Reducing the Forwarding State Requirements of Point-to-Multipoint Trees Using MPLS Multicast

George Apostolopoulos
ICS-FORTH, Greece
georgeap@ics.forth.gr

Ioana Ciurea
Redback Networks
iciurea@redback.com

December 14, 2005

Abstract

IP multicast was first proposed to improve the performance of applications that require group communications. Still, many years after its initial proposal, IP multicast is not widely used in the Internet today. One of the main roadblocks to the adoption of multicast is the potentially large multicast forwarding state in the routers. In this work we investigate this state scalability problem in the context of an IP VPN service built using MPLS. We focus on point-to-multipoint applications such as distribution of TV programs to multiple clients of a VPN provider. Using recent work on multicast MPLS, we show that with some very simple modifications to the signaling protocols used to establish the point-to-multipoint MPLS distribution trees it is possible to achieve significant reduction in the forwarding state required for setting up a large number of distribution trees. These savings in forwarding state can be achieved without having to send multicast traffic to receivers that do not want to receive it, also known as traffic leakage.

1 Introduction

1.1 Overview

IP multicast has been developed to enable the efficient operation of a variety of network applications that require communication between multiple parties. Typical multicast applications are media distribution or broadcasting which are by nature one-to-many, and many-to-many applications such as videoconferences or multi-player games. Although today most routers support IP multicast forwarding, multicasting is not widely deployed in the Internet. One of the main roadblocks to the adoption of multicast is the state scalability problem. Unlike unicast IP addresses that are now allocated in a hierarchical fashion and can be aggregated quite effectively, multicast destination addresses do not lend themselves to aggregation. This results in very large multicast forwarding routing tables.

Today, we also witness a significant penetration of IP Virtual Private Network (IP VPN) applications, where providers offer various services and connectivity between sites over a common shared IP infrastructure. Many believe that IP VPNs will become the predominant driver of provider revenues as we go forward. Among the many potential technologies that can be used to build the networks that support IP VPNs, Multi-Protocol Label Switching (MPLS) [3] has become quite popular due to its prevalence (practically all routing equipment vendors have implemented it in their products) and its ability to support traffic engineering in a natural way. MPLS is currently used in mostly unicast mode, with point-to-point tunnels between the routers at the edge of the provider's network where the customers are connected.

While until now most of the VPN applications needed only unicast communication over the backbone, newer applications emerge where unicast is not sufficient any more. VPN applications like Virtual Private LAN Services (VPLS) [4] where the backbone can be used to emulate a shared broadcast network, or media distribution, where a content provider distributes content to many of the customers of the VPN service, require more effective use of the backbone resources that can not be accomplished through simple unicast support. The media distribution applications are of particular interest since they are both very relevant (we are aware of at least 3 major providers that are investigating such a service for TV content) and

can potentially have extreme scalability requirements with thousands of channels and tens of thousands of receivers.

The need to support multicast communication over IP backbones in VPN applications has led to efforts to extend the existing unicast tunnel setup mechanisms in IP VPNs to support establishment of multicast tunnels [34], and for formulating and standardizing multicast MPLS [5, 6]. Multicast MPLS relies on a multicast MPLS label that controls the replication of incoming data to multiple outgoing interfaces and control protocols for managing the installation of the multicast MPLS labels in the network.

While there has been a large body of work investigating methods for reducing the state requirements of multicast, there are only few works that address the state scalability issue in the context of using MPLS multicast in the backbone of a VPN service provider. The state scalability problem in this environment is not much different than that in the conventional IP multicast cases. Similar to IP multicast, state refers to the amount of memory needed to maintain the multicast entries in the routing table as well as those in the forwarding engines. In MPLS multicasting the state is the amount of memory (usually in the forwarding hardware) that is required to store the multicast labels in the MPLS forwarding tables. Similarly to IP multicasting, reducing the amount of forwarding state needed is essential for successful deployment of multicasting, since the amount of forwarding state can affect both the cost of the forwarding hardware as well as its performance. Furthermore, some of the applications we will consider in this work require point to multipoint (P2MP) communication and result in *dense* multicast groups, where the number of receivers in a multicast group is a relatively large percentage of the total nodes in the network. Most of the existing work on reducing the state requirements of multicast has focused on sparse groups and these results are not directly applicable in the case of dense groups.

In this report we will show that taking advantage of the MPLS label forwarding paradigm and the signaling mechanisms that already exist in MPLS based IP VPN backbones for setting up P2P and P2MP LSPs, we can achieve dramatic savings in the state requirements for the P2MP tree construction in the backbone nodes. Unlike other methods for reduction of forwarding state, in our method this reduction can be achieved without delivering traffic to nodes that have not requested it (also known as traffic leaking) and with very simple modifications to the signaling protocol used to setup the P2MP LSPs. P2MP MPLS trees can be used in a wide variety of other applications, for example, virtual private LAN services (VPLS), multiplayer games, where players participating in the same game form a short lived multicast group, or videoconferences between multiple customer sites. Thus, we believe that the investigation of the scalability and implementation issues of P2MP MPLS multicasting is important and relevant for a large variety of applications.

1.2 Unicast and Multicast IP forwarding

1.2.1 IP routing and forwarding process

Currently, the Internet operates based on the TCP/IP suite of protocols for routing and application level communication. Packets are forwarded to their destination based on the destination address that is contained in the IP packet header. IP packets are forwarded through a network from a source to one or more destinations by a sequence of IP aware devices called routers. Using the router as a reference, we call the direction of a packet received by the router ingress direction, while the direction of a packet transmitted by the source is called egress direction. The ingress interface of a router is the interface on which a packet is received, while the egress interface is the interface where the packet is transmitted. Incoming packets are forwarded based on information in the forwarding information base (FIB). Such a forwarding entry contains a mapping (source address, destination address, ingress interface) $\rightarrow$ (destination interface, action).

A router matches an ingress packet, based on the information contained in the packet header, to a forwarding entry and executes the corresponding action. Since IP routing is destination based, a router will need to maintain one routing entry for each possible destination in the Internet. This could lead the routing table to become of an unmanageable size. Indeed, in the mid-90s there was a concern about the growth of the Internet routing tables and several techniques were developed to address the problem. Today addresses are allocated in a hierarchical fashion and can be aggregated in network prefixes of variable size. This allows routers to keep track only of destination prefixes which dramatically reduces the size of the routing tables. Now, packet forwarding relies on the router determining the "longest prefix match", i.e. the most specific destination prefix that contains the destination address of the packet. The hierarchical

aggregation of the IP addresses was made possible by their one-to-one nature where a single IP address corresponds to a single end-host that exists in a single logical location in the network. With proper coordination it is possible to assign similar IP addresses to hosts that belong to the same logical entities (i.e. service providers) so that these addresses can be aggregated. In this way, a router will need to know only a single prefix that allows it to reach the provider's network and not a route for each host in the provider's network.

1.2.2 Multicast Communication in IP networks

An unidirectional communication flow is said to be multicast if the traffic originates from one source and has to be forwarded to several destinations/receivers. Multicast communication may exist in any kind of switching networks; here we will consider only the case of IP multicast. Internet protocol [1] from its early days contains addressing schemes for multicast traffic. IPv6 architecture expands such addressing schemes for multicast traffic [2]. Typical multicast applications are media distribution or broadcasting applications like video on demand or video conferencing. The IP multicast allows a sender to send traffic to multiple hosts that belong to a "group". This group is identified by an IP address (allocated from the special space of class D addresses) and any host can send traffic to this group address. Multicast routers need to maintain routing state in order to be able to deliver traffic to multicast groups. This will be at least one routing entry for each group that passes the router.

A large number of different protocols that build different types of trees has been proposed over the years. Some protocols build source specific trees, where each source is using a different tree to reach the receivers in the group, while other approaches can have shared trees. In a shared tree, each router needs to keep one entry for each multicast group that is crossing it while in a source specific tree, each router will need to maintain an entry for each combination of source and group address that it serves, increasing in this way significantly the state requirements.

Unlike unicast IP addresses, group addresses reflect a logical group of nodes that does not have a well established logical location and it can span all the Internet. As a result there is not enough topological locality that would allow the hierarchical aggregation of the multicast IP routing entries. Over the years, this problem and the need for global coordination when allocating multicast IP group addresses have proven to be the main roadblocks to the deployment of multicast IP routing.

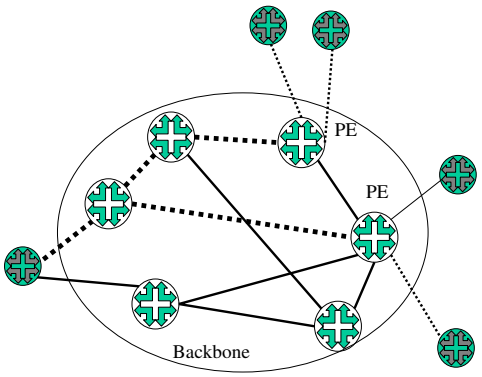


Figure 1: Example of a VPN

1.3 New multicast applications

Common applications that need multicasting, such as for example wide-area media distribution, operate at the Internet scale and require multicasting service between any two Internet connected hosts. Recently, there have been other areas that can benefit from multicasting and do not involve global deployment of multicast forwarding but instead they only need a much more limited multicasting capability within the network of a single network provider. These applications are related with Virtual Private Network (VPN) services.

1.3.1 Virtual Private Networks

A Virtual Private Network (VPN) represents a collection of hosts, belonging to the same organizational entity which are connected over a public (i.e. shared) transport infrastructure. Essentially, a VPN is a service agreement between two agents: (a) The customer, namely the organizational entity to which the VPN hosts belong. (b) The service provider (SP), which owns and manages the shared transport infrastructure. A collection of hosts that (a) are directly (physically) interconnected, without any transit through a 3rd party infrastructure and (b) are

part of the same VPN will be called customer permises (CP).

A customer edge router (CE) represents the demarcation point, on the customer side, between the customer network and service provider network. The role of the CE is to provide packets to the SP network in the format and at the rate which has been previously agreed. A service provider edge router (PE) performs an address mapping (or address translation) function. Packets received by PE from CE still reflect the customer's addressing conventions, which are irrelevant for the SP network. The role of PE is to map an address from customer's address space to an address meaningful for the SP network. Such address mapping is technology specific, and is realized mostly through encapsulation of packets into *tunnels*. Inside an SP network, the forwarding process occurs between an ingress PE and an egress PE, regardless of the final destination in the customer's network. A core router (CR) exists inside the SP network, and carries traffic based on internal SP addressing. An example of a VPN is shown in Figure 1.

In this work, we will use a simplified VPN model where customer premises are considered to be connected to the SP's network through a single point to point link. Also, the provider's backbone consists of a single domain.

1.3.2 Tunneling and encapsulation

In all VPN applications, the service provider must ensure an absolute segregation of traffic between different VPNs sharing the same physical infrastructure. Several layer 2 protocols, like Frame Relay and ATM ensure such segregation through their connection oriented nature; for example, an ATM SVC (Switched Virtual Circuit) represents an end-to-end agreement on VC allocation, such that all nodes along a selected path have agreed that traffic with specific VC designation belongs to an unique VPN. In many cases, the providers employ a connectionless technology in their backbones, more commonly these days plain IP forwarding, in the form of the so called IP VPNs [19, 20]. In such cases, the PE routers need *tunneling* mechanisms that will let them distinguish between packets that belong to different VPNs. Tunnels are unidirectional flows of traffic established between an ingress PE (which receives packets from a CE) and an egress PE, which distributes the packets to a CE, part of the same VPN, on the "opposite" side of the SP network. The ingress PE will multiplex all packets received from the CE into the tunnel. The actual multiplexing process consists of encapsulating each incoming packet into a specific format, labeled with the tunnel address. The egress PE will strip off the encapsulation for each packet received and forward the packet to the CP network. Core routers will forward the encapsulated packet based exclusively on the tunnel identification. Several IP tunneling mechanisms have been proposed, for example IPSec [31], L2TP [32], GRE [33], and MPLS [3].

1.3.3 Multicasting in VPNs and Point to Multipoint (P2MP) tunnels

Offering unicast VPN services is straightforward. It is rather more complicated though to provide multicast VPN services. In such services, VPNs should be able to carry broadcast or multicast traffic, originating from any site of the customer's network and destined to any of the other customer sites. Therefore, an ingress PE must be able to forward a packet to all other PEs in the VPN, or to a selected number of PEs. Attempting to satisfy these requirements with unicast tunnels may lead to a fully meshed infrastructure of tunnels. Using this approach, a single VPN would require $O(N^2)$ unicast tunnels in order to provide the necessary connectivity between all the PEs that participate in the VPN, since in general each PE will need to talk to each other VPN. This is usually referred to as the N^2 problem and can severely limit the scalability of a VPN solution since core routers will need to keep track of a very large amount of state for each of these tunnels for each VPN.

In order to reduce the number of tunnels needed to connect the PEs that participate in the same VPN one could consider Point to Multi-point (P2MP) tunnels, that have a single source and multiple receivers. These P2MP unidirectional multicast is also known as Source Specific Multicast (SSM) [37]. Using P2MP tunnels a VPN with N members needs only N P2MP tunnels to ensure that all the PEs that participate in the VPN can communicate with each other. From the IP tunneling mechanisms proposed, only GRE encapsulation and MPLS can support P2MP tunnels. GRE encapsulates customer traffic into an IP packet so it can use IP multicast. MPLS although initially a P2P technology is currently being extended to support packet replication through multicast labels [6, 23]. Due to the additional operational complexity, many providers prefer not to deploy IP multicast in their networks. As a result, in many cases MPLS is the most attractive technology for offering support for multicast VPN services.

1.3.4 VPN Multicast applications

Here we present some examples of multicast VPN applications. For each case we highlight how these applications can be implemented effectively with multicasting support over the backbone in the form of P2MP tunnels.

Media distribution One of the provider's customers is the media provider that originates the multicast traffic. Customers subscribe to the service through some application level protocol that notifies the source that a new receiver is added to the distribution list of a particular program. In most cases, for scalability reasons there will be multiple sources and the customer will be connected to the source closest to him. When a new customer joins, its CE router will have to receive the multicast traffic stream corresponding to the program. This means that the PE router that connects this CE router to the backbone should become a receiver for this multicast traffic stream. In this application, the flow of traffic is from the source to the receivers, resulting in unidirectional source specific multicast trees. An example of the network architecture for this application is shown in Figure 1. Each PE serves a large number of CE routers. A PE joins a multicast group (shown with broken lines) after at least one of its CE routers joins that group. An example of how P2MP trees could be used in this VPN application is shown in Figure 2(a). If a given VPN is serviced by PEs 1,2,4,5, and 6, each PE will be the source of a P2MP tree that has the other PEs as receivers. In the figure we show two of these P2MP trees, one with PE1 as source and one with PE6.

Given that a single PE or CE can also serve individual users it is not uncommon to have multiple thousands of receivers behind a single PE router. With so many receivers the chances that a PE will need to receive a given program are quite high, resulting in multicast groups that span most of the PE routers. Also, in general, the ratio of PE/P routers in a IP VPN network can be quite high, with more PE routers being constantly added as the customer base grows. As a result, we expect a large percentage of the total network nodes to be receivers for each multicast group, resulting in a dense multicast group scenario.

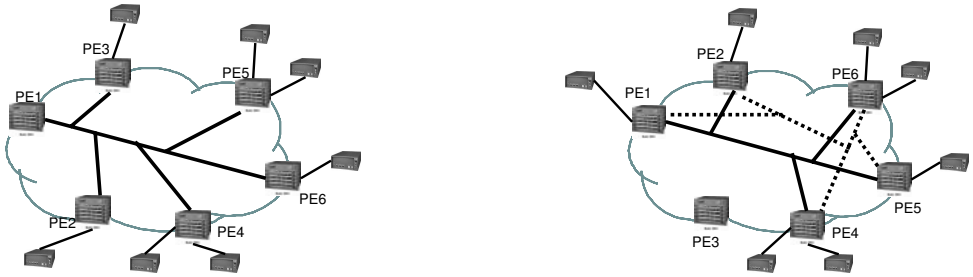
To give some example scaling numbers, in such an application we would expect to have thousands of multicast groups (TV programs). The size of the provider's backbone (both PE and P routers) will be in the order of the low hundreds of nodes. The total number of end-users (domestic and corporate) could be in the order of tens of thousands or even larger. These are the numbers we will use as guidelines in the evaluation of our approach.

Multicast customer traffic A natural example of the need of multicasting in VPNs is the case where the customer's traffic is multicast and needs to be delivered to multiple destination customer sites. In this case the PE attached to the source site needs an effective way to send traffic to multiple destination PEs. Current proposals in the Internet Engineering Task Force (IETF) (the organization responsible for the development of standards for the operation of the public Internet) propose the usage of a single shared tunnel for carrying multicast traffic for each VPN but since this can be very inefficient due to traffic concentration, they allow the establishment of source specific trees. An example of how P2MP tunnels can be used in a multicast VPN application is shown in Figure 2(b) where PE1 is the media source, and the receivers are behind PEs 2,4,5, and 6.

VPLS Another related application is Virtual Private LAN Services (VPLS) [4] where the provider offers a service that emulates a shared broadcast network and customer traffic is in the form of Ethernet frames. In order to handle broadcast traffic and Ethernet frames with unknown destination MAC addresses, a PE must broadcast them to all the other customer sites, i.e. to the PEs responsible for these sites. In current IETF proposals [29, 30] a PE replicates traffic to a separate tunnel for each destination PE. Clearly, a P2MP tunnel from the source PE to all the other PEs in the VPN can improve the performance of the backbone. In large provider networks with a large number of VPNs, the total number of P2MP trees can become very large.

In all three applications above there can be multiple P2MP tunnels from a single PE to a number of other PEs. These tunnels will originate from the PE and terminate to a set of other PEs that also carry the same VPN. Each PE may need to source as many as one P2MP tunnel per VPN it serves (it is also possible that there are multiple P2MP tunnels per VPN, if for example different QoS is required for each). The set of destination PEs for each tunnel will in general be different for different P2MP tunnels originating from the same PE.

The above set the context for our work in this report. We will investigate methods to improve the scalability of VPN applications that require multicast support in the backbone and implement this through MPLS based P2MP tunnels (or trees). Our focus will



(a) Media distribution

(b) multicast VPN

Figure 2: Examples of P2MP tree used in VPN applications

be on the reduction of the multicast MPLS forwarding state.

1.4 Report organization

The report is organized as follows: In Section 2 we survey the large volume of work on multicast state scalability and point out how our work fits in this area. Section 3 presents the issue of state requirements in the case of MPLS forwarding and Section 4 discusses the details of our proposal of how to achieve state aggregation when MPLS multicast forwarding is used. Finally, in Section 4.4 we evaluate the effectiveness and overhead of our approach, in Section 5 we discuss some finer points of our proposal and in Section 6 we summarize our findings and present our plans for future work.

2 Overview of related work

There is a large volume of work that attempts to address the state scalability problem of IP multicast. One approach focuses on sparse groups,where the tree contains many routers that are not branching points [7, 8, 9]. The long sequences of non-branching routers do not need to maintain group state if multicast traffic is tunneled through them. [7] proposes constructing dynamic tunnels between the branching points of the multicast tree. The protocol proposed requires multicast state to be maintained only by the branching points, the root and the leaves of a distribution tree; the multicast data sent between branching points is encapsulated and sent unicast, transparently to the intermediary nodes. Both [8] and [9] are using recursive unicast to reduce the state requirements of multicast. In [8] the branching nodes send unicast packets to one of the receivers that is downstream of an outgoing link, while in [9] packets are sent to the next branching node in the tree. The destination address of the packet it rewritten at each branching point. In [39] the authors propose a method that also depends on a dedicated node Network Information Manager System (NIMS) to build and have a global view of the MPLS multicast trees in the network. Branching nodes are informed by the NIMS on the identity of their next-hop branching node in the tree; intermediate routers do not maintain multicast state information and they are using the existing LSPs to forward data. Branching routers will inspect the multicast MPLS packets while the non-branching routers will not make any distinction between labeled multicast and unicast packets. All the above solutions generally introduce data processing overhead for the encapsulation/decapsulation of data at the tunnel endpoints as well as control packets overhead and protocol complexity for maintaining the soft state of the dynamic tunnels. These techniques do not really apply to dense multicast groups since most of the routers that belong to such a group will be branching points.

Another approach is to reduce IP multicast forwarding by merging routing table entries, even if they do not send traffic to the same set of output interfaces. In this way, some traffic may be forwarded

over links that it would not normally be sent, causing traffic *leaks*. Clearly, in these approaches, care needs to be taken so that the amount of traffic leakage is not excessive. [10] proposes to aggregate (and potentially leak traffic for) only groups that carry small amounts of traffic, in an attempt to reduce the overall amount of traffic leakage. This approach needs to determine the bandwidth usage of a certain group by periodic measurements of the number of packets sent to a group over a given period of time. This results in a lot of complexity, especially since the amount of leakage depends also on the aggregation actions taken by upstream routers. A different approach to leaky aggregation has been proposed in [11]. Multiple groups are forced to use a common shared tree called "aggregated tree", thus reducing the number of trees that need to be maintained in the network. The complexity of maintaining per group multicast state is pushed to the nodes that are at the domain boundaries, while the core nodes will only maintain shared aggregated tree multicast state. Since the aggregated tree may not match exactly the shape of the trees it is carrying, traffic leakage can occur. The authors suggest that the aggregated trees (which are essentially tunnels) can be constructed with any convenient tunneling technology (MPLS, IP-in-IP, or GRE) without providing more details. The same authors propose a variation of the aggregated tree concept in [12] for providing QoS multicast in a MPLS network. The forwarding in the core network is done using MPLS labels that are used to identify the aggregated tree. To some extent this work is quite relevant to our work. It is also using MPLS to build the aggregated trees and it is tailored for using over a MPLS backbone Since it is one of the the methods that addresses multicast MPLS, it will be one of the main "competing" methods that we will compare our solution against. There are also other similar approaches that attempt to aggregate whole P2MP trees on an end-to-end basis. In [40] a label aggregation scheme that pushes the complexity of aggregating the label entries at the edge routers is proposed. Each edge router maintains a table (Tree Node Table) having the IP addresses of all routers in the multicast tree. Using this table the ingress node will determine the label associated with the multicast flow thus allowing different flows that share the same tree to use the same label. In [41] the forwarding state is reduced by allowing different groups to share the same tree if their shapes are identical. In order to identify these same shape trees, a Tree Numbering technique is proposed where all the possible multicast trees in the MPLS network have associated a number. This number is computed from the leaves to the source by assigning a weight to every outgoing interface. Each router will participate in this computation and different multicast sessions will have the same number associated at the ingress. This number is dependent on the number of egress routers that are reachable through the tree and not by the number of intermediate core routers and is used to classify the multicast flow into the correct P2MP tunnel. In the two above approaches only trees that have exactly matching shapes are aggregated. This is in contrast to the aggregated multicast where traffic leaking is allowed and a significant part of the complexity of the solution has to do with finding trees that can be aggregated without causing too much leaking.

Finally, there is a large class of works [13, 14, 15, 16, 17, 18] that follow an end system multicast approach where the multicast tree is built as an overlay network of point-to-point connections between end systems, allowing the network to remain unaware of multicasting. These efforts attempt to solve more general problems with multicasting and not only the state scalability issue. One of these works focuses on MPLS networks: In [18] two protocols are proposed to solve the issue of using multicast in an MPLS environment; the solution is pushing the complexity of maintaining multicast state to the edge routers of the networks; this is accomplished by allowing only edge routers to be branching points; in one of the solutions (ERM), the core routers are involved in constructing the multicast tree while the second one does not involve them at all and relies on the existence of a Multicast Management router to construct a Steiner-heuristic tree; the data is forwarded among the edge routers and mapped into existing LSPs.

3 Multicast MPLS and its state requirements

3.1 Multicast MPLS

In MPLS, packets are forwarded based on information at the beginning of a packet (after the layer 2 header) called shim headers [3]. A shim header is 32 bits long and contains 20 bits of label used in switching, 3 experimental bits (called the EXP bits), a bit to signal the bottom of the label stack and 8 bits for the TTL (time to live) value. Multiple shim headers can be carried in MPLS packets in what is

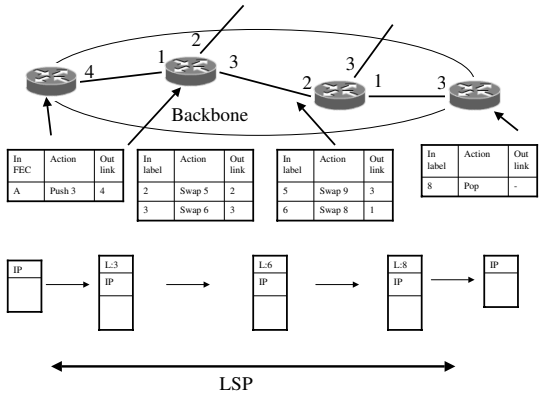


Figure 3: Operation of MPLS forwarding

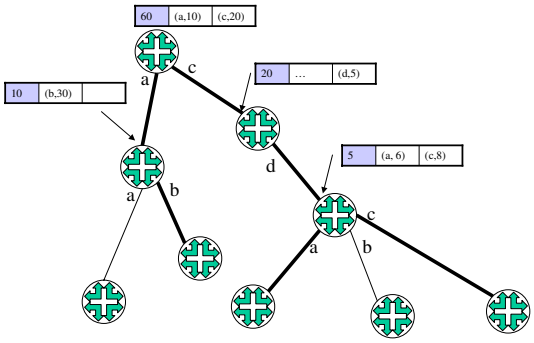


Figure 4: Example of a MPLS P2MP LSP

called the MPLS stack. MPLS switches, also called Label Switch Routers (LSRs) switch incoming packets based on their labels and the contents of their MPLS forwarding information base (FIB). The label contained in an incoming MPLS packet is looked up in the MPLS FIB. When a match is found the FIB will contain information for the action that should be performed. Actions could be (a) *swap* where the label at the top of the stack is replaced by another label and the packet is forwarded to some interface, (b) *pop* where the label at the top of the stack is removed exposing the label below or the original packet if it was the last label in the MPLS stack, and (c) *push* where a new label is pushed at the top of the MPLS stack.

The establishment of the MPLS FIB state across the nodes in a network is coordinated by a number of protocols that are collectively known as the MPLS control plane. The traffic engineering (TE) extensions of the Resource Reservation Protocol (RSVP-TE) [21] and Label Distribution Protocol (LDP) [22] are commonly used to create MPLS FIB entries. All the nodes in a MPLS network backbone must participate in the above protocols as well as a routing protocol that discovers routes between the nodes. LDP is mainly used to create MPLS FIB state that forces traffic to follow the routing protocol best-effort path, while RSVP-TE is usually used to setup explicitly routed paths that can be used for traffic engineering. These control protocols establish paths by installing appropriate entries in the MPLS FIBs of the nodes that participate in the path. After the forwarding state is created, packets are forwarded (or tunneled) over the path that the MPLS FIB entries create. These paths are called Label Switched Paths (LSPs). An example of how the MPLS FIBs of the different MPLS nodes cooperate to create an LSP and how the packet looks while being forwarded over the LSP is shown in Figure 3.

The MPLS control protocols also maintain state for their operation. For example RSVP-TE maintains some control state for each of the LSPs that crosses each node. Thus, we need to make a distinction between *control* and *forwarding state*. Control state exists on the control plane, usually in the control processor, where memory resources are relatively plentiful and cheap while forwarding state exists in the specialized hardware where resources are more scarce and definitely more expensive. Another consideration is the *scope* of the labels. A label can have a *platform* space and exist only once in all the FIBs in the system, or *interface* scope where the same label can exist in more than one interface with different actions associated with it.

Now we will discuss how the above concepts can be extended in the case of multicast data distribution, and in particular P2MP tunnels. While a unicast MPLS FIB entry has the form $inlabel \rightarrow (action, params)$, a multicast MPLS FIB entry can be seen as specifying a different action for a number of outgoing interfaces, e.g. $inlabel \rightarrow (action_{i1}, params_{i1}), (action_{i2}, params_{i2}), ...$ where $i1, i2, ...$ are some of the outgoing interfaces. For explaining P2MP tree operation, we need only focus on the swap operations. Considering only swap, a multicast FIB entry for some interface looks

like: $inlabel \rightarrow (i1, outlabel_{i1}), (i2, outlabel_{i2}), ...$ A packet that arrives at this interface and has $inlabel$ at the top of the stack, will be sent to interfaces $i1, i2, ...$ with the label at the top of the stack replaced with $outlabel_{i1}, outlabel_{i2}, ...$ respectively. This operation is consistent with the first implementations of the multicast MPLS labels, for example [23]. Assuming that the signaling protocols are extended to support the setup of P2MP LSPs (we will discuss this more in Section 3.3), these multicast labels can be organized so that they create a P2MP tree. An example of such a tree is shown in Figure 4.

3.2 The role of irreducible state

In this work we will focus on how much we can aggregate the forwarding state in the backbone. We will investigate how we can reduce the forwarding state, i.e. the number of MPLS multicast labels needed in the backbone nodes, without violating the requirement that a receiving PE router should be able to uniquely determine which group an incoming packet belongs to. To ensure this, we employ the often used technique of pushing a demultiplexing label in the MPLS stack of the multicasted packets. The source of the group will push into the stack an "inner" label that uniquely identifies the group before it pushes the label that will be used for forwarding the packet over the P2MP tree. When a packet arrives at a receiver, the receiver will execute a pop operation that will reveal the inner label, allowing the determination of the multicast group where the packet belongs. The mapping between inner labels and groups can be communicated between the source and the receivers through some application specific mechanism. In this context, all our state reduction work aims at reducing the forwarding state required for handling the outer labels.

The receivers will of course have appropriate forwarding state to handle the inner labels, that will be exposed when traffic is delivered to a receiver and the outer label is popped. In this work, we do not consider this state since this is to a large degree independent of the state aggregation mechanisms used in the backbone. The only information we require in the backbone is knowledge when traffic from a particular tunnel should be delivered locally at a receiver. When traffic contained in a tunnel should be delivered locally the multicast forwarding table entry for the tunnel contains an entry for a "local delivery link". This ensures that the forwarding plane will tap into the traffic in the tunnel and deliver it locally. We are not concerned with what happens with this traffic after it is delivered locally. A separate component of the router, or even a separate router altogether will maintain the table of the inner labels and will demultiplex this traffic to the appropriate receivers.

This separate component will usually serve a fairly large number of customers. It is not uncommon for a PE router to serve many hundreds of customers through access technologies such as ATM or 802.1q that allow a physical port to be split into multiple virtual circuits that can be assigned to different customers. In this setup, although the PE router essentially performs a multicast forwarding operation sending the information it receivers from the backbone to the corresponding CE routers, due to the extremely high fanout, it make sense to assume that the forwarding state needed for the customer demultiplexing of traffic will be kept separate from the backbone forwarding state. In our work all this state is abstracted behind the local delivery link and methods to aggregate it are beyond the scope of our work. Aggregating this state may not even be possible since the receiver will need to how to handle the traffic for each group; this is why this state is sometimes called *irreducible* [11].

3.3 Signaling protocol details for setting up the P2MP tree

A large number of signaling schemes for creating MPLS P2MP trees as well as IP multicast trees based on unicast data forwarding have been proposed [23, 7, 8, 25, 9, 34]. They all require some form of JOIN/LEAVE messages between the receivers and the source. Some approaches [23, 34] introduce a multicast explicit route that describes the whole tree to be used for setting up the P2MP LSP. Here we will present an abstract signaling protocol that is simple and has the properties necessary for implementing our proposal. As we will describe, the only feature we require of the signaling protocol is the ability for downstream controlled label allocation [22].

The signaling protocol first has to be able to identify the P2MP LSP. For a TV program distribution application the P2MP LSP can be identified by the address of the source node, and the session identifier that corresponds to the application that generates the traffic (there may be multiple applications that create P2MP LSPs from the same source node, for example a TV distribution application, a video conference application and so on). Finally, within a session, we

need to be able to identify an individual program, and for this we will use a program or group identifier. Unlike the P2P case, the establishment of P2MP LSPs is made complicated by the fact that there are multiple receivers. These receivers may potentially have different QoS requirements and this makes the LSP setup more complex. Also, in the P2MP case, one needs to deal with the cases were receivers leave and join the P2MP LSP over time. All these issues have been well studied in the context of IP multicast and an overview of the related issues can be found in [25]. Here we will ignore QoS and present an idealized description of how the P2MP LSPs can be setup, and then show how this operation can be mapped to RSVP-TE signaling. There is some very recent work in the Internet Engineering Task Force (IETF) that addresses the same issue [34].

Each receiver will individually determine a path to the source and then initiate the signaling for joining the (potentially) pre-existing tree for this source. Each JOIN message will contain the source/session identifier and the group identifier. In this project we will ignore the details of how the receiver will determine a path to the source, and whether he will need to have knowledge of the already existing tree for this group. We assume that some entity, either at the source or the receiver or network management, will determine the path between the source and a new receiver that wants to join the tree. This could be a fully specified source route, a partial source route or the first interface towards the source to be used for hop-by-hop forwarding. The receiver will be notified of this route information and will initiate the signaling for joining the tree. We will also ignore QoS issues, since other papers have investigated some of the issues related to creating receiver initiated QoS trees [25]. The JOIN message could be source routed to the source or just be propagated hop-by-hop.

The JOIN message from each receiver will be forwarded towards the source of the tree. To ensure that the distribution topology for a given group is always a tree, when the JOIN message first encounters a node that is already receiving in this group, the receiver joins the tree at that node. In order to be able to perform this check, each node needs to maintain state that identifies all the (source,session,group) that it actively receives. In our model, after a receiver joins the tree, there is no need for the join messages to be propagated to the source. Although there could be applications where the source will need to keep track of group membership, for example if each receiver has heterogeneous QoS requirements, in the TV distribution application each receiver requires the same QoS, so the source can be agnostic of the number of receivers that have joined the tree for a particular group.

As the JOIN message makes its way to the source, it creates control and forwarding state in each of the nodes it crosses until it joins the existing tree. In each node, a control state entry is created for the (source, session, group) along with a multicast label entry. Initially, this entry will contain only one outgoing interface. The JOIN message is updated with the label used by the downstream node, and carries this label to the upstream node, which is going to use it as the outgoing label in its own multicast label entry. Essentially we follow the downstream unsolicited model [22] for generating and propagating label state, since the labels are suggested by the downstream node. We also follow the ordered model, where the upstream node can generate labels for its upstream nodes only after it has received label mappings from its downstream nodes. When the JOIN message arrives at a merge node, there is no need to create control state there (since the node already has state for this source, session and group combination), and the multicast label at the merge node is updated to send traffic to the downstream interface it received the JOIN message from, using the downstream label that is contained in the message. There are no further actions required on the part of the tree between the merge node and the source.

When a receiver does not want to receive a group any more, it originates a LEAVE message towards the source that again contains the (source,session,group). The processing of this message is symmetric to the JOIN message. In each node that is traversed by the LEAVE message, the multicast label is updated so that it does not send traffic to the link that the LEAVE message arrived from. If there are no more downstream links active for a given multicast label, the label and the corresponding control state is removed. In this case, the LEAVE message is propagated upstream. If the processing of the LEAVE message does not cause the deletion of the control state at a given node, the message is not propagated upstream.

The above abstract protocol can be fairly easily mapped to an existing signaling protocol. For example RSVP-TE [21] can be extended to accommodate the above operations. In particular, the JOIN messages can be mapped to RSVP PATH messages. Un-

like the case of RSVP-TE where the label switched path is created in the same direction as the flow of PATH messages in our case we want the path to be created in the opposite direction. While in RSVP-TE the labels assigned from the downstream nodes are put in the RESV messages, in our case the node that sends the PATH message will propose the label to be used by its upstream node. This is very similar to what is proposed in [34].

If a soft state protocol link RSVP-TE is used, periodic message exchange will be required to refresh state that would otherwise time-out and be deleted. In general each node will need to refresh state by sending PATH messages to the upstream node (i.e. its parent in the tree for this P2MP LSP). Similar to the join PATH messages, these refresh PATH messages will not be forwarded all the way to the source, avoiding the explosion of the refresh messages that the source will have to process.

3.4 State requirements of MPLS multicast and state aggregation strategies

In the straightforward P2MP case, as shown in Figure 4, each node will need to carry one multicast MPLS label for each group it carries. Clearly, this is not scalable. A technique that is quite often used in the P2P LSP case is that of tunneling. If multiple LSPs need to be setup between a source node S and a destination node D, then only a single LSP (the "outer" LSP) is setup (thus only one label is needed in each backbone node) and the multiple LSPs are multiplexed over this LSP using the label stack in each packet. An additional label is pushed into the stack that specifies which "inner" LSP this packet belongs to. When the packet arrives at D, the outer label is popped and the inner label is used to identify the inner LSP.

The tunneling technique can also be used in the P2MP case. If there are multiple multicast flows from the same source that must be delivered to the same group of receivers, then only a single outer P2MP LSP would be needed. Although the forwarding state on the routers at the edge of the network would not be reduced since they should maintain state for each of the inner LSPs in order to be able to demultiplex traffic, the routers at the core of the network would only need to know the outer label. Certainly, finding which multicast flows (and consequently trees) to aggregate into a single P2MP tunnel can be challenging. Multicast trees may differ in single links even if they share the same set of receivers. One will need to keep track of the detailed tree shape (i.e. which links belong to a given tree) in order to be able to determine if two trees could be combined in a single tunnel. Furthermore, trees may not have the exact same set of receivers. Combining those trees in the same tunnel could result in traffic leaking, with traffic arriving at the receivers that are not interested in it. Finally, it may be possible to just combine only parts of the multicast trees into tunnels and achieve a partial reduction of the forwarding state.

Until now, there have been two main strategies for using tunneling for reducing the state requirements of MPLS multicast trees. One is the approach where whole multicast MPLS trees are combined in the same tunnels. When two trees have similar shapes they are combined into the same P2MP tunnels. Tree similarity is defined as the cost of traffic leaking when the two trees are combined. A centralized entity keeps track of the individual multicast trees and the changes in their shape as receivers join and leave them and determines how best to aggregate them into shared P2MP tunnels. Examples of this approach are [11, 40, 41].

Tunneling does not have to be applied to the whole P2MP trees: long shared segments of the multicast tree that do not contain branching nodes can be also be combined into a single (P2P) tunnel, thus reducing the state requirements on the non branching nodes down to a single entry per tunnel instead of a single entry per group passing the node. An example of such approach is the one in [39].

Finally, a third approach, which is the proposal brought forward in this project is to aggregate common sub-trees of the P2MP trees. While two P2MP trees may not be exactly the same they may share common subtrees. Thanks to the link local semantics of the MPLS labels, it is possible to just share the state required for these common subtrees, reducing in the way the overall state requirements and completely avoiding traffic leaking. We will see that this approach does not require any signaling between the source and the receivers (other than what is needed to setup the P2MP tree). Backbone nodes will individually detect if they can aggregate forwarding state and will coordinate this aggregation with the upstream neighbor(s) in the tree.

To give an example of how the local state in a node can be aggregated in this way, we assume that a node maintains a multicast MPLS FIB for each one of its

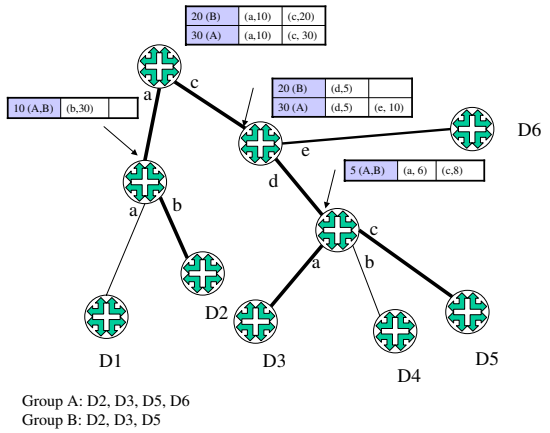


Figure 5: Example of state aggregation for two MPLS P2MP LSPs

interfaces. An entry in this FIB table will look like
$inlabel_1 \rightarrow ((i1, outlabel_{i1}), (i2, outlabel_{i2}), (i3, outlabel_{i3})...)$.

If there is a different $inlabel_2$ that has exactly the same set of outgoing interfaces and labels, for example
$inlabel_2 \rightarrow ((i1, outlabel_{i1}), (i2, outlabel_{i2}), (i3, outlabel_{i3})...)$

then these two entries can be collapsed into a single entry, since the actions that have to be performed for both $inlabel_1$ and $inlabel_2$ are the same. In this way, we can achieve a reduction in the size of the MPLS FIB of this particular incoming interface without any traffic leakage. Clearly, if we were to collapse these two entries into a new common entry, with potentially a new incoming label, $inlabel_3$, the upstream node should be informed so that they will start using $inlabel_3$ as the label they swap at the top of the label stack in the packets they send to this router.

It is important to realize that this kind of aggregation is only possible because we use MPLS that does packet forwarding based on labels with local scope. In IP multicast, the packet is forwarded based on its global scope multicast IP destination address. Even when aggregation of some multiple IP forwarding entries is possible, it may not be possible to achieve maximum savings since the aggregation can happen only by reducing the size of the prefix mask of the address. If there are some addresses that do not belong to the new range after the aggregation, this range has to be split into smaller ranges, reducing the amount of aggregation. In some cases it is not possible to aggregate state at all without introducing some leakage. With MPLS, each node has control on the label it will instruct the upstream label to use for the traffic it sends to it for some particular group. As we will show, each node can choose these labels so that it can reduce the amount of labels it needs to maintain, and at the same time increase the commonality of labels in the upstream node, so this node can also reduce its state.

The MPLS multicast forwarding functionality we have discussed so far is not much different than proposals such as the ones in [8, 9, 24]. In all these cases, the usual IP forwarding entries are replaced with entries that contain the unicast addresses of the receiver for each outgoing link. The destination address of the packet is replaced with the address of these nodes as the packet is forwarded through the multicast tree towards its destination. Performing these operations with MPLS is more efficient since usually MPLS forwarding is implemented in fast hardware. The non-standard IP forwarding that the above proposals require may make it much slower. Also, all these works focused on saving state in the non-branching routers in sparse groups. To the best of our knowledge we are the first ones to propose exposing and exploiting the commonality of state between the forwarding entries of multiple groups in dense group applications.

This is exactly where the MPLS forwarding paradigm can help since MPLS packets can have their label rewritten when they are sent out to the outgoing link. In both multicast and unicast MPLS, the significance of the MPLS label is limited to a single link. A node has the capability of choosing the label that directly connected neighbors will use to send traffic to it. It is also possible to modify this label while traffic is being received, without affecting nodes other than the node that was using this label to send it traffic. This is very different than the IP forwarding model where the destination address has global scope and can not be changed. This local nature of the MPLS labels can allow the aggregation of the MPLS forwarding state in a hierarchical fashion, so that common subtrees can use the same state independently of how many different trees they belong to. An example is shown in Figure 5. Being able to reuse label 5 for sending traffic to receivers D3 and D5, allows us to reduce the forwarding state of the parent node of D3 and D5. In this way, instead of requiring two forwarding entries for the two P2MP trees, in some nodes we can save one entry reducing the overall amount of forwarding state required in the network.

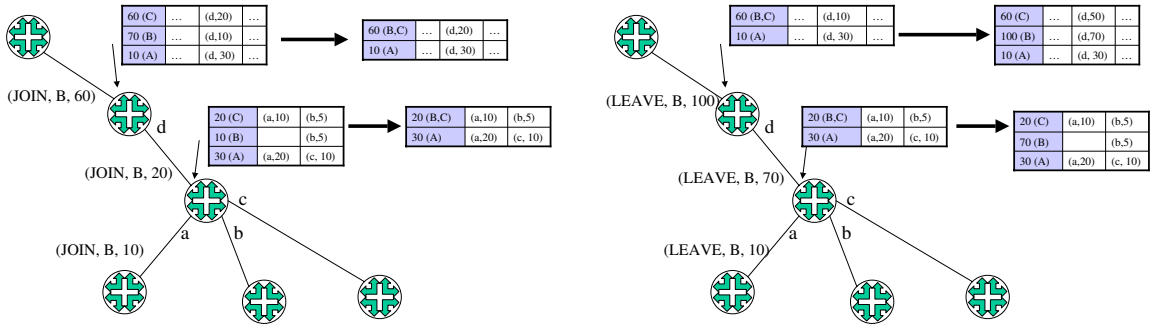
4 Reducing forwarding state using MPLS

As we described in the previous section, we follow the downstream on-demand label assignment model, where downstream nodes tell their upstream neighbors which label to use when forwarding traffic to them. This model allows each node to control what label it advertises to its parent and to optimize the selection of this label according to its own label state. If a node is to select a label that will reduce the overall amount of label state in the network, clearly it is preferable to reuse an existing label. Since we do not want to have traffic leaks, the existing labels should cover exactly the same set of outgoing links, and the same outgoing labels on each link. Since we have assumed locally scoped labels, each interface in the node will have its own separate MPLS multicast FIB. The selection of a preexisting label should happen only within the FIB of the interface that will be used to receive traffic from the parent node, since this is where the incoming traffic will arrive. Thus, the node simply has to look in the MPLS FIB for this interface and find another label that has the same set of outgoing links and uses the same outgoing labels on each of these links. If such a label is found, then this is the label advertised in the parent node and there is no need to create a new label entry in the FIB. This way, the node has achieved reduction in its label state. Moreover, by using a preexisting label, the parent node will also have more commonality in the outgoing labels it is using in its FIBs. This will enable the parent node to combine more of its own labels, reducing its own forwarding state too. This in turn increases the commonality of labels in its own parent and so on. Since we want to achieve leak free operation, labels will have to be de-aggregated when a receiver leaves a group. In these cases, a node will have to create a copy of an existing label and remove from the copy the interface that is not active in the group any more.

An example of the operation of this scheme is shown in Figure 6. In this figure we show the multicast forwarding state maintained by some of the nodes. There are three groups (A,B and C) and initially there is one label entry for each of the groups. When group B becomes active on link a, the upstream node of link a receivers the JOIN message from the downstream node, and updates its FIB. After the update labels 20 and 10 can be aggregated since they have the exact same set of outgoing links and outgoing labels. Thus, a new label is created (that logically corresponds to both groups B and C). It is this label that is advertised to the upstream node over link d. In a similar fashion, the upstream node determines that it can collapse two of its labels to a common label, allocates a new label and sends the new label to its own upstream node. As a result of this aggregation, there are two fewer labels in use. When group B is not active any more on link a, the upstream node has to recreate the label state for group B. A new incoming label is allocated for this label entry and this label is advertised to the upstream now, which in its turn has to de-aggregate its shared label that it used for both groups B and C. Eventually, there is again one label entry per group.

In essence this scheme aggregates locally in a simple distributed way trees that share the same shape. If a node advertises the same upstream label for two groups, this means that these trees share the same shape downstream from this node. This is true because the trees share the same set of outgoing interfaces in this node and the fact that their outgoing labels on each of these interfaces are the same, means that the shape of the trees downstream from each of these links is also the same. What is very attractive with this approach is its simplicity and the fact that it can be piggy-backed on top of the preexisting signaling mechanism for the setup of the P2MP LSPs, as long as this mechanism is using the downstream on-demand label allocation mode. Similar approaches for increasing the sharing of state for IP multicast such as [7, 8, 9] also need to use a signaling mechanism to exchange control state for their schemes. In IP networks this is an additional requirement and source of complexity since such signaling mechanisms do not exist. On the contrary, in the MPLS domain the incremental cost of performing state aggregation is much smaller since there are only simple changes required to the pre-existing signaling protocols.

In order for this aggregation to work correctly, a node that receives a suggested label from its downstream node has to incorporate this label in its FIB before it selects the label that it will suggest to its own upstream parent node. Incorporating the proposed downstream label may result in a label entry being aggregated with some other label entry. Typically, when the tree is being setup and receivers are added, we expect to see a lot of label aggregations. As more receivers for different groups are added, more and more groups will be able to discover common shapes in their trees and multiple label entries will



(a) Label aggregation

(b) Label de-aggregation

Figure 6: Example of a label entry aggregation and de-aggregation

be combined into a single entry. On the other hand, when receivers leave a group this will cause a change in the shape of the tree for the affected groups. This will result in label entry de-aggregations as the group is removed from the label entries it was sharing.

In general, the label received from the downstream node may cause either the aggregation of existing entries, or the splitting of a single existing entry into two entries. In both cases, the receiving node must locate its FIB entries that are affected and change some of the existing outgoing labels on these entries. This is not hard to do since the signaling message will contain the group identifier for which this operation is performed. The node will need to maintain a structure that maps each group to the label entry used to receive traffic for that group. If many groups share the same label (as will be the case when these entries are aggregated) many groups may point to the same label. Using this information, the node can quickly locate the entry being used. Then, it will need to modify the outgoing label in one of the outgoing links. To do so, it can check if the new label entry, after the change of the outgoing label, can be aggregated with an existing entry. If yes, then it simply updates the label entry pointer for this group to point to that label. If there is no appropriate label entry for aggregation then the node will have to create a new entry. In this case, the node will have to allocate a new incoming label for the new entry and send this entry to its parent node.

It is important to note that in general each time a receiver joins or leaves a group, the signaling message will have to propagate all the way to the group's source. This is necessary since each addition or deletion of an outgoing interface in a label entry or even the change of an outgoing label in one of the outgoing links will cause a change in the upstream label this node advertises to its parent for this group. This is in contrast to the signaling behavior we described in Section 3.3 where the messages for receiver joins and leaves need only propagate until a node that already have state for the group in question. As a result, we expect an increase in the volume of signaling messages when the label aggregation techniques are employed. We will further characterize this increase in Section 4.4.

4.1 Implementation description

In Algorithms 2 and 3 we present the pseudo-code for the operations that each node performs on reception of a JOIN or LEAVE message. Algorithm 1 shows a helper function used by both the JOIN and LEAVE processing. These operations occur in the control plane of the node. A node will receive the JOIN/LEAVE message from *downstream_link* and this message will contain a *downstream_label*. Each node should maintain a search structure which it can search based on a given forwarding state (which labels are used for which downstream interfaces). A search for the *old_state* will use the outgoing labels before adding the newly received label, while a *new_state* is after the *downstream_label* has overwritten the old outgoing label for the *downstream_link*. The node can retrieve the *old_state* for its control plane where un-

Algorithm 1 Helper function

```
check_for_state(state) {
 node = tree_lookup(state)
  if (node exists) {
   node->ref_cnt ++
   upstream_label = node->upstream_label
  } else  {
   node = new_node()
   node->ref_cnt = 1
   node->upstream_label = alloc_new_label()
   upstream_label = node->upstream_label
   node->key = new_state
   tree_insert(node)
 }
 return upstream_label
}
```

Algorithm 2 JOIN processing

```
if (G is new tree) {
 upstream_label = check_for_state(new_state)
} else {
 /* tree exists */
 node = tree_lookup(old_state)
 node->ref_cnt --
 if (node->ref_cnt == 0) {
  tree_delete(node)
 }
 upstream_label =
    check_for_state(new_state)
}
return upstream_label
```

Algorithm 3 LEAVE processing

```
node = tree_lookup(old_state)
node->ref_cnt --
if (node->ref_cnt == 0) {
    tree_delete(node)
}
upstream_label =
    check_for_state(new_state)
return upstream_label
```

aggregated information is maintained for each P2MP tree.

As we discussed in the previous section, each node will need to search efficiently in all the forwarding states it maintains to determine if it can reuse an existing state for incoming JOIN and LEAVE messages. Note that this operation will be performed in the control plane, so we have relatively more flexibility for the amount of time and memory we have available compared to the forwarding plane operations.

4.2 Signaling issues

A node will change the incoming label for a label entry in two cases: when this entry is aggregated with another entry, or when it is a new entry that resulted from de-aggregation. In the case of aggregation, the node should not remove the old (and now redundant) label immediately since it will take some time for the parent to process the signaling message and stop using this label for traffic sent to the node. If the label was deleted immediately, data could get lost. When the incoming label changes as a result of de-aggregation, there will be a temporary traffic leak until the parent updates its outgoing labels, since it will be using the old label that will forward traffic to the link that the group just stopped being active on.

To handle the case where the old label must not be deleted prematurely, it would be useful if the delivery of the signaling messages was reliable and acknowledged. Then a node would start a timer for removing the old label entry when it received acknowledgment from its parent that it received the signaling message. Protocols like RSVP and RSVP-TE are soft state and they do not require acknowledgments for the exchanged messages. Messages are exchanged periodically to ensure that state does not timeout. This is clearly not sufficient for our needs. Nevertheless, mechanisms to ensure reliable delivery of RSVP messages have already been proposed [26].

The above label aggregation scheme does not introduce operational problems in case a signaling message gets lost. If a signaling message for adding a new receiver to a tree gets lost, this will just delay the flow of traffic to this new receiver. If a signaling message for deleting a receiver from a group is lost, this will result in traffic leaking until the parent node eventually receives information that it should stop sending traffic to the downstream link. If the messages lost are between nodes that already had state for the group and they are only to inform parent nodes for changing of the incoming labels, the effects of a message loss

will be similar. If the change of the label is because of traffic aggregation, traffic will flow using the unaggregated state for a while, and the deletion of the unneeded previous label entry will be delayed until the parent node receives and acknowledges the lost message. If the change of incoming label is caused because of de-aggregation of a previous label entry, then traffic will leak until the parent node receives the lost message.

4.3 Implementation example

Each node will have to be able to check in an efficient way which of its local labels it can aggregate. In order to do so, it will need to check which of its local labels have the same set of outgoing interfaces and are using the same outgoing labels on each of these interfaces. A simple and efficient method to perform this test is to use a tree as a lookup structure. Each label entry (which corresponds to a group) is mapped to a tree key. Assuming that the node has N outgoing interfaces and it is using a 20-bit label for each of them, then the key can be an array of N 20 bit numbers, each position of the array corresponding to one of the outgoing interfaces. If a group is not using a particular outgoing interface, its position in the bitmap can be set to 0. Else, the k-th position of the key contains the outgoing label used in outgoing interface i_k. This key can be used to enter the group in a lookup tree. All groups that have the same set of outgoing interfaces and labels (and thus can be aggregated) will end up in the same node of the tree. A simple linked list can be used to link all these groups together. If a group is to be delivered locally, we can use an imaginary "local delivery" link which will contain the label to be used for the local delivery. Since this label will always be the same (a pop) all the groups that are to be delivered only locally can be readily aggregated.

To make sure we have aligned access to memory, the number of bits allocated in the key for each interface can be 32 (instead of the 20 bits of the MPLS label). In this way, a node that has 16 outgoing interfaces will need (16+1)*32 = 544 bits = 68 bytes for the key if we include the special local delivery link. This does not necessarily mean that if we have 10,000 label entries we will need 680,000 bytes of storage. As the labels are being aggregated, the tree will be correspondingly smaller since it will need to store only the aggregated labels. Any library code that implements a lookup tree can be used, making implementation easy. With the right implementation, performance can be quite high. For example, using a balanced tree implementation on a Pentium 4 system running at 1.8 GHz, building a tree of 32K distinct label entries each with 36 outgoing interfaces, each with a 32 bit outgoing label, takes 256 milliseconds. Performing a lookup for a label entry that does not exist in this tree takes 7 microseconds.

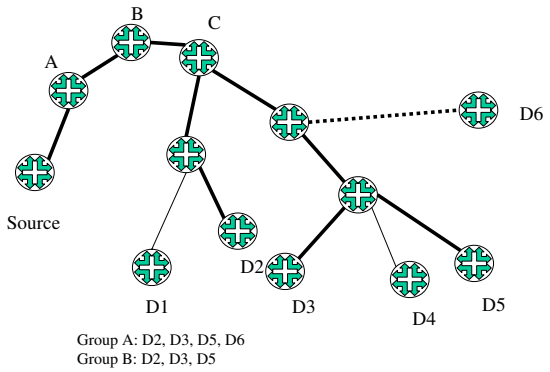


Figure 7: Examples of state aggregation methods

4.4 Evaluation

We evaluate the characteristics of the different tree schemes and the effectiveness of the distributed state aggregation using simulation. We start by describing the details of our simulation setup.

4.4.1 Methods Compared

We evaluate our method against the baseline approach of no forwarding state aggregation where each P2MP tree consumes one forwarding entry in each of the network nodes it is crossing and against some representative alternative methods for forwarding state aggregation: multicast state aggregation. These methods are:

Tunneling: In this method, if there is a sequence of consecutive nodes that are not branching it is possible to setup a unicast tunnel over these nodes. This tunnel will consume one forwarding state entry in each of these nodes. If there exist multiple P2MP trees that all share this segment of non-branching nodes it is possible to tunnel all of them over the unicast tunnel established over the non-branching nodes. In this way, it is possible to avoid allocating additional forwarding state for all these P2MP trees in the

nodes of the tunnel, reducing in this way the overall state requirements. In the example of Figure 7, there are two P2MP trees/groups with a slightly different set of receivers. Both trees will have to cross nodes A and B and none of these nodes is branching. As a result, a unicast tunnel can be setup over these two nodes using only a single forwarding entry per node instead of two. Of course this scheme is more effective when there are multiple P2MP trees that have relatively long sections of non-branching nodes. This is more likely in cases where the multicast groups are sparse, i.e. the number of receivers is relatively low compared to the overall size of the network. A typical implementation of this scheme would require the tunnels to be setup and torn down dynamically depending on the underlying P2MP trees. Further, each tunnel has to be signaled and its forwarding state needs to be maintained explicitly or be refreshed if it is soft state.

Aggregated Multicast: In this method, when two P2MP trees share the same set of receivers, it is possible to reuse the same P2MP trees instead of setting up a new one. In this way, only forwarding state for a single P2MP tree has to be established in the network. Two trees do not have to share exactly the same set of receivers. It is possible to still reuse a P2MP tree if its set of receivers includes all the receivers of a second tree. In this case, it is possible that there will be some receivers that will get traffic that is not intended for them and we will have traffic leaking. By calculating the cost of this additional traffic it is possible to balance the cost of leaking with the benefit of forwarding state aggregation achieved. A typical implementation of this scheme would use a centralized server that would compute the shapes of the different P2MP trees and for an incoming P2MP tree would decide if it should reuse one of the existing trees. Each time a P2MP tree changes (due to a receiver departure or arrival) this tree matching has to be performed again. In our simulations we will evaluate both non-leaking and leaking versions of the aggregated multicast.

In the non-leaking version, the two trees have to match completely in order to be combined into a single tree. In leaky aggregated multicast two trees are compared and the number of links they differ by is considered to be the leak when these two trees are combined into a single aggregated tree. In the example of Figure 7, the two P2MP trees can be aggregated, but the aggregated tree will contain the link shown with the broken line. Traffic for both groups will flow over this link although only group A needs it. As a result, there will be traffic leaking in this leak equal to the volume of traffic for group A. If this leak is below a threshold then the two trees can be combined but having one of the trees extended appropriately so it covers all the receivers of the other tree. Each time a new group has to be placed in the network, its tree is computed first and then the centralized controller searches all the existing trees for finding the minimum leak match with the new tree. If this minimum leak is below the threshold, the incoming tree is combined with the existing tree else the incoming tree is added to the network. We evaluate cases where the leaking allowed for the aggregated multicast is 20%, 40% and 60%.

4.4.2 Comparison Metrics

We will compare the above methods in a number of different metrics. Some will be measured through our simulations while others are harder to measure and quantify.

Forwarding State: This is of course our main metric. This is the size of the forwarding table maintained at each node. For each method, we measure through simulation the total amount of forwarding state needed in all nodes in the network, and we report this value as a percentage of the forwarding state needed without aggregation. As we discussed in Section 3.4 the forwarding state we consider is only the state kept by the core routers and does not include the irreducible state at the edges of the provider's network.

Overheads: It is rather difficult to compare the above methods in terms of their relative costs and overheads because the nature of cost is very different for each method.

- In the case of the proposed label allocation method, the largest overhead is the number of additional signaling protocol messages that will be required to be sent to the source of the P2MP tree, to handle cases where the label allocated to the downstream subtree changes. These signaling messages are in addition to the messages needed regularly by the signaling protocol since typically the signaling message will need to propagate only until a node that has state for the group is found. We measure this increase in our simulations and we show it in our results. We ignore the message size and we count only the number of messages. Without assuming RSVP

or any other particular protocol, we count the JOIN messages towards the source and for each JOIN we also count one acknowledge message. Remember that without state aggregation we have assumed that JOIN and JOIN messages will stop propagating to the source as soon as another node that belongs to the tree is found. With state aggregation, messages will in general need to travel all the way to the source. An additional source of overhead in the proposed method is the lookup that each node does for matching the forwarding state of incoming P2MP trees with that of existing trees. This search though as discussed can be implemented efficiently with a search structure (a tree for example) with complexity of $O(logN)$ where N is the size of forwarding state per node. Further, since in many nodes the forwarding state is significantly reduced, this cost becomes even smaller there.

- In the case of tunneling, the overhead lies mostly in the signaling messages needed to establish and maintain the tunnels. Each time a tunnel is discovered a signaling pass is required to set it up and similarly to tear it down when it is not needed. Also, if the signaling protocol used is soft state, signaling messages will be needed to refresh the tunnels and keep them active. While these messages are signaling messages like the ones in the label aggregation method, their purpose is very different and they are not directly comparable to the messages needed in the label aggregation method.

- For the aggregated multicast case the cost lies in the computations for determining if an incoming P2MP tree can be merged with one of the existing P2MP trees. The algorithm that makes these decisions must attempt to maximize the aggregation of forwarding state (by reusing existing trees) without resulting in excessive bandwidth leaking due to the imprecise match of the trees being aggregated. This is usually accomplished by setting a limit in the leaking and never aggregating two trees is the resulting bandwidth leaking exceed this limit and also by choosing for aggregation trees that result in the less possible bandwidth leaking. The computation for the best aggregation match can be rather time consuming since it is linear to the number of established P2MP trees, that as we discussed at the beginning can become very large in certain VPN applications. Again, due to its nature this cost does not appear in the other two aggregation methods, and it is not possible to compare it against the overheads of the other methods. An additional cost of aggregated multicast is the signaling cost for notifying the tree manager for establishments or tear downs of P2MP trees as well as changes in the receiver set of existing P2MP trees. Since all the tunnel aggregation is performed at a centralized location by the tree manager, the latter must be notified of all changes in the network.

Ease of implementation and deployment: It is always hard to argue about ease of implementation of different solutions. Nevertheless, it appears that the method we propose is simpler to implement than the two other competing methods. Our method requires some modest changes in the signaling protocol mostly in the rules for originating and propagating signaling messages and the execution of some relatively simple code which runs locally on each node and does not interact with the calculations in other nodes. Receiver additions and deletions are simple to handle and they do not require special procedures.

The tunneling scheme has to dynamically establish the tunnels as well as maintain them with state refreshes. Nodes need to keep track of how many P2MP trees use the tunnels and tear down the tunnels that are not needed. In cases that the shape of the P2MP trees changes due to receiver departures or arrivals, tunnels may need to be spliced together or spliced apart [7] resulting in some complex operations.

The aggregated multicast approach requires the operation of a centralized tree manager [11] that needs to be informed of the establishment or tear down of each P2MP tree in the network as well as changes in the receiver sets of existing P2MP trees. In cases of receiver changes the tree matching algorithm has to be performed again to see if a P2MP tree can reuse an existing P2MP tree. The tree manager must maintain data structures that describe all the existing P2MP trees in the network, and ensure that these data structures always reflect the situation in the network, even in the case of loss of signaling messages and potentially failures of the network or the tree manager itself. This will require significant protocol complexity to achieve. Also, if leaking is allowed, then some thresholds must be set that control the maximum amount of leaking that is allowed. Determining appropriate thresholds for a given network may not be a trivial task.

In terms of ease of deployment, all three approaches require that all nodes in the network implement the method, so they do not offer the possibility of gradual introduction.

4.4.3 Simulation Setup

Here we discuss the parameters of our simulation models.

Topologies: We performed experiments with a large number of topologies both realistic and artificially generated. For the artificially generated topologies we used 50 node topologies created using the BRITE [27] tool. One type of topologies follows the older Waxman model [35], while the others are based on the newer Barabasi model [36] that is adapted to account for the recently discovered power laws in some of the topology metrics of the Internet. The average degree of the Barabasi and Waxman topologies is 4. In addition we use topologies that correspond to real ISP networks as measured through the Rocketfuel project [43]. From the multiple ISP topologies available, we report results on two ISP and ISP2. ISP has 42 nodes and average degree 3.5 while ISP2 has 60 nodes and average degree 4.5. All the links in all topologies have the same cost and capacity.

Routing: The multicast tree for a group is built by picking the shortest path from the receiver to the source and setting up a path. When this path hits a node that is already member of the tree we terminate. This ensures that we build a tree. This is in essence a minimum delay tree (as compared to a minimum cost tree). Since there may be multiple paths (with the same cost) between a receiver and the source, we can load balance by choosing different paths each time in an attempt to spread the load equally and not overload any single path. Load balancing can be turned on and off in our simulations and we investigate its effect in the shape of the P2MP trees and the effectiveness of the state aggregation methods.

Group Model: The selection of group model is very important for the realistic and effective evaluation of forwarding state aggregation methods. Factors such as the clustering of receivers, their number relatively to the size of the network and their relative position to the source can significantly affect the performance of the various schemes. Still, selecting an appropriate group model for the VPN based applications we consider in this work is not simple, mainly because there is no data for the behavior of multicast enabled VPN applications. Actually there is not sufficient data even for the basic VPN statistics, i.e. the distribution of VPN sizes in a typical service provider, the relative positions of the sites of the same VPN and so on. The only information that exists anecdotally is that the large majority of VPNs are small (i.e. consisting of a handful of sites) while there are very few but very large VPNs that have sites attached to almost all the PEs of a provider. Such large VPNs could correspond for example to companies with national coverage that have presence (and sites) in all the cities that the provider serves. Attempting to model the multicast requirements of these VPNs becomes even harder. The few existing studies of multicast group membership and location (for example see [38]) found different types of behavior for media distribution and game playing but focused on Internet scale multicasting. In a VPN the situation is rather different since we are interested in the multicast patterns in the backbone while the multicast traffic follows the customer applications. Since the customer multicast receivers are "hidden" behind PE routers we should somehow attempt to "map" the customer multicast patterns into backbone multicast patterns. This is by no means trivial and is beyond the scope of this report. Thus we employ a random group model where group sizes and receivers are uniformly distributed and we consider the following aspects:

- **group number and placement:** We do experiments with the number of groups ranging from 1000 to 5000 groups. In the case of the media distribution the number of groups corresponds to the number of trees originating from the sources of the media. Each receiver will attempt to connect to the media sources that is closer to it. For the VPNs, the number of groups corresponds to the total number of trees setup (and not the number of VPNs). Say we have 100 VPNs with 10 members each, then each VPN will contain 10 trees for a total of 1000 trees (and groups). The groups are placed randomly.
- **group membership density:** We have three different densities
  - **small groups (sparse):** where the relative size of the group is small compared to the network size (average group size 5),
  - **large groups (dense):** where the average group size is 33 for ISP1, 50 for ISP2 and 45 for the 50 node topologies
  - **mix:** where the average group size is uniformly distributed between the above min

and max values.

- **group dynamics:** While one would expect receivers to be added/deleted to groups after they are established, we do not simulate the evolution of the network over time. Rather the size and the receivers belonging to each group are determined in advance and all receivers are placed on the network at the same time. There are multiple reasons that we feel this is not a significant issue with our simulations. One is that for the applications we consider group membership should be rather static. In VPN applications the number and location of VPN sites does not change often. In media distribution, usually each PE router serves enough customers/receivers for a given media stream to mask their frequent arrival and departure. Another reason is that our method deterministically depends on the shapes of the P2MP trees established and not on the order these trees are built. Thus, after a long run of receiver additions and deletions, the degree of aggregation will depend only on the statistics of the group sizes at that moment. This is not necessarily true for the aggregated multicast case since new aggregation decisions are made each time one of the P2MP trees changes shape.

Applications: we evaluate multicast as will be used in two different applications:

- **media distribution:** Where there are multiple media sources that multicast content to multiple groups. The overall content to be served is split among the different sources, so groups of receivers that are interested in accessing certain content will send JOIN messages to the corresponding source thus forming a single source specific tree for each group.

- **multicast VPNs:** A subset of the nodes of the network participates in a VPN. Then, from each member of the VPN we set up a source based tree with all the other members of the VPN as receivers.

Simulation statistics: All the experiments were performed multiple times with different random generator seeds. In all cases the 95% confidence intervals were computed for the results and they are very small (usually less that 1% of the reported numbers) and definitely do not affect the reported results.

4.4.4 Simulation results and discussion

In Figures 10 to 33 we present three types of results. We present results for 4 topologies (Waxman, Barabasi, ISP, and ISP2) two applications (Media distribution and VPN) and all group types (dense, sparse and mix). For the media distribution application we show results with dense and mix groups since in we expect groups to be large as a result of multiple customers viewing similar media streams. On the other hand, for the VPN application we show results for mix and sparse groups since we expect the size of most VPNs to be relatively small when compared to the size of the network. In Figures 10 to 17 we show the improvement in the state requirements that each method managed to achieve, in Figures 18 to 25 we show the increase in signaling messages that our method incurs, when compared to the signaling methods required to set up the P2MP trees when our method is not used. This increase is expressed as a percentage of the messages our method requires when compared to the baseline number of messages. In Figures 26 to 33 we show the amount of leaking required by aggregated multicast. This amount of leaking is expressed as a percentage of the base traffic (what would be required to deliver the multicast traffic to its destination without leaking). Finally, in Figures 34 to 37 we compare the performance and overhead trade-off of aggregated multicast with that of our method in the case of placing 5000 groups. Aggregated multicast can achieve an increasing amount of aggregation if one is willing to allow more leaking of traffic. Our method, in the form presented in this report, does not allow traffic leaking. Below we summarize some conclusions that can be drawn from these data.

- For the media distribution applications, in all the dense cases our approach performs better than aggregated multicast with 20% leaking but worse than the cases with 40% and 60% leaking threshold, while in the mix cases, our method performs worse only with aggregated multicast with a 60% leaking theshold. This is significant since our scheme does not introduce any traffic leaking. On the contrary, aggregated multicast introduces leaking that can be up to 50% of the overall bandwidth used in the network as can be seen in Figures 26 to 33. Especially in the mixed case, aggregated multicast has to incur a rather high amount of traffic leaking in order to match the performance of our scheme. The situation

Topology	Nodes	Links	Avergage Degree
ISP	42	85	3.6
ISP2	60	135	4.5
Waxman	50	100	4
Barabasi	50	97	4

Figure 8: Network Topology Parameters

Parameter	Value
Topology	ISP, ISP2, Barab50, Wax50
Link Cost	1
Link Size	Infinite
Routing Algorithm	Shortest Path
Routing	Source Routing
Load Balancing	Random among minimum length shortest paths
Number of groups to place	1000 - 5000
Applications	VPN and media distribution
Group placement	Random, Uniform
Sparse group density	Size is $\leq$ 10% of network size
Dense group density	Size is $\geq$ 90% of network size
Mixed group density	Size is between 10% and 90% of network size
Group dynamics	Receivers join once, never leave

Figure 9: Simulation Parameters

is similar for the VPN applications where our method performs somewhat better when compared to aggregated multicast.

- For the VPN applications the situation is different mainly due to the fact that we evaluate them using mixed and sparse groups. We see that in all cases our method performs the best. Only by increasing very much the leaking thresholds of aggregated multicast (up to 100%) it becomes possible to match the performance of the label based aggregation. But using this very high leaking thresholds results in dramatic increase in the amount of extra traffic introduced to the network that can reach up to 70% and 80% as can be seen in Figures 26 to 33. Thus, in cases where the size of the groups is small and consequently there is less similarity in their shape of the corresponding P2MP trees aggregated multicast does not perform well and requires substantial leaking in order to match the performance of label based multicast. Another reason for the low performance of aggregated multicast is that the total amount of groups are spread among many more sources, so the number of groups per source is rather low. This limits the amount of aggregation choices available to aggregated multicast since it can only aggregate trees that have the same source (see Section 5 for more details). On the other hand, our scheme can discover similarity even in subtrees of P2MP trees with different sources and can achieve significant state savings exploiting these cases.

- Aggregated multicast performs quite well in cases where multicast groups are dense. Indeed, this results in P2MP trees that span most of the nodes in the backbone and as a result have fairly similar shapes that aggregated multicast has many opportunities to aggregate. On the other hand, when groups are sparse, then there are less opportunities for aggregation within the given leaking limits, since the various trees can differ significantly in shape. In all the sparse cases, our method outperforms aggregated multicast even though it does not introduce any traffic leaking.

- Aggregated multicast without leaking can not really achieve any noticeable reduction in the for-

warding state.

- The effectiveness of aggregated multicast increases with the number of groups placed in the network but this leads to an increase in the amount of leaking as can be seen in Figures 26 to 33. Indeed, in cases where the increase is more pronounced this happens at the expense of a rather large increase in the amount of leaked state. Our method does not improve as fast with the number of groups since there are less leak free aggregation opportunities as the number of groups grows.

- In Figures 34 to 37 we show the aggregation performance versus traffic leaking trade-off that can be achieved by the aggregated multicast method and we compare it with the performance of our method. In the x-axis we show the amount of state required and at the y-axis we show the corresponding traffic leaking. Our method does not introduce any leaking but for clarity we show it as a vertical line (instead of a point on the x-axis). We see that in most cases our method achieves state aggregation performance that aggregated multicast can only achieve through significant traffic leaking. In the cases of mixed group, aggregated multicast needs leaking of the order of 40% to 60% and and even more for the sparse VPN applications. Only for dense groups aggregated multicast can outperform our method with a reasonable amount of traffic leaking which is still around 20%.

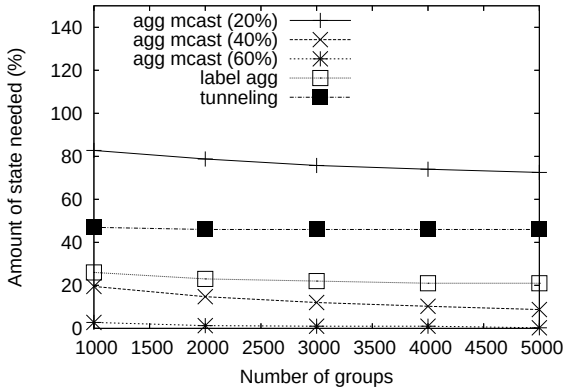
- Our method always outperforms tunneling as can be seen in Figures 10 to 17.

- Load balancing does not have an important effect on the relative performance of the different schemes (thus we only show results with load balancing enabled, as would be the case in a real network).

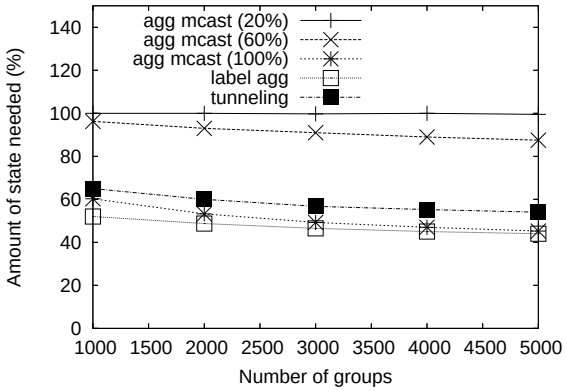
- Different random topologies of the same type do not affect the results (thus we show results only from one representative topology for each type).

- The relative performance of the various schemes does not seem to depend much on the network topology, while the density of the groups makes a more pronounced difference.

- When using MPLS based forwarding state aggregation there is a fairly large increase in the number of signaling messages required as can be seen in Figures 18 to 25. We also notice in these Figures that the increase in the messages is independent of the number of P2MP tunnels established throughout the network. The increase in the number of signaling messages is expected since when MPLS label based aggregation is used the signaling messages need to be propagated all the way to the source. Note, that this increase exists only because here we assumed that the basic signaling protocol will not propagate JOIN and LEAVE messages after they arrive at a node that already belongs to the group. This may not necessarily be the case. In an application with receivers with heterogeneous QoS, the join or leave of a receiver may result in modification of the reservations for the P2MP LSP all the way to the source, in this case the signaling messages will have to be propagated up to the source so our scheme will not incur any additional overhead.

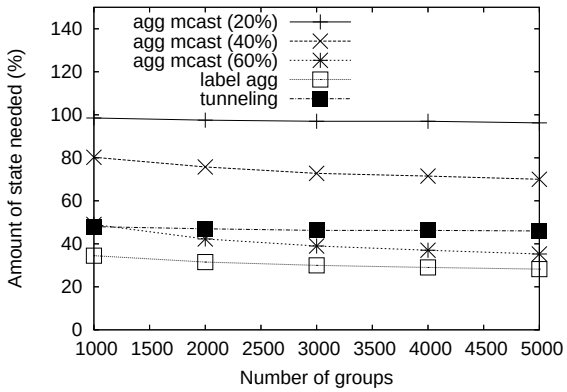


Barabasi, dense groups, media

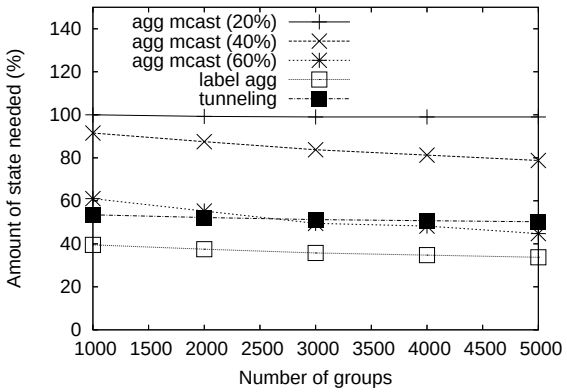


Barabasi, sparse groups, VPN

Figure 10: State reduction for Barabasi topology

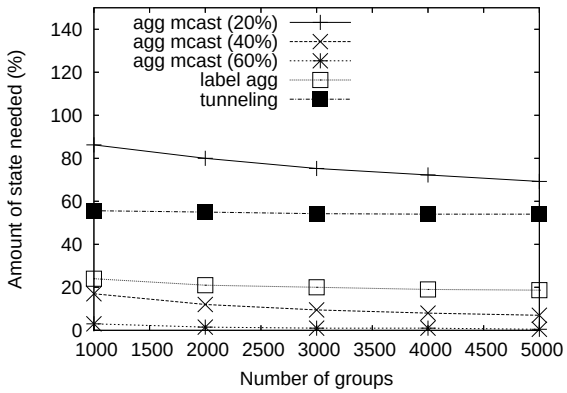


Barabasi, mixed groups, media

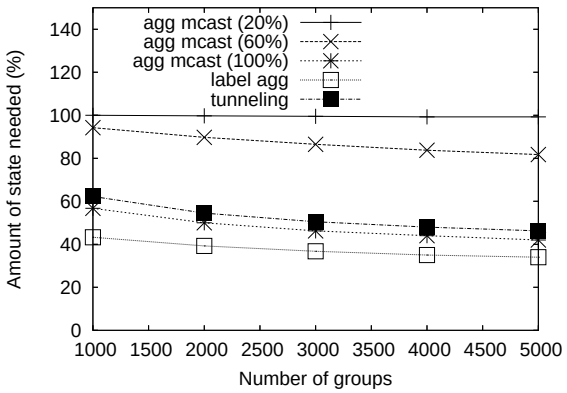


Barabasi, mixed groups, VPN

Figure 11: State reduction for Barabasi topology, mixed groups

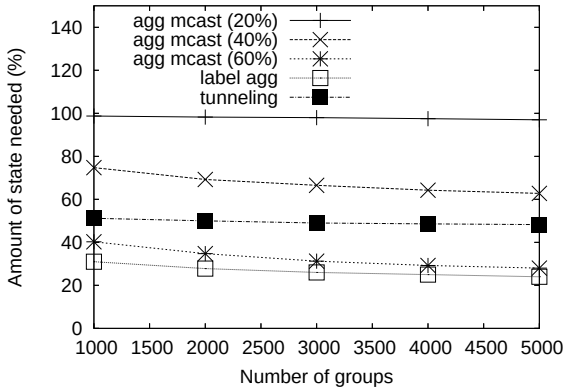


Waxman, dense groups, media

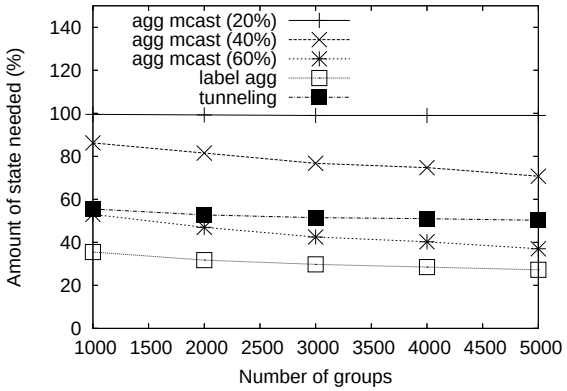


Waxman, sparse groups, VPN

Figure 12: State reduction for Waxman topology

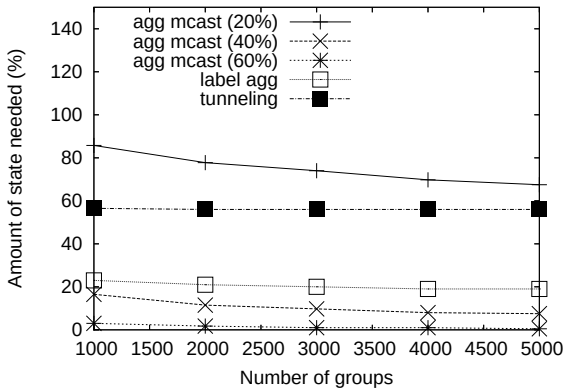


Waxman, mixed groups, media

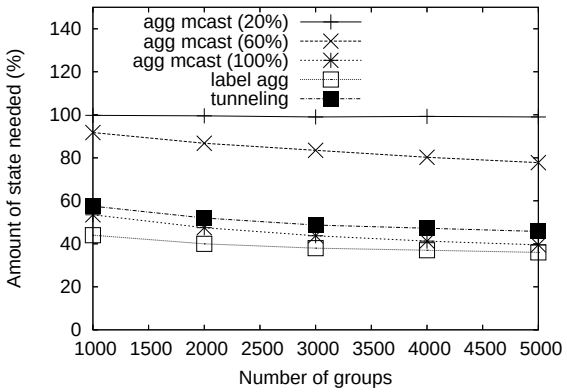


Waxman, mixed groups, VPN

Figure 13: State reduction for Waxman topology, mixed groups

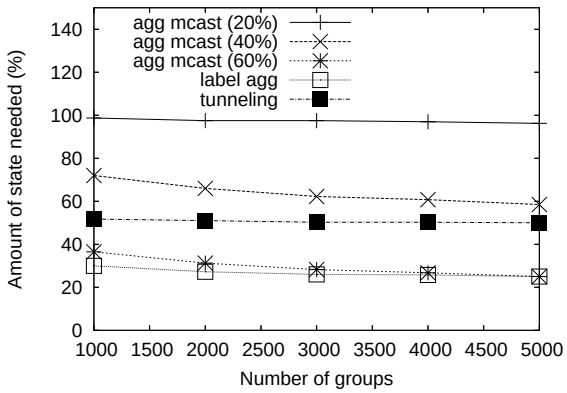


ISP, dense groups, media

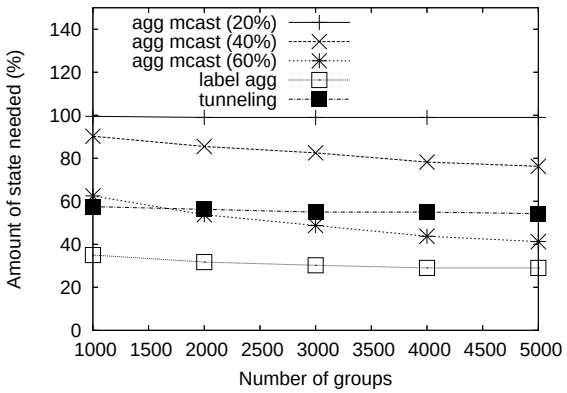


ISP, sparse groups, VPN

Figure 14: State reduction for ISP topology

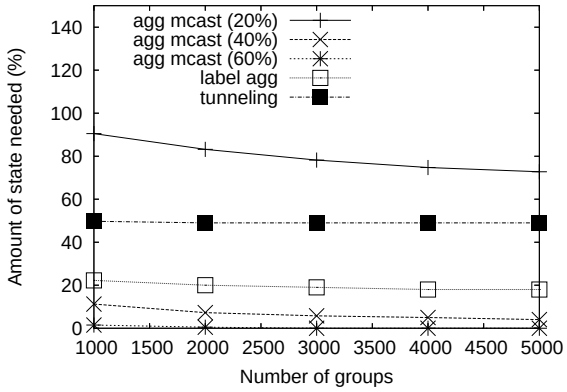


ISP, mixed groups, media

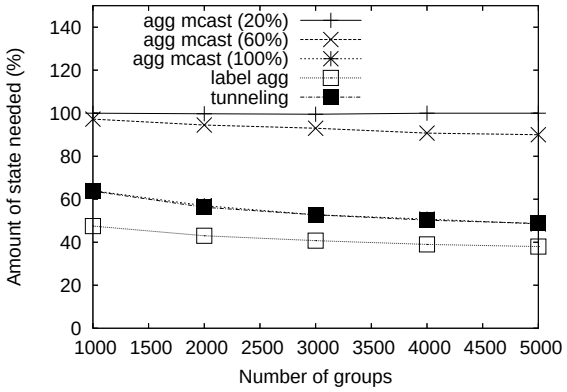


ISP, mixed groups, VPN

Figure 15: State reduction for ISP topology, mixed groups

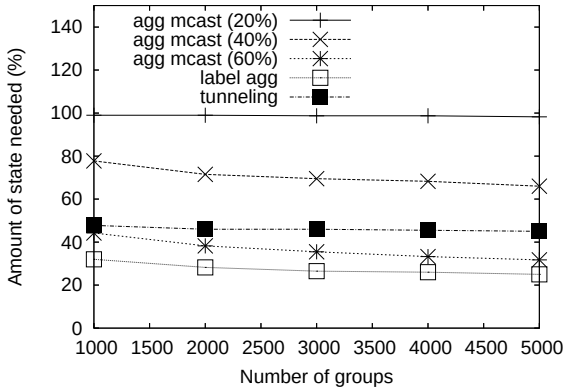


ISP2, dense groups, media

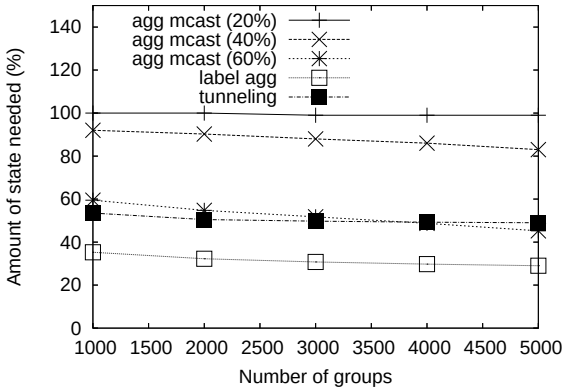


ISP2, sparse groups, VPN

Figure 16: State reduction for ISP2 topology

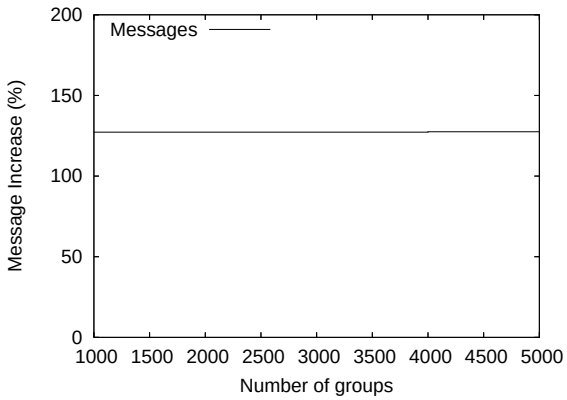


ISP2, mixed groups, media

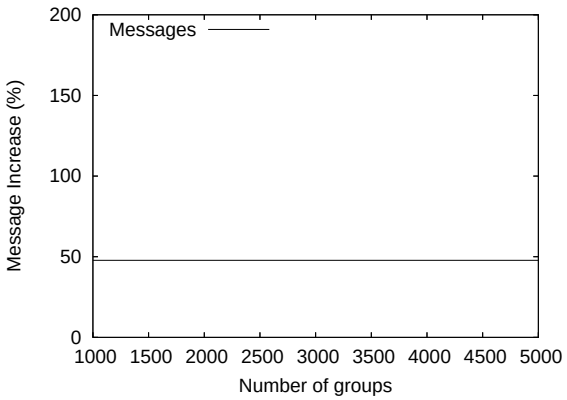


ISP2, mixed groups, VPN

Figure 17: State reduction for ISP2 topology, mixed groups

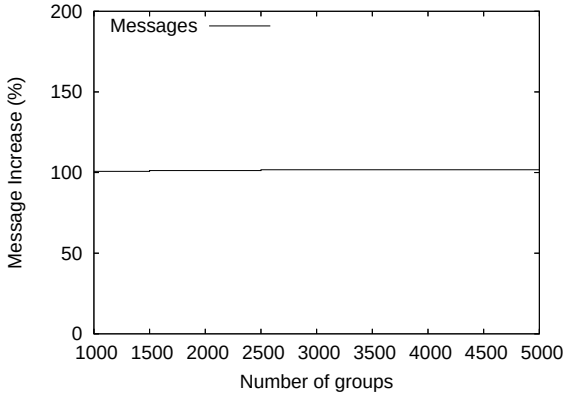


Barabasi, dense groups, media

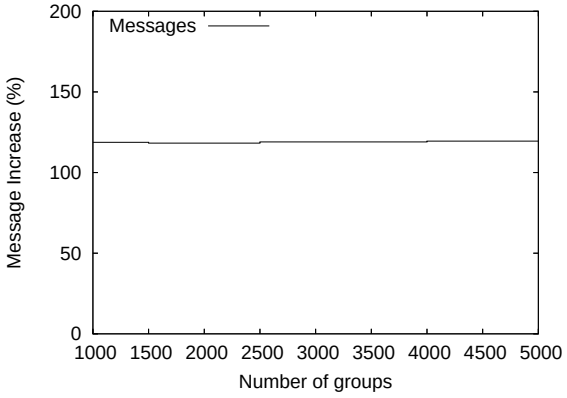


Barabasi, sparse groups, VPN

Figure 18: Signalling message increase for Barabasi topology

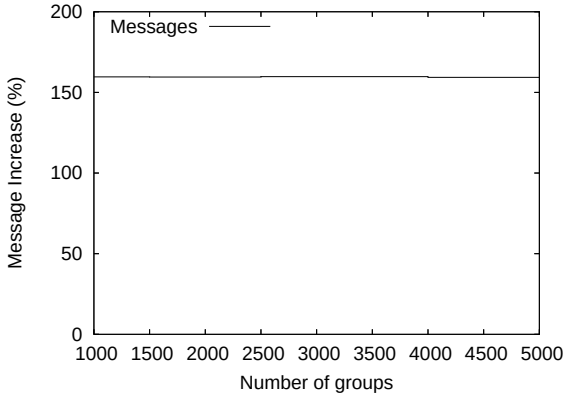


Barabasi, mix groups, media

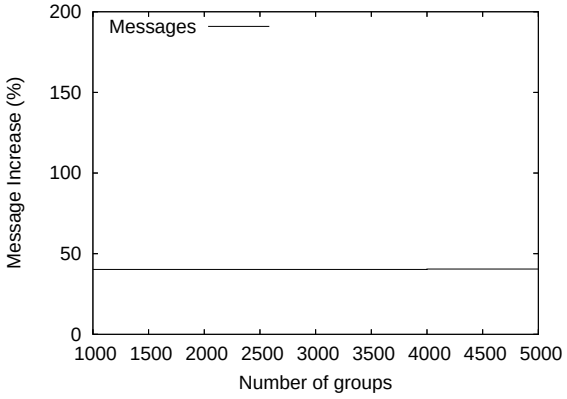


Barabasi, mix groups, VPN

Figure 19: Signalling message increase for Barabasi topology, mixed groups

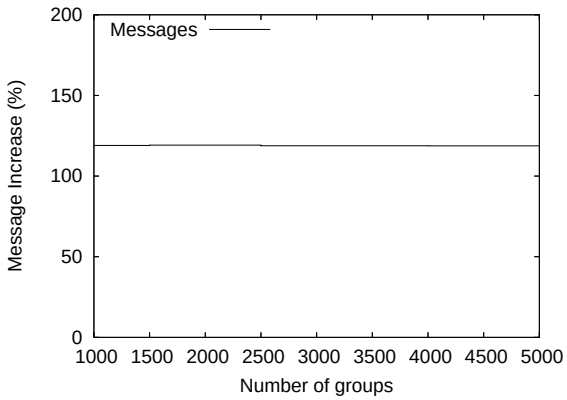


Waxman, dense groups, media

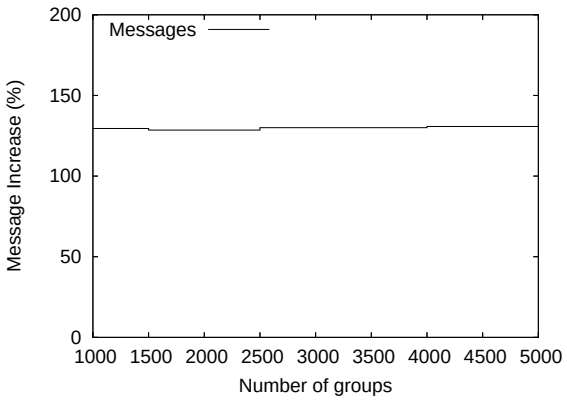


Waxman, sparse groups, VPN

Figure 20: Signalling message increase for Waxman topology

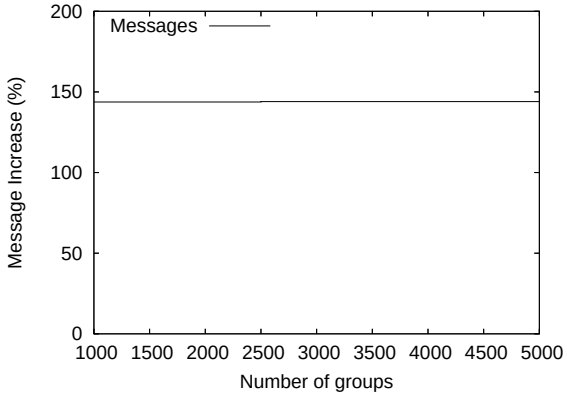


Waxman, mix groups, media

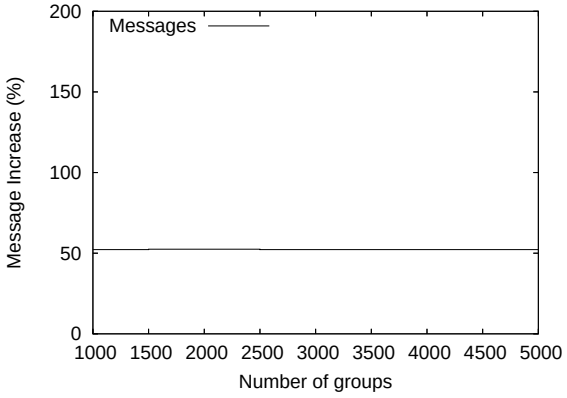


Waxman, mix groups, VPN

Figure 21: Signalling message increase for Waxman topology, mixed groups

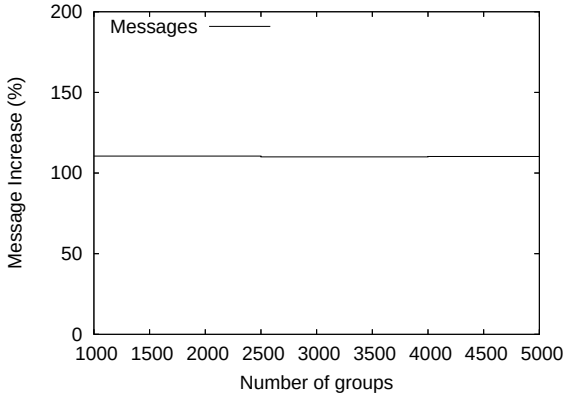


ISP, dense groups, media

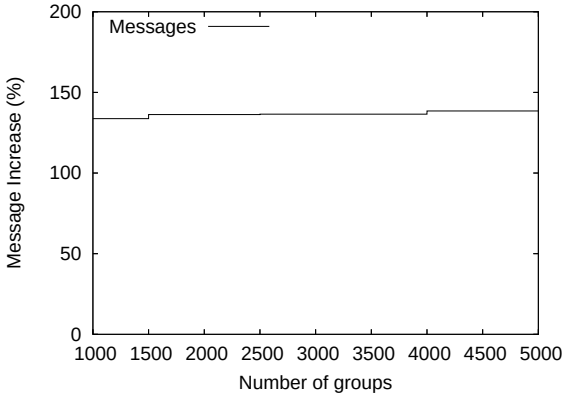


ISP, sparse groups, VPN

Figure 22: Signalling message increase for ISP topology

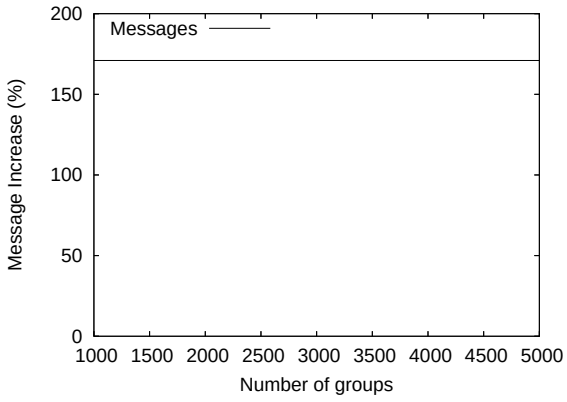


ISP, mix groups, media

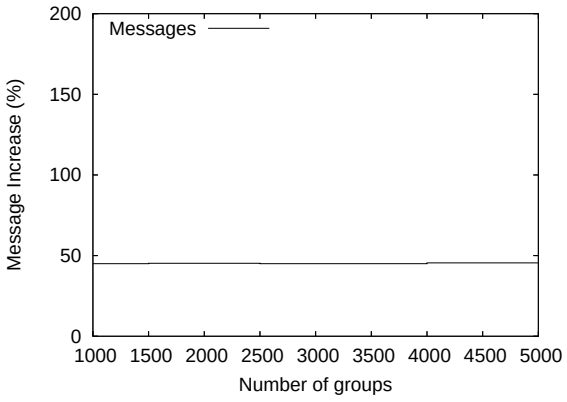


ISP, mix groups, VPN

Figure 23: Signalling message increase for ISP topology, mixed groups

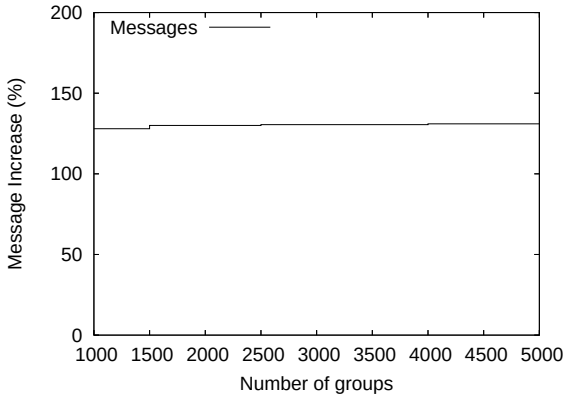


ISP2, dense groups, media

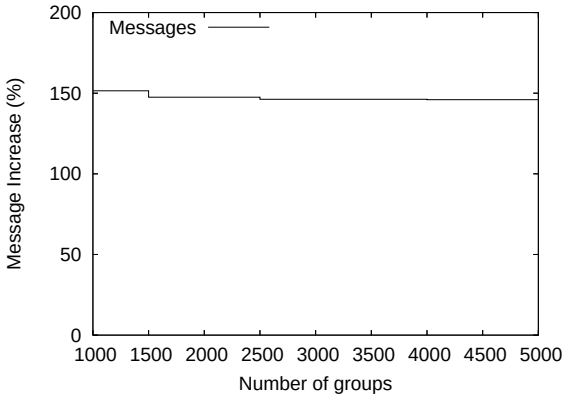


ISP2, sparse groups, VPN

Figure 24: Signalling message increase for ISP2 topology

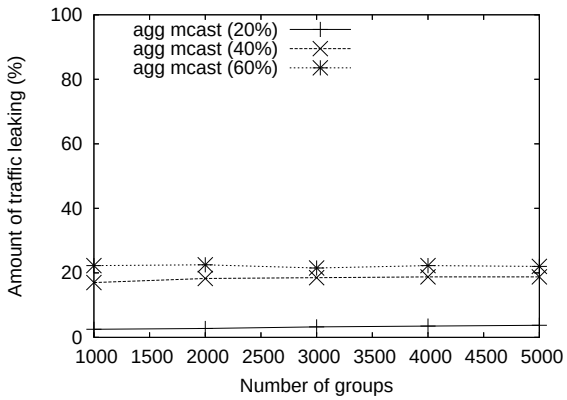


ISP2, mix groups, media

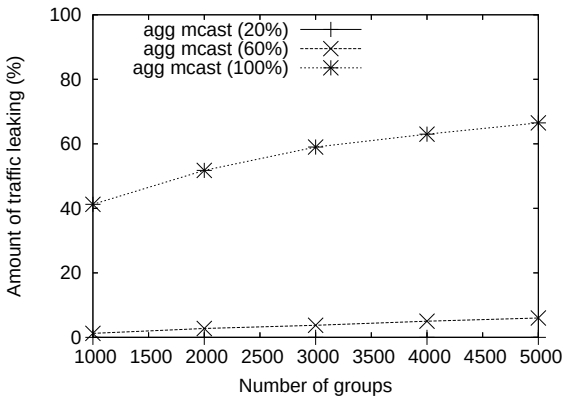


ISP2, mix groups, VPN

Figure 25: Signalling message increase for ISP2 topology, mixed groups

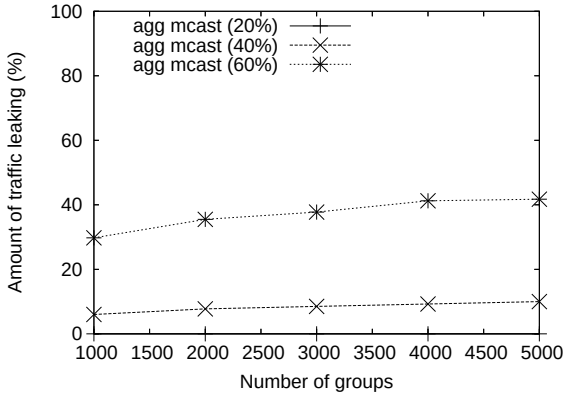


Barabasi, dense groups, media

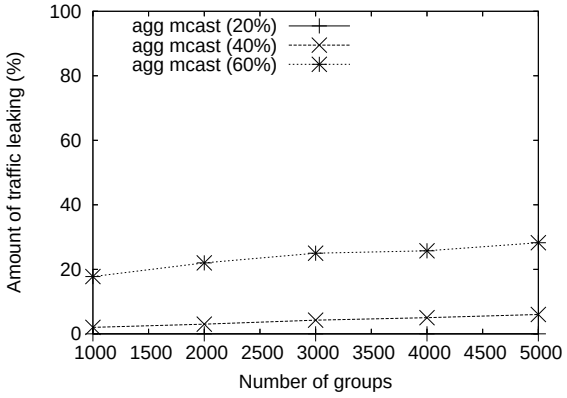


Barabasi, sparse groups, VPN

Figure 26: Traffic leaking for Barabasi topology

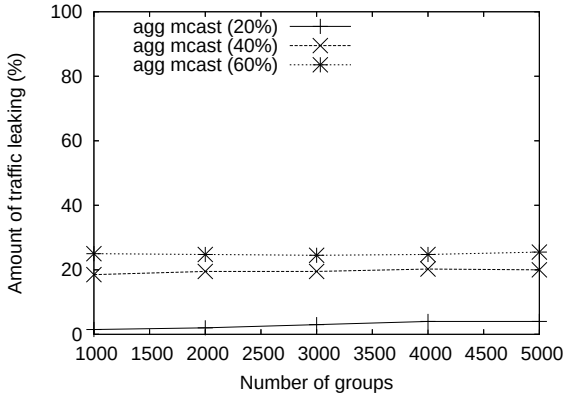


Barabasi, mix groups, media

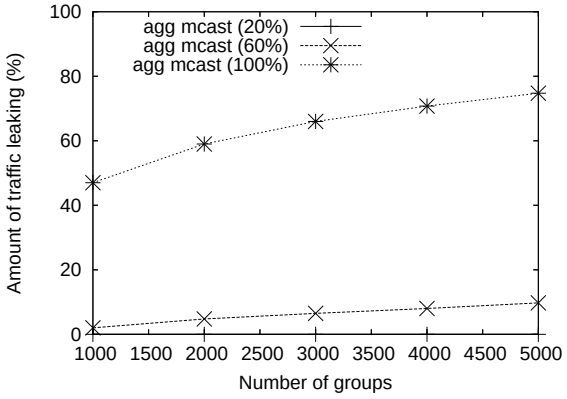


Barabasi, mix groups, VPN

Figure 27: Traffic leaking for Barabasi topology, mixed groups

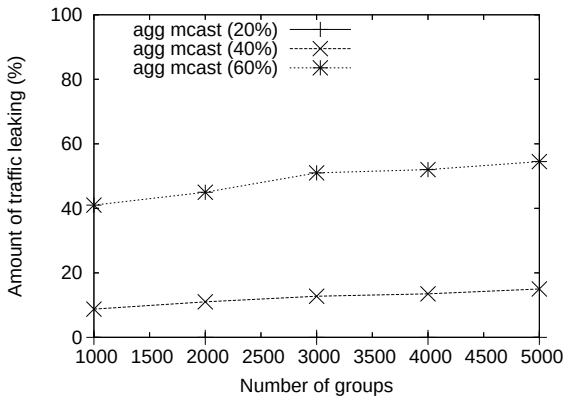


Waxman, dense groups, media

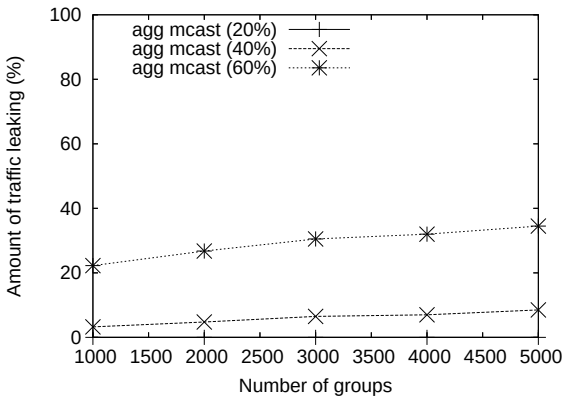


Waxman, sparse groups, VPN

Figure 28: Traffic leaking for Waxman topology

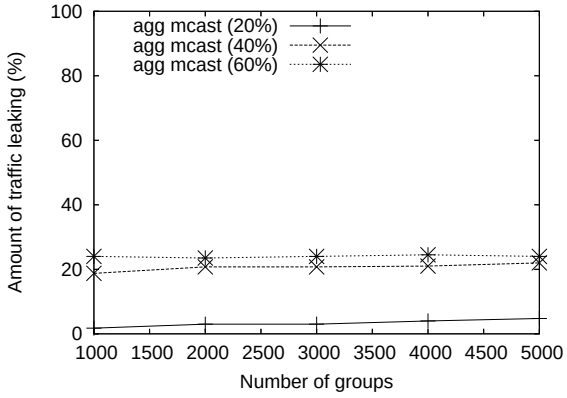


Waxman, mix groups, media

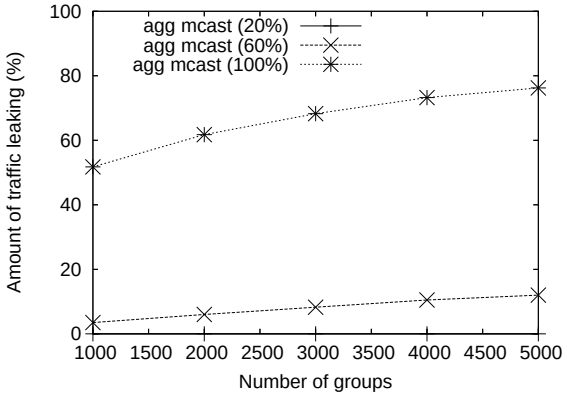


Waxman, mix groups, VPN

Figure 29: Traffic leaking for Waxman topology, mixed groups

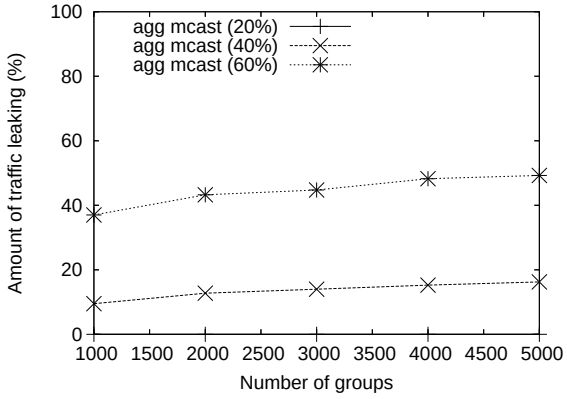


ISP, dense groups, media

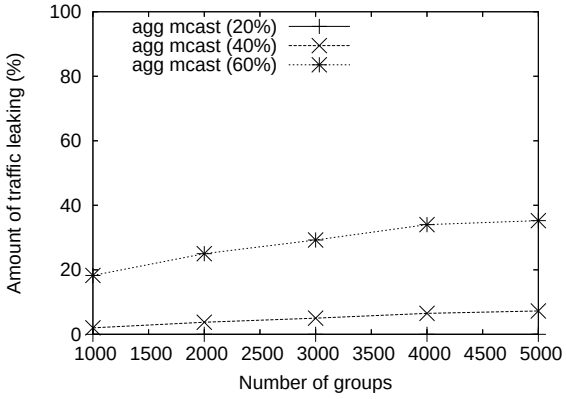


ISP, sparse groups, VPN

Figure 30: Traffic leaking for ISP topology

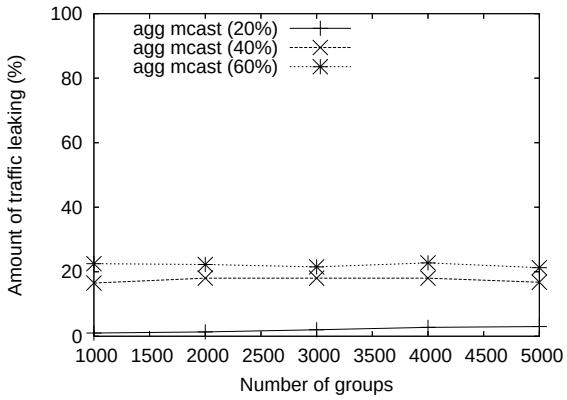


ISP, mix groups, media

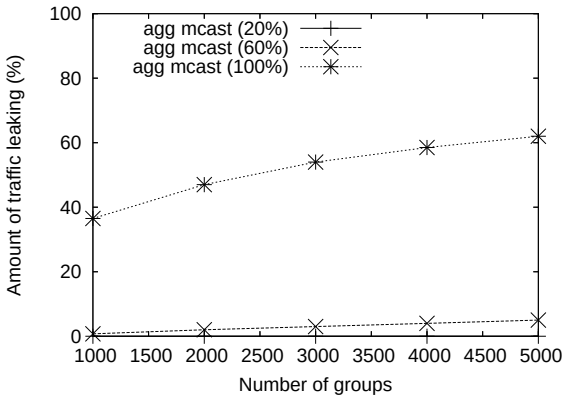


ISP, mix groups, VPN

Figure 31: Traffic leaking for ISP topology, mixed groups

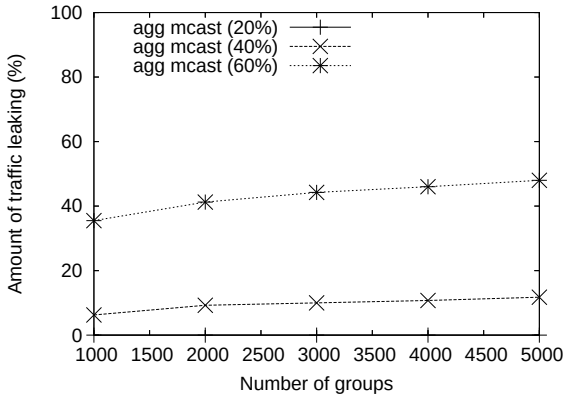


ISP2, dense groups, media

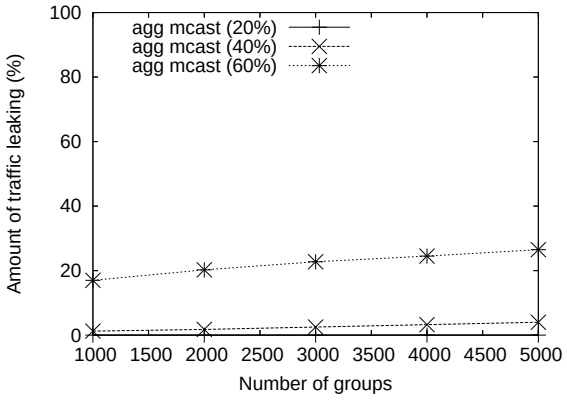


ISP2, sparse groups, VPN

Figure 32: Traffic leaking for ISP2 topology



ISP2, mix groups, media



ISP2, mix groups, VPN

Figure 33: Traffic leaking for ISP2 topology, mixed groups

5 Discussion

5.1 Comparison of the recursive and the end-end approach

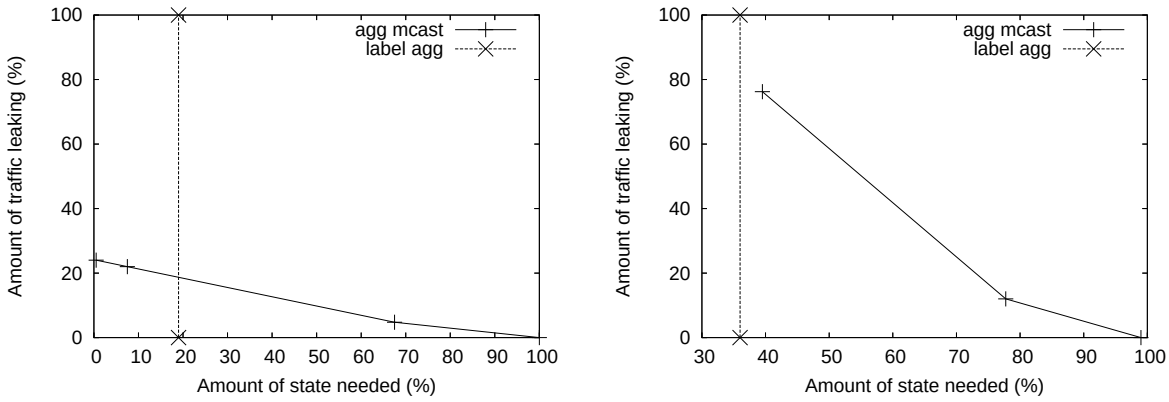
All approaches for reducing the forwarding state in multicast communication rely on discovering and exploiting commonality in the multicast trees. The tunneling approach discovers commonality in the form of long non-branching paths in the trees, the aggregated multicast finds whole tress that have similar shapes and our method finds common subtrees in trees. One could consider the aggregated multicast the "end-to-end" approach where all the complexity of the method lies at the edge of the network, while our method will require some processing in the internal nodes of the network. While one could argue against adding complexity in the core of the network we believe that by doing so we gain significantly in a number of areas. So, if we compare aggregated multicast with our label based scheme, we find that:

- Aggregated multicast necessarily has a centralized nature, in the sense that some controller entity needs to keep track of all the trees in the network and determine how new trees can be combined with the existing trees. The centralization of this operation could become an issue since the controller needs to constantly have a correct picture of the multicast trees that exist in the network and all operations that can modify the shape of a tree have to "pass through" the controller. Clearly, this centralization also prevents this scheme to be applicable to a wide area environment, i.e. the whole Internet. Of course, the dependence on MPLS inherently limits the scheme we discuss in this work to the confines of a single provider's backbone but it is not hard to imagine extending all schemes to the Internet in general by using a part of the IP packet header as a label (the IPv6 header already includes some room for a label). At this scale a centralized scheme could never work. Our method works hop-by-hop and is inherently distributed and scalable. Of course, one could propose to perform aggregated multicast in a distributed way by performing the aggregation decisions for subtrees of the multicast trees. In these cases though, it is not clear how one would control the leaking of the overall tree.

- Aggregated multicast can only aggregate trees that share the same source. While in general the concept of aggregating trees based on their shape is applicable to all types of trees, it is not quite clear what aggregation means in single source unidirectional trees, such as the P2MP trees we consider in this work. Indeed, merging two P2MP trees will not result in a P2MP tree unless the trees have the same source. This inherently limits the choices that aggregated multicast has when it determines how to best aggregate trees. In some of the scenarios we studied above, for example the VPN cases, the total number of trees in the network will be spread among most or all the nodes in the network. This will result in fewer choices for aggregation. On the other hand, our scheme while also handling only P2MP trees, has the ability to aggregate shared subtrees. Thus, if there are two P2MP trees with different sources but with a shared common part, our scheme will be able to avoid state reuse for the shared part. This is the reason why our scheme is able to achieve significant aggregation even in the sparse VPN cases that are difficult for aggregated multicast to handle.

- It is possible to deploy our scheme incrementally. Since the aggregation decision is local to each router and there is no extra protocol exchanges between routers, it is possible to implement label aggregation in only a subset of routers in the network. The amount of aggregation achievable in these cases will depend on the number of routers that implement our scheme and on how many groups are actually crossing these routers. Aggregated multicast on the other hand could be deployed incrementally, but it is not clear how hard it would be to do so. Since the tree aggregation decisions are taken in the centralized tree manager, routers do not need modifications. The tree manager will have to be notified when receivers leave or join groups and the paths that they use to do so, in order to discover the shape of the P2MP trees. If these notifications can be implemented without changes in the routers then this scheme can be deployed incrementally otherwise all routers will need to be modified for correct operation.

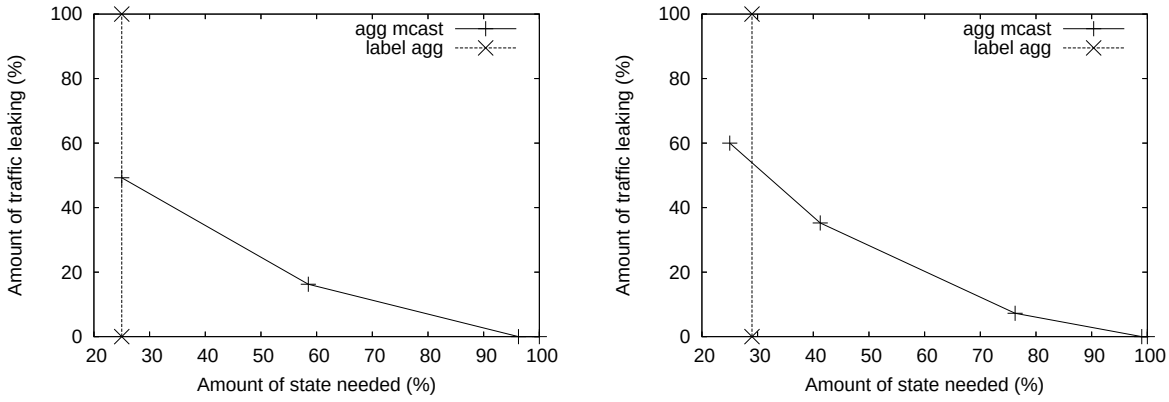
- Aggregated multicast inherently requires traffic leaking and has to balance the amount of leaking with the aggregation it can achieve. Thus it has to determine good tree matches under the leaking constraints. Most of the related algorithms



ISP, dense groups, media

ISP, sparse groups, VPN

Figure 34: State/performance trade-off for ISP topology



ISP, mixed groups, media

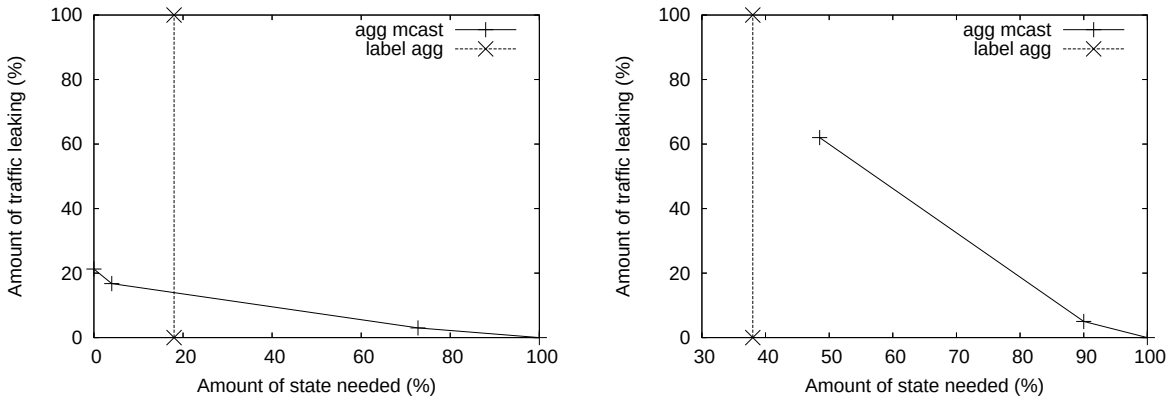
ISP, mixed groups, VPN

Figure 35: State/performance trade-off for ISP topology, mixed groups

have been shown to be NP-complete [42] and can only have a heuristic solution. Our approach is able to perform quite well without requiring leaking and avoids this problem alltogether.

- Initially, aggregated multicast may appear to be more general than our scheme since it can aggregate arbitrary trees, while our scheme seems to be most naturally applied to P2MP trees since it takes advantage of the signaling performed for joining each receiver to the multicast tree. This is not necessarily true though. Our approach can also be applied without any modification to core-based trees where all members send traffic to the core of the tree, which in turn sends it to the receivers. In these cases, receivers send JOINs to the core, so aggregation can happen for the part of the tree downstream from the core. Data from the sources could be tunneled to the core so that only a single label is required in the part of the tree between the source and the core. In bidirectional trees, labels will most probably be distributed through a multicast routing protocol and not through the join/leave mechanism assumed in this work, so the applicability of our method has to be investigated more.

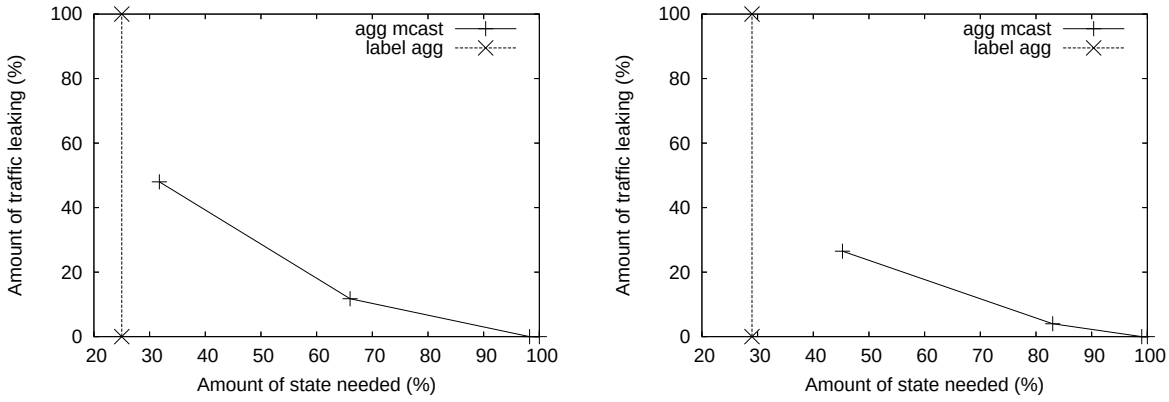
- Another advantage of the aggregated multicast approach is that the reduction in the overall number of trees is not limited to the savings in the amount of forwarding state but also allows for reduction of overheads in the control plane.



ISP2, dense groups, media

ISP2, sparse groups, VPN

Figure 36: State/performance trade-off for ISP2 topology



ISP2, mixed groups, media

ISP2, mixed groups, VPN

Figure 37: State/performance trade-off for ISP2 topology, mixed groups

Fewer trees will have to be established through the core so that there will be less keep-alive messages and control plane state. Still, there is nothing preventing our scheme to achieve similar reductions in the overheads of the control plane. It is not strictly necessary to maintain control state for each individual (unaggregated) group, instead it is possible to maintain state for each of the labels allocated. This allows to exchange state refresh and keep-alive messages that are proportional to the amount of forwarding state allocated and not to the number of groups. The details of such a control plane are beyond the scope of this work though.

5.2 Additional issues

Here we present a brief discussion of some of the finer points in our proposed scheme.

Aggregation efficiency at the source We should note that the aggregation of state becomes less efficient as we get closer to the source. Since each label represents a unique tree, necessarily the source of multiple P2MP tunnels will have to know at least as many labels as the P2MP tunnels. In this respect, the source will have to maintain state equivalent to the state needed by a scheme that aggregates tunnels with the same shape. The difference with the latter would be that using MPLS the forwarding state in the nodes further down the trees is significantly reduced. Thus the overall state requirements in the

backbone network are reduced.

Signaling between receivers and the source Some form of communication will be required between the receivers and the source. When a receiver wants to join a multicast group, somehow it needs to determine which is the source for the group and what is the session and group identifiers it should use to join the P2MP LSP for this group. Ideally, we would like the multicast MPLS operation and the aggregation to be transparent to the CE routers and the application server that is connected to the source PE. These nodes could operate using normal IP multicast and join and leave IP multicast groups. End node will discover which groups to join in order to receive a particular TV program by talking to the TV distribution application server. The PEs at the edge of the backbone network will have to intercept these IP multicast JOIN/LEAVE operations and convert them to P2MP LSP JOIN/LEAVE operations. In order to intercept the IP JOIN/LEAVE messages a multicast routing protocol should execute between the PE and CE routers. For converting the IP multicast operations to P2MP LSP operations, the PEs need to able to find which P2MP LSP (identified by its source, session and group identifiers) corresponds to each IP multicast group. This is a problem faced by other proposals that change the nature of the multicast destination address. Many schemes, e.g, [9, 24], replace the destination address with the pair (S,D) where S is the IP address of the source and D is the multicast destination address. In these schemes, the conversion of the IP multicast address to the new address is trivial. Others, like [8], do not use the IP multicast address as part of the new address but instead they use a port number allocated by the source, but they do not discuss how the different nodes agree on this mapping. In any case, there is a variety of options for exchanging this information between the PE routers. In VPN applications, PE routers typically exchange information between them. In L3VPNs, for example, BGP is used to exchange routing information between PEs. In other applications like VPLS, both BGP and LDP have been proposed for this role. The mapping between an IP multicast address and the P2MP identifier that corresponds to this group can be communicated by the source to the receivers using these mechanisms.

Extra signaling traffic A potential problem with our proposed approach is that each time a receiver joins or leaves a P2MP LSP, signaling messages will need to be sent all the way to the source of the LSP. This is in contrast to the usual signaling behavior where the messages for receiver JOINs and LEAVEs need only propagate until a node that already has state for the group. In an application like TV program distribution this should not be a problem since we expect very dense and fairly static groups since the receivers are PE routers serving a large number of CE routers. Still, in applications like multi-player games where a quite variable number of players may need to form transient groups with a single video game server that serves multiple games, the volume of signaling messages that the source has to process may become a problem.

Luckily, our scheme does not really require the new incoming label assignments to be advertised to the parents immediately. As we discussed in Section 4.2 delayed notification of the parent will result in delayed aggregation of state (since the deletion of state that can be aggregated is delayed) or temporary traffic leaks. The potential signaling overload from these messages could be prevented using techniques such as message throttling. Indeed throttling is proposed in [34] and could be applied to the label aggregation scheme proposed here. In our scheme, as in [34], delaying the sending of the JOIN messages upstream will result in a delay in the setup of the P2MP trees but it will not interfere with the label allocation. On the contrary, it may make it possible to aggregate multiple JOINs into a single message, another solution that has been proposed for reducing the volume of signaling traffic.

Alternatively, it is possible to devise a scheme where a node notifies its parents only after a significant amount of change in its aggregated state has occurred. The notification is performed through a new type of message that contains cumulative information for all the groups that need to change their incoming label. The processing of this message by the parent will be the same as what we described in Section 4. Appropriate triggering mechanisms can be devised so that we can control the trade-off between the volume of signaling traffic that the source has to process and the effects of the delayed notification.

Label allocation mode The proposed scheme requires the label allocation to be performed by the downstream node, and the proposed label be advertised to the upstream parent node. This is known as the downstream unsolicited mode [22]. It is also an ordered label control scheme since the upstream node can further advertise labels to its upstream neighbors after it has received labels from its own downstream

nodes. Our scheme could also work in the downstream on-demand mode, where the upstream node requests a label mapping from its downstream neighbor. As long as it is the downstream node that gets to choose the label to be used, we can achieve state aggregation. Also, the ordered operation is not really necessary. Independent control, where a node can send label assignment to its upstream nodes without any request it is also possible. In fact, if we were to implement the protocol we discussed above for sending label assignments asynchronously to the upstream neighbor, this would be the label control model we would be using.

Broadcast network considerations All schemes that perform downstream label allocation have problems when multiple nodes are connected with a broadcast network. In this case, to avoid sending multiple packets over the broadcast network, all the downstream nodes need to agree on a single label that can be used by the upstream node to send traffic to them. This may not be trivial since a particular label may be available in some of the downstream nodes but be used for some different group in others. Proposed ways around this problem involve running a simple protocol among all the nodes connected to the broadcast link, or partitioning the label ranges so that downstream routers can allocate labels that do not collide with labels allocated in other downstream routers connected to the same broadcast link. Of course, as it is argued in [28], in the backbone environments that we focus on in this work, there are really very few broadcast links, and typically routers are connected with point to point links, most commonly Packet over SONET or ATM. As a result, we do not expect this to become a serious issue.

Control plane aggregation The aggregation of the state in the forwarding plane can be combined with aggregation of the control state. Since multiple forwarding entries are collapsed together, it may be possible to avoid maintaining detailed control (i.e. RSVP) state per group. In protocols such as RSVP, the state maintained per group can be considerable, since this protocol requires timers for retransmission of messages and maintaining a lot of the properties of the LSP. The main problem with aggregating control state is how to identify the aggregated control state. Without any control state aggregation, the state for each group can be identified based on the information contained in each signaling message, i.e. the source, session and group identifiers. If we aggregate control state new identifiers should be allocated for the aggregated state, and the other nodes should be notified about these new identifiers so they start using them in their signaling messages. Similar to the local scope of the MPLS labels, these identifiers will have local scope (between a node and its parents) instead of the end-to-end scope of the commonly used LSP identifiers. While this is not impossible, it definitely requires careful study. We will revisit this issue in our future work.

QoS routing In this work we overlooked the issue of QoS. We assumed that all groups have the same QoS requirements and we did not make any assumptions on how the nodes that compute the paths take into account these QoS requirements as well as the state of the network. A very good discussion of the issues related with QoS and multicast tree construction can be found in [25]. In addition to the problem of how to keep track which groups use a particular link (since this is necessary to compute how many resources need to be reserved in a link), computing a QoS path from the receiver to the source of the tree may not follow the shortest path, and if added to an existing tree, the resulting topology may not be a tree any more. This problem will definitely affect our scheme, since we assume that we are working with trees, and every node has a single ancestor per group. [25] proposes some methods to avoid these non-tree topologies. As long as the QoS routing component can extend the existing tree without breaking its tree shape, our scheme can work without modifications. The fact that each receiver may have different QoS requirements does not really affect our scheme which is based on discovering commonalities in the shape of the trees only and ignores the amount of resources used on links. Since we avoid long term traffic leaks, we do not consider the case where a receiver has reservations for large or small volumes of traffic.

6 Conclusion and future work

In this report we demonstrated how with very simple modifications to the signaling protocol used to setup P2MP MPLS LSPs in MPLS backbones, it is possible to achieve significant reduction of the amount of labels that are needed to implement a large number of P2MP LSPs. This is accomplished without any traffic leakages and thus our solution does not suffer from the complexity needed for carefully balancing state requirement reduction and traffic leakage. Despite its simplicity, our method can achieve performance comparable and in many cases superior to other sig-

nificantly more complex methods such as aggregated multicast. In particular, our method performs better in cases where there are many sparse groups, a situation that can arise in many modern applications (VPNs, networked games). Also, since our method does not require global knowledge of the whole multicast tree it can be used in a wide-area context and it is not limited to the network of a single service provider.

In future work we plan to investigate if there are scenarios where leaky aggregation would be needed and how we can modify our scheme to achieve leaky aggregation in such a case. In addition, we plan to address data protection, which is a necessary feature for applications such as TV program distribution. We believe that we can extend our techniques and implement state efficient protection of the P2MP LSPs that will allow the traffic to flow uninterrupted to the receivers even when backbone network links or nodes fail. Furthermore, we will investigate if the label aggregation approach can be used for different types of trees, in particular shared trees (Core Based Trees for example). Finally, it is important to investigate how our scheme can be combined with other routing algorithms and in particular constrained based routing, or QoS routing. The additional constraints on path selection will impose different shapes on the P2MP (or shared) trees that will affect the effectiveness of our label based aggregation schemes. In our future work we plan to investigate these effects.

References

[1] J. Postel editor, Internet Protocol protocol specification, Request for Comments 791, September 1981

[2] S. Deering, R. Hinden, Internet Protocol version 6 (IPv6) protocol specification, Request for Comments 1883, December 1995

[3] E. Rosen et al., MPLS Label Stack Encoding, Request for Comments: 3032, January 2001

[4] M. Carugi, D. McDysan, Editors, Service requirements for Layer 3 Provider Provisioned Virtual Private Networks, *draft-ietf-ppvpn-requirements*, Internet draft, work in progress

[5] D. Ooms, B. Sales, W. Livens, A. Acharya, F. Griffoul, F. Ansari, Framework for IP Multicast in MPLS, *draft-ietf-mpls-multicast*, internet draft, work in progress

[6] D. Ooms, B. Sales, W. Livens, A. Acharya, F. Griffoul, F. Ansari, Overview of IP Multicast in a Multi-Protocol Label Switching (MPLS) Environment, Informational Request for Comments: 3353, August 2002

[7] J. Tian, G. Neufeld, Forwarding State Reduction for Sparse Mode Multicast Communication, *in Proceedings of the IEEE INFOCOM*, March 1998

[8] I. Stoica, T.S.E. Ng, H. Zhang, REUNITE: A Recursive Unicast Approach to Multicast, *in Proceedings of IEEE INFOCOM*, March 2000

[9] Luis H.M.K. Costa, S. Fdida, Otto C.M.B. Duarte, Hop By Hop Multicast Routing Protocol, *in Proceedings of ACM SIGCOMM*, August 2001

[10] P. I. Radoslavov, R. Govindan, D. Estrin, Exploiting the Bandwidth-Memory Tradeoff in Multicast State Aggregation, Technical report, USC Dept. of CS Technical Report 99-697 (Second Revision), July 1999.

[11] A. Fei, J. H. Cui, M. Gerla, M. Faloutsos, Aggregated Multicast: an Approach to Reduce Multicast State, *in Proceedings of Sixth Global Internet Symposium (GI2001) in conjunction with Globecom 2001*, November 2001

[12] J. H. Cui, A. Fei, M. Gerla, M. Faloutsos, An Architecture for Scalable QoS Multicast Provisioning, UCLA CSD Technical Report #010030, August 2001

[13] Y. S. Chu, S. Rao, H. Zhang, A case for end system multicast, *in Proceedings of ACM Sigmetrics*, June 2000

[14] P. Francis, Yoid: Extending the internet multicast architecture, Technical report, AT&T Center for Internet Research at ICSI (ACIRI), April 2000

[15] D. A. Helder, S. Jamin, End-host multicast communication using switch tree protocols, *in Proceedings of the Workship for Global and Peer to Peer Computing on Large Scale Distributed Systems (GP2PC)*, May 2002

[16] J. Janotti, et al., Overcast: Reliable multicasting with an overlay network, *in Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, October 2000

[17] D. Pendarakis, S. Shi, D. Verma, ALMI: An application level multicast infrastructure, *in Proceedings of 3rd Usenix Symposium on Internet Technologies and Systems (USITS)*, March 2001

[18] B. Yang, P. Mohapatra, Edge Router Multicasting with MPLS Traffic Engineering, *in Proceedings of IEEE International Conference on Networks (ICON)*, August 2002

[19] Rosen, E. et al., BGP/MPLS VPNs, *draft-ietf-ppvpn-r4fc2547bis*, internet draft, work in progress

[20] Knight, P. et al. Network based IP VPN Architecture Using Virtual Routers, internet draft, *draft-ietf-ppvpn-vpn-vr*, work in progress

[21] Awduche, D. et al., RSVP-TE: Extensions to RSVP for LSP Tunnels, RFC 3209, December 2001.

[22] L. Andersson et al., LDP Specification, Request for Comments 3036, January 2001

[23] http://www.cs.virginia.edu/ mngroup/projects/mpls/documents/thesis/

[24] H. Holbrook, D. Cheriton, IP Multicast Channels: Express support for Large scale single-source applications, *in Proceedings ACM SIGCOMM*, September 1999

[25] B. Rajagopalan, R. Nair, Multicast Routing with Resource Reservation, *Journal of High Speed Networks*, January 1998, Volume 7 Issue 2

[26] L. Berger, D. Gan, G. Swallow, P. Pan, RSVP Refresh Reduction Extensions, Request For Comments, RFC 2961, April 2001

[27] A. Medina, A. Lakhina, I. Matta, J. Byers, BRITE: An Approach to Universal Topology Generation, *in Proceedings of the 9th International Workshop on Modeling, Analysys and Simulation of Computer and Telecommunication Systems (MASCOTS)*,August 2001

[28] D. Thaler, M. Handley, On the aggregability of multicast forwarding state, *in Proceedings of IEEE INFOCOM*, March 2000

[29] M. Lasserre, V. Kompella Editors, Virtual Private LAN Services over MPLS, Internet Draft, work in progress

[30] K. Kompela, Y. Rekhter Editors, Virtual Private LAN Service, Internet Draft, work in progress

[31] S. Kent and R. Atkinson, Security Architecture for the Internet Protocol, Request for Comments, RFC2401, November 1998

[32] W. Townsley et. al, Layer Two Tunneling Protocol (L2TP), Request for Comments, RFC2661, August 1999

[33] D. Farinacci et. al, Generic Routing Encapsulation (GRE), Request for Comments, RFC 2784, March 2000

[34] R. Aggarwal, D. Papadimitriou, S. Yasukawa Editors, Extensions to RSVP-TE for Point to MultiPoint TE LSPs, internet draft, work in progress

[35] B. Waxman, Routing of Multipoint Connections, IEEE J. Selected Areas in Communications, December 1998

[36] A. L. Barabasi and R. Albert, Emergenece of Scaling in Random Networks, Science, 286:509-512, October 1999

[37] S. Bhattacharyya Editor, An Overview of Source Specific Multicast (SSM), Informational Request for Comments, RFC3569, July 2003

[38] J-H Cui et al, Measuring and Modelling the Group Membership in the Internet, in IMC 2003, October 2003

[39] A. Boudani and B. Cousin, A New Approach to Construcing Multicast Trees in MPLS Networks, *in Proceedings of ISCC 2003*, Italy

[40] Y. Oh et al., Scalable MPLS Multicast Using Label Aggregation in Internet Broadcasting Systems, *in Proceedings of ICT 2003*, March 2003

[41] O. Banimelheim et al., Deploying Multicast Communication over MPLS Networks Using Tree Numbering, *in Proceedings of ISCC 2005*, Cartagena, Spain

[42] L. Lao, J.-H. Cui, and M. Gerla, Tracking the Group Tree Matching Problem in Large Scale Group Communication, Technical Report, UCLA CSD TR040022, June 2004

[43] *Rocketfuel: An ISP Topology Mapping Engine*, http://www.cs.washington.edu/ research/networking/rocketfuel/, accessed on July 2005