Forwarding State Reduction for Sparse Mode Multicast Communication

by

Jining Tian

B. E., Tsinghua University, China

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

we accept this thesis as conforming
to the required standard

The University of British Columbia

August 1997

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

The University of British Columbia
2366 Main Mall
Vancouver, Canada
V6T 1Z4

Date: Oct 1, 1997

Abstract

Reducing forwarding state overhead of multicast routing protocols is an important issue towards a scalable global multicast solution. In this paper, we propose a new approach, Dynamic Tunnel Multicast, which utilizes dynamically established tunnels on unbranched links of a multicast distribution tree to eliminate unnecessary multicast forwarding states. Analysis and simulation results show promising reduction in the state overhead of sparse mode multicast routing protocols.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor, Dr. Gerald Neufeld, for his guidance, encouragement, and support during my thesis work. I would also like to thank Dr. Alan Wagner, who kindly accepted to be my second reader, and provided many valuable comments to this thesis. I'm also grateful to Dr. Norm Hutchinson and Dr. Son Vuong for their advises on various topics.

I'm particularly grateful to Mark McCutcheon, who provided many constructive comments throughout my research and writing of the thesis.

I would also like to thank folks in the Distributed Systems Group: David Finkelstein, Dwight Makaroff, Roland Mechler, Peter Smith, and Alistair Veitch for their generous help during the past two years.

JINING TIAN

The University of British Columbia

August 1997

Chapter 1

Introduction

Multicast service can deliver packets to a set of destinations identified by a multicast group, rather than a single destination. The IP multicast model [1], developed in 1988 by Stephen Deering, is an effort to provide multicast service over the Internet. In this model, neither the senders nor the receivers need to know the location of each other, and the membership can evolve dynamically. It is the responsibility of the multicast routing protocols to keep track of the membership information of a multicast group, and to establish multicast distribution trees to deliver packets from a sender to all the receivers. The multicast routing protocol is the center component of this model. In this chapter, we will first give out a brief overview of the IP multicast model.

1.1 IP multicast Model

The IP multicast model [1] was proposed in late 80's to support multi-point communication within a Wide Area Network(WAN). It did not gain the momentum until early 90's when multimedia conferencing over the Internet became possible. Before

that, only two types of delivery were supported on the Internet: unicast delivery which is used in traditional point-to-point communication, and limited broadcast delivery which is used to reach every node on a subnet.

Multicast, as a new type of delivery, is implemented as an extension to the Internet Protocol (IP) [2]. In this model, class D IP addresses(224.0.0.0 to 239.255.255.255) are allocated to multicast traffic. Each multicast session will occupy a class D multicast address. New socket Application Programming Interfaces(APIs) are designed to let the application join or leave a multicast group. The sender of a group can simply send packets to the class D address of the session, and the packets will be delivered, using the traditional best effort delivery mechanisms, to all the receivers that have joined the multicast group.

A multicast router is a router that supports multicast forwarding service, i.e. forwarding packets that arrived on one incoming interface to more than one outgoing interface. Multicast routers use multicast routing protocols to exchange membership information and to build distribution trees that connect each sender to all the receivers. One multicast router must be selected as the Designated Router for each subnet that have potential multicast senders or receivers. An auxiliary protocol, the Internet Group Management Protocol (IGMP) [3] is used for the end hosts to convey the group membership information to their Designated Routers. Two types of distributions trees can be built for a group: shared trees or source specific trees. A shared tree for a group has its root at a special center point, with all the members at its leaves. All the sources of a multicast group can use the same shared tree to deliver packets to the receivers. A source specific tree be built for each source and is used only to deliver traffic from that source.

In order to support the multicast forwarding service, each multicast router

must maintain a multicast forwarding table. The entries in the table are also referred to as multicast forwarding states. Each entry in the table has an incoming interface and one or more outgoing interfaces. When a packet arrives at a router, first the multicast forwarding table entry with matching group address (for shared trees) or with matching group address and source address (for source specific trees) is found, then the packet is forwarded onto all the outgoing interfaces stored in the entry. The multicast forwarding table entries on all the multicast routers collectively define the multicast distribution tree. The key differences between multicast forwarding and unicast forwarding are: in unicast the forwarding table lookup is based on the unicast address of the destination node and the packet is forwarded only onto one outgoing interface, while in multicast the lookup is based largely on the multicast group address and the packet may be forwarded to more than one outgoing interface.

1.2 Existing Multicast Routing Protocols

There are several multicast routing protocols currently available, namely Distance Vector Multicast Routing Protocol (DVMRP) [4], Multicast Extension to OSPF (MOSPF) [5], Core Based Trees (CBT) [6], Protocol Independent Multicasting Sparse Mode (PIM-SM) [7], and Protocol Independent Multicasting Dense Mode (PIM-DM) [8].

These multicast routing protocols can be classified into two categories: dense mode protocols and sparse mode protocols. Dense mode protocols such as DVMRP and PIM-DM are designed for the situation where group members are densely populated. On the other hand, sparse mode protocols, such as PIM-SM and CBT, are designed for the case in which group members are sparsely located. More detailed introductions of various multicast routing protocols are given in the next chapter.

1.3 Motivation and Problem Definition

When multicast service is to be provided globally, the scalability of multicast routing protocols becomes an important issue. The scalability of a protocol can be defined as its ability to maintain an acceptable performance level when some parameters of the network or application become very large. The scalability of a multicast routing protocol can be evaluated in two aspects: scalability with respect to the number of receivers and scalability with respect to the number of multicast groups. Although dense mode multicast routing protocols can handle a large number of receivers, all the existing multicast routing protocols will face scalability problems when the number of groups becomes very large.

The multicast forwarding table explosion is one of the major reasons that caused the scalability problem with the growth in the number of groups. According to the current multicast routing protocols, each multicast router has to maintain a multicast forwarding table entry for every group whose distribution tree passes through the router. When there are numerous groups, the forwarding table will be very large, which will directly lead to high router cost and low forwarding performance.

In unicast, clever hierarchical address assignment which reflects the geometrical proximity of the network nodes in their address prefixes can lead to significant reduction of the routing table size [9]. For example, if all the routers and hosts in US bear the same address prefix, then the routers in Canada will only need one forwarding table entry for all the destinations in US. However, in multicast there is no restrictions on the physical location of the host that can join a group. The group membership can also change dynamically. So one can not make any assumption about the locations of the receivers of a group, and hence forwarding table entries

for different multicast groups can not be aggregated.

In this thesis, we will provide a solution to reduce the size of multicast forwarding tables, and therefore improve the scalability of sparse mode multicast routing protocols.

1.4 Observation and Proposal

Most of the multicast groups are sparse when looked at on a global scope. A lot of the locally dense groups will become sparse in the backbone. This situation is not uncommon. In fact, we estimate that most of the medium or small scale conferencing groups will be very sparse in the backbone.

One observation we have is that, when the members of a group are sparsely located, the distribution tree of the group is likely to contain some long, unbranched paths. Routers on these paths are unnecessarily using the multicast forwarding mechanism to achieve an unicast forwarding function. We call the multicast forwarding state that has only one immediate downstream receiver "uni-multicast" forwarding state.

Based on this observation, we propose a new approach, namely the Dynamic Tunnel Multicast, as a general optimization of the existing sparse mode multicast routing protocols. Our approach can eliminate the uni-multicast forwarding states by using the dynamically established tunnels between the start and end points of those unbranched paths. After dynamic tunnels are established, usually only the root node, branching nodes and leave nodes of the original multicast distribution tree need to maintain state information about the group. The unbranched nodes are bypassed by the tunnel, and do not have to know about the group since the packets sent to the group are forwarded via unicast between tunnel end points.

The elimination of the uni-multicast states on the unbranched nodes can greatly reduce the overall forwarding state requirement and hence considerably improve the scalability of existing multicast routing protocols.

For example, two researchers at UBC, Vancouver and one researcher at ETH, Zurich want to have a video conference as shown in figure 1.1. There are 20 routers on the multicast distribution tree, but 17 of them (router $a, b, \cdots, q$) are uni-multicast routers. In the this example, a dynamic tunnel can be established

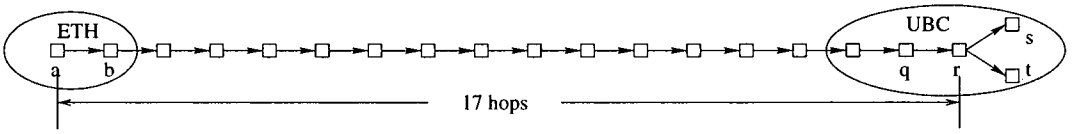


Figure 1.1: Dynamic Tunnel Example

between router a and r, bypassing 16 uni-multicast routers. Before the tunnel is established, all the 20 routers have to know about the multicast group, but after the tunnel is established, only four of them (router a, r, s and t) have to keep this information. Dynamic tunnels can thus lead to great savings on multicast forwarding states on the routers.

1.5 Thesis Contributions

My thesis is that, unicast IP forwarding will significantly reduce multicast state information and there by make large numbers of very sparse multicast groups feasible.

Following is a list of the contributions of this thesis:

1. Investigate the reasons that are causing scalability problems in current multicast routing protocols when the number of active groups in a network is

large.

2. Propose a Dynamic Tunnel Multicast model, which can greatly reduce the state information in multicast routers. It is compatible with both PIM-SM and CBT.

3. Present the protocol specification for PIM-Dynamic Tunnel(PIM-DT), the Dynamic Tunnel Multicast protocol with PIM-SM as the underlying multicast routing protocol.

4. Implement PIM-DT on LBNL network simulator and verified the state reduction through simulation results.

5. Discuss solutions for containing control packet overhead.

1.6 Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 introduces some related works on multicast routing, including DVMRP, PIM and CBT. Chapter 3 presents the basic concepts and the operational model of the dynamic tunnel multicast scheme. Chapter 4 contains the protocol specification for PIM-DT. Chapter 5 includes an analysis and simulation of the Dynamic Tunnel Multicast. Chapter 6 concludes the thesis and discusses the future work.

Chapter 2

Related Work

The Dynamic Tunnel Multicast that we propose is designed as a general optimization on top of some other existing multicast mechanisms. Before presenting our work, we first examine some of the existing multicast routing protocols. We start with a classification of the multicast routing protocols, and then discuss the basic mechanisms behind DVMRP [4], CBT [6] and PIM-SM [7].

2.1 Classification of Multicast Routing Protocols

A key role of a multicast routing protocol is to rendezvous the sender with the receivers by constructing a multicast distribution tree. Existing multicast routing protocols can be classified into two categories, dense mode protocols and sparse mode protocols, according to the way the distribution trees are constructed.

Dense mode protocols assume receivers or senders exist on each subnet, unless otherwise indicated. In order to rendezvous the senders with the receivers, either the data from each sender have to be flooded to all the possible receivers, or the membership information has to be flooded to all the possible senders. DVMRP and

PIM-DM [8] choose the former, and MOSPF [5] chooses the latter.

In DVMRP and PIM-DM, the data from each sender are periodically flooded to all the subnets, and a distribution tree rooted at the sender with leaves on each subnet is established during the flooding process. Subnets that do not have receivers later prune themselves off the distribution tree to improve the forwarding efficiency. In MOSPF, when a host joins a group, the membership information is flooded to all the routers in the network. A source specific distribution tree rooted at each sender can be constructed since each multicast router in the network knows the location of every receiver of a group.

Sparse mode routing protocols on the other hand, do not assume the membership of any subnet. A Rendezvous Point(RP) or Core is defined for each group to help the senders meet with the receivers. When a receiver joins a group, it talks to its Designated Router(DR) using IGMP, and the DR will send an explicit Join message towards the RP or Core of the group. A distribution tree rooted at the RP/Core can be established as the Join messages are forwarded. Senders will send data to the RP or Core of the group, which will further forward the data in the reverse direction along the paths that the Join messages have traversed.

2.2 Examples of Multicast Routing Protocols

In this section, we will look at three multicast routing protocols. The first one is DVMRP, which is the first and the most widely deployed multicast routing protocol in the world. We will explain in more detail some of the fundamental techniques, such as Reverse Path Forwarding(RPF), and the static tunnels that are used in the protocol. Then we will look at CBT and PIM-SM, which can be used as the underlying multicast support for our Dynamic Tunnel Multicast.

2.2.1 DVMRP

The Distance Vector Multicast Routing Protocol [4] is the earliest development in multicast routing. It is a dense mode protocol and uses flood and prune strategy to establish the distribution tree from the source to the receivers. When a receiver joins the group, the membership information is only propagated to its Designated Router via IGMP. The first packet from a sender is flooded to all the subnets in the same domain, and the distribution tree is built using a technique called Reverse Path Forwarding (RPF). During this process, when each router receives a packet, it will first check if the packet arrives on the interface that it uses to send packets in the reverse direction towards the source. If it is, the packet is forwarded to all the other interfaces. A multicast forwarding table entry is created for the group and the source. The incoming interface of the entry is just the interface on which the packet arrived, and the outgoing interface contains all the other interfaces. If a packet does not arrive on the "correct" interface, it is discarded. A message is also sent to the previous hop router of the discarded packet, telling that router to remove the interface on which this message arrives from its outgoing interface list in the multicast forwarding table entry. Designated Routers that do not have any member on the local subnet or downstream routers sends prune messages upstream to prune themselves off the distribution tree.

In DVMRP, only source specific trees are built, and the trees are uni-directional. This implies data can only flow from the root to the leaves. Each link in the network is configured with a cost and threshold value. Packets transmitted on the link will have the TTL field in the IP header of the packet decreased by the cost value of the link. A packet with remaining TTL value less than the threshold value of the link will not be forwarded onto the link. The threshold is used to control the scope that

a packet can reach.

Static tunnels can be established as a virtual link to interconnect multicast routers separated by non-multicast-capable routers. The topology map maintained by multicast routers can be different from the map maintained by unicast routers. Each tunnel is manually established, and configured with a cost and threshold. A tunnel can be used by all the groups. It is bi-directional, but different direction can have different cost and threshold values.

2.2.2 CBT

Core Based Tree is designed for sparse groups and it establishes shared trees only. A Core router is defined for each group. Each Designated Router with members on its local subnet sends a Join-Request messages towards the Core. Each CBT router forwards the Join-Request towards the Core and records a transient state if it does not have the forwarding state for the group. When the Join-Request reaches the first router that is already part of the corresponding distribution tree, a Join-Acknowledgment is sent hop by hop back to the requesting DR. Each CBT router will change the transient state for the group into a fixed multicast forwarding state as the Join-Acknowledgment traverse it.

In CBT, the tree branches are bidirectional. Data received on any valid interface of the multicast forwarding state will be forwarded onto all the other interfaces except the incoming one.

2.2.3 PIM-SM

PIM-SM is designed for sparse groups. It supports both shared trees and source specific trees. Distribution trees are uni-directional.

A Designated Router (DR) that has local member sends Join message towards the Rendezvous Point (RP) of the group, which is similar to the Core in CBT. A shared tree rooted at RP can thus be built. Sources of the group send data encapsulated directly to the RP, and the RP forwards the data further down the distribution tree to all the receivers. If the data rate for a source is high, the DR can switch to join the source specific tree of the source by sending Joins towards the source. Once the DR starts to receive data from the source specific tree, it can inform the shared tree not to deliver packets from that particular source to the DR to avoid duplicates.

Joins need to be sent upstream periodically in order to keep the forwarding state alive on the parent router. If a Join is not received on an outgoing interface for a certain period of time, then that outgoing interface is deleted from the outgoing interface list of the routing table entry. When sending the periodical Joins, multiple Joins for different groups/sources can be sent in a single packet.

Chapter 3

Tunnel Operations

In this chapter, we introduce the operational model of the Dynamic Tunnel Multicast. The major roles of the DTM include:

- establish and destroy dynamic tunnels on demand.
- maintain the tunnel states on the end points of tunnels.
- adjust the dynamic tunnels in case of membership changes.
- adjust the dynamic tunnels to cope with route changes and various failure conditions.

Since Dynamic Tunnel Multicast is designed to be an optimization of the existing sparse state multicast routing protocols, we will clarify the functions that are necessary to support Dynamic Tunnel Multicast.

3.1 Assumptions

We assume there exists some form of underlying multicast mechanism on all the routers that want to support dynamic tunnels. In this section we summarize our

assumptions about the underlying multicast support.

Although only Protocol Independent Multicast Sparse Mode(PIM-SM) and Core Based Tree(CBT) are considered in this document, the dynamic tunnel multicast can also work with other multicast routing protocols as long as the following assumptions hold.

- The distribution tree of a group can be either a shared center based tree or a source specific shortest path tree. Each distribution tree must have a root node. In PIM-SM, the root node is either the Rendezvous Point(RP) or the designated router of the source; in CBT, the root node is the Core router.

- We define upstream to be in the direction towards the root, and downstream to be in the direction away from the root(or towards the leaves). Each router participating in the routing protocol must have some mechanism to determine which is the upstream interface towards the root. In Reverse Path Forwarding(RPF), a commonly used multicast distribution tree construction method, the interface towards the root, as per unicast routing, is selected as the upstream interface. In the Multicast Extension to BGP(MBGP) [10], the upstream interface towards the root can be determined by explicit policies, and RPF is not used.

- Routers that want to join the group must explicitly send Join messages on the upstream interface towards the root. The message will be forwarded further upstream until a router already on the distribution tree is reached. A successful join operation can be confirmed by the fact that the downstream receiver receives data on the native tree(in PIM-SM), or from the fact that an explicit Join-Acknowledgment is received(in CBT).

- Processing of the Join messages at the routers installs multicast forwarding state on the routers. All the forwarding states on the routers collectively define the multicast distribution tree. The multicast forwarding state contains at least the group address, an upstream interface, and a list of downstream interfaces.

- Each interface in the forwarding state can be either uni-directional or bi-directional. In PIM-SM, the interfaces are uni-directional as the data can only be forwarded from the upstream interface onto the downstream interfaces. In PIM-SM, the upstream interface can be also referred to as the incoming interface, and downstream interface can be referred to as the outgoing interface. In CBT, the interfaces are bi-directional, and data can be forwarded from any interface to all the other interfaces except the incoming one. In CBT the terms "incoming interface" and "outgoing interface" make sense only when packets are being forwarded.

3.2 Concepts

Following are the basic concepts that are used in the definition of Dynamic Tunnel Multicast model.

Native Multicast Distribution Tree (Native Tree) We define the Native Multicast Distribution Tree, or simply the native tree, to be the distribution tree constructed by the underlying multicast routing protocol. The native tree may not exist in some part of the network after the dynamic tunnels are established. In this case, we use the term native tree to refer to the tree that would have been established by the multicast routing protocol, if there had

been no dynamic tunnels.

Uni-multicast Forwarding State A multicast forwarding state that has only one immediate downstream receiver and has no local member for a distribution tree is called an Uni-multicast Forwarding state for the distribution tree. A router that has uni-multicast forwarding state for a distribution tree is called an uni-multicast router on the distribution tree.

Potential Tunnel End Points Potential tunnel end points on a multicast distribution tree include branching nodes(non-uni-multicasting nodes), the root node, the leaf nodes, and all the nodes that cannot be bypassed by dynamic tunnels due to reasons specific to the underlying multicast routing protocol, or due to administrative concerns.

Dynamic Tunnel Dynamic tunnels can be established between adjacent potential tunnel end points to eliminate the uni-multicast forwarding states. Dynamic tunnels are different from the static tunnels in some existing multicast routing protocols (i.e. DVMRP). In the following discussion, the term *tunnel* always refers to *dynamic tunnel* unless otherwise specified. Following are some of the unique properties of dynamic tunnels:

- Dynamic tunnels are distribution tree specific. Each tunnel is created for a certain distribution tree, which can be either source specific or shared. Only packets for the corresponding distribution tree can be forwarded into the tunnel. When source specific distribution trees are used, only packets generated from a certain source can be forwarded into the tunnel.
- Each dynamic tunnel has two end points, an upstream tunnel end point and a downstream tunnel end point.

- Dynamic tunnels are created and destroyed on demand.
- Dynamic tunnels can be either uni-directional or bi-directional depending on the nature of the interface in the forwarding states of the underlying multicast routing protocol running on the tunnel end points. Dynamic tunnels are uni-directional in PIM-SM, and bi-directional in CBT.
- Dynamic tunnels do not affect route calculations. The establishment of tunnels does not change the topology map used in the underlying unicast or multicast routing protocol.
- Dynamic tunnels can have dynamically assigned costs and thresholds that can be derived from the underlying multicast or unicast routing protocol.
- A dynamic tunnel is not used as a new virtual interface, instead, it is treated as a new attribute associated with an existing interface. The associated interface of a dynamic tunnel is defined below.

Native Path Each dynamic tunnel has a corresponding native path, which is the path on the native tree between the tunnel end points. Routers on the native path are bypassed by the tunnels and may no longer keep state information for the group.

Associated Interface of a Dynamic Tunnel The upstream(or downstream) associated interface of a dynamic tunnel is the interface through which the native path of the tunnel is connected to the upstream(or downstream) tunnel end point. The tunnel is also said to be associated with that interface. For any distribution tree, there is at most one tunnel associated with any interface.

Interface States An interface in a multicast forwarding table entry can be in one of the following four states: Idle, Native, Tunnel or Dual. An interface in

Idle state may not be included in the forwarding table entry. An interface is in Native state if it is performing the normal multicast forwarding function. Data can be sent and received on Native state interfaces in native format. An interface is in Tunnel state if there is a dynamic tunnel associated with it. Data can be sent and received from the tunnel associated with the interface in encapsulated format. An interface in Dual state operates as if it is in both Native and Tunnel states. Data can be received in both native and encapsulated format for an interface in Dual state. Data will be sent twice for an interface in Dual state, once in native format, once in encapsulated format for the tunnel. The Dual state is introduced to minimize the interruption in data delivery caused by tunnel establishment, tear-down and adjustment.

Dynamic Tunnel Tree A Dynamic Tunnel Tree, or Tunnel Tree, is the distribution tree with some branches replaced by dynamic tunnels.

3.3 Optimization Goals

The primary goal of the dynamic tunnel approach is to reduce the uni-multicast state while at the same time keeping the tunnel tree topology as close as possible to the native tree. It is always assumed the native multicast distribution tree created by the underlying multicast routing protocol to be the optimum.

We are aware that dynamic tunnels also introduce data processing and control overheads. Containing these overheads are also part of our ultimate goals, but they are not considered primary goals in the operational model of the dynamic tunnels. Additional mechanisms can be devised later to achieve these goals.

3.4 Uni-multicast State Detection

Each of the routers on the distribution tree can detect the existence of uni-multicast forwarding states from the fact that the router has only one direct downstream receiver for a multicast group. If all the out-going links of a router are non-multi-access links, the uni-multicast state can be determined from the fact that the router has only one downstream interface in the multicast forwarding state.

It is slightly more difficult to determine the number of direct downstream receivers on a multi-access link since some multicast routing protocol such as PIM supports join suppression, which allows only one of the direct downstream receivers on the multi-access links to send Join messages. The problem does not exist in CBT since it does not support join suppression.

There are a number of ways to solve the problem. First, we can disallow tunnels to span across multi-access links. In this case, if the uni-multicast state exists on the link, it will never be deleted. Though this solution might potentially reduce the average length of tunnels, it is actually not a bad choice since its simple, and since most of the dynamic tunnels are established in the backbone where the multi-access links are rare. It has the additional advantage of not having to worry about Early Tunnel Terminations as discussed later in section 3.11. Second, we can disable the join suppression on the multi-access link. On multi-access links with join suppression disabled, the number of downstream receivers can be determined from the number of different downstream routers that are sending Join messages. This works best when the number of routers on the link is small. Third, we can modify the join suppression algorithm on multi-access links in order to determine whether a router is in uni-multicast state. Basically we allow at most two of the downstream receivers to send Join messages instead of at most one as in the original

join suppression algorithm. In this case, a uni-multicasting router will receive Join messages only from one downstream receiver, while a real multicasting(non uni-multicasting) router will receive from two.

3.5 Tunnel Tree Establishment

One way to establish a tunnel is to start from the potential downstream tunnel end points; that is, either from the leaf nodes or from the branching nodes on the native distribution tree. Once the multicast distribution tree becomes stable, these potential downstream tunnel end points start sending Tunnel Request messages upstream towards the root. The Request messages are sent on the same interface as the Join messages are sent in the underlying multicast routing protocol. The Request message includes the multicast group, the downstream tunnel end point, the original TTL value used in the IP header when the packet is sent, a cost and a threshold value which indicate the cost and threshold of the path through which the Request message has traversed.

Each router on the distribution tree that receives the message first checks if the Request can be further forwarded. If the router is not a potential tunnel end point, then it tries to forward the Request message further upstream. The cost and threshold values in the outgoing Request message are updated.

If the router that receives the Request message is a potential tunnel end point, then the request is not forwarded any further. The router will check if a tunnel can be established. The router can impose a lower limit on tunnel lengths to contain the maintenance overheads of the tunnels. The router can derive the length of the native path from the original TTL value in the message and the current TTL value in the IP header. If the length can not meet the minimum length requirement,

the router that receives the request can optionally send a tunnel reject message back to the requesting router indicating the reason. Otherwise, the router becomes the upstream tunnel end point. It sends a tunnel Setup message back to the downstream tunnel end point and the tunnel state is recorded.

The newly created tunnel is associated with the interface on which the Request message arrives. The interface is set to Dual state. The interface changes to Tunnel state if a Prune/Quit message is received. The interface reverts to Native state when the associated tunnel is torn down.

When the tunnel is established, data packets that arrive at the upstream tunnel end point are forwarded onto all the outgoing interfaces according to the forwarding rule of the underlying multicast routing protocol. If an outgoing interface is in Native state, packets are forwarded in native format as usual. If an outgoing interface is in Tunnel state, packets are encapsulated and sent unicast directly to the other end point of the associated tunnel. If an outgoing interface is in Dual state, the packets are sent twice on the interface, once in native format and once encapsulated.

If the tunnels are uni-directional, data packets can only come from the upstream tunnel and be forwarded into downstream tunnels. If the tunnels are bi-directional, data coming from a tunnel will be forwarded into all the other tunnels except the incoming one.

When the potential downstream tunnel end point receives the tunnel Setup message, it first checks if the interface towards the upstream tunnel end point is the same as the interface towards the root. If RPF is used to construct the multicast distribution tree, unicast routing can be used to determine the interface towards the root and the interface towards the upstream tunnel end point. If the two interfaces

are not the same, the tunnel Setup message is discarded. Although passing this check will not guarantee that the upstream tunnel end point is on the path towards the root, it can reduce the chance of forming routing loops. If the two interfaces are the same, the tunnel state is recorded and the tunnel is established. The associated interface of the tunnel is set to be in Dual state. Join messages are no longer sent towards the root by the downstream tunnel end point. Instead, tunnel Request messages are sent periodically towards the root to refresh the tunnel state on the upstream tunnel end point.

In PIM-SM, since the intermediate uni-multicast routers are bypassed by the tunnel and no longer receive Join messages, their unnecessary multicast forwarding state will eventually timeout and be deleted. The downstream tunnel end point can send Prune message upstream to speed up the process. In CBT, explicit Quit messages are sent upstream to remove the uni-multicast states.

When the upstream tunnel end point stops receiving Join messages or receives Prune/Quit messages on the interface associated with the tunnel, the interface is set to be in Tunnel state. The Prune or Quit messages are not forwarded further.

Figure 3.1 illustrates the tunnel tree establishment procedure with PIM-SM as the underlying multicast routing protocol. Here A, B, C are three routers on a distribution tree. A and C are branching points that have more than one immediate downstream receivers, B is a uni-multicasting router that has only one immediate downstream receiver. C sends tunnel Request messages upstream, and B forwards them on. When A receives the tunnel Request message, it replies with a tunnel Setup message and sets the interface on which the request message arrives to be in Dual state. This will cause the subsequent data in the downstream direction to be forwarded twice for that interface: once in Native state, once in encapsulated

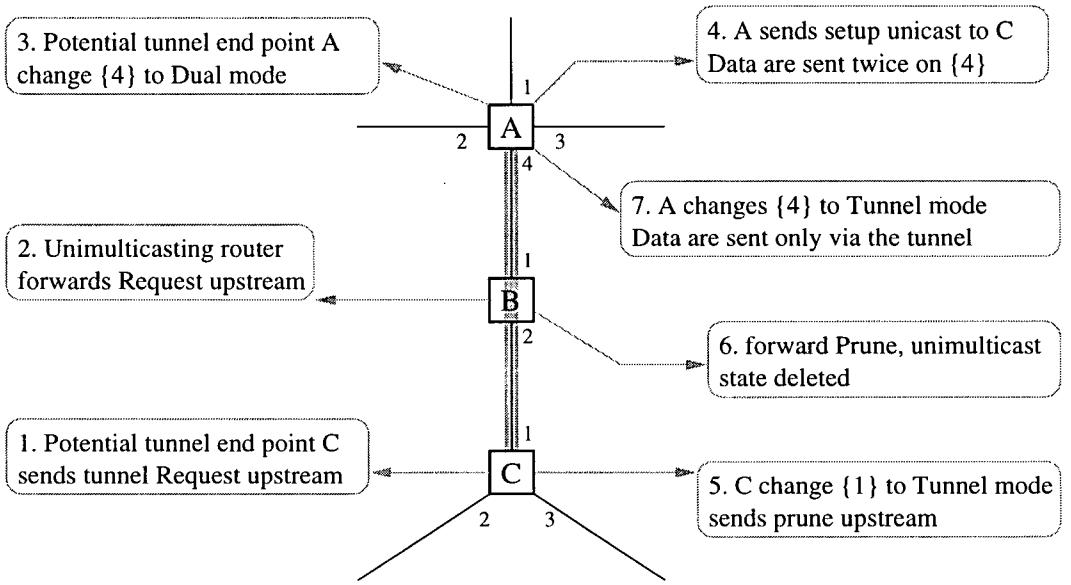


Figure 3.1: Tunnel Establishment

state. When C receives the tunnel Setup message, it sets the interface on which the tunnel Request messages are sent to be in Tunnel state, and sends a Prune message upstream. The multicast forwarding state on B will be deleted as the Prune message propagates along the way. When A receives the Prune message, the interface on which the Prune message is received is set to be in Tunnel state, and subsequent data in the downstream direction are forwarded only once through the tunnel on that interface.

3.6 Tunnel Encapsulation

Several encapsulation techniques can be used when sending data in the dynamic tunnels, namely IP in IP Tunneling [11], Generic Routing Encapsulation(GRE) [12], and Minimal Encapsulation within IP [13]. The IP in IP Tunneling is a strait forward

encapsulation technique. It wraps the original IP packet directly in another standard IP header. GRE is a general purpose solution which can be used to encapsulate any type of network layer packet in any other type of network layer packet. The Minimal Encapsulation within IP minimizes the encapsulation overhead by compressing the inner IP header. Since the dynamic tunnels are expected to be used extensively, it is important to reduce the encapsulation overhead. Under this consideration, the Minimal Encapsulation within IP is selected as the default encapsulation mechanism for Dynamic Tunnel Multicast.

3.7 Tunnel State Maintenance

Dynamic Tunnels use soft tunnel state. A downstream tunnel end point periodically sends tunnel Request messages to the upstream tunnel end point in order to keep the tunnel state alive. The upstream tunnel end point timeouts and deletes the tunnel state if no more tunnel Request messages are received within a certain timeout period.

The periodic Request messages can also be used to detect route changes. The message will simply be forwarded if the router that receives it has no forwarding state for the group, or if the router is uni-multicasting and the message arrives at the downstream interface of the multicast forwarding state. A tunnel Request message arriving at a router under other conditions is an indication of route change or membership change which usually will trigger tunnel adjustments. Those conditions will be discussed later in section 3.13.

If there is no route change or membership change, the Request message arrives at the upstream tunnel end point and the tunnel state is refreshed. The message is not forwarded any further by the upstream tunnel end point.

3.8 Tunnel Tear Down

When a downstream tunnel end point no longer has downstream receivers, it can discard the tunnel by not sending tunnel Request messages to the upstream tunnel end point. The tunnel state information at the upstream tunnel end point will eventually expire and be deleted. The downstream tunnel end point can speed up the tear down process by sending an explicit tunnel Destroy message to the upstream tunnel end point.

3.9 Tunnel Splice

After a member leaves the group, the upstream end of an existing tunnel, which was previously a branching node in the tunnel tree, now may have only one downstream interface left in the multicast forwarding state, and becomes an uni-multicasting router on the distribution tree of the group. In this case the upstream tunnel and the downstream tunnel of the former branching point can be spliced. The router at the splice point that connects the upstream tunnel and the downstream tunnel stops sending tunnel Request messages upstream since it is no longer a potential tunnel end point. When it receives the tunnel Request messages from the downstream tunnel, the router at the splice point no longer sends back tunnel Setup messages. Instead, it appends a tunnel Destroy message at the end of the received tunnel Request message, and forwards the new message upstream. When the upstream end point of the upstream tunnel receives the tunnel Request/Destroy message, it destroys the old tunnel to the splice point, and sends a tunnel Setup message back to the requesting router to establish a new tunnel.

Figure 3.2 illustrates this procedure. Originally there were three tunnels

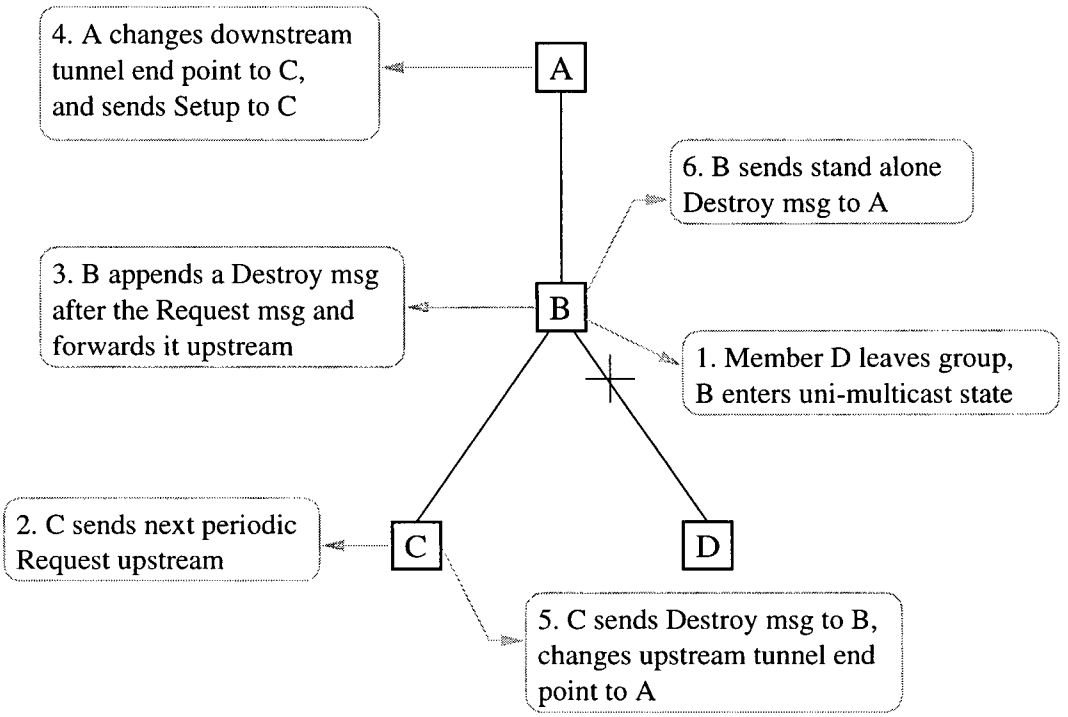


Figure 3.2: Tunnel Splice

established, A-B, B-C, and B-D. Now the B-D tunnel is destroyed, and tunnel A-B and B-C can be spliced. When router B at the splice point receives the tunnel Request message from C, it appends a tunnel Destroy message in the received tunnel Request message indicating the old A-B tunnel can be replaced, and forwards the new message towards the root. When router A receives the message, it sends a tunnel Setup message back directly to the requesting router C, and the spliced tunnel is established. Then A stops sending packets via the old A-B tunnel. After C receives the Setup message of A-C tunnel, it can send a tunnel Destroy message to B to remove the old A-B tunnel.

3.10 Tunnel Split

When a new member joins the group, it might be necessary to add branches in the middle of the tunnel. Consider the case shown in figure 3.3:

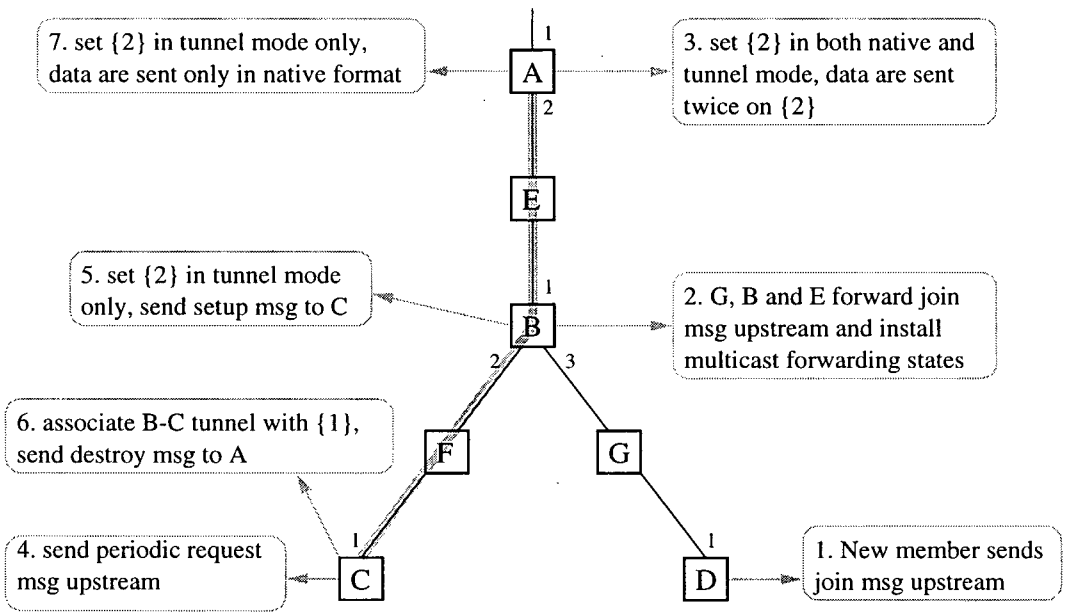


Figure 3.3: Tunnel Split

There is a tunnel established from A to C for a multicast distribution tree. Router E, B and F are on the native path of the A-C tunnel, but they do not have any forwarding state information for the distribution tree since they are bypassed. Router D is a new member that wants to join the group. It sends a Join message towards the root. The Join message would have stopped at B if there were no tunnels set up and B would have been the branching node of the distribution tree. Since now the forwarding states on routers between B and E have already been deleted, the Join message propagates all the way to the upstream tunnel end point A. Processing of the Join message reinstalls forwarding states on all the routers

between B and A.

Now the topology of the tunnel tree and the topology of the native tree are no longer aligned. If the routes are symmetric, packets might be sent twice on some of the links between A and B, once in native format on the native tree to D, the other in encapsulated format through the tunnel between A and C. If the routes are asymmetric, duplicates may still occur on some of the links. In most cases, duplicates do exist. In order to avoid duplicates, we need the ability to add a new branch in the middle of the A-C tunnel.

The situation can be corrected when C sends the next periodic tunnel Request message upstream. When the Request message reaches B, B will not forward it further since the Request message arrives on an interface other than the one on which the Join message arrives. B sends back a tunnel Setup message to the requesting router C setting up the new tunnel. When C receives the Setup message, it changes the upstream end point of the tunnel from A to B and sends a tunnel Destroy message to A. Tunnels between A,B and between B,C can be established later following the tunnel establishment procedure described in section 3.5.

The tunnel request from C may be rejected by router B because the tunnel is too short. In this case, router B sends a tunnel reject message back to C indicating the reason. When router C receives the rejection, it changes the upstream interface to Native state, and starts sending normal Join messages upstream to reinstall multicast forwarding state between C and B. The old A-C tunnel can be torn down either when C starts receiving data from the native tree, or after a timeout.

3.11 Early Tunnel Termination

A problem may occur when tunnels are allowed to span across multi-access links and a router on a multi-access link has more than one immediate downstream members on the link forwarding tunnel Requests. Allowing multiple tunnels to be associated with a multi-access interface not only complicates the implementation, but also generates duplicates since packets are sent multiple times for each of the outgoing tunnels on the multi-access link. This problem only occurs when the join suppression on the link is disabled or modified as discussed in section 3.4.

To solve this problem, the downstream routers on the multi-access link should listen on the link for Join or Request messages from other routers. If there are other sibling members on the multi-access link, then the router should become a potential tunnel end point and respond to tunnel Requests from routers further downstream. Downstream routers on multi-access links should not generate Request message upstream.

3.12 Dynamic Tunnels in PIM-SM

Some problems are unique to PIM-SM since it allows a source specific tree and a shared tree to exist at the same time for a given multicast group. These problems include tunnel sharing among the source specific trees and the shared tree of a multicast group, and source specific prune state on the shared tree.

In most cases dynamic tunnels are established separately for source specific trees and shared tree of the same multicast group. Within the part where a source specific tree and the shared tree overlap, the tunnel can be shared among the different trees of the same multicast group.

Source specific prunes, i.e. the (S,G)RPT forwarding states, are filters on the shared tree used to prevent packets from being delivered to members that have already switched to source specific trees. They always coexist with (*,G) forwarding states, and no separate tunnels are established for them. These states are generated at the point where a source specific tree and the shared tree diverge, and are propagated upstream along the shared tree until the next branching point. When dynamic tunnels are established, this diverging point may be shifted downstream to the next tunnel end point. On a router where both (S,G) and (*,G) forwarding states exist, if the incoming interfaces of the two forwarding states are the same but are associated with different tunnels, the two interfaces should be considered different. Source specific prunes should be sent upstream along the shared tree, via the dynamic tunnel, if one has been established.

3.13 Fault Tolerance

In this section, various failure conditions are considered. The goal of our approach is to reduce as much as possible the interruptions in data delivery.

3.13.1 Failure in the Middle of a Tunnel

If a link or a router on the native path is down, it will be automatically routed around by the unicast forwarding mechanism. If the route change caused by the failure does not affect the location of tunnel end points, no adjustment is necessary. If the native tree is changed, as long as the tunnel end points are still mutually reachable, the data delivery via the tunnels will not be disturbed. However, the topology of the tunnel tree may no longer be the optimum. This situation is discussed in more detail in section 3.13.3.

3.13.2 Failure of the Tunnel End Point

Failed upstream tunnel end point will be detected by the unicast or multicast routing protocol running on its neighbors. The branching point of the tunnel tree must be adjusted accordingly, otherwise the tunnel tree topology and the native tree topology are no longer aligned. The next Request message sent towards the root will be sent via a different route resulting in a "upstream branching point shift" as described in section 3.13.3.

If a downstream tunnel end point only has Native state downstream interfaces, its failure can be detected by the underlying multicast or unicast routing protocols running on its immediate downstream receivers. The next Join messages from those receivers are sent on an alternative path towards the root, and new tunnels can be established later.

3.13.3 Branching Point Changes

Route changes can alter the branching points and hence the topology of the native tree. The branching points of the native tree can be moved either upstream or downstream. The branching points of the tunnel tree have to be adjusted accordingly, otherwise the tunnel tree topology will not be optimal. Here we assume that the native tree constructed by the underlying multicast routing protocol is the optimum.

In this section we introduce several mechanisms that can align the topology of the tunnel tree with that of the changed native tree. There is always a trade off between the tunnel tree efficiency and the control overhead. Since most of the routes on the Internet are expected to be fairly stable, route changes are considered scarce events. Under this assumption, the simple mechanisms are actually favored.

In this document we simply point out the problems and their possible solutions.

Branching Point Downstream Shift

After a route change, tunnel Request may arrive at a different interface of the upstream tunnel end point. If the new interface has no tunnel associated with it, the upstream tunnel end point simply changes the upstream associated interface of that tunnel. If the new interface is associated with another tunnel which indicates the native paths of the two tunnels now share some common prefix, then the branching point of the native tree has moved somewhere downstream.

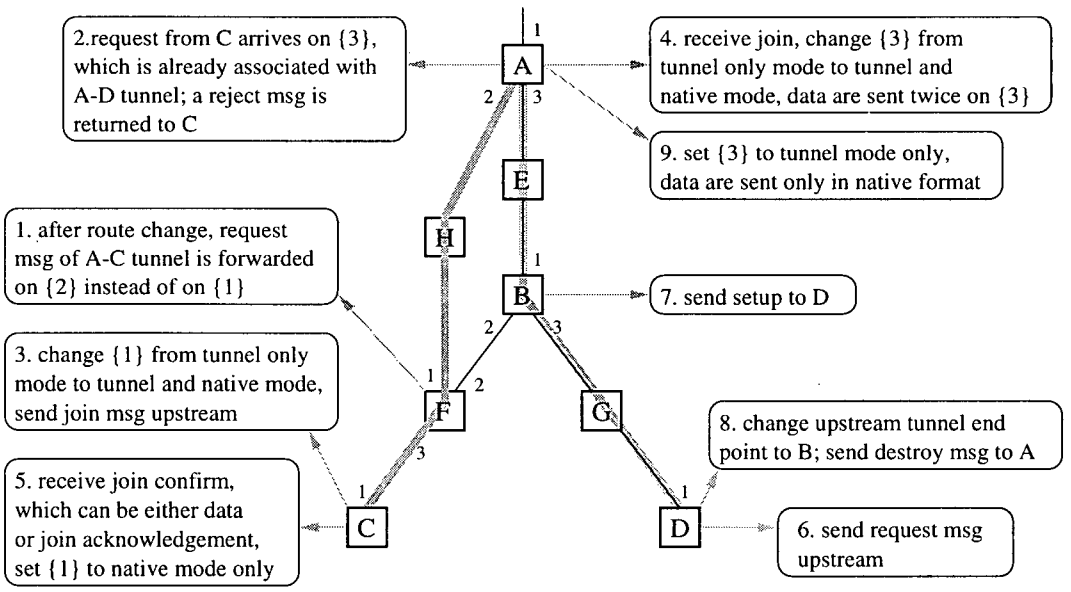


Figure 3.4: Route Change: Branching Point Shift Downstream

Figure 3.4 shows an example that illustrates the situation. Two tunnels *A-C* and *A-D* are already established. Their native paths were completely different before the route change. The native path of tunnel *A-C* is $A - H - F - C$ while the native path of tunnel *A-D* is $A - E - B - G - D$. After the route change, tunnel

A-C's native path passes through B, which is also on the native path of A-D tunnel. Now tunnel Request messages of the two tunnels arrive at the same interface of A. The branching point of the native tree has moved from A to B.

In order to find the new location of the branching point, we tear down the tunnel whose native path has changed. The upstream tunnel end point sends a tunnel reject message to the downstream end point of the tunnel about to be torn down. When the downstream tunnel end point receives the rejection, it changes its upstream interface to native and Tunnel state, and starts sending Join message upstream to reinstall multicast forwarding state on all the routers on the native path of the tunnel. When the downstream end of the tunnel that is about to be torn down starts receiving data from the native tree, it destroys the tunnel. The next tunnel Request message from the remaining tunnel will trigger a tunnel split operation as described in section 3.10, and the new branching point is found.

In our example, A sends rejection to C, C switches to Dual state, and sends a Join message upstream. When C starts receiving data from the native tree or when C receives join acknowledgment, it sends a Destroy message to A, tearing down the A-C tunnel. The next tunnel request message from D will trigger a split of the A-D tunnel, a situation already discussed in section 3.10, and the new branching point B will be found.

Branching Point Upstream Shift

The changes in the route may cause the upstream tunnel send point to be no longer on the correct path from the downstream tunnel end point towards the root. In this case, the periodic tunnel Request message can no longer reach the current upstream tunnel end point. The message will be propagated towards the root until it reaches

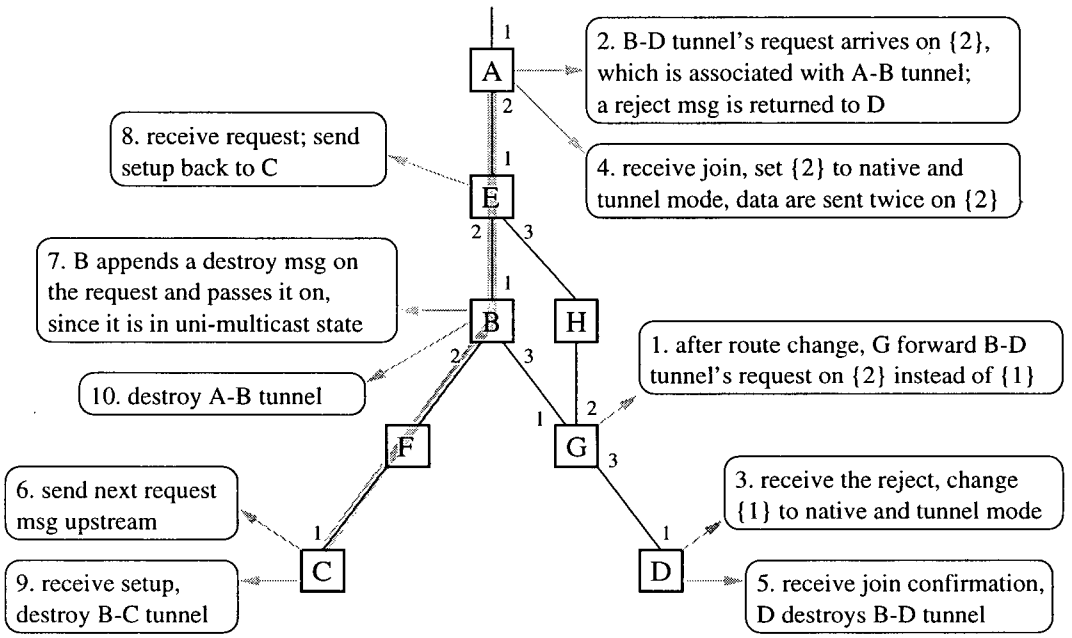


Figure 3.5: Route Change: Branching Point Shift Upstream

a node that already has multicast forwarding state for the distribution tree. The interface on which the message arrives can be in either Native state, Tunnel state, or Dual state. The message arriving at an interface in Native state causes a tunnel Setup or Reject message to be returned. The message arriving at an interface in Tunnel state causes a tunnel Reject message to be returned. The message arriving at an interface in Dual state is discarded.

If the downstream end point of the tunnel being affected by the route change receives a Setup message, it sets up the new tunnel and destroys the old upstream tunnel. The upstream end point of the new tunnel is the correct new branching point. If the downstream end point of the affected tunnel receives a reject message, it switches its upstream interface to Dual state and sends Join messages upstream. The old upstream tunnel can be torn down after its downstream end point receives

confirmation of the join, which can be either data from the native tree or explicit acknowledgment. The new branching point subsequently can be found after another tunnel split.

In the example shown in figure 3.5, a route change causes Request message of tunnel B-D no longer to pass through its upstream tunnel end point B. Instead, it reaches router A. From the point of view of router A, the request comes from a brand new requesting router, but the request arrives on an interface that is already associated with another tunnel. A then sends a tunnel reject message back to D. When D receives the rejection, it sets interface $\{1\}$ to Native state, and starts sending Join messages upstream. In this example, the Join message is propagated all the way to A. When A receives the join, it sets interface $\{2\}$ to Dual state and starts sending data in both native format and in encapsulated format on interface $\{2\}$. When D receives data from the native tree or receives explicit join acknowledgment, it destroys the old B-D tunnel. The next periodic tunnel Request message from C will trigger a new round of combined tunnel splice and split. When B receives the tunnel Request message from C, it appends a Destroy message in the request and forwards it upstream according to the tunnel splice procedure described in section 3.9. When E receives this Request/Destroy message, it ignores the destroy part since it has no tunnel established to B. E processes the Request message and sends a tunnel Setup message to C following the tunnel split procedure described in section 3.10. When C receives the tunnel Setup message, it destroys the B-C tunnel following the tunnel splice procedure. When B receives the tunnel Destroy message from C, it transitively destroys the A-B tunnel in which the old A-C tunnel will be split into A-E and E-C tunnel. Tunnel E-D can also be established later.

Chapter 4

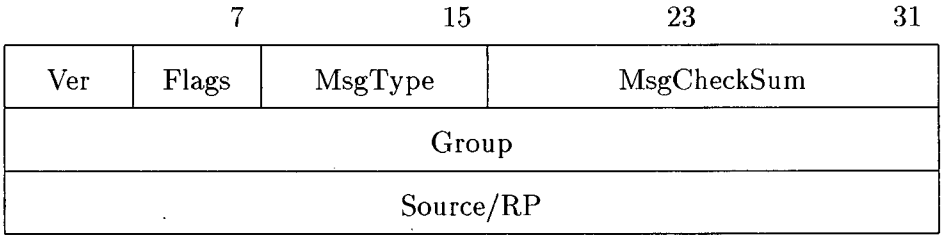
Protocol Specification

In this chapter, we describe the specification of the PIM-DT protocol, and the Dynamic Tunnel Multicast protocol with PIM-SM as the underlying multicast support.

4.1 Message Types

A DTMP control message contains a common message header followed by one or more Tunnel Control Objects. There are four objects defined in this document: request, setup, reject and destroy. The objects that are included in the control message is indicated in the common message header.

4.1.1 Common Header



The contents of each fields are:

- *Ver* (4 bits)

  Protocol version number. This document defines DTMP version 1.0.

- *Flag* (4 bits)

  Control flags. Reserved.

- *MsgType* (8 bits)

  Message type. There are five message types related to tunnel management in DTMP:

  1. MsgType 0 = Tunnel Request
  2. MsgType 1 = Tunnel Setup
  3. MsgType 2 = Tunnel Reject
  4. MsgType 3 = Tunnel Destroy

  A Tunnel Request message contains one Tunnel Request object followed by an optional Tunnel Destroy object. A Tunnel Setup message contains one Tunnel Setup object. A Tunnel Reject message contains one Tunnel Reject object, and a Tunnel Destroy message contains one Tunnel Destroy object.

- *MsgCheckSum* (16 bits) The message checksum. Although UDP header has a checksum field, the calculation of checksum is optional. So we need our own checksum to ensure message integrity.

- *Group* 32 bits The address of the multicast group.

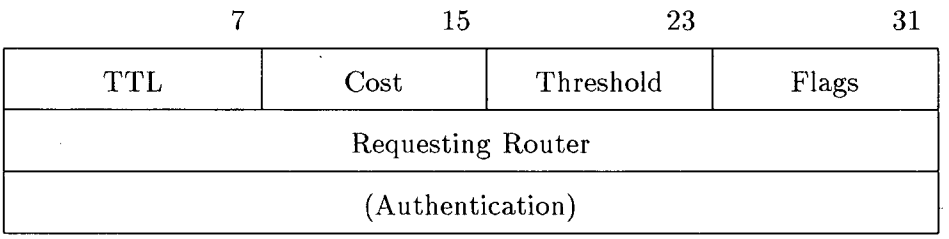
- *Source/RP* (32 bits) This field contains the address of the source if the router is on a source specific tree, and contains the address of the Rendezvous Point(RP)

if the router is on a shared tree.

Following are the definition of each tunnel control objects. Each object may contain an optional Authentication field at the end, which is used by the receiver to verify the validity of the object. Only valid object from a trusted router will be accepted.

4.1.2 Tunnel Request Object

The format of a Tunnel Request Object is:



- *TTL* (8 bits): The initial TTL value of the IP packet when it is originated from the source. This is used to calculate the distance between the message sender and receiver in terms of hop count.

- *Cost* (8 bits): This field contains an estimation of the cost of the path from the router where the request message has last traversed to the requesting router. The field is updated at each hop. When the Request message is forwarded further upstream, the value of the cost field in the outgoing message should be the value in the incoming Request message plus the cost of the link on which the Request message is received. If a tunnel is established, the updated estimation will be used as the cost of the tunnel.

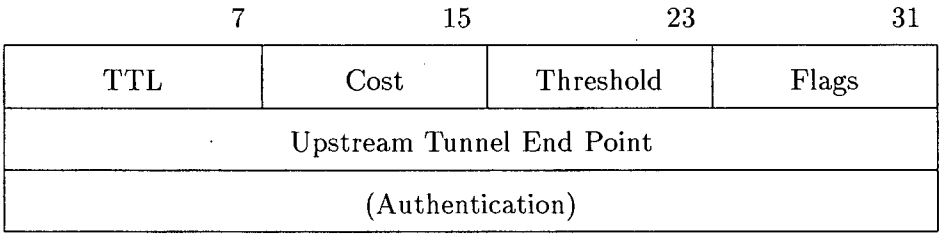
- *Threshold* (8 bits): This field contains the threshold value for the path between the router where the request message has last traversed and the requesting router. This value is also updated by each router as the request message is forwarded upstream. The updated Threshold value should be the larger of the threshold value of the link on which the request message has arrived, and the threshold value in the incoming request message plus the cost of the last link on which the massage has traversed. If the request is forwarded upstream, the threshold field of the outgoing message should contain the updated value. If a tunnel is established, the updated value will be used as the threshold for the tunnel. An incoming data packet will not be forwarded on the tunnel if its remaining TTL value is less than the tunnel's threshold value.

- *Flags* (8 bits):

- *Requesting Host* (32 bits): The address of the originator of the Request message. The originator will become the downstream tunnel end point if a Setup message is received later.

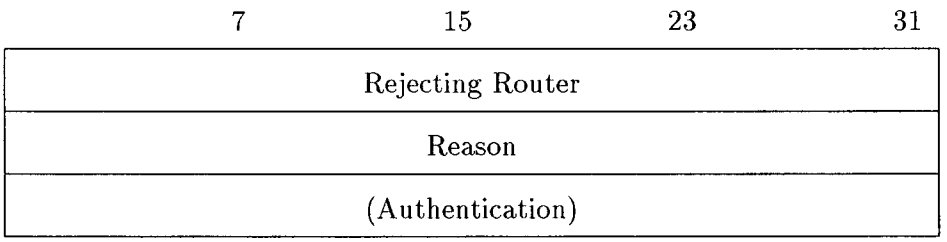
- *Authentication* (variable length, optional): This field is used to authenticate the request and is optional. The format of the Authentication Object is yet to be determined.

4.1.3 Tunnel Setup Object



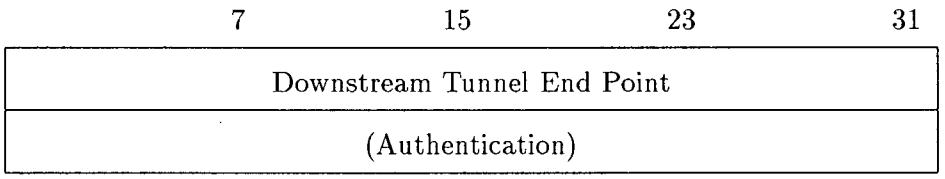
- *TTL* (8 bits): The TTL value of the IP packet when it is originated.
- *Cost* (8 bits)
- *Threshold* (8 bits):
- *Flags* (8 bits):
- *Upstream Tunnel End Point* (32 bits): The address of the originator of the Setup message. The originator now is the upstream tunnel end point.
- *Authentication* (variable length, optional): Used to authenticate the setup.

4.1.4 Tunnel Reject Object



- *Rejecting Router* (32 bits): The address of the originator of the Reject message.
- *Reason* (32 bits): Indicate the reason why the request is rejected.
- *Authentication* (variable length, optional): Used to authenticate the reject.

4.1.5 Tunnel Destroy Object



- *Downstream Tunnel End Point* (32 bits): The address of the originator of the Destroy message, which was the downstream end of the tunnel.
- *Authentication* (variable length, optional): Used to authenticate the destroy.

4.2 Message Processing

4.2.1 State Transition

Each routing table entry has one incoming interface, *iif* , and n outgoing interface, *oif*s . In order to describe the processing of the control messages, two state machines are used, one for the *iif* and one for the *oif* . Usually, the *iif* state machine of an upstream router exchange messages with the *oif* state machine of a downstream router. Sometimes events on an *oif* state machine will trigger transitions on the *iif* state machine of the same router.

Outgoing Interface States

An outgoing interface(*oif*) of a multicast routing table entry can be in one of the following four states: Idle, Native, Tunnel or Dual state.

An *oif* in Idle state has no downstream member. Data are not forwarded on Idle *oif*s . *oif*s in Idle state can be deleted from the *oif* list of the routing table entry. An *oif* in Native state has a downstream member requesting native data

on the link connected to the *oif* . The *oif* should keep receiving Join messages in order to stay in the Native state. Incoming data will be forwarded on to a Native *oif* in native format. An *oif* in Tunnel state has an outgoing tunnel established to a downstream receiver. The *oif* should keep receiving Request messages in order to stay in the Tunnel state. Incoming data are forwarded in encapsulated format to the downstream end point of the outgoing tunnel associated with the *oif* . An *oif* in Dual state has both outgoing tunnels and immediate downstream router that are requesting native data. Data are sent twice for an *oif* in Dual state: once in native format to the immediate downstream member, once in encapsulated format to the downstream end point of the tunnel associated with the *oif* .

Outgoing Interface State Transition

Figure 4.1 describes the transitions among the four *oif* states. In the labels beside transitions, inputs from downstream routers are shown above the bar, and outputs to downstream routers are shown below the bar. Forwarded messages to upstream routers are also put below the bar but with a prefix "F:". conditions are shown in square brackets. There are two conditions defined in the diagram, [U] and [M]. Condition [U] holds when the routing table entry of the distribution tree does not have other *oif*s than the interface on which the input message arrives, and when there is no local member for the group and source. Condition [M] holds when the entry has other *oif*s or has local member.

When the first Join message for the distribution tree arrives on an interface in Idle state, i.e., on an interface that is not in the *oif* list of the routing table entry for the group, the interface is inserted in the *oif* list of the routing table entry. The *oif* changes to Native state from Idle state. If the Join arrives on a router that does

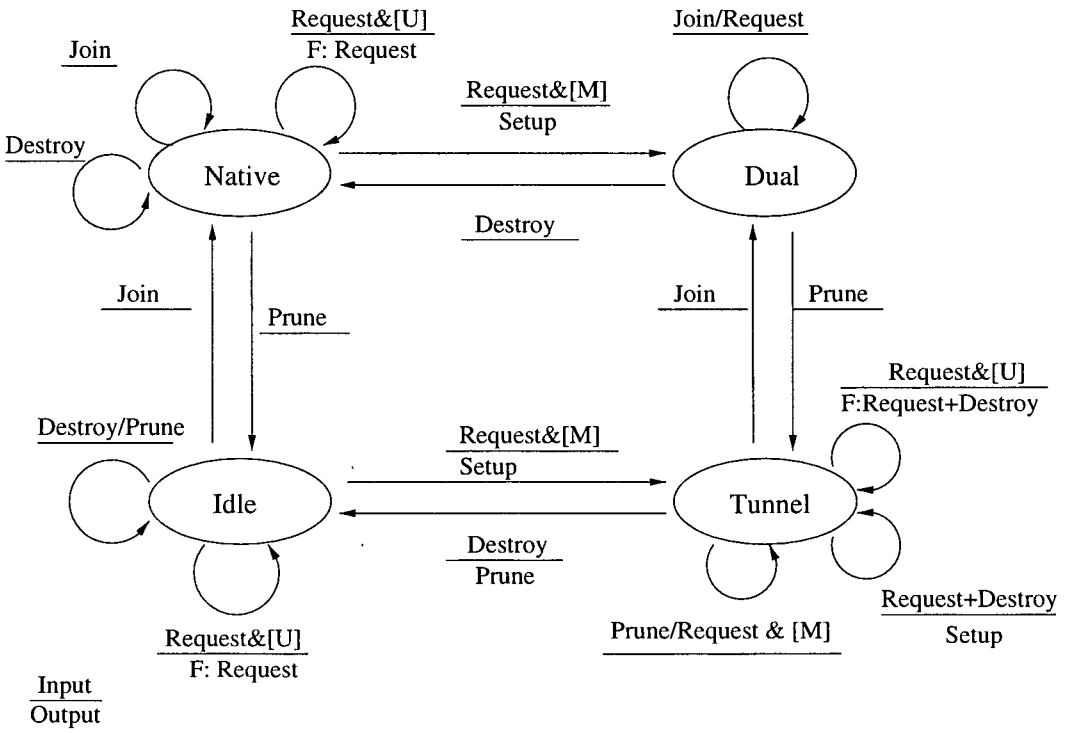


Figure 4.1: Outgoing Interface State Transition Diagram

not have a routing table entry for the group and source, an entry will be created, and the Join will be forwarded upstream. When a Request message arrives on an interface in Idle state, and the associated routing table entry has *oif* other than the one on which the Request arrives, a Setup message is sent back to the requesting router and a tunnel is established. The interface changes from Idle state to Tunnel state. When a Request message arrives on a router that does not have associated routing table entry for the distribution tree, i.e.. when the [U] condition holds, then the Request is forwarded further upstream.

An *oif* in Native state enters Dual state when a Request message is received

on the *oif* and condition [M] holds, i.e., the entry either has more than one *oif*s or has local member. A Setup message is returned to the requesting router, and a tunnel is established. If the Request arrives on an *oif* in Native state and the condition [U] holds, i.e.. the entry has only one *oif* and no local member, then the entry remains in Native state and the request is forwarded further upstream. An *oif* in Native state changes to Idle state if a Prune is received on the interface. The *oif* can be deleted from the *oif* list. If the *oif* list hence becomes empty and there is no local member, a prune is sent further upstream.

An *oif* in Tunnel state changes to Dual state when a Join is received. It changes to Idle state when a Destroy message is received and there is no local member, and the *oif* can be deleted from the *oif* list. If the *oif* being deleted was the last in the list and there are no local members, a Prune or Destroy message is sent upstream depending on the state of the *iif* . If a Request message is received on a Tunnel state *oif* and condition [U] holds, a Destroy object is appended in the Request message and the combined message is sent upstream. If a Request arrives on an *oif* and condition [M] holds, the tunnel is refreshed. If a combined Request and Destroy message is received and the tunnel associated with the *oif* is the same as the one in the Destroy message, then that tunnel is destroyed, and the new tunnel is setup as requested. The combined Request and Destroy message is used in tunnel splice operation.

An *oif* in Dual state changes to Tunnel state if a Prune is received on the *oif* . An *oif* in Dual state changes to Native state if a Destroy is received. Other messages are discarded. It is designed primarily to avoid interruptions in data delivery during the tunnel setup and destroy. An entry should stay in Dual state as short as possible, since the router very likely is generating duplicated data packets on the same link.

Incoming Interface States

The incoming interface(*iif*) of a multicast routing table entry can be in one of four states: Idle, Native, Tunnel or Dual.

A routing table entry with *iif* in Idle state will not accept any incoming data. Routing table entries with Idle *iif* can be deleted. A routing table entry in Native state can accept incoming data in unencapsulated format, i.e., native format. Data will be forwarded on to all the outgoing interfaces of the entry. Data arrived in encapsulated format will be discarded, and a tunnel Destroy message may optionally be sent back to the originator of the encapsulated tunnel data. If the routing table entry for a distribution tree is not an uni-multicast entry, i.e., it either has local member, or has more than one different immediate downstream receiver, then a tunnel Request message is sent periodically on the *iif* , trying to establish a tunnel.

A routing table entry with *iif* in Tunnel state has an incoming tunnel associated with it. Only packets from the correct upstream tunnel end point can be accepted and forwarded on. Incoming data in native format will be discarded and optionally a Prune message can be sent upstream. Incoming encapsulated data from a router other than the upstream tunnel end point will be discarded and a tunnel Destroy message may be sent to the originator of the encapsulated data.

Incoming Interface State Transition

Figure 4.2 illustrates the transitions among the four *iif* states. The conventions used here are the same as those in the *oif* state transition diagram. The two new conditions are defined here. Condition [J] holds when there is some downstream receiver or there is local member. Condition [P] is the opposite of [J].

When the first Join message for the distribution tree arrives on an *oif* or

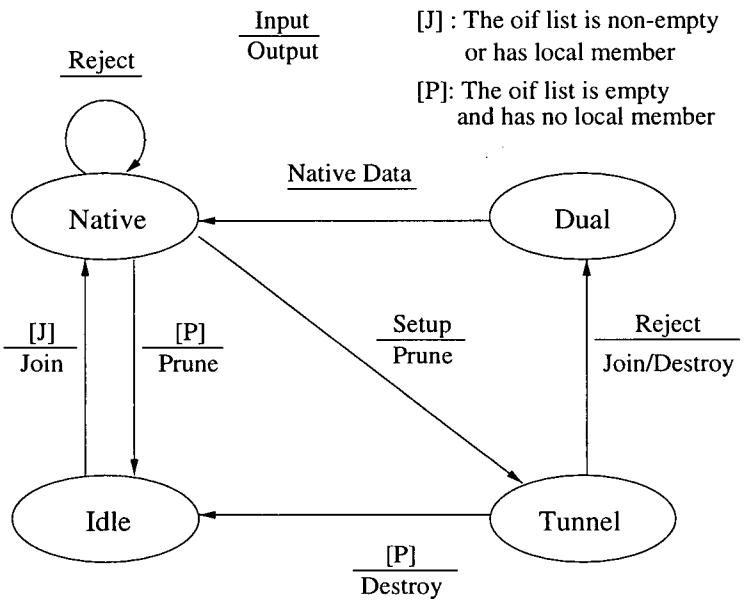


Figure 4.2: Incoming Interface State Transition Diagram

when the first local member joins the group, a routing table entry is created, a Join message is sent upstream, and the *iif* of the entry becomes in Native state.

When the last downstream or local member leaves the distribution tree, the *iif* of the multicast routing table entry for the group changes to Idle state. A Prune is sent upstream if the *iif* is in Native state, a Destroy is sent if the *iif* is in Tunnel state. The entry itself can then be deleted. Other messages received when *iif* is in Idle state are discarded.

If a Setup message is received for a *iif* in Native state, a tunnel is established and the *iif* changes to Tunnel state. An *iif* in Dual state changes to Native mode when data in native format starts to arrive on the *iif* . If a Reject message is received for a *iif* in Tunnel state, a Destroy and a Join are sent upstream, and the *iif* changes to Dual state. Unexpected Setup message received in any state should

always trigger a Destroy message being sent back to the originator of the Setup message.

4.2.2 Pseudo Codes

Following we present the pseudo codes for the message processing. They gives more accurate definitions for the state machine. Real implementations can be derived strait forward from the pseudo code.

Process of Join Message

```
process_join(Join message)
{
    determine the arriving interface aif of request_message;
    lookup the routing table entry i of the requested group;
    if ( no such entry ) {
        create a new entry for the group;
        insert aif in the oif list;
        set the aif in Native state;
        set iif in Native state;
        send Join message further upstream;
        return;
    }
    if ( aif in Tunnel state ) {
        set aif in Dual state;
    }
    if ( iif in Tunnel state )
        return;
    old_iif = iif;
    calculate the new iif;
    if ( iif != old_iif ) { // iif has changed
        send Join message further upstream;
    }
    return;
}
```

Process of Prune Message

```
process_prune(Prune message)
{
    determine the arriving interface aif of request_message;
    lookup the routing table entry i of the requested group;
    if ( no such entry )
        return;

    if ( aif in Dual state ) {
        set aif in Tunnel state;
    } else if ( aif in Tunnel state )
        return;
    } else {
        // native state, delete the oif
        set aif to Idle state and delete aif from oif list;
        if ( oif list is empty ) {
            if ( iif in Native state )
                send Prune upstream;
            else
                send Destroy upstream;
        }
    }
}
```

Process of Request Message

```
process_request(Request message)
{

    determine the arriving interface aif of request_message;
    lookup the routing table entry i of the requested group;

    if (no routing table entry) {
        if (tunnel bypassing is allowed)
            // forward the request further upstream;
            forward_request(group, source, pkt); // exit #1
        else return; // exit #2
    }

    if (a Destroy object is included in the message)
```

```
    process_destroy(Destroy object);

determine the outgoing interfaces oif_list of the routing table entry i;
if ( aif is in Idle state ) {
    if ( distance(requesting_router, this_router) <
        minimum tunnel length requirement) {
            discard Request message and return; // exit 3.5
    }
    if ( there exists another tunnel t1 to the requesting router )
        delete_tunnel(t1);

    // performing tunnel split operation
    set aif in Tunnel state and insert aif into oif_list;
    setup a tunnel to the requesting_router on aif;
    return;  // exit #3
}

// aif is in oif_list, trying to forward the request further upstream
// check if the router is doing uni-multicast for the group
// is_umcast() returns true if the group has no local member and
// has only one immediate downstream receiver;
// it tests the U1 condition
if ( is_umcast(group, source) && is_bypass_allowed() ) {
    if ( no incoming tunnel for i ) {
        forward the Request upstream;
        return; // exit #5
    } else {
        // exist incoming tunnel ti for i
        append a destroy_object in the request_message;
        forward the new Request message upstream;
        return; // exit #6
    }
}

// can't forward the request upstream, check the length limit
if ( distance(requesting_host, local_host) <
    minimum tunnel length requirement) {
        discard Request message and return; // exit 6.5
}

if ( aif in Tunnel state ) {
    determine aif's associated tunnel t2
```

```
        if ( the other end of tunnel t2 is the requesting_router ) {
            reset the tunnel_refresh_timer of t2;
            return; // exit #9
        }
    } else { aif in Native state or the requesting router is
        if ( there exists another tunnel t3 to the requesting_router )
            delete_tunnel(t3);
        if ( aif in Native state ) {
            setup a tunnel to the requesting_router on aif;
            return; // exit #7 8
        }
        // reject the request since the aif is occupied by another tunnel
        send_reject to the requesting router
        return; // exit #10 11
    }
}
```

Process of Setup Message

```
Process_setup(Setup message)
{
    lookup the routing table entry i of the group;

    if ( no routing table entry found ) { // no such group
        send Destroy message to the sender of the Setup message
    } else {
        setup the tunnel;
        if (iif in Native state) {
            send Prune message upstream;
        } else { // iif is in Tunnel state
            send Destroy to the upstream end point of the old tunnel;
        }
    }
}
```

Process of Reject Message

```
Process_reject(Reject message)
    lookup the routing table entry i of the group;
    if ( no routing table entry found )
```

```
        return; // discard the message

    // starting send joins again if there was a tunnel established
    if ( iif in Tunnel state ) {
        set iif to Native state;  // discard incoming tunnel
        if ( have local member or have downstream receiver ) {
            send Join message upstream;
        }
        send Destroy message upstream;
    }
}
```

Process of Destroy Message

```
Process_destroy(Destroy message/object)
{
    determine the arriving interface aif of request_message;
    lookup the routing table entry i of the requested group;

    if (no routing table entry)
        return;  // no such group, discard the Destroy

    if ( aif in Tunnel or Dual state ) {
        if ( requesting router ==
            downstream end point of the tunnel associated with aif) {
            delete the tunnel;
            if ( aif in Tunnel state ) {
                set aif to Idle state and delete the aif from the oif list of i;
                if ( oif list is empty ) {
                    if ( iif in Native state )
                        send Prune upstream;
                    else
                        send Destroy upstream;
                }
            } else {
                set aif to Native state;
            }
        }
    }
}
```

4.3 Timers and Refresh Message Generation

In PIM-SM, there are three types of required timers related to a source specific forwarding table entry: Join/Prune-Timer, Oif-Timer, and Entry-Timer. One Join/Prune-Timer is maintained for each entry to generate periodic Join/Prune messages. When the Join/Prune-Timer goes off, all the routing entries are checked, and aggregated Join/Prune packets that may contain multiple Join or Prune messages are sent upstream. n Oif-Timer, one for each *oif* in the entry, are used to timeout each *oif* . When an Oif Timer goes off, the associated *oif* is removed from the forwarding table entry. The timer is refreshed(the timer value is set to its initial value) each time a Join message arrives on the *oif* . One Entry-Timer is used to timeout the entry itself. When this timer goes off, the associated multicast forwarding table entry is deleted. The timer is refreshed each time a data packet is received.

In PIM-DT, all these three timers are kept and the processing of the timer events remains largely unchanged. The differences in the timer event processing between PIM-SM and PIM-DT are: 1) When a valid request message is received on an *oif* that is in Tunnel or Dual state, the Oif-Timer of the interface is refreshed. The timer values used can be different from the values selected in PIM [14]. 2) When the entry timer goes off and the *iif* of the entry is in Tunnel state, a Destroy message instead of a Prune message is sent upstream.

In PIM-DT a new Request/Destroy timer is introduced to generate the periodic Request messages from the downstream tunnel endpoints to keep the tunnel state on the upstream tunnel end point. When this timer goes off, for each entry that either has more than one immediate downstream receiver or has local member and with *iif* in Tunnel state, a Request message is sent upstream.

Chapter 5

Analysis and Simulation

PIM-DT is intended to be implemented as an optimization on top of PIM-SM. In this chapter, we will evaluate the effectiveness of Dynamic Tunnel by comparing the performance of PIM-DT to that of PIM-SM. The performance of the protocols is measured through both analysis and simulation.

5.1 Performance Analysis

The efficiency of the Dynamic Tunnel Multicast routing protocol can be evaluated in terms of state information requirement, tree cost, data processing efficiency and control overhead. The state information requirement can be measured using the average multicast routing table size or the average multicast forwarding cache size. In the existing multicast routing protocols the two sizes are the same in most cases. The tree cost can be evaluated using the total cost of the links traversed by all the copies of a packet when it is delivered to all the receivers. The data processing overhead can be measured in terms of average number of instructions executed at each router in order to forward the packet. The control overhead can be measured

using the total number of control packets sent to all the links in order to maintain the correct protocol behavior.

In this analysis, we will focus on the states information requirement and control overhead of the Dynamic Tunnel Multicast protocol. The state information requirement can be measured using the average multicast forwarding table size. The control overhead can be measured using the total number of control packets sent on all the links that are needed to maintain the protocol states.

For simplicity, we only analyze and simulate the behavior of PIM-DT, the Dynamic Tunnel Multicast with PIM-SM as the underlying multicast routing protocol, and we only consider the case in which all the traffic is delivered on source specific shortest path trees (SPT). Dynamic tunnels with shared trees and bi-directional trees are likely to have similar behavior as those with source specific trees. The tree cost, data processing efficiency, control overhead and detailed tunnel dynamics on a full fledged version of PIM or CBT will be analyzed in our future work.

5.1.1 Network model

In the following analysis, each node in the network topology represents a router. Each router can be viewed as being connected to a local network omitted in the topology map. Routers are considered having local members if some hosts in its connected local network want to receive traffic. The receivers of a multicast group always join the source specific trees, thus no shared trees will be created in the network.

5.1.2 Evaluation Matrix

Average multicast routing table size

In PIM terminology, the multicast routing table entries on a source specific tree of a multicast group G rooted at a source S is a (S, G) entry.

First, let us define an α parameter of a distribution tree t to be the average number of multicast routing table entries per router for the tree:

$$\alpha(t) = \frac{N_e}{N_t} \tag{5.1}$$

where N_e is sum of the total number of multicast routing table entries, i.e., the total number of (S,G) entries, on all the routers for distribution tree t, and N_t is the number of routers on the tree.

When no tunnels are established, each router on a source specific distribution tree has one (S,G) routing table entry for the distribution tree, in which case $N_e = N_t$ and the value of the α parameter is always 1.0. 1.0 is the maximum α value for source specific trees. The minimum α value for any particular tree is defined by the following equation:

$$\alpha_{min}(t) = \frac{N_b + N_l + N_r}{N_t} \tag{5.2}$$

where N_b is the number of branching points on tree t, N_l is the number of leaf nodes on the tree, N_r is the number of root node of the tree which is always 1, and N_t is the total number of nodes in tree t. The α parameter of a tree reaches its minimum when all the uni-multicast routers on the tree are bypassed by dynamic tunnels. In conclusion, for source specific trees, the following condition holds:

$$0 < \frac{N_b + N_l + N_r}{N_t} < \alpha \leq 1.0$$

The α value shows what fraction of the routers on the distribution tree still have the state information after the tunnels are established.

Now we can use following formula to calculate the average number of multicast routing table entries in the entire network:

$$E = \bar{T} \cdot \frac{\bar{N}_e}{N} = \bar{T} \cdot \frac{\bar{N}_t}{N} \cdot \bar{\alpha} \tag{5.3}$$

where $\bar{T}$ is the average number of active multicast groups in the network, $\bar{N}_e$, $\bar{N}_t$ and $\bar{\alpha}$ are the average N_e, N_t and α values for all the distribution trees in the network respectively, and N is the total number of nodes in the network.

When no tunnels are established, the average number of multicast routing table entries E' is:

$$E' = \bar{T} \cdot \frac{\bar{N}_t}{N} \tag{5.4}$$

The percentage of multicast routing table entries saved due to the establishment of dynamic tunnels can be calculated as the follows:

$$\bar{\gamma} = \frac{E' - E}{E'} = (1 - \bar{\alpha}) \tag{5.5}$$

Thus the effectiveness of the dynamic tunnels in terms of reduction in multicast routing state is directly related to the $\bar{\alpha}$ parameter of the distribution trees. The smaller the $\bar{\alpha}$ value is, the more effective the tunnels are.

The α_{min} value of some example distribution trees

In this section, we will look at a number of distribution trees, and calculate their minimum possible α values. The minimum α values reveal the potentials of dynamic tunnels.

In the example shown in figure 1.1, the α parameter is 1.0 for the native tree. Since there are only 4 routers that are aware of the multicast group on the

tunnel tree, the α_{min} parameter is $4/20 = 0.2$ when the tunnel is established. The reduction in routing table size is 80%.

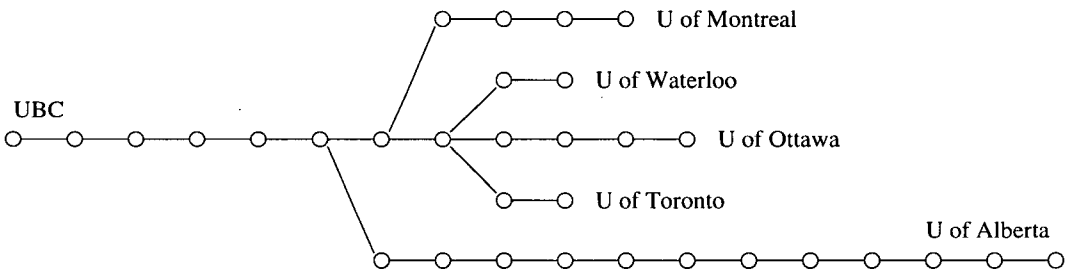


Figure 5.1: A Conference Example

In another example, 6 researchers from 6 universities in Canada want to have a video conference. The distribution tree rooted at UBC is shown in figure 5.1. There are 32 routers involved, including 1 root node, 5 leaf nodes, and 3 branching nodes. If tunnels are established, only 9 of them have to remember the forwarding state. Therefore the α_{min} parameter of the tunnel tree is $9/32 = 0.28$. The maximum reduction in forwarding state is 72%.

Finally we consider the real network routes collected by Vern Paxon in his Internet routing research [15], and analyze the possible α parameters of the trees. In Paxon's work, traces between 37 sites located all over the world are recorded using the `traceroute` utility. We pick one site as the sender, n other sites as the receivers, and construct a distribution tree based on the traces. One such tree rooted at Advanced Network and Services, NY, is shown in figure 5.2. The leaf nodes are labeled in the figure. The x axis is the distance in hop count between each site and the Advanced Networks and Services.

The minimum α parameter can be calculated using formula 5.2. The average value of the minimum α values with the number of receivers varying from 2 to 20

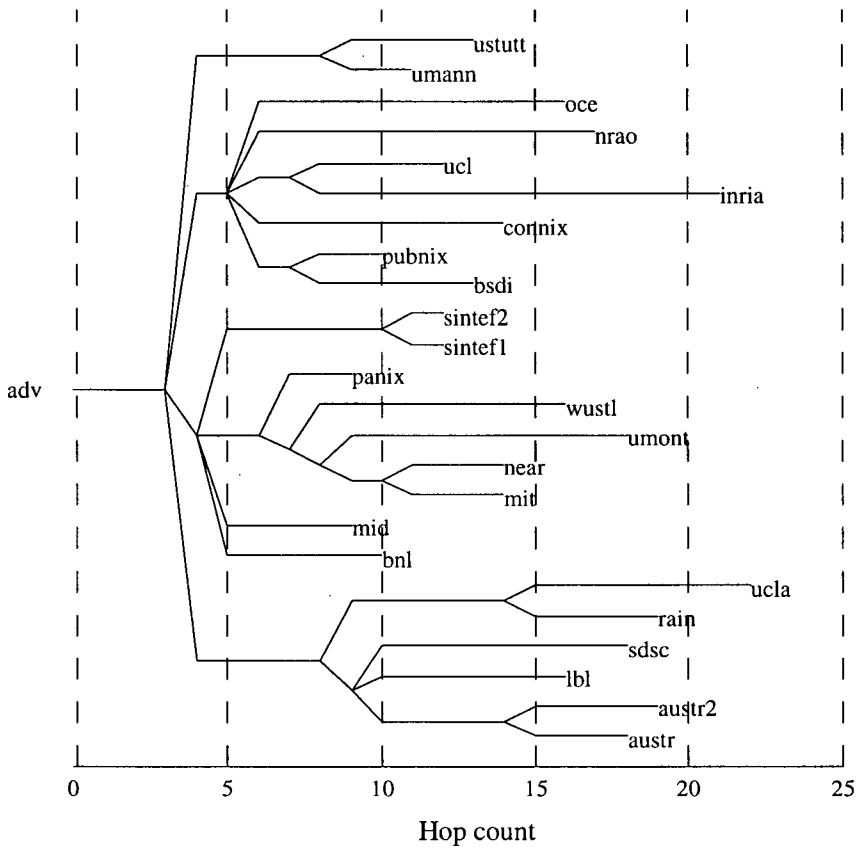


Figure 5.2: One distribution tree constructed from the traces

are shown in figure 5.3.

From the figure we can see that when the tunnels are all established, the α values are constantly smaller than 20%, which indicate over 80% reductions in forwarding table size.

5.2 Simulation

In the previous section the performance of PIM-DT is analyzed theoretically. In this section, we will use the LBNL Network Simulator, `ns` , to validate the basic protocol

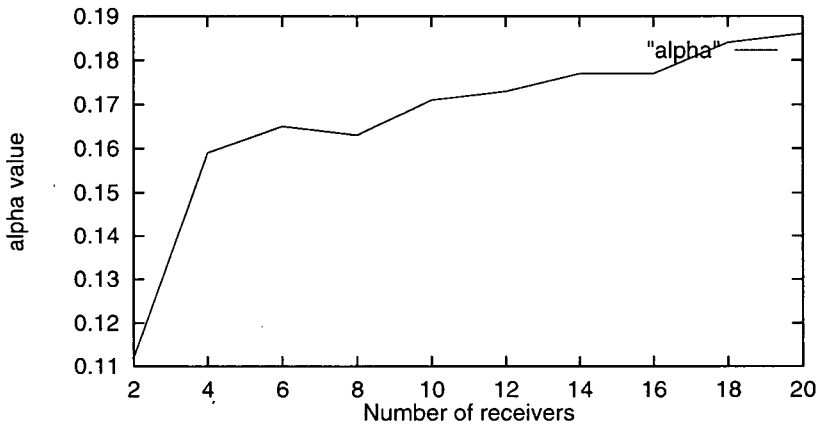


Figure 5.3: Average α_{min} for the trees

design presented in chapter 3 and the analysis given in the previous section.

5.2.1 Overview of ns

The LBNL Network Simulator, ns [16], is a simulation tool developed by the Network Research Group at the Lawrence Berkeley National Laboratory. It is an event-driven network simulator implemented as an extension to the Tool Command Language. The simulation engine is written in C++, and the simulation is controlled and configured via a Tcl interface.

There are three primitive building blocks in ns : nodes, links and agents. The nodes and links collectively define the network topology which is configurable through a Tcl script. Agents can generate and consume packets. Protocol entities such as sources, sinks and relays are implemented as agents which can be deployed among the nodes. Each node has an unique address and each agent is attached to a unique port on that node. A central scheduler keeps track of all the events such as packet arrivals and timeouts that occur on all the nodes, links or agents.

Statistics in the network such as byte or packet counts can be collected at any time during the simulation using Tcl commands. The simulated network behavior can also be recorded in the traces, which can be analyzed in detail after simulation. The traces can be visualized using the LBNL Network Animator where different message types, message sizes, and protocol states can all have different visual representations.

A new version of **ns** , **ns** version 2, is currently under development in the LBNL. We did not choose it because it was not stable at the time when we developed our simulations. **ns** has already been used as a powerful tool in many research areas, such as in the analysis and comparison of several flavors of TCP [17], in the analysis of router queuing and scheduling behavior [18], and in multimedia multicast delivery [19].

5.2.2 ns Multicast Extensions: PIMLite and SPIM

Our simulation of PIM-DT is developed based on Daniel Zappala's multicast-extensions for **ns** version 1.0b4 [20]. Daniel's extension includes a extremely simplified version of PIM called PIMLite. PIMLite supports only Joins but no Prunes. Joins are not sent periodically, and are not aggregated.

In our simulation, first, SPIM, another simplified version of PIM is implemented. SPIM supports many PIM functions that are not available in PIMLite, such as Prunes, aggregated Join/Prune messages, periodic timeout and refresh of routing table entries. In SPIM, all the receivers join the source specific tree from the very beginning and all the data are delivered via source specific trees. The implementation is also ported to ns version 1.4, the latest development of **ns** version 1. SPIM serves as the basis of implementation and the target of comparison

for PIM-DT.

The PIMLite/SPIM protocol entities are implemented as agents in `ns` . They are deployed on all the nodes in the network. The address space is partitioned and part of it is allocated to multicast. Nodes in PIMLite/SPIM are modified so that they can forward multicast traffic. Each PIMLite/SPIM agent maintains a multicast routing table and each node maintains a multicast forwarding cache, which is installed by the PIMLite agent attached to it. Management and look up functions for multicast routing tables and multicast forwarding caches are provided. Simple multicast sources and sinks are also implemented in the extension.

5.2.3 PIM-DT Simulation

Our simulation of PIM-DT focuses on two aspects of the protocol behavior: state information requirement and control overhead. Some protocol details which have no or little impact on these two parameters are omitted.

In our simulation, PIM-DT is implemented as a new type of agent derived from SPIM. It preserves all the functions in SPIM, and supports all the protocol features that are described in Chapter 2 and 3. Current implementation of PIM-DT does not support shared trees. This simplification will not significantly affect the evaluation of the dynamic tunnel mechanism.

Basic Test Network for Protocol Validation

The basic protocol features of PIM-DT are validated on a 15-node basic test network shown in figure 5.4. In this network, four receivers r1, r2, r3 and r4 on node 0, 1, 13 and 14 respectively join two source specific trees rooted at node 13 and 14, where two low bit rate Constant Bit Rate sources s1 and s2 are located. All the basic

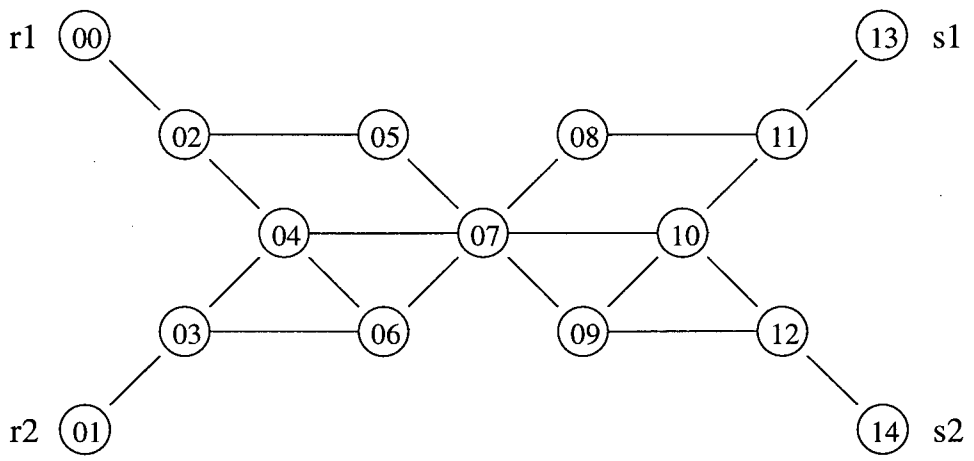


Figure 5.4: Basic Test Network Topology

tunnel operations such as tunnel establishment, tear down, split, splice, branching point shifts are tested on this configuration.

Experiment Setup for Performance Evaluation

The network topologies used in the simulation are $n \times n$ gird mesh topologies with $n = 8$. All the links in the network are identical bidirectional links whose bandwidth is 10Mb and delay is 3ms. T sources and N_l receivers are randomly deployed in the network. The duration of the test multicast session, i.e. the time between the first receiver joins the session and the last receiver leaves the session is D_s seconds. The average duration a receiver participate in a session is D_r seconds. Table 5.2.3 summarize the parameters used in the simulation.

5.2.4 Experiment Result

First of all, the simulation result obtained on the basic test network given in figure 5.4 is visualized using the LBNL network animator `nam`. The basic protocol operations

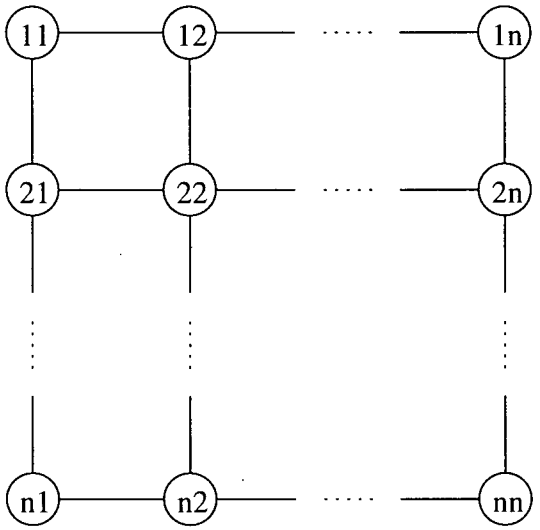


Figure 5.5: Experiment Network Topology

N	64	number of nodes in the network $= n \times n$
T	8,16,24,32,40	number of sources(distribution trees) in the network
N_l	4,8,16,32	number of receivers(leaves) for each source
D_s	500 sec	duration of the session
D_r	400 sec	duration that each receiver stays in the session

Table 5.1: Summary of simulation parameters

are all verified. We observed Requests and Setups being exchanged and tunnels being established. Next, several simulations are run on a 8×8 experimental network topology. Communication statistics of various links and nodes are logged to files, which were analyzed to verify if the alleged state reduction is achieved. In the experiment, we have only one sender for each group.

Routing Table Size

The routing table size on each router is sampled every 6 seconds. The average of the sampled value are calculated for both SPIM and PIM-DT. The average table size is shown in figure 5.6.

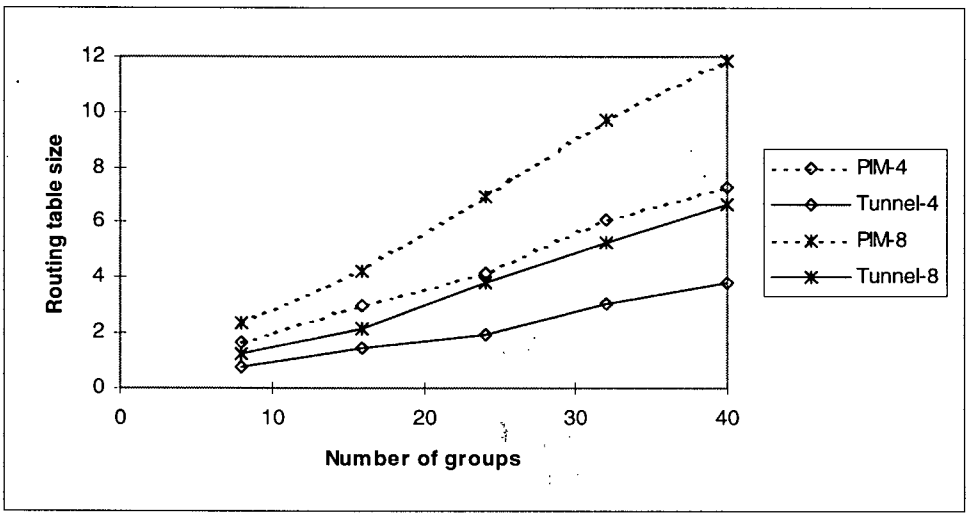


Figure 5.6: Average Routing Table Size

The horizontal axis is the number of groups that are active in the test network, and the vertical axis is the average routing table size. The poly-lines labeled PIM-4 and PIM-8 show the average routing table sizes for SPIM protocol when the maximum number of receiver per group are 4 and 8 respectively, and the poly-lines labeled Tunnel-4 and Tunnel-8 are the average routing table size for PIM-DT protocol with maximum number of receivers being 4 and 8.

From the figure we can see that when the number of receivers are the same, the routing table size of PIM-DT is much smaller than SPIM. The absolute routing table size grows with number of active groups and number of receivers, as predicted

in formula 5.3.

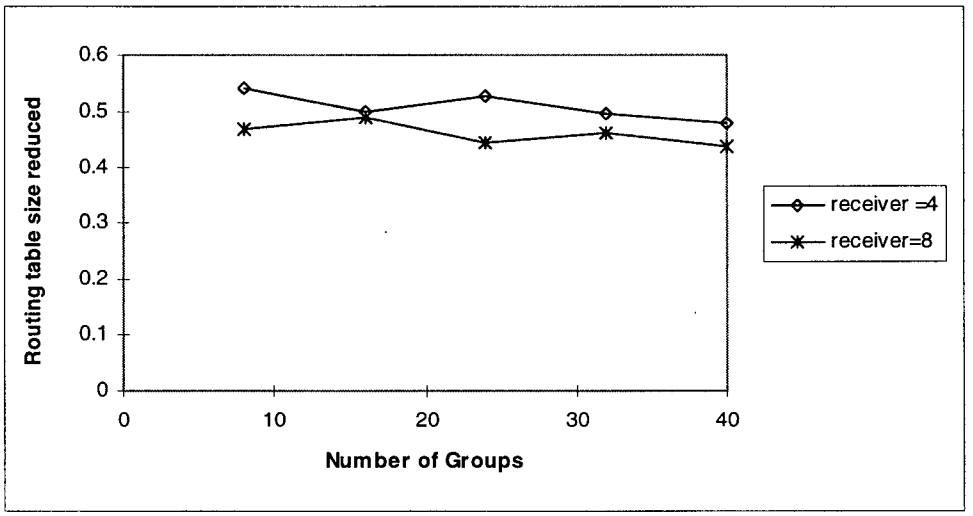


Figure 5.7: Reduction in Routing Table Size

Figure 5.7 shows the relative state information reduction achieved by PIM-DT. The horizontal axis is again the number of active groups, the vertical axis is the γ value as defined in formula 5.5. The figure shows a roughly 50% reduction in forwarding state information.

We manually checked the shape of some of the distribution trees generated during the simulation, and calculated their α_{min} parameters. The α_{min} values of the trees are around 0.5, which agrees with the γ values shown in figure 5.7 according to formula 5.5.

We believe running the simulation on a larger topology map can lead to more significant reductions in multicast forwarding states, we run one simulation on a 20×20 grid mesh network and observed around 80% reduction in forwarding states. However, further experimentations are necessary to confirm this result.

Control Overhead

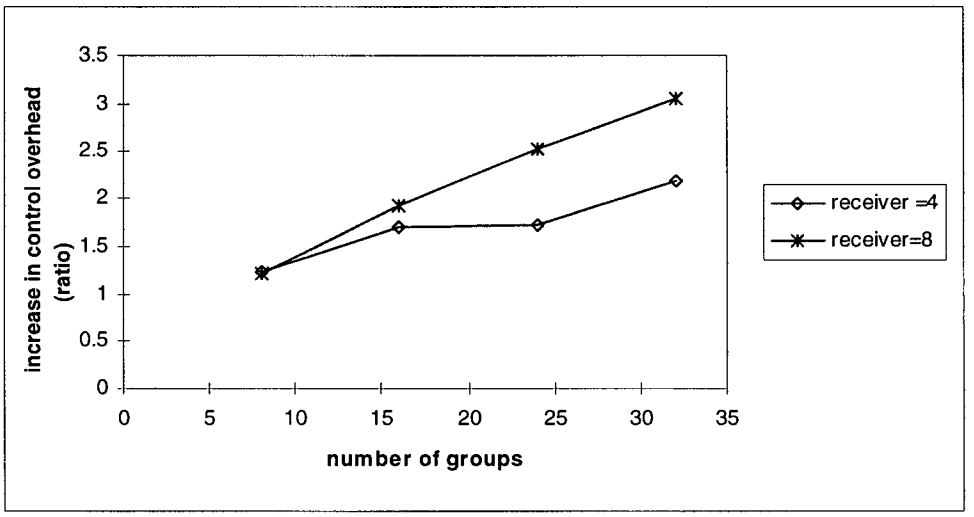


Figure 5.8: Control Overhead vs. Number of Groups

After we confirmed the routing table reduction, we performed several experiments to analyze the control overhead of PIM-DT. Usually the control overhead is measured as a ratio between total bandwidth spent on control bits and total bandwidth spent on data bits [21]. Since we are comparing the performance of PIM-DT and SPIM, and the total bandwidth consumed by data bits are roughly the same for PIM-DT and SPIM in our experiments, we simply use the absolute number of control packets to measure the control overhead. The differences in packet sizes are not considered in the simulation.

Figure 5.8 shows the ratio of the number of control packets generated in PIM-DT relative to the number of control packets generated in SPIM. It demonstrates that PIM-DT with a simple fixed rate refreshing strategy increases the total number of control packets faster than SPIM as the number of active groups grows. The total

number of control messages generated in PIM-DT almost tripled when there are 32 distribution trees in the network and each tree has 8 receivers.

The main cause of the extra overhead is the periodic Request messages sent from the downstream tunnel end point to the upstream end point. In PIM, though the Join messages are also sent periodically, the number of control packets are much less, since multiple Joins can be aggregated and sent in a single packet. In PIM-DT, refresh packets for different tunnels are sent individually. For simplicity, the Joins are not aggregated in the current version of PIM-DT.

The result indicates more sophisticated refreshing strategies are needed in order to contain the control overhead, otherwise the gains in state reduction may be overshadowed by the cost of excessive control packets. The next stage of the dynamic tunnel protocol design focuses on the reduction of control overhead. Possible solutions to the problem are discussed in the next section.

5.2.5 Containing Control Overhead

In this section, we discuss 3 methods that can reduce the control overhead.

Adaptive Refresh Period

Currently, the refresh period for Request messages is fixed. It is possible to adapt the refresh period to the data rate of the flow, and perhaps extend it to consider the length of the tunnel as well. The basic idea of this method is the same as that of the Scalable Timer approach [22], which "fixes the control bandwidth instead of refresh interval".

The periodical Request message has two major functions: first it serves as a "keep alive" message, to inform the upstream tunnel end that the receiver still

requires the data; second it ensures the route is correct and the tunnel tree is aligned with the native tree. A higher refresh rate together with a shorter timeout period will generate more control overhead, but will ensure tree branches that lead to no receiver to be pruned quickly and will cause the tunnel tree to react more promptly to route changes. Lower refresh rates and longer timeout periods can have just the opposite result.

If we assume that the real control overhead should be measured using the ratio between the total number of control bits and total number of data bits, then frequent refresh messages will not be a problem for high bit rate flows. In this case, it is actually desirable to have frequent refresh messages and shorter timeout periods to ensure quick termination of the data flow when the receiver quit from the group, and quicker convergence of tunnel tree topology to native tree topology when the route changes.

For low bit rate flow, the refresh interval can be increased. An upper limit on the percentage of control traffic in the total traffic can be defined, in order to guarantee that low bit rate flows always have even lower bit rate control traffic.

Request Aggregation

Request messages can be divided into two classes, the initial Request messages which are used to setup tunnels, and refresh Request messages which are used to keep the tunnels alive. In the current PIM-DT simulation, the request messages are always forwarded immediately by the intermediate routers.

In fact, it is desirable to forward the initial request quickly, as it may contain digital signatures and time stamps from the downstream requesting router for security purposes which may not be valid if excessive delay is encountered. The

initial Request message can carry an "Urgent" flag to indicate that it should not be delayed. The refresh Request messages on the other hand, usually are not so urgent. We can introduce a "holding time" on each router to let the routers hold the refresh Request packets for some time before forwarding them upstream, trying to aggregate multiple Request Objects into the same packet to reduce overhead.

If the refresh Request messages are to be delayed, we need to change the DTM protocol to make the refresh Request messages untrusted, which means refresh Request messages arriving on a wrong interface or wrong router will not cause new tunnels being established. Instead, a Tunnel Adjust message is returned to the downstream tunnel end point, to trigger another Request Message being sent with the Urgent flag set.

Piggy-back Initial Request in Joins

Another way to reduce the Request packets is to request tunnels from the beginning of the session. Originally we introduced some delay before a router starts sending Request packets to avoid the case that at the beginning of a session, multiple users join the same group at approximately the same time and tunnels are being established and adjusted frequently. As for sparse groups, the chances for two receivers to join the same distribution tree at the same time is very small. In which case, we can allow tunnels to be established immediately after the members join the session. If a combined Join and Request message is received by a router, the join is processed first, then the Request is processed. The processing of the Join and Request are still the same as the procedure defined in chapter 3.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we proposed that by establishing dynamic tunnels, unnecessary uni-multicast forwarding states can be erased. Thus significantly reduce multicast forwarding states, and thereby making large number of sparse multicast groups feasible.

The general architecture of Dynamic Tunnel Multicast is defined. PIM-DT, an instance of DTM with PIM-SM as the underlying multicast support, is specified and is validated using simulation. We confirm, via simulation, that dynamic tunnels can reduce multicast forwarding states.

Simulations also reveals that a more sophisticated refreshing strategy is needed to contain the control overhead.

6.2 Future Works

Following is a list of Future works:

1. Containing Control Overhead

Suggestions for containing control overhead are already discussed in section 5.2.5. Further work is needed to determine which strategy or combination of strategies is best.

2. More Complete Simulations and Tests

   How will the tunnels behave under a full fledged PIM-SM implementation is yet to be examined. Dynamic tunnel over CBT also needs to be analyzed. Simulations of more realistic scenarios are needed. The test network can be larger, have more groups and participants, and can be generated using some existing topology generation packages. The simulation implementation may also need to be ported to ns version 2 at some point.

3. Security Issues

   Control messages such as tunnel Request and tunnel Setup can be digitally signed, so that tunnels can be established only between trusted routers. The most suitable security measurement to be used with DTM is yet to be identified.

4. Dynamic Tunnels with RSVP

   Dynamic tunnels works best for best effort flows and for networks where the bandwidth is abundant. Dynamic tunnels should not be established for flows that require fine grained resource reservations, since for these flows state information has to be remembered by all the routers on the distribution tree anyhow. An interesting question is how dynamic tunnels interacts with RSVP to provide QoS guarantee while at same time keep the state requirement low. This question should be investigated in the future.

Bibliography

[1] Stephen Edward Deering. *"Multicast Routing in a Datagram Internetwork"*. PhD thesis, Stanford University, December 1991.

[2] "Internet Protocol". RFC 791, 1981.

[3] William C. Fenner. "Internet Group Management Protocol, Version 2", May 1996.

[4] T. Pusateri. "Distance Vector Multicast Routing Protocol". draft-ietf-idmr-dvmrp-v3-03.ps, sep 1996.

[5] J. Moy. "Multicast Extensions to OSPF". RFC1584, March 1994.

[6] A. Ballardie. "Core Based Tree(CBT) Multicast Routing Architecture". draft-ietf-idmr-cbt-arch-**.txt, 1997.

[7] Deborah Estrin, Stephen Deering, Van Jacobson, and etc. "Protocol Independent Multicast-Sparse Mood(PIM-SM): Protocol Specification". draft-ietf-idmr-PIM-SM-spec-09.ps, Sep 1996.

[8] Deborah Estrin, Van Jacobson, and etc. "Protocol Independent Multicast-Dense Mood(PIM-DM): Protocol Specification". draft-ietf-idmr-PIM-DM-spec-01.ps, Jan 1996.

[9] Y. Rekhter and C. Topolcic. "Classless Inter-Domain Routing(CIDR)". RFC 1520, September 1993.

[10] A. Ballardie and M. Tatham. "Extending BGP to Support Inter-Domain Multicast Routing (M-BGP)", April 1997.

[11] W. Simpson. "IP in IP Tunneling". RFC 1853, October 1995.

[12] S. Hanks, T. Li, D. Farinacci, and P. Traina. " Generic Routing Encapsulation(GRE)". RFC 1701, October 1994.

[13] C. Perkins. "Minimal Encapsulation within IP". RFC 2004, October 1996.

[14] Ahmed Helmy. "Protocol Independent Multicast-Sparse Mode(PIM-SM): Implementation Document", August 1996.

[15] Vern Paxson. "End-to-End Routing Behavior in the Internet". In *SIGCOMM*. ACM, 1996.

[16] S. McCanne and S. Floyd. The LBNL Network Simulator. software online(http://www-nrg.ee.lbl.gov/ns).

[17] Kevin Fall and Sally Floyd. "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP". *Computer Communications Review*, July 1996.

[18] Sally Floyd. "Ns Version 1 Simulator Tests for Class-Based Queueing".

[19] Steven McCanne and Van Jacobson. "Receiver-driven Layered Multicast". In *SIGCOMM'96*. ACM, 1996.

[20] Dianiel Zappala. Ns Version 1 Multicast Extension. software online(http://netweb.usc.edu/daniel/research/sims/).

[21] Tom Billhartz, J. Bibb, Ellen Farrey-Goudreau, Doug Fieg, and Stephen Gordon Batsell. "Performance and Resource cost Comparisons for the CBT and PIM Multicast Routing Protocols". *IEEE Journal on Selected Areas in Communications*, 15(3):304–315, April 1997.

[22] Puneet Sharma, Deborah Estrin, Sally Floyd, and Van Jacobson. "Scalable Timers for Soft State Protocols". In *INFOCOM*. IEEE, April 1997.

[23] "Introduction to IP Multicast Routing", March 1996.

[24] Stephen Deering, Deborah Estrin, and etc. "The PIM Architecture for Wide-Area Multicast Routing". *IEEE/ACM Transaction on Networking*, 4(2):153–162, April 1996.

[25] Stephen Deering and etc. "Protocol Independent Multicast-Sparse Mode (PIM-SM): Motivation and Architecture". draft-ietf-idmr-pim-arch-04.ps, Oct. 1996.