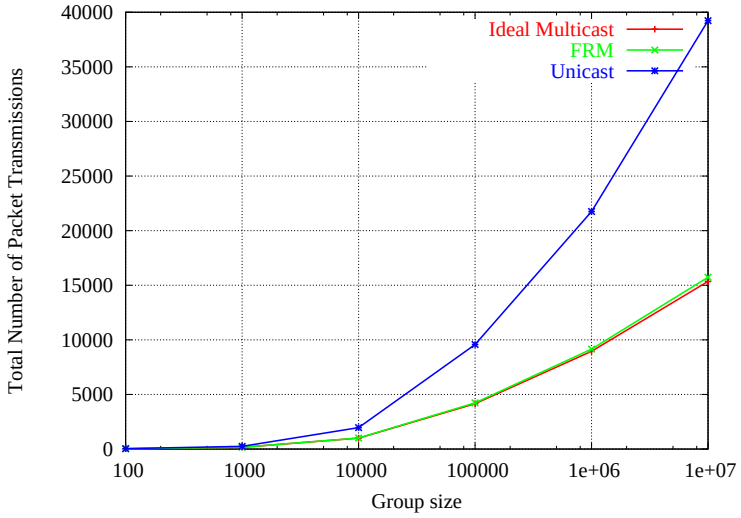


Figure 6: **The total number of packet transmissions (N^{pkt}_{FRM}) for increasing group sizes.**

5.2.2 Packet Forwarding Overhead

The bandwidth penalty due to FRM forwarding is threefold:

1. The per-packet shim header containing the encoded subtree.
2. The redundant transmissions incurred in situations where subtrees are too large to be encoded in a single header.
3. Unnecessary transmissions due to false positives in the $TREE\_BF$ encoding.

We begin by examining the overhead due to redundant transmissions (item 2). Given a dissemination tree T_S rooted at a domain S, we quantify this overhead using the following two metrics:

- $N^{pkt}_{FRM}(e)$ (for $e \in T_S$): The number of transmissions over e required to multicast a single packet using our scheme from the source to all group members.
- N^{pkt}_{FRM}: The total number of packet transmissions required to multicast a single packet using our scheme from the source to all group members:

$$N^{pkt}_{FRM} \doteq \sum_{e \in T_S} N^{pkt}_{FRM}(e).$$

We calibrate FRM's performance against (1) *ideal multicast*, in which precisely one packet is transmitted along each edge of the dissemination tree ($\forall e \in T_S : N^{pkt}_{FRM}(e) = 1$) and (2) simple *per-domain unicast*, in which the source transmits a copy of the packet to each member domain individually.

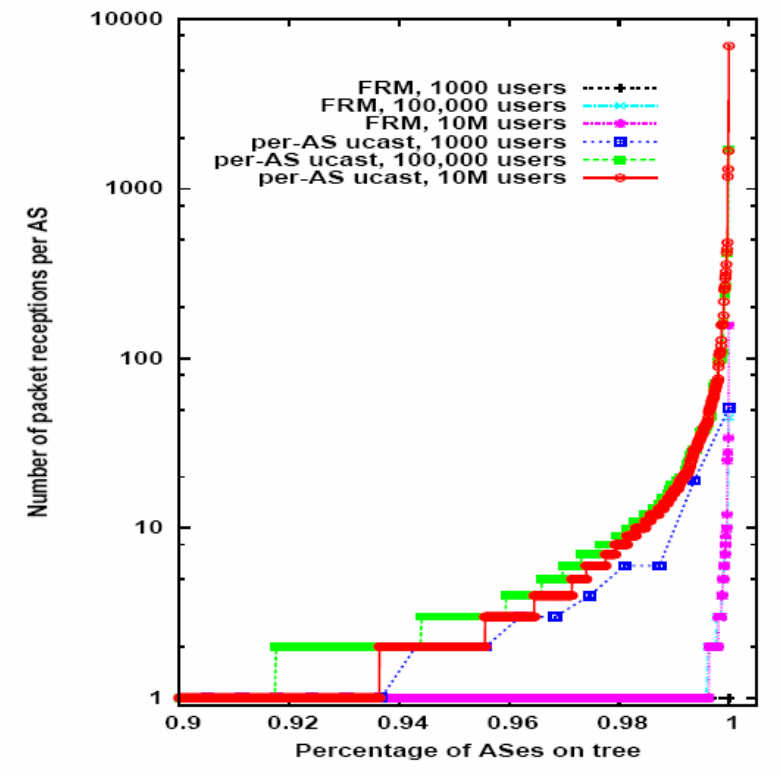


Figure 7: **CDF of $N_{FRM}^{pkt}(e)$ for FRM and unicast.**

Figure 6 show N_{FRM}^{pkt} as a function of group size, providing an objective comparison between FRM, ideal multicast, and unicast[4]. We observe that for all group sizes, the number of transmissions incurred by FRM is quite close to that of ideal multicast (0-2.4% higher). As expected, the difference between FRM and the ideal forwarding scenario grows with the increasing number of group members, since larger trees result in a larger number of shim headers (and hence an increased number of duplicate transmissions). In all cases, however, our scheme achieves improvement over plain unicast, which can require more than twice the bandwidth of FRM for large multicast groups (over 1 million users).

We now turn to examining the per-edge bandwidth overhead. Figure 7 shows the CDF of $N_{FRM}^{pkt}(e)$ for FRM and per-domain unicast for three different group sizes and we see that in all cases, over 90% of edges see exactly one transmission. However, in the case of simple unicast, the worst-case number of transmissions per link is nearly 4 orders of magnitude greater than that of ideal multicast for very large groups. By comparison, FRM forwarding offers significant saving in bandwidth overhead, allowing over 99.5% of edges to see exactly one packet transmission. The worst-case $N_{FRM}^{pkt}(e)$ value for FRM with 10 million members is

[4]In this experiment we assume the use of a fixed 100-byte shim header, which amounts to approximately 10% per-packet overhead on typical data.

157 - a substantial improvement over 6,950 transmissions for plain unicast.

While the worst-case overhead of 157 transmission incurred by our scheme is certainly non-negligible, it should be noted that our tests simulate a fairly stressful scenario: 10 million multicast users with no topological locality results in every domain having a group member. This is equivalent to broadcasting to the entire Internet and one might argue that for such cases, FRM's overhead of 157 packet copies on a single link represents a reasonable penalty. Additional examination revealed that the highest $N_{FRM}^{pkt}(e)$ values are seen by large ISPs, whose connectivity is characterized by a high AS degree and a large number of downstream domains.

We also note that one might consider adopting a number of optimizations to reduce this transmission overhead even further. For instance, if the number of tree edges from a domain A ($(A \rightarrow B), (A \rightarrow C), ...$) constitutes a large fraction of A's entire edge set, the source border router could replace these edges by a single *aggregate edge* $(A \rightarrow *)$ that instructs A to forward the received packet to all neighboring domains (except the one on which the packet has arrived). This optimization could reduce the total number of edges in the $TREE\_BF$ encoding and, correspondingly, the number of duplicate packet transmissions, while sacrificing the precision of the encoding and making it more susceptible to false positives. We refer the reader to [28] for a more extensive discussion of link aggregation, as well as several other optimizations to the basic FRM forwarding scheme presented here.

Evaluation of Bandwidth Efficiency. While the number of redundant packet transmissions provides a meaningful metric for evaluating the bandwidth cost of FRM forwarding, we remind the reader that the total bandwidth overhead incurred by our scheme is a combination of three distinct factors: the per-packet cost of the shim header, the overhead of redundant transmissions, and the penalty of unnecessary transmissions incurred by false positives in the $TREE\_BF$ encoding. Hence, one might argue that the most objective performance comparison can be achieved by evaluating our design using a metric that takes into account all of these sources of overhead. Toward this end, we will examine the FRM forwarding scheme from the standpoint of *bandwidth efficiency*. Informally, we are interested in the ratio between the number of *bytes* transmitted using the FRM scheme and the minimal amount that could be achieved by deploying an "ideal" multicast protocol. We will examine this ratio for individual edges on the dissemination tree, for the tree as a whole, and for the entire topology.

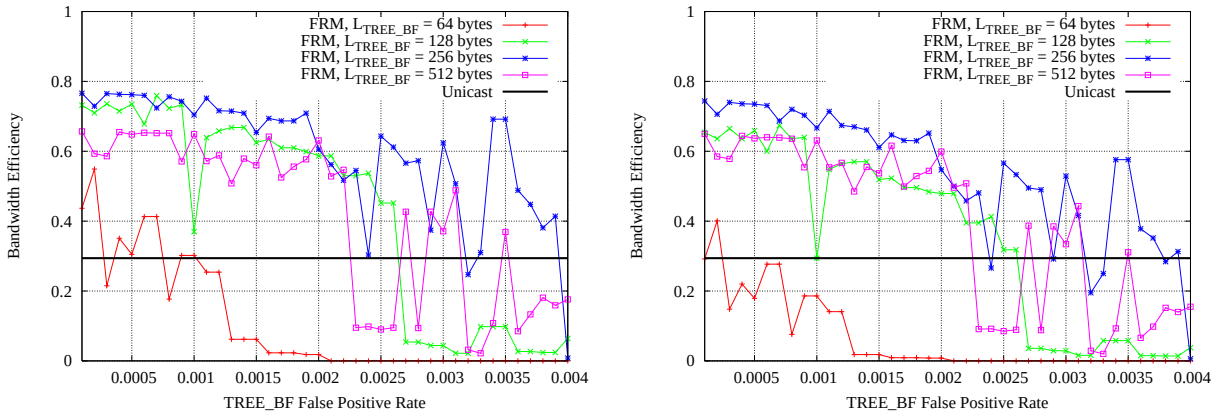
Consider a scenario in which a domain S multicasts a single data packet of length L_{PKT} to a set of receivers in a topology T along a dissemination tree $T_S \subset T$. For each edge $e \in T_S$, we define the bandwidth efficiency achieved by FRM as

$$E_{FRM}(e) \doteq \frac{L_{PKT}}{N_{FRM}^{bytes}(e)}.$$

$N_{FRM}^{bytes}(e)$ denotes the total number of bytes transmitted along e using the FRM scheme:

$$N_{FRM}^{bytes}(e) \doteq N_{FRM}^{pkt}(e) \cdot (L_{TREE\_BF} + L_{PKT}),$$

where $N_{FRM}^{pkt}(e)$ is the number of packet transmissions along e and L_{TREE_BF} is the length of the shim header in bytes. Note that $E_{FRM}(e) = 1$ corresponds to the "ideal" forwarding



(a) E_{FRM}^{tree} - efficiency along the dissemination tree.

(b) E_{FRM}^{top} - overall efficiency in the topology.

Figure 8: **FRM bandwidth efficiency as a function of the target false positive rate for different choices of L_{TREE_BF}.**

scenario, which is unattainable in our scheme even with the optimal number of transmissions due to the overhead of the shim header.

We also consider the efficiency of bandwidth usage for the dissemination tree as a whole:

$$E_{FRM}^{tree} \doteq \frac{|T_S| \cdot L_{PKT}}{\sum_{e \in T_S} N_{FRM}^{bytes}(e)},$$

where $|T_S|$ denotes the number of edges in the tree.

The bandwidth efficiency of unicast forwarding to the same set of receivers provides a meaningful basis for comparison:

$$E_U^{tree} \doteq \frac{|T_S| \cdot L_{PKT}}{\sum_{e \in T_S} \left(N_U^{pkt}(e) \cdot L_{PKT} \right)} = \frac{|T_S|}{\sum_{e \in T_S} N_U^{pkt}(e)}.$$

The penalty due to false positives in the subtree encoding can be quantified in terms of the number of bytes transmitted over non-participant edges: $N_{FRM}^{bytes}(e)$ for $e \in T \backslash T_S$.

Finally, we define the topology-wide efficiency of FRM forwarding as the ratio of the minimum bandwidth usage achievable by "ideal" multicast to that incurred by our scheme (the latter includes the overhead of transmissions along off-tree edges due to false positives):

$$E_{FRM}^{top} \doteq \frac{|T_S| \cdot L_{PKT}}{\sum_{e \in T} N_{FRM}^{bytes}(e)}.$$

Note that $T_{FRM}^{tree} \approx T_{FRM}^{top}$ represents the desirable case, in which most of packet transmissions occur along the forwarding path defined by the dissemination tree, while the non-participant routers do not experience a large penalty because of false positives in the encoding.

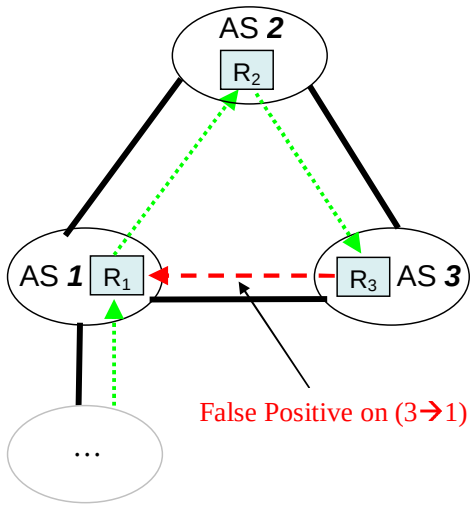


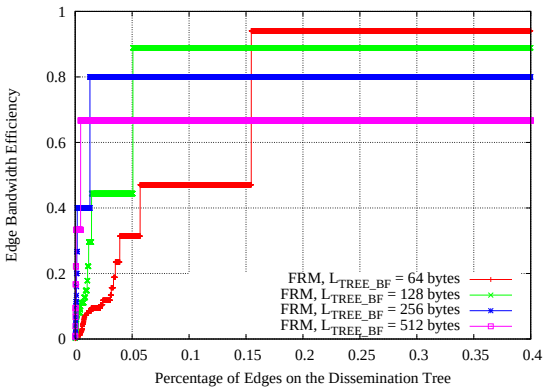
Figure 9: **Emergence of a routing loop as result of a false positive in the $TREE\_BF$ encoding.**

We ran a set of simulations to evaluate the FRM forwarding scheme with respect to the bandwidth efficiency metrics described above. In these simulations, we used an Inet-generated [29] topology with 30,000 AS nodes, 20,000 of which were assigned to have group members. We generated a dissemination tree rooted at a leaf domain with a single upstream provider, computed its subtree encodings, and simulated the FRM forwarding process for a single data packet of length 1KB.

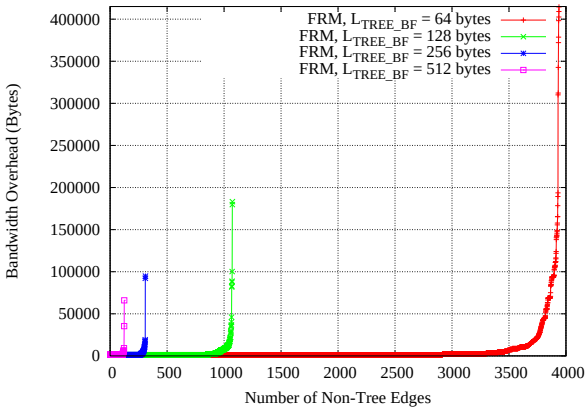
Figure 8(a) plots the resulting value of E_{FRM}^{tree} as a function of the target false positive rate for different values of L_{TREE_BF} and provides a comparison with E_{U}^{tree} (the bandwidth efficiency of unicast forwarding). Figure 8(b) shows the corresponding values of E_{FRM}^{top}, which include the overhead of extraneous transmissions along edges that do not belong to the tree. We see that for both metrics, FRM offers a substantial improvement over plain unicast and comes fairly close to achieving optimal efficiency with the right choice of the encoding parameters. Using a 256-byte shim header and the target false positive rate of 0.01%, our scheme achieves $E_{FRM}^{tree} = 0.766$ and $E_{FRM}^{top} = 0.744$. By comparison, unicast achieves only 0.294.

Our results also suggest that the overall efficiency of FRM forwarding is affected to a significant extent by the choice of the shim header length (L_{TREE_BF}). A small shim header decreases the per-packet overhead, but at the same time reduces the maximum size of a subtree that can be transmitted in a single packet, thus increasing the number of duplicate transmissions and vice versa. In our simulations, a moderately large shim header of length 256 bytes yielded the best bandwidth efficiency results.

As expected, raising the false positive threshold $Rate_{fp}$ increases the difference between E_{FRM}^{tree} and E_{FRM}^{top} by permitting a larger number of unnecessary transmissions along the non-participant edges. Increasing $Rate_{fp}$ also reduces the number of duplicate transmissions and can thus be expected to increase the tree bandwidth efficiency (E_{FRM}^{tree}), but our simulation results suggest that this intuition may not be true. To the contrary, the net effect of admitting



(a) The cumulative distribution of $E_{FRM}(e)$ for $e \in T_S$ and different choices of L_{TREE_BF}. (Note the x-axis scale).



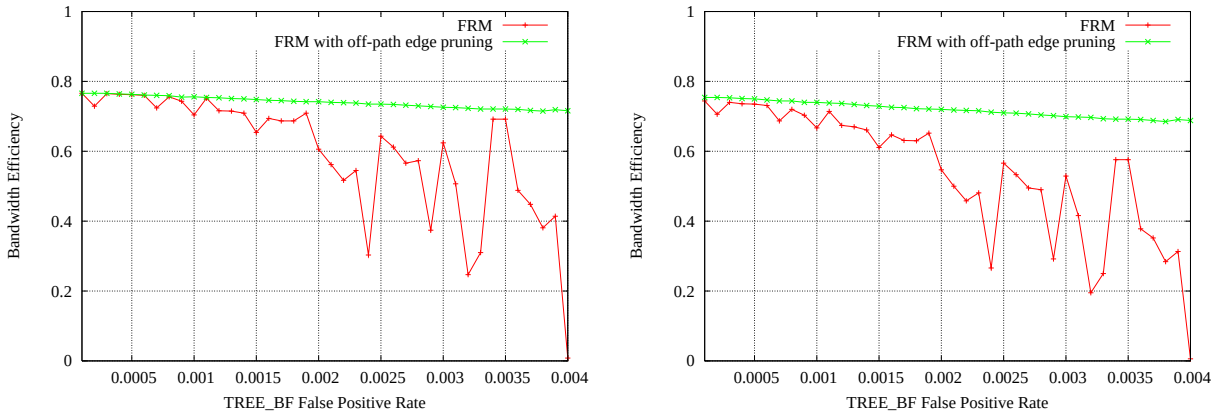
(b) The distribution of $N^{bytes}_{FRM}(e)$ for $e \in T/T_S$ and different choices of L_{TREE_BF}. A point positioned at (x, y) on this plot indicates: x non-tree edges see the overhead of y bytes or less.

Figure 10: **The distribution of bandwidth overhead across (a) edges in T_S and (b) edges in T/T_S.**

a larger number of false positives is a reduction in bandwidth efficiency despite the savings in the number of duplicate transmissions. Overall, a conservatively-chosen $Rate_{fp}$ value (below 0.1%) appears to be a generally safe choice. Further investigation revealed that the sharp decline in bandwidth efficiency beyond $Rate_{fp} = 0.2\%$ can be attributed to emergence of routing loops - a scenario illustrated in Figure 9. In this Figure, a multicast packet is being forwarded along the path ($AS1 \rightarrow AS2 \rightarrow AS3$). When the packet reaches $AS3$, a bloom filter lookup on ($AS3 \rightarrow AS1$) in the shim header results in a false positive and causes R_3 to forward the packet back to R_1, thus creating a loop ($AS1 \rightarrow AS2 \rightarrow AS3 \rightarrow AS1 \rightarrow ...$). In our simulations, routers discard a packet copy once its TTL, initially set to 32, reaches 0.

Figure 10(a) plots the CDF of $E_{FRM}(e)$ for $e \in T_S$ with the false positive rate fixed at 0.01% and we see that our scheme achieves efficient utilization of bandwidth for the vast majority of edges on the tree. If we examine the CDF for L_{TREE_BF} = 256 bytes, which, according to Figure 8(b) achieves the highest level of efficiency for the topology as a whole, over 98.7% of edges in T_S are utilized with the efficiency of 0.8 (which corresponds to exactly one packet transmission) and only 0.2% (36 edges in absolute terms) see the efficiency of 0.4 or less. The worst-cast efficiency value experienced by a single edge in T_S is 0.004, which corresponds to 194 packet transmissions.

Figure 10(b) evaluates the bandwidth penalty imposed upon the off-tree edges ($T \backslash T_S$) as result of false positives in the $TREE\_BF$ encoding. We plot this quantity for different choices of L_{TREE_BF} and a fixed false positive threshold of 0.01%. As expected, a small shim header increases the number of distinct subtree encodings and, accordingly, the number of non-tree edges that receive a packet transmission as result of a false positive hit. For L_{TREE_BF} = 256, only 310 edges see unsolicited packet transmissions, which might well represent a reasonable



(a) E^{tree}_{FRM} - efficiency along the dissemination tree.

(b) E^{top}_{FRM} - overall efficiency in the topology.

Figure 11: **FRM bandwidth efficiency as a function of the target false positive rate with and without edge pruning ($L_{TREE\_BF} = 256$).**

penalty considering the scale of our simulated scenario. The worst-case overhead of 94,720 bytes (the equivalent of 74 packet transmissions) is experienced by only one of the edges and over 86% of them see exactly one transmission, which amounts to 1,280 bytes of overhead.

Optimizing Bandwidth Efficiency. While the results presented in the preceding section suggest that the bandwidth overhead incurred by the FRM forwarding scheme remains within reasonable bounds even under very stressful scenarios, we now consider an additional optimization that allows us to reduce the bandwidth penalty even further.

The high-level question we would like to answer is: to what extent does additional knowledge about the underlying topology and the unicast routing paths help in improving the bandwidth efficiency of our scheme? Clearly, replacing the underlying unicast routing infrastructure with a policy-compliant link state protocol would altogether obviate the need for a shim header, eliminate the problem of false positives and, in fact, allow us to achieve "ideal" multicast with respect to bandwidth efficiency. Since this approach would face a daunting barrier to deployment in the current Internet, we instead consider the following optimization that operates within the constraints of the current BGP infrastructure:

> Suppose there exists a mechanism that allows any transit router R, whose AS neighbors in the topology are given by $Neighb(R)$, to determine, for any source domain S, the subset of its neighbors $Neighb^*(S,R) \subset Neighb(R)$ such that $\forall N \in Neighb^*(S,R)$: R's domain is *not* on the unicast routing path currently used by S to reach N.

Knowledge of this additional information allows us to optimize the transit forwarding functionality in FRM: transit routers can simply avoid propagating packets along edges that are known *not* to be on the tree. Our simulation results suggest that this optimization, which we

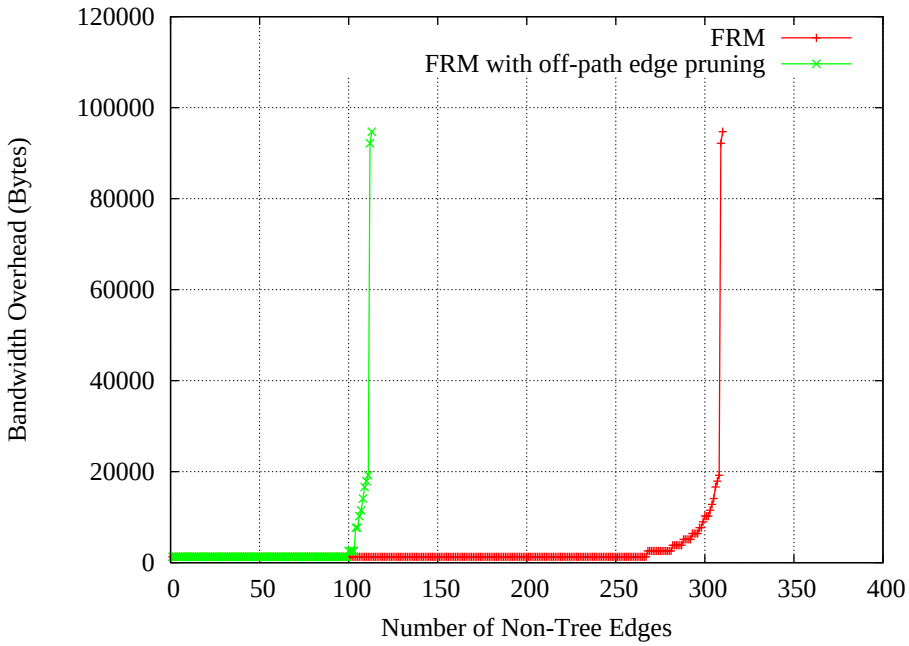


Figure 12: **The distribution of bandwidth overhead across edges in T/T_S with and without edge pruning (**$Rate_{fp} = 0.01\%$**,** $L_{TREE\_BF} = 256$**).**

call *off-path edge pruning*, produces substantial savings in the forwarding bandwidth cost. More specifically, this optimization:

1. Produces a measurable improvement in bandwidth efficiency with respect to both E_{FRM}^{tree} and E_{FRM}^{top}.
2. Makes the occurrence of routing loops less likely.
3. Reduces the number of extraneous transmissions due to false positives in the encoding and hence lessens the bandwidth penalty imposed upon the routers off the dissemination path.

Figure 11 quantifies the improvement in bandwidth efficiency and plots E_{FRM}^{tree} and E_{FRM}^{top} with L_{TREE_BF} = 256 as a function of the false positive rate with and without edge pruning. We see that although edge pruning does not improve the maximum achievable E_{FRM}^{tree} value (both schemes achieve the best-case value of 0.766 with $Rate_{fp}$ set to 0.01%), it does offer improvement over the basic scheme under higher false positive threshold values. For instance, $Rate_{fp}$ = 0.32% yields E_{FRM}^{tree} = 0.247 (below that of unicast) and edge pruning improves this value to 0.723. Investigation revealed that improvements come primarily from the reduced number of routing anomalies, such as the one shown in Figure 9, that would have otherwise occurred due to a large number of false positives.

More importantly, however, edge pruning improves the topology-wide bandwidth efficiency (E_{FRM}^{top}), as can be seen from Figure 11(b), by reducing the number of unsolicited

packet transmissions along edges that do not belong to the tree. In the best-case scenario ($Rate_{fp} = 0.01\%$), edge pruning improves E^{top}_{FRM} from 0.744 to 0.754 and, as expected, the magnitude of improvement grows with the false positive threshold.

To further understand the effects of this optimization on reducing the overhead of erroneous transmissions, we examined the distribution of bandwidth overhead along the edges in T/T_S for $Rate_{fp} = 0.01\%$ and L_{TREE_BF} = 256. The results are shown in Figure 12. Without edge pruning, 310 non-tree edges see unsolicited traffic and 43 of these edges see two or more packet copies. The pruning optimization reduces the number of edges penalized by our scheme to 113 and only 14 of them see more than one packet.

6 FRM: Implementation

We have implemented a software-based prototype of an FRM router that runs under the Linux operating system and makes use of the eXtensible Open Router Platform (XORP) [30]. Our implementation supports all main elements of the design, as described in the preceding sections and has been successfully deployed and benchmarked in a controlled test environment.

Figure 13 illustrates the overall structure of the prototype. At a high level, our implementation consists of a Linux kernel module that performs the functions of the forwarding plane and a user-level component that maintains group membership state and handles the propagation of membership update reports to neighboring domains. The user-level module runs in the execution context of the XORP BGP daemon ($xorp\_bgp$) and communicates with the kernel-side FRM module via the Linux netlink mechanism [31] - a standard feature of the Linux OS that allows the kernel to request service from a user-level process and vice versa through a generic socket-like interface.

In the kernel-side module, the FRMHdrCache data structure caches forwarding state for groups with active sources in the router's local domain, while the BGPPeerTable holds the encodings of edges to neighboring domains used to forward transit packets. Group membership bloom filters are maintained by the $xorp\_bgp$ daemon as a component of its Routing Information Base (RIB).

At present, our FRM prototype lacks support for interfacing with intra-domain multicast routing protocols. As an interim mechanism, we implement intra-domain forwarding using the LocalGrpMembers data structure in the kernel module that stores the IP addresses of local group members for each active group. A more scalable implementation might, for instance, store the IP address of the group's local rendezvous point (assuming the use of PIM-SM within a domain).

6.1 Packet Processing

When a multicast packet arrives on one of the router's interfaces, Linux delivers it to the FRM kernel-level module, which in turn invokes its packet forwarding code path (Figure 14).

Since in the core FRM forwarding scheme, packet processing actions in the source domain are somewhat different from those associated with forwarding transit packets, we first examine the packet's source IP address to determine its origin.

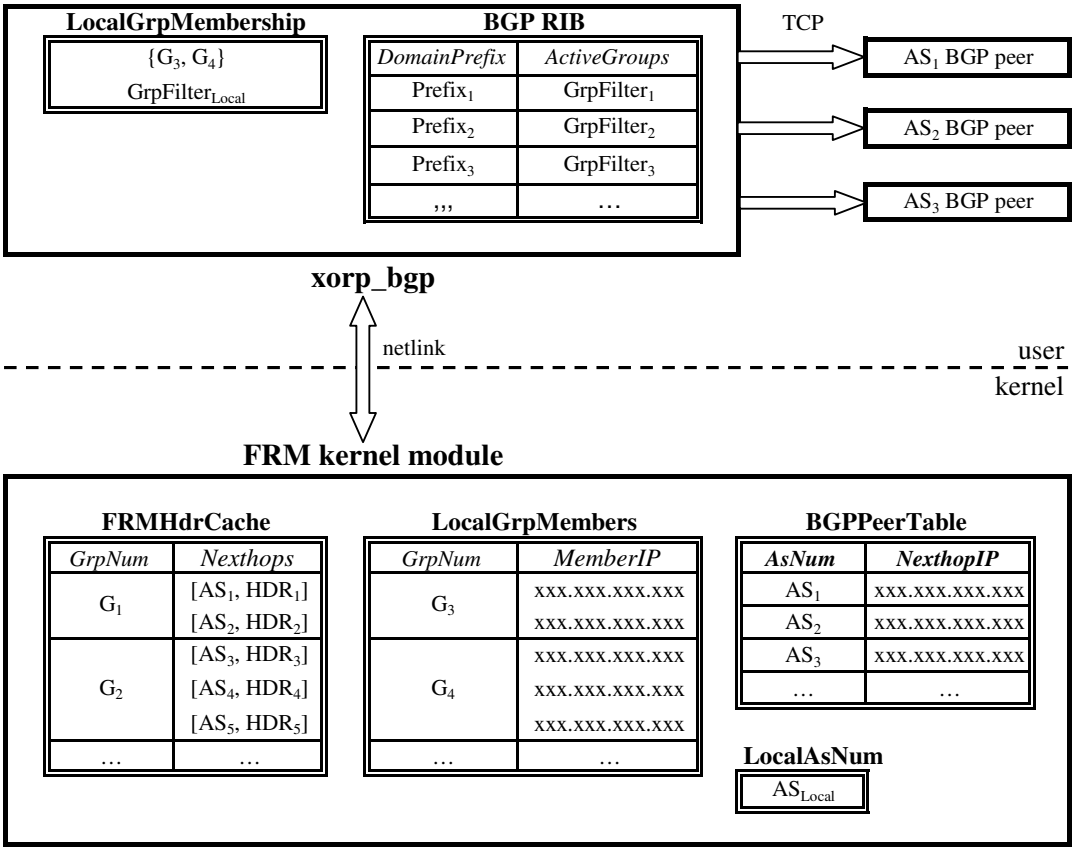


Figure 13: **Software architecture of the FRM prototype.**

6.1.1 Source Domain Processing

If the source address indicates that the packet originated in the router's local domain, we invoke the *source forwarding code path* (Figure 15). Recall from Section 4.2.1 that forwarding in the origin domain involves constructing the multicast dissemination tree for the destination group and propagating copies of the packet to the source's immediate children in the tree, augmenting each copy with a shim header that contains a bloom filter encoding of the subtree rooted at that node.

We first check whether the requisite forwarding state for the packet's destination group is present in the FRMHdrCache data structure.

Source domain: cache miss In the event of a cache miss, the kernel makes an upcall to $xorp\_bgp$ to request the dissemination tree for the packet's destination group. Upon receipt of the kernel's request, the BGP daemon scans its RIB, identifies those destination prefixes whose GRP_BFs indicate membership in the specified group, and computes the tree from the union of the corresponding unicast routes. The daemon splits up the resulting tree into one or more

```
/**** Main entry point for FRM forwarding at a border router  ****/
static void frm_forward(struct packet *pkt) {
  uint16_t chksum_in_packet, chksum_computed;
  struct frmhdr *frmh;
  struct iphdr *iph;
  uint16_t hdrlen;

  iph = (struct iphdr *)pkt->data;

  // Perform several checks and discard invalid packets
  if ((pkt->len < MIN_IPHDR_LEN) ||      // Invalid header format
     (!MULTICAST(iph->daddr))    ||      // Invalid destination group address
     (iph->ttl == 0))                    // TTL has expired
    goto drop;

  iph->ttl--;                            // Decrement the TTL value
  if (iph->ttl == 0)
    goto drop;

  if (addr_in_local_domain(iph->saddr)) {
    // The source address is in the local domain; we are the source border router

    // If this is an IGMP group membership report, update the LocalGrpMembers table
    if (iph->protocol == IPPROTO_IGMP)
      frm_process_igmp(pkt);
    else
      frm_forward_source(pkt);           // Invoke the source forwarding codepath

  } else {
    // The packet originated in a remote domain; we are a transit router
    frmh = (struct frmhdr *)(pkt->data + (iph->ihl * 4));
    if (!IS_VALIDFRMHDR(frmh))           // Verify presence of the shim header
      goto drop;

    // Verify checksum in the FRM header
    chksum_in_packet = FRMHDR_GETCHKSUM(frmh);
    FRMHDR_SETCHKSUM(frmh, 0);
    hdrlen = sizeof(struct frmhdr) + FRMHDR_GETBLOOMLEN(frmh);
    chksum_computed = frm_shimhdr_chksum((char *)frmh, hdrlen);
    if (chksum_in_packet != chksum_computed)
      goto drop;                         // Invalid checksum

    // Restore the checksum in the header (it was reset to zero above)
    FRMHDR_SETCHKSUM(frmh, chksum_in_packet);

    frm_forward_transit(pkt);            // Invoke the transit forwarding codepath

    // Forward the packet to all active members in the local domain
    frm_forward_local(pkt);
  }

done:
  return;

drop:
  log_packet_drop(pkt);
  goto done;
}
```

Figure 14: **Implementation of the main packet forwarding code path (non-essential details are omitted).**

```
/**** Packet forwarding at the source border router  ****/
static void frm_forward_source(struct packet *pkt) {
  uint16_t iphdrlen_bytes, shimhdrlen_bytes;
  uint32_t group_num, payload_bytes;
  struct hdrcache_item *cacheitem;
  struct iphdr *iph, *newiph;
  struct msg_needheader msg;
  struct packet newpkt;

  iph = (struct iphdr *)pkt->data;
  group_num = IPADDR_TO_GRPNUM(iph->daddr);

  // Look up the destination group address in the shim header cache
  if ((cacheitem = hdrcache.lookup(group_num)) == NULL) {
    /* Header not found in the cache. Enqueue the packet and request the
       dissemination tree from the BGP daemon. */
    waittable.addPacket(pkt, group_num);

    // Send the header request signal to the XORP daemon
    msg.group_num = group_num;
    sendmsg_to_bgp(MSG_NEEDHEADER, &msg, sizeof(msg));

  } else {
    // Found a cached header for this group

    // Allocate a new packet buffer that contains a placeholder for the shim header
    iphdrlen_bytes = iph->ihl * 4;
    shimhdrlen_bytes = config.getMyShimHdrLen();
    payload_bytes = pkt->len - iphdrlen_bytes;

    newpkt.len = pkt->len + shimhdrlen_bytes;
    newpkt.data = (char *)malloc(newpkt.len);

    newiph = (struct iphdr *)newpkt.data;
    newpayload = newpkt.data + iphdrlen_bytes + shimhdrlen_bytes;

    // Copy the IP header into the new packet buffer
    memcpy(newiph, iph, iphdrlen_bytes);

    // Copy the payload into the new packet buffer
    memcpy(newpayload, pkt->data + iphdrlen_bytes, payload_bytes);

    /* Increment the 'length' field of the new IP header to reflect the
       addition of the shim header and recompute its checksum. */
    newiph->tot_len = htons(newpkt.len);
    newiph->check = 0;
    newiph->check = compute_ip_csum(newiph);

    // Forward the packet to children on the dissemination tree
    frm_forward_source_cachehit(&newpkt, cacheitem);

    // Deallocate the packet buffer
    free(newpkt.data);
  }
}
```

Figure 15: **Implementation of the source forwarding code path (non-essential details are omitted).**

```
/****
   This routine is invoked at the source border router to initiate the dissemination
   of a multicast packet once a valid header cache item (struct hdrcache_item) has been
   obtained. The cache item contains the set of next hops and pre-computed shim headers
   (one for each child on the tree).
****/
static void frm_forward_source_cachehit(struct packet *pkt, struct hdrcache_item *item) {
  struct neighbour_as *neighb;
  struct frmhdr *frmh;
  struct iphdr *iph;
  char *hdrdata_ptr;
  uint16_t as_num;
  unsigned int i;

  iph = (struct iphdr *)pkt->data;
  frmh = (struct frmhdr *)(pkt->data + (iph->ihl * 4));

  hdrdata_ptr = item->hdrdata;
  for (i = 0; i < item->num_nexthops; i++) {   // For each child on the tree...
    as_num = item->nexthop_asnums[i];

    /* Resolve the child AS number into a neighbor structure that contains the requisite
       forwarding information. */
    neighb = config.getNeighbour(as_num);

    memcpy(frmh, hdrdata_ptr, item->hdrlen);   // Copy the shim header into the packet buffer

    frm_sendpkt_nexthop(pkt, neighb);          // Forward the packet to the child domain

    hdrdata_ptr += item->hdrlen;               // Advance the header buffer pointer
  }
}
```

Figure 16: **Implementation of the source forwarding code path: cache hit (non-essential details are omitted).**

subtrees (one for every direct child of the local AS) and responds to the kernel with a set of structures of the form $(AS_x, SubTree_x)$, where AS_x is the AS number of a direct child node and $SubTree_x$ is a list of edges representing the subtree rooted at AS_x. The kernel parses this response, encodes the supplied edges into bloom filters, and constructs a set of FRM shim headers - one for every child node AS_x.

Once the headers are computed, a copy of the packet, augmented with an appropriate shim header, is made for each child domain and then forwarded toward that child's next hop address. For efficiency reasons, we use an auxiliary data structure (BGPPeerTable) in the kernel to map between the AS number of a BGP peer and its corresponding next-hop IP address and a subsequent lookup in the kernel's main forwarding table resolves the next-hop IP into a pointer to the outgoing network interface. Finally, we insert the destination group address and the set of shim headers for each child AS into FRMHdrCache. This data structure is indexed by group address and uses a basic LRU replacement scheme.

Source domain: cache hit In the event of a cache hit, packet processing is extremely simple (Figure 16). A lookup in FRMHdrCache produces the set of next-hop AS numbers and

the associated shim headers. A copy of the packet is made for each child entry AS_x associated with the destination group; the packet is augmented with the cached shim header and sent to AS_x. Note that the upcall to the XORP BGP daemon is required only in the event of a cache miss.

The use of FRMHdrCache can greatly reduce the per-packet processing overhead at the source border router, since a cache lookup is vastly more efficient than full recomputation of the tree. However, as we explain below, this improvement comes at the cost of having to invalidate a (potentially large) number of cached elements in response to a group membership event in a remote domain.

6.1.2 Transit Domain Processing

If the packet did not originate in the router's local domain, we invoke the *transit forwarding code path* (Figure 17). We decrement the packet's TTL, iterate through the BGPPeerTable and, for each peer AS_x, check for the presence of the edge $(AS_{Local} \rightarrow AS_x)$ in the packet's $TREE\_BF$. If the edge is present, we forward a copy of the packet toward the next-hop address for AS_x. As the last step, we strip off the FRM shim header and forward a copy of the packet to every active local group member listed in the LocalGrpMembers table.

6.1.3 Packet Processing Overhead

We measure the forwarding latency for each of the forwarding code paths described above - *source cache hit*, *source cache miss*, and *transit*. Our measurements were conducted on a 1.8GHz IBM Thinkpad with 1GB RAM running the prototype FRM code under Linux RedHat 9, kernel level 2.4.20-8. For each code path, we measure the time interval between the packet's handoff to the FRM module and the time at which the last copy of the packet is enqueued for transmission over the outgoing interface. The latency results presented below are averaged over 1,000 incoming packets.

Table 2 illustrates the average per-packet forwarding time for the *source cache hit* code path and we test performance for different packet sizes and fanout values (i.e., the number of outgoing copies). For calibration, we also measure the processing latency achieved by the standard Linux kernel implementation of multicast forwarding. As expected, FRM processing time scales linearly with the number of outgoing packet copies, while larger packets take longer to process due to the higher memory copy overhead. Relative to native multicast forwarding, FRM exhibits similar scaling behavior but is always slower in the absolute and further examination revealed that the difference in performance is primarily due to the fact that our implementation incurs one additional buffer copy for every packet sent. In standard multicast, an identical copy of the packet is sent to all outgoing interfaces, while our scheme requires generating a distinct copy (with the appropriate shim header) for every neighbor and hence replicating the original buffer.

To measure the forwarding time for packets that suffer a cache miss, we populate the RIB with an Oct'04 snapshot of the Internet BGP table containing 117,519 prefix entries and initialize a fraction F of these entries to indicate membership in the packet's destination group. In this experiment, we maintain the group membership data in bloom filters of length 2KB and

```
/**** Packet forwarding at a transit router  ****/
static void frm_forward_transit(struct packet *pkt) {
  uint32_t bloomlen_bytes, num_hash;
  struct neighbor_as *neighb;
  struct frm_tree_edge edge;
  struct frmhdr *frmh;
  struct iphdr *iph;
  uint16_t forward;
  char *bloomdata;

  iph = (struct iphdr *)pkt->data;
  frmh = (struct frmhdr *)(pkt->data + (iph->ihl * 4));

  bloomdata = ((char *)frmh) + sizeof(struct frmhdr);
  bloomlen_bytes = FRMHDR_GETBLOOMLEN(frmh);
  num_hash = FRMHDR_GETNUMHASH(frmh);

  if ((bloomlen_bytes * BITS_PER_BYTE) == config.getMyShimBloomLen()) {
    /* The 'fast' path: lookup each neighbor in the shim header using
       pre-computed encodings. */
    while ((neighb = config.getNextNeighbor(neighb)) != NULL) {
      forward = frm_bloom_lookup_from_bits(bloomdata, bloomlen_bytes,
                                           neighb->bloom_bitpos, num_hash);
      if (forward)
        frm_sendpkt_nexthop(pkt, neighb);
    }
  } else {
    // The 'slow' path: perform a full bloom filter lookup
    edge.src_as = config.getMyASNum();
    while ((neighb = config.getNextNeighbor(neighb)) != NULL) {
      edge.dst_as = neighb->as_num;
      forward = frm_bloom_lookup(bloomdata, bloomlen_bytes,
                                 num_hash, &edge);
      if (forward)
        frm_sendpkt_nexthop(pkt, neighb);
    }
  }
}
```

Figure 17: **Implementation of the transit forwarding code path (non-essential details are omitted).**

6 hash functions are used to generate the encoding. Table 3 lists the forwarding latency for a single 512-byte packet at R_s for an increasing number of member prefixes included in the tree. The reported latency includes the cost of scanning the BGP RIB, constructing the dissemination tree, generating the appropriate $TREE\_BF$ encodings (we use 256-byte bloom filters with 6 hash functions), replicating the outgoing packet, and enqueuing the outgoing copies for transmission. We see that in the worst case where every prefix has an active group member, it takes approximately 303.2ms to forward the packet and further investigation revealed that the processing time is dominated by the cost of scanning the RIB. While clearly expensive, we do not view the processing latencies of cache misses as cause for concern for two reasons: First, these measured latency values are entirely dependent on the processor speed and other hardware characteristics of the router which, in our case, is a uniprocessor IBM Thinkpad and in reality, header construction can be parallelized and optimized on SMPs. Second, this latency

Fanout	Linux multicast 1-byte packets	FRM 1-byte packets	FRM 128-byte packets	FRM 1024-byte packets
1	0.4	0.7	0.8	1.2
128	25.4	64.8	76.2	89.5
256	50.7	132.5	154.2	177.5
512	101.2	262.7	308.6	351.4

Table 2: **Forwarding latency (in microseconds) at R_s when the destination group is in FRMHdrCache.**

Member prefixes	0 (F = 0)	459 (F = 1/256)	1836 (F = 1/64)	7345 (F = 1/16)	29380 (F = 1/4)	117519 (F = 1)
Processing time	65.8	68.3	74.5	89.1	124.8	303.2

Table 3: **Forwarding latency (in milliseconds) for a 512-byte packet at R_s when the destination group is not in FRMHdrCache.**

is only incurred on the first packet sent to a group and cache misses can be rendered even more infrequent via pre-computation and an appropriate choice of the cache size.

Finally, Table 4 reports the forwarding latency at R_t for transit packets. We measure the processing time for a single packet 512-byte packet under different tree fanout values and repeat the measurement for different sizes of the BGPPeerTable. The size and format of the $TREE\_BF$ encoding is the same as in the previous experiment. We observe that as with source-domain forwarding, the processing time scales linearly with the number of outgoing packet copies and, as expected, also exhibits linear dependence on the domain's AS degree. Overall, transit forwarding is efficient and only marginally more expensive than a cache hit at the source border router for the same fanout value.

In summary, the design of FRM admits a straightforward and efficient implementation of the cache hit and transit forwarding code paths that achieve efficiency comparable to that of the native multicast forwarding mechanism in the Linux kernel. For cache misses, we believe that a combination of hardware and software optimizations along with a sufficient cache memory allotment can make the performance impact of misses negligible. However, an exploration and evaluation of performance optimizations merits further study, particularly in the context of realistic router hardware configurations.

6.2 Advertising Group Membership Updates

When an endhost joins or leaves a multicast group, an IGMP membership report is generated and delivered to its designated router (DR). In our current implementation, we modify DRs to relay these membership events directly to the source FRM router R_s. Upon receipt of this group membership event, the kernel-side FRM component updates its LocalGrpMembership table. If the event in question triggers a domain-level membership change (i.e., activation of

Fanout	AS degree 1	AS degree 32	AS degree 128	AS degree 256	AS degree 512	AS degree 1024
1	7.6	10.9	17.0	27.7	45.6	81.4
32		38.8	43.8	54.5	73.5	113.4
128			127.1	137.1	159.4	204.5
256				220.7	248.8	308.0
512					402.2	465.2
1024						748.7

Table 4: **Transit forwarding latency (in microseconds) for a 512-byte packet at R_t.**

a previously-inactive group or vice versa), the kernel module issues an upcall to $xorp\_bgp$, notifying it of the membership update. In response, the BGP daemon updates the local domain's membership bloom filter and calculates the delta from its previous value (expressed as the list of modified bit positions). Having done that, the daemon initiates global dissemination of the updated membership state by re-advertising the domain's prefix to its BGP peers and augmenting the advertisement with the FRM_GRP_UPDATE path attribute that contains the membership delta. Multiple successive membership updates may be combined and sent in a single BGP advertisement to reduce the overhead of BGP traffic at the expense of increased join latency.

When a peer $xorp\_bgp$ daemon receives an FRM_GRP_UPDATE, it extracts the list of modified bit positions and applies this update to its GRP_BF state for the corresponding prefix. Since the update may cause some subset of cached tree encodings in FRMHdrCache to become stale, the daemon then issues an invalidation request to the kernel module, which, in turn, purges all affected entries from FRMHdrCache.

It is easy to verify that for a given set of modified bit positions B_m and a given group G whose bloom filter representation is a set of bit positions B_G, the cache item associated with G requires invalidation if and only if $B_G \cap B_m \neq \emptyset$. Since a full linear scan of FRMHdrCache to identify such entries can be quite expensive, our implementation uses an auxiliary kernel-level data structure to efficiently resolve a bit position into a set of pointers to FRMHdrCache entries associated with that bit.

Note that in our current implementation, every incoming FRM_GRP_UPDATE triggers a kernel invalidation request even if none of the cached entries require invalidation. As a potential optimization, the BGP daemon can keep track of the current content of FRMHdrCache and issue an invalidation call only if the cache is known to contain stale entries. This would prevent some unnecessary user-kernel crossings, but would make cache eviction more expensive because additional coordination between the kernel module and the daemon would be required to keep the two structures in sync. We believe that both approaches are valid and come with their trade-offs, which we plan to explore more fully through subsequent deployment experience.

In our evaluations, the processing overhead for a single FRM_GRP_UPDATE message with one group activation event that modifies 6 bits in the membership bloom filter and invalidates a single FRMHdrCache entry (with 1,024 entries present in the cache) incurs 18.67

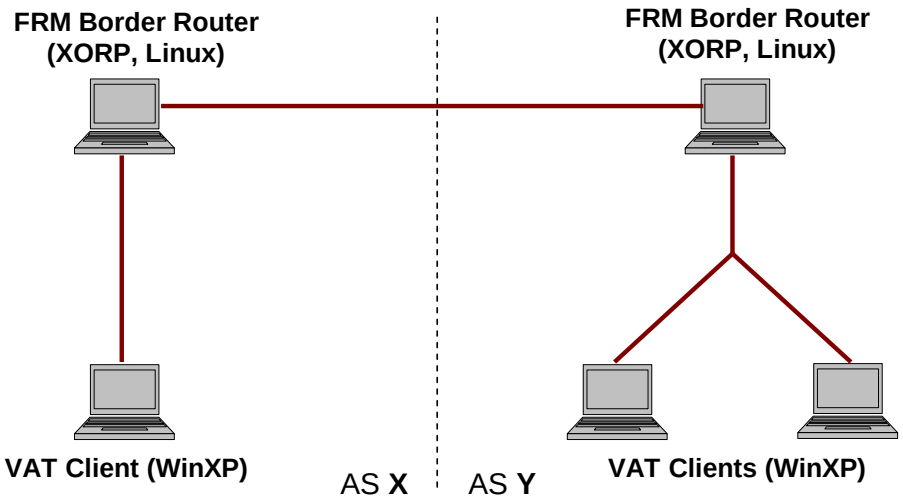


Figure 18: **FRM deployment set-up.**

μsec of processing time. It takes 0.34 μsec to update GRP_BF and 18.33 μsec to identify and invalidate the stale cache entry.

6.3 Deployment Experience

To test the end-to-end functionality of our FRM implementation, we deployed the prototype in a local testbed consisting of 2 interconnected FRM routers and 3 Windows desktop clients. The set-up is illustrated in Figure 18. We deployed an unmodified version of the VAT [32] audio conferencing tool on the client machines and observed packet delivery from the source client to receivers, demonstrating that our FRM implementation can forward packets end-to-end using legacy OS and application stacks.

7 Conclusion

In this report, we presented our work on Free Riding Multicast - a different approach to network-level multicast routing. Our design explores a novel set of tradeoffs, offering a simpler (no distributed tree computation) and easier-to-configure (no need for rendezvous point) solution at the expense of reduced bandwidth efficiency and greater resource demands on border routers - tradeoffs that we believe are worth exploring given current technology trends.

In conclusion, we note that since FRM makes no use of hierarchical address aggregation, its implementation represents a fairly general abstraction and in effect provides the ability to route on flat identifiers. Hence, our general scheme could be applied to more general routing services such as IP-layer anycast or name-based routing. The primary difference is that mul-

ticast requires matching *all* subscriptions, whereas the above require matching *any*. The only implication to our design is that false positives would be even more undesirable and a simple solution would be to enumerate subscriptions or use an alternate encoding scheme that admits only false negatives.

The goal of this project was to tackle a purely technical barrier to inter-domain deployment of multicast in the Internet, but other barriers do exist. However, given the growing adoption of Internet broadcasting, multimedia conferencing, massively multiplayer online games, and other networked applications, we conjecture that time may be right to revisit IP multicast and re-evaluate its chances.

8 Acknowledgements

The work presented in this report includes contributions from Sylvia Ratnasamy and my advisor, Professor Scott Shenker.

I would like to thank Sylvia for offering me the opportunity to collaborate with her on this exciting project and for her invaluable guidance, without which the FRM prototype would have never seen the light of day.

I would like to thank my advisor, Scott, for his valuable suggestions, feedback, and his generous support.

I also thank the anonymous reviewers of our SIGCOMM submission [28] for their valuable input and comments that helped improve this work.

References

[1] Stephen E. Deering and David R. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Trans. Comput. Syst.*, 8(2):85–110, 1990.

[2] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, 1984.

[3] Edward Castronova. Network technology, markets, and the growth of synthetic worlds. In *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, pages 121–134, New York, NY, USA, 2003. ACM Press.

[4] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. The effect of latency on user performance in warcraft iii. In *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, pages 3–14, New York, NY, USA, 2003. ACM Press.

[5] Internet Television News and Marketplace. `http://www.iptv-industry.com/`.

[6] Tony Ballardie, Paul Francis, and Jon Crowcroft. Core Based Trees (CBT). In *SIGCOMM*, pages 85–95, 1993.

[7] Stephen Deering, Deborah L. Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, and Liming Wei. The PIM Architecture for Wide-Area Multicast Routing. *IEEE/ACM Transactions on Networking*, 4(2):153–162, 1996.

[8] Hugh W. Holbrook and David R. Cheriton. IP Multicast Channels: EXPRESS Support for Large-scale Single-source Applications. In *SIGCOMM*, pages 65–78, 1999.

[9] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang. A Case for End System Multicast. In *Measurement and Modeling of Computer Systems*, pages 1–12, 2000.

[10] NewArch project: Future-Generation Internet Architecture. `http://www.isi.edu/newarch/`.

[11] GENI: Global Environment for Network Innovations. `http://www.geni.net`.

[12] FIND: Future Internet Network Design. `http://find.isi.edu`.

[13] Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming the Internet Impasse Through Virtualization. In *Proceedings of the 3rd ACM Workshop on Hot Topics in Networks (HotNets-III)*, November 2004.

[14] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, 1970.

[15] Yakov Rekhter and Tony Li. RFC 1771 - A Border Gateway Protocol 4 (BGP-4), 1995. Internet Engineering Task Force, Inter-Domain Routing Working Group.

[16] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. Scalable High Speed IP Routing Lookups. In *SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 25–36, New York, NY, USA, 1997. ACM Press.

[17] Pankaj Gupta. Algorithms for Routing Lookups and Packet Classification, December 2000. PhD thesis, Stanford University, December 2000.

[18] Content Addressable Memory, Cypress Semiconductor. `http://www.cypress.com`.

[19] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On Power-law Relationships of the Internet Topology. In *SIGCOMM*, pages 251–262, 1999.

[20] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. Design and Implementation of a Routing Control Platform. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, May 2005.

[21] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed Internet Routing Convergence. In *SIGCOMM*, pages 175–187, 2000.

[22] Sylvia Ratnasamy, Andrey Ermolinskiy, and Scott Shenker. Revisiting IP Multicast. Intel Research Technical Report.

[23] Venkata N. Padmanabhan and Lili Qiu. The Content and Access Dynamics of a Busy Web Site: Findings and Implications. In *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 111–123, New York, NY, USA, 2000. ACM Press.

[24] Lakshminarayanan Subramanian, Sharad Agarwal, Jennifer Rexford, and Randy H. Katz. Characterizing the Internet Hierarchy from Multiple Vantage Points. In *Proc. of IEEE INFOCOM 2002, New York, NY*, Jun 2002.

[25] Cisco Systems. Access List Configuration in Cisco's Gigabit Ethernet Interface. (Reports that GigE module supports up to 256K TCAM entries).

[26] Brad Cain, Steve Deering, Bill Fenner, Isidor Kouvelas, and Ajit Thyagarajan. RFC 3376 - Internet Group Management Protocol, Version 3, 1996. Internet Engineering Task Force, Inter-Domain Multicast Routing Working Group.

[27] BGP Table Statistics from AS 1221. `http://bgp.potaroo.net/as1221/`.

[28] Sylvia Ratnasamy, Andrey Ermolinskiy, and Scott Shenker. Revisiting IP Multicast. *SIGCOMM Comput. Commun. Rev.*, 36(4):15–26, 2006.

[29] C. Jin, Q. Chen, and S. Jamin. Inet: Internet Topology Generator, 2000.

[30] Mark Handley, Eddie Kohler, Atanu Ghosh, Orion Hodson, and Pavlin Radoslavov. Designing Extensible IP Router Software. In *Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI '05)*, Boston, MA, USA, May 2005.

[31] Linux man pages: NETLINK(7) .

[32] Van Jacobson and Steven McCanne. Visual Audio Tool. `http://ee.lbl.gov/vat/`.