UNIVERSITY of CALIFORNIA
SANTA CRUZ

SECURE HIERARCHICAL MULTICAST ROUTING AND MULTICAST INTERNET ANONYMITY

A dissertation submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Clay Shields

June 1998

The dissertation of Clay Shields is approved:

Professor J.J. Garcia-Luna-Aceves, Chair

Professor Tracy Larrabee

Professor Darrell Long

Dean of Graduate Studies

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUN 1998		**00-06-1998 to 00-06-1998**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Secure Hierarchical Multicast Routing and Multicast Internet Anonymity	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**102**	

Abstract

Secure Hierarchical Multicast Routing

and Multicast Internet Anonymity

by

Clay Shields

In a computer network, multicast provides an efficient many-to-many service by constructing a delivery tree across all the members of the multicast group. There are a number of existing protocols for performing multicast routing. This work improves the field of multicast routing by presenting a new protocol that can be used to construct a hierarchical multicast tree composed of heterogeneous multicast domains. It also shows how this protocol can also be made secure, so that only authorized multicast members may use the multicast tree to send and receive data. Finally, this work presents multicast as a method of providing anonymity for participants in Internet communication.

Contents

List of Figures

List of Tables

For all my family.

Acknowledgements

This thesis would not have been possible without the outstanding guidance of my advisor, J.J. Garcia-Luna-Aceves. I can't imagine how, in any way, I could have had a better advisor. My thanks also to Tracy Larrabee for her friendship, advice and support. Darrell Long deserves any blame or credit that might go along with turning me loose on the security world.

I also have to thank my friends, particularly the biology geeks, who were a continuing source of humor and sanity. And it was my particular pleasure to have had my close friend and colleague, Brian Levine, going through the same grad school experience with me. I am certain that I would have left long before this work was completed had memory requirements I had to suffer the insanity alone. I am sure I'll memory requirements see him at the shrimp buffet again.

The text of this dissertation includes a reprint of previously published material [48]. The co-author listed in this publication directed and supervised the research which forms the basis for the dissertation.

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Grant Numbers F19628-96-C-0038 and F30602-97-1-0291.

Chapter 1

Introduction

1.1 Multicast Routing

Multicasting is a network service that provides many-to-many communication. A host on the network may use this service to communicate with many other hosts, without the need to transmit separately to each host in the multicast group. Any message sent on a multicast address traverses any network link only once on its way to the receivers, with copies being made and sent at the points where the paths to different receivers diverge. Multicasting improves efficiency in two ways. It conserves bandwidth by eliminating duplicate transmissions, and reduces the processing required of a host that is communicating with many other hosts, as the sending host need only send any message once.

A *multicast routing protocol* is used to determine the paths multicast messages take through the network. These protocols build a multicast tree connecting a source with its intended receivers. There have been many proposals for creating multicast routing trees.. Some assume that there is complete knowledge available about the topology of the network, and use that to construct trees that attempt to minimize the delay incurred in delivering a packet to all receivers [32, 39]. Those used for IP multicast cannot assume complete topology information and must build the multicast

tree in some manner that does not require that information.

There are two general approaches to building a *multicast tree* in an IP network. In some protocols, such as the Distance Vector Multicast Routing Protocol (DVMRP) [17], Multicast Extensions to OSPF(MOSPF) [35] and the Protocol Independent Multicast-Dense Mode (PIM-DM) protocol [18] the receiving group is assumed to be fairly dense. In these protocols the sender initiates the multicast assuming all routers in the network are interested in receiving the transmission and, initially, the multicast is sent to all receivers. If any receiver does not wish to receive the multicast, it must take explicit action and send a message called a *prune* to remove itself from the tree. These prune messages have a limited lifetime; every so often the prune messages expire and the multicast goes to all possible recipients, which again have to prune the branch if they still do not wish to receive the multicast. These types of protocols are termed *sender initiated* as the receivers are not required to take any action to receive the multicast. In each of these protocols the routing tree is formed along the shortest path from a sender to each receiver. This *shortest path tree* attempts to minimize the delay in the delivery of any data packet, but it can create formidable routing requirements as the number of multicast groups and sources grow. The storage overhead at a router on a shared tree is $O(n \cdot s)$, where n is the number of multicast groups and s is the number of sources in the group

Other protocols assume a different approach in forming the tree and the means of initiating reception of the group. In the Core Based Tree (CBT) multicast protocol [6], the Ordered Core Based Tree (OCBT) protocol [47, 46] and in the Protocol Independent Multicast-Sparse Mode (PIM-SM) protocol [21], a single *shared tree* is created for all sender and receivers in the group, and receivers initiate their own connection to the tree. In each of these *receiver initiated* protocols a well known router exists that accepts connection requests from other routers. This router is known as the *rendezvous point* in PIM ; in CBT and OCBT it is called a *core*. The returning acknowledgment builds a branch of the tree back to the initiator along the reverse path of the connection request. Instead of forwarding each packet on a per-group per-source basis, each data

packet is instead forwarded over every on-tree link for that group except the one on which it was received. Accordingly, the router does not have to maintain information about each source for each group and has a single entry for each group. The router overhead is therefore $O(n)$, giving the shared tree approach superior scalability. However, because each such packet no longer travels over its shortest path to each receiver shared trees incur longer average delay in the delivery of a data packet. There have also been protocols developed that use both methods of building the multicast tree, depending on the distribution of members in the topology. PIM [19] switches between these two modes depending on the location of source in the network. MIP [38] starts with a shared tree but relaxes that requirement as necessary to allow formation of shortest-path branches to lower the delay.

1.2 Multicast Security

While multicast routing provides an efficient many-to-many service, it does not limit or control who can use the service — any host can send to a multicast group or elect to receive the traffic from that group. Because of this, most of the efforts to provide a secure multicast service have focused on providing mechanisms for distributing encryption keys to those participants authorized to have them, for authenticating messages, and re-keying the group as required when members join or leave. The first efforts to provide secure multicast [37, 31] were unscalable, because either they required a single server compute the key for a group, or they required extensive knowledge about the group membership. More recently, distributed and scalable methods of keying a multicast group have been proposed [55, 58, 13]. In these protocols, the encrypted data are still available to any interested receiver, and in the case of flood and prune protocols, will actually be sent to all possible receivers, unless they prune themselves from the tree.

This approach to providing security, while viable, is inefficient and vulnerable to attacks against the encryption being used [8, 57], because every interested attacker can have5 access to the data. While it may be possible to use stronger encryption to help thwart this attack, this

approach may be limited by the political and legal policies regarding the encryption use of encryption. Additionally, an attacker may be able to gain useful information without breaking the encryption by using *traffic analysis* [25]. Multicasting also provides an efficient means for a *denial-of-service* attack, through which an attacker can send large amounts of data to the multicast address being attacked. This data is copied and sent to all receivers, possibly creating congestion in the network that prevents legitimate users from being able to participate in the group.

It is therefore desirable to limit participation in a secure multicast group to only those group members authorized to participate. Gong and Shacham [27] were the first to point out the need for some type of authentication and authorization mechanism for multicast. They also clearly stated the goals that a secure multicast protocol design should meet: *compatibility* with existing protocols, *scalability* to the scope of the global Internet, *transparence* to higher-level protocols, *localizablity* for gradual introduction of the technology and *flexibility* to support a variety of policies. However, the authors did not create any protocol meeting these criteria.

The first attempt to provide for authentication and authorization in an existing multicast routing protocol came in some simple extensions to CBT [6] that attempted to regulate access to the multicast tree at the first hop router [5]. Ballardie and Crowcroft pointed out the need for *Secure IGMP*[1], which could present cryptographic credentials from the host to the router. However, their protocols did not meet any reasonable design requirements. To begin with, there was no mechanism for key distribution, and since all authorization was done at the leaf router on the tree, a corrupt router compromised the entire scheme. Additionally, rather than preventing flooding attacks, the protocol attempted to detect and squelch such attacks by randomly sampling packets and, upon detection of unauthorized traffic, sending messages towards the putative source that prevented it from forwarding traffic onto the tree. The problem with this scheme is that it leads to a simple and effective denial of service attack. An attacker, in conjunction with one corrupt router, could send unauthorized traffic that was forged with the source address of the target of the attack. When these

[1] Internet Group Management Protocol (IGMP) [24] is the protocol that hosts uses to communicate with an attached router to initiate its connection with the multicast group.

packets were detected, the innocent target would be removed from the tree, victim of the forgers.

Gong and Shacham [28] examined the problems inherent in maintaining the efficiency of multicast routing while providing a secure service in. This work introduced four methods of reducing the size and number of control messages needed to authenticate group members and to distribute encryptions keys to the group. First, they pointed out that a multicast tree consisted of branches, each of which could utilize different control information than other branches. A node on the tree could then tailor messages for each branch separately, rather than send information needed by only one branch to all branches. Second, they showed that an intermediate node could do some message re-processing, including re-arranging or re-encrypting the message so long as the message's integrity and origin are maintained. This is significant, because it means that a sender does not need to know the topology or group membership to trim control information from a message. Instead, simply knowing a few nodes down different branches, a node can combine this with the first point to tailor messages for several small branches, at the bottom of which the messages are reprocessed for sub-branches. Gong and Shacham also point out that shared tree protocols are ideal for this type of re-processing, because protocols that have distributed cores can use them as natural spots for message re-processing, in effect breaking one flat tree down into a hierarchy of smaller trees, each of which has its own control traffic. Third, they point out that group re-keying need not take place only at the time a member joins or leaves the group, but can be pre-computed; they call this *hot start authentication*. Finally, they extend the idea of hot start authentication to *continuous authentication*, under which each member needs to periodically re-authenticate to receive the current key. These four ideas re-occur in later works in the area [34, 55].

Some of the above ideas appeared in a RFC that again attempted to produce a secure version of CBT [3]. Under this scheme, called the Scalable Multicast Key Distribution (SMKD), the central CBT core is given an authorization list that it uses to verify signed join requests from receivers, each of whom has some public/private key pair. As the tree grows, the access list and a shared group key are distributed along each branch of the tree. The major problems with this

scheme are that no provision is made for re-keying the group should some member leave, and no mechanism is supplied for updating the access list should it change while the group is in existence. The CBT protocol has changed since the time this RFC was released [4]; CBT is no longer a hard-state protocol nor does it support multiple cores. Both the use of multiple cores and the hard state were needed for the scalability of the original key distribution mechanism.

Recently, an application-level implementation of multicast security called Iolus [34] has been proposed. Iolus uses multiple multicast groups, each group with a different multicast key, connected by "group security controllers (GSC)" that re-key and forward traffic between groups. The use of different multicast groups reduces the problem of changing the key, because when some member leaves the group only that group needs to receive new keys, rather than all the groups in the session. Iolus clearly follows the first two ideas presented by Gong and Shacham [28], with control messages being destined for a specific multicast group instead of a particular branch of the multicast tree and with the GSC doing re-processing of the messages that need to travel between groups. While Iolus provides for secure key distribution and re-keying when necessary, it is still necessary to implement multicast security at the routers. Implementing network security at the application level does provide for authentication, in that it gives the appropriate encryption keys to qualified receivers. However, it does not provide for authorization at the network level; therefore, it does not protect the routing infrastructure against unauthorized senders mounting flooding attacks and it does not prevent unauthorized receivers from joining the tree and receiving encrypted data.

Iolus is also inefficient in utilizing network resources. An Iolus session uses multiple multicast addresses. The mechanisms that claim multiple multicast addresses and make sure that the correct ones are distributed to their local area are certain to be more costly than those to distribute a single address globally. Iolus can also lead to multicast packets being duplicated repeatedly over the same link. This is antithetical to multicast routing protocol design, and can occur when the GSC is placed improperly. Because the GSC communicates with different multicast groups, traffic from one multicast group may arrive and be destined to go out to one or more other multicast groups. If the

path to any receiver in another group lies along the same path as an incoming packet, that packet will cross the link again on its way out from the GSC. This duplication can occur a number of times equal to the one less than the number of multicast groups the particular GSC is servicing for the session. The problem arises as the placement of the GSC is not necessarily related to the network topology. Even if care is taken to place the GSC, any receiver obtaining the incorrect address for the local multicast group for the session creates a situation where packet duplication can occur.

1.3 Anonymity

The rapid public acceptance of the Internet as a means of communication and information dissemination is creating previously inconceivable new opportunities for gathering information about individuals. The problem is inherent in the way that packets are transmitted over the Internet. Each packet carries the IP address of the *initiator*, which is the machine that sent it, as well as the IP address of the *responder*, who is the intended recipient [2]. Any *eavesdropper*, a machine that sits on the network along the path that the packet travels, can read this information and can determine which entities (as represented by their IP numbers) are communicating; responders always receive the IP address of the initiator so that they have an address for replies. Therefore, the IP addresses of senders and receivers are known to each other and to eavesdroppers. While IP addresses do not necessarily uniquely identify an individual, it may be possible to link dynamically assigned IP numbers to an individual if they access different services with the same assigned address, or if records are available about whom was assigned which address during a particular period.

By monitoring the IP address of packets that reach some server or that pass through some particular point in the network, an organization can learn about an individual and can use this information in ways that might adversely affect the individual. For example, an insurance company could set up an on-line service area and collect the IP addresses of the individuals that apply for or access information about an account. It could then also set up a seemingly unrelated web site that distributes accurate and useful information about diseases, and then collect the IP addresses of

accesses to that server and match them to the addresses from the first server to learn who may have a disease. The company could thus deny insurance or move to close accounts before costly medical bills accrue.

Similarly, a government may choose to monitor Internet traffic to dissident sites that archive forbidden literature and information, either outside or inside the country's Internet. After learning the sender's IP address, the government might then be able to learn the identities of other dissidents, or even to arrest them. In each of these cases, the individual's privacy has been compromised by the lack of anonymity inherent in Internet communications. The relative ease of associating an individual with an IP address can also limit the effectiveness of anonymous electronic cash schemes. Though the identity of the individual spending the cash is not evident to the bank issuing the funds, the communication channel (e.g., the Internet) may reveal the sender's identity to the merchant, and the merchant may inform the bank. Widespread anonymous communication protocols would allow for the creation of truly anonymous commerce in which the buyer and seller are not known to each other.

1.4 Overview

Chapter 2 presents a new, flexible approach to inter-domain multicast routing. The HIP protocol introduces the idea of "virtual routers" as a method of organizing the control of an entire domain so as to appear as a single router on a higher-level shared tree. HIP then routes between domains using the Ordered Core Based Tree (OCBT) protocol, and allows recursive application of virtual routers to create a multi-leveled hierarchy while providing efficient methods of distributing the location of the center point for the tree. The advantages of this approach include improved robustness, the ability to route between heterogeneous domains, and a significant reduction in the amount of bandwidth consumed in control traffic and in the amount of state stored at each router over existing hierarchical multicast protocols.

We next present a secure version of HIP called Keyed HIP (KHIP), a secure, hierarchical

multicast routing protocol. We show other shared-tree multicast routing protocols are subject to attacks against the routing that can isolate receivers or domains or introduce loops into the structure of the tree. KHIP protects the routing against attacks that could form branches to unauthorized receivers and limits the effects of flooding attacks. Trusted members must obtain cryptographic credentials to join the multicast tree or to receive the proper keys for data transmission. Untrusted routers that are present on the path between trusted routers cannot change the routing and can mount no denial-of-service attack stronger than simply dropping control messages. GNY logic is used to show the proper operation of the authentication between trusted parents and children. KHIP breaks down the multicast routing tree into sub-branches, each of which has a trusted router as a parent and can share a common key for data transmissions across the sub-branch, removing the need for a single encryption key shared across the entire multicast tree.

The current set of Internet routing protocols do not provide any anonymity for the parties involved. Our approach to designing anonymous protocols is uniquely comprehensive, focusing not only on maintaining anonymity for the initiator, but also for the responder, and for the initiator and responder simultaneously. We introduce the use of multicast routing in the design of protocols that provide anonymous communication. Further, we improve upon existing anonymous protocols using multicast and other mechanisms. We show how these and previous initiator-anonymous protocols can be easily modified to maintain responder anonymity, which is useful for applications such as anonymous web serving. A similar approach yields simultaneously anonymous protocols. We present the results of an implementation of multicast-aided anonymity.

Chapter 2

HIP — Hierarchical Multicast Routing

There are two compelling reasons for developing hierarchical multicast routing protocols. The first is to provide a means of routing between heterogeneous domains that might use any multicast protocol internally. The second, and perhaps more pressing, is that the protocol currently being used for multicast routing on the Internet, the Distance Vector Multicast Routing Protocol (DVMRP) [41], is unable to scale with the exponential growth of the Internet. DVMRP computes and maintains its own unicast routing table between DVMRP capable routers; as the number of multicast capable subnets has grown exponentially, so has the size of the table maintained at each router. This table is rapidly growing too large to be easily stored and, unless some solution is found, it will become increasingly difficult for any single host to maintain the routing information it needs to function properly. A well known solution to this problem lies in reducing the amount of information that needs to be stored by routing *hierarchically*. The goal of a hierarchical multicast protocol is to build a single distribution tree across all receivers, but to do it in such a way that the control information regarding the tree is limited to a particular domain or level. A hierarchical scheme can

also allow for heterogeneity of multicast protocols at different levels.

There have been two general ways proposed to build a hierarchical multicast tree: a tree of trees, in which the leaf nodes of a higher-level tree can each be the root node of a lower-level tree; or trees within trees, in which a node of a higher-level tree actually contains a lower-level tree. The first approach was one of the motivations for creating the Ordered Core based Tree protocol (OCBT) [47, 46]. OCBT is a *hard-state* protocol, so once a branch of the tree is constructed, it is not removed until a link failure occurs on the branch or all receivers lower on a branch quit from the tree; this reduces the control traffic required to construct and maintain the tree. Additionally, OCBT uses the existing unicast routing table to make its routing decisions; routers do not need to maintain a separate table to locate other multicast routers. OCBT functions similarly to the Core Based Tree protocol (CBT) [7] in that a router wishing to join the tree sends a join request towards the closest core. When the join request reaches that core or an on-tree router, the receiving router generates a join acknowledgment that traverses the reverse path of the join request and instantiates that branch of the tree. In OCBT, every core and on-tree router also maintains an integer *logical level*. The level of each core is fixed and the level of the router is set by the returned join acknowledgment, which is marked with the level of the core or on-tree router that was reached. When a lower-level core receives a join request and it is not already part of the multicast tree, it must join to a higher-level core, and does so in the same way by sending a join request towards the next highest core. The join request is marked with the level of the core for which it is intended; if it reaches a branch of the tree of that level or higher, then the join acknowledgment is marked with that level and builds a branch of that level back to the sender. If the request reaches a lower-level branch, that branch breaks to allow formation of the higher-level branch. It is this labeling and breaking mechanism that ensures that OCBT remains loop free.

If it were possible to define each logical level of core as an actual level of a hierarchy, it would be an simple task to create such a tree. However, both OCBT and HPIM [29], a similar construction for PIM [18, 21], suffer in hierarchical application from the fact that it is very difficult to come up

with a hierarchical placement of cores or RPs without extensive knowledge of the network topography and the receiver set. This type of hierarchy also suffers from the fact that it is homogeneous; it does not easily provide for multicast protocols or domains of other than OCBT or PIM, except as leaf nodes of the tree.

Most hierarchical proposals therefore use the second method and construct a tree from domains that contain trees themselves. Under this type of hierarchical scheme, a single flat routing *region* is divided into several non-overlapping *domains*, each of which runs its own internal routing protocol. These domains are connected and packets are routed between them by another routing protocol at the higher level. Each member now routes packets only to subnets in its domain, and to reach subnets in other domains each member sends packets to the interface of the higher level of the hierarchy, which forwards the packets to the appropriate domains. The end result is that each member only has to keep routes to the subnets in its domain, resulting in smaller routing tables and lower overhead. This method still creates a single spanning tree; at the highest level it appears as a tree of domains.

Hierarchical DVMRP (HDVMRP) [52] divides the current flat routing region into disparate parts and assigns each a unique name. DVMRP is then used within each routing domain for *intra-domain* routing and between each region for *inter-domain* routing. Data packets travel normally within a domain, but are encapsulated for routing between domains to prevent the packet from being processed as a native data packet, which could result in duplicate packets being delivered to hosts. However, the encapsulated packets might traverse the same links as un-encapsulated packets; this is a weakness because every multicast protocol tries to prevent data packet from being sent multiple times across the same link. In fact, in the worst case, it is possible to have a duplicate packet for every level of the hierarchy in this scheme. HDVMRP also introduces additional network traffic by duplicating the unicast routing. As many more routers become multicast capable, the unicast and multicast topographies in the Internet converge, and the separate routing performed by DVMRP becomes an unnecessary burden on network bandwidth.

Just as using shared trees allowed reduction in router state and in control traffic for a flat routing domain, using shared trees reduces router state and control traffic in a hierarchical scheme. There are other compelling reasons to use shared trees at a higher hierarchical level. Domains are relatively static, and running a separate unicast routing protocol between them can consume bandwidth unnecessarily. Domains are also generally connected by fewer higher-speed links, which reduces the inherent shared-tree problem of link congestion. Recognizing these benefits, the IDMR working group of the IETF has also developed a shared-tree, hierarchical protocol currently called the Border Gateway Multicast Protocol (BGMP) [33, 51]. Along with the Multicast Address Set Claim protocol (MASC) [23], BGMP proposes a mechanism for building associating groups of multicast addresses with particular domains and then building a single level tree of domains to the domain associated with that multicast address.

This chapter presents a new protocol called the HIP protocol (HIP) that uses OCBT as the inter-domain routing protocol in a hierarchy that can include any multicast routing protocol at the lowest level. The goal of HIP is to provide hierarchical multicast routing at a highly reduced cost in terms of network traffic and storage requirements at routers. The HIP protocol introduces the idea of a *virtual router* (VR) that is formed by all border routers of a domain operating in concert to appear as a single router in the higher-level tree. Virtual routers provide a convenient level of abstract on and enhanced robustness, while providing a data delivery service that does not duplicate packets over any link. A domain can use any multicast protocol internally while acting as a virtual router on the higher-level tree. OCBT is used to route between actual and virtual routers at a higher level in the multicast routing hierarchy, and if more hierarchical levels are required, an OCBT domain containing virtual routers can itself be made into a virtual router for a higher-level tree. Additional functionality gives HIP the option to allow a convenient and efficient method of center point location[1]. HIP's functionality comes primarily through the operation of a domain's border routers, and is therefore amenable to easy deployment by aligning the multicast domains with

[1] Note that we designate the router that forms the root of the tree as the center point (CP) rather than the core to avoid confusion with cores that may be local to a single domain and do not effect the entire tree.

existing domains. HIP always attempts to maintain short paths by choosing the border router with the shortest path to the center point to provide connectivity for the entire domain. Furthermore, HIP also provides additional robustness at the highest level by replacing a single center point of the shared tree with several routers that operate together in a distributed fashion and can tolerate failure. HIP also provides a possible framework for center-point location distribution at a domain level.

The next section describes the construction and operation of virtual routers and how they are used to build a single level of hierarchy. Section 2.2 details HIP's recursive application of OCBT to create a multi-level hierarchical tree and mechanisms for center-location distribution. Section 2.3 presents an example of how HIP distributes center-point information and builds a hierarchical tree. Section 2.4 offers a proof of HIP's correctness.

2.1 The HIP Protocol

The correct operation of OCBT is not dependent on the order in which messages are received at the router, though the resulting multicast tree may differ in shape from a tree produced by a different ordering of messages. This property allows us to operate an entire domain such that it can accept the same input messages as an OCBT router and produce correct, if not always identical, output. By carefully organizing the *border routers* (BR — those routers that define the edge of the domain and "speak" the protocol of both the internal and external domains), the input arriving on the external interfaces can be processed collectively in such a way that the output of the virtual router can simulate the output of a single router in place of the domain. This organization turns out to be very simple. For each multicast group, one border router is selected to be the *controller* for the group. This selection can be dynamic, with all border routers agreeing by communicating over an All-Border-Routers (ABR) internal multicast address, or it can be pre-configured. All other border routers, called *subordinate routers*, forward all virtual router control traffic to the controller, which keeps all state for the virtual router and sends instructions back to the border routers. The

controller can communicate with the border routers over individual TCP connections to insure that control messages are not lost, or over the ABR address using some reliable multicast protocol. A simple analogy can be drawn between an OCBT router and an OCBT virtual router: a normal router keeps all the routing state and sends messages to its neighbors via given interfaces; a virtual router controller keeps almost all the routing state and communicates with it neighbors via given border routers. The only state that subordinate routers maintain is their own list of on-tree interfaces so that they can forward data as it arrives without it going through the controller. The subordinate routers do not change any state on their own. Only the controller makes state changes and notifies the subordinates when to change their state.

When a virtual router is part of the tree, the controller selects which subordinate will serve as the *exit router* to connect the domain to its parent on the higher-level shared tree, and always chooses for this role the border router with the shortest path to the given center point. Selecting the exit router in this way helps maintain short paths from lower to higher levels. The controller chooses the exit router by sending a message asking each BR to query its routing table for the distance to a specified address, and then chooses the BR with the shortest reported distance as the exit router. Border routers that have on-tree links for a particular group outside the virtual router must join the same multicast address internally. This way, when data are received from outside the VR, it is forwarded across the internal multicast domain to other border routers and then onto the other links attached to the virtual router. Data flows just as it does on a single-level shared tree. The actual internal operation of the virtual router will differ depending on the internal multicast protocol used. Source-routed protocols, such as DVMRP and PIM-DM, use reverse path forwarding checks to accept or drop a packet. Data may arrive and be distributed from different BRs than those that pass the reverse path forwarding (RPF) checks. In these cases, some data packets might need to be encapsulated and forwarded to the correct border router for distribution internally so that the RPF checks are passed. Shared tree protocols, such as PIM-SM and CBT can use whatever core dissemination mechanism they choose internally, and the border routers can connect or send

directly to core or RP as needed. For OCBT, the border routers themselves can be used as cores, with the exit router serving as a level two core and all other BRs as level one cores. This way, the internal routers can be pre-configured to contact any BR as a core, and the tree formed from there as required. This is illustrated clearly in Section 2.1.2.

There are advantages and disadvantages to the virtual router approach. Clearly, a message arriving at a border router needs to travel to the controller and the reply returned before the subordinate can reply for the VR. This increases the response latency of the virtual router. For an OCBT virtual router, the maximum incurred delay will the twice the round trip time from the receiving BR to the controller (as the control message is forwarded and an answer received), plus the maximum round trip time from the controller to the furthest border router (if the controller needs to query the subordinates to receive routing distance information). While this latency grows as the domain size increases, there is a simple trade-off between bandwidth and latency that might help. Each time a border router receives a message likely to cause the controller to issue a request for routing information, that border router can transmit the address to be looked up on the ABR address as it forwards the control message. When the controller receives the control message, it then needs only to wait for the replies and does not have to send out a request itself; unnecessary replies can just be dropped. This will lower the latency to twice the round trip time from the receiving BR to the controller, plus a time less than or equal to the maximum time from the receiving BR to any other BR plus the maximum time from any BR to the controller.

The other disadvantage of a virtual router set up is that the overall paths traversed by a link can be made much longer by the need to traverse a virtual router. If a connection to a center point encounters a virtual router, then the path needs to traverse the VR, even if a short path is available. This seems to be a problem with any shared tree hierarchical protocol; as a particular domain may have only one parent on the shared tree, some receivers within the domain or connecting through the domain will have worse paths than they might have in a flat routing scheme. This is the design tradeoff – lower state and lower control overhead in exchange for longer paths, just as might

be intuitively expected by changing from a source tree to shared tree. The actual delay incurred by the longer paths is dependent on the network topography, though it appears that the difference between a source tree and shared tree might not be as significant as intuitively expected.

While using virtual routers can increase join latency and transmission delays, these are known design trade-offs to gain the benefits that virtual routers can bring. Virtual router provide a convenient level of abstraction in building hierarchical protocols. Many protocols exist that provide flat multicast routing and there is significant experience with these protocols. Using virtual routers allows these protocols to be used in an inter-domain role without significant modification. Additionally, virtual routers can be configured as desired on a policy basis; the domains that are routed between are configured manually rather than automatically. While this imposes a small burden in configuration it should not be excessive, as most organizations are fully aware of their domain boundaries and where the border routers are. This can be a major benefit as the virtual router can be made to use policy-based unicast routing scheme, such as BGP [44], to route to other domains. The scheme is also flexible enough to allow for virtual routers within virtual routers if necessary. This can be used for as many levels of hierarchy as needed or desired, as demonstrated in Section 2.2. Finally, use of virtual routers can significantly reduce the amount of state maintained by routers and the amount of control traffic between them, depending on what routing protocol is used between domains. To this end we propose using OCBT as the inter-domain routing protocol.

2.1.1 OCBT as an Inter-domain routing protocol

OCBT has several properties that make it a good candidate for use as an inter-domain routing protocol. Besides being proven correct and loop free at all times, OCBT relies directly on the unicast routing protocol used to route between domains, so that policy-based routing such as BGP can be used. OCBT is also a very simple protocol, and can be easily represented by a finite state machine as shown in Figures 2.1 and 2.2. In these figures, the possible transitions are marked with the type of message that causes the transition on top and the type of messages the router

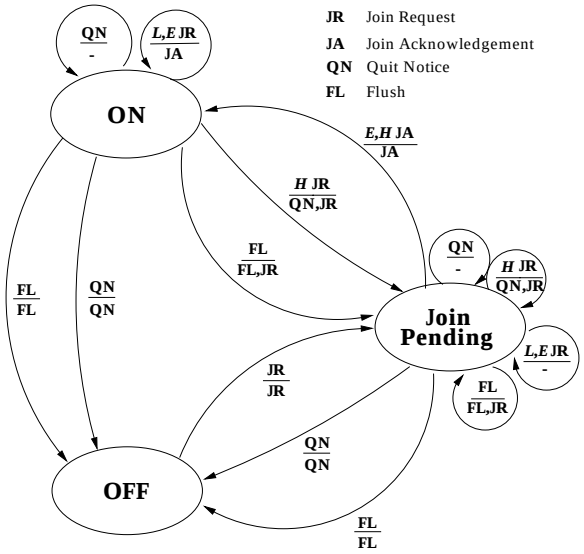


Figure 2.1: OCBT state transitions for a non-core router.

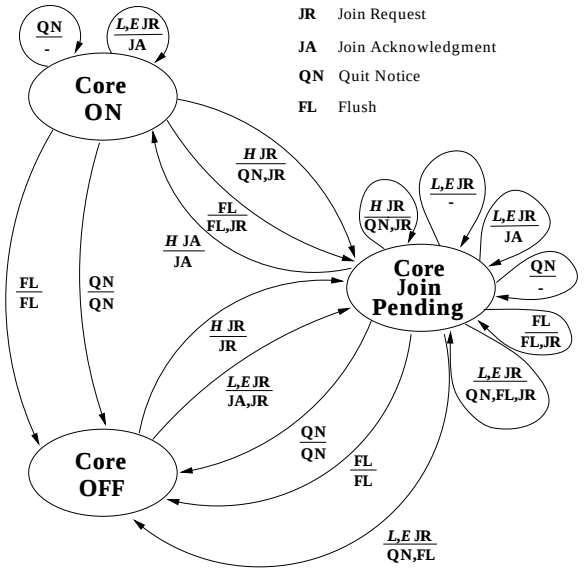


Figure 2.2: OCBT state transitions for a core router.

emits before changing state on the bottom of the transition label. The italic *L,E,* and *H* indicate whether the level of the received message is of logical level lower, equal to or higher than that of the receiver; a comma means any of the listed levels. Under normal operation, by which we mean in the absence of network partitions, each OCBT router is in one of only six states, and there are only four

types of messages related to the construction and destruction of the tree. Each router also sends periodic keep-alive messages to its parent to ensure that the link is alive and that the state between routers is consistent. The messages themselves are very small, the largest containing two integers representing the message type and message level and three IP addresses representing the message source, the message originator (if the source is forwarding a message) and the core for which the message is destined. This is 14 bytes of information, and most messages are smaller; many consist of only six bytes of control information.

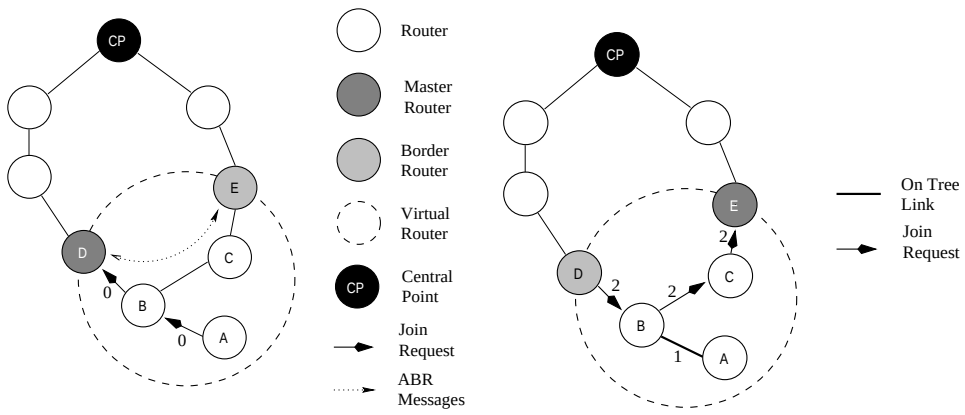
Internally, each OCBT router needs to keep very little state for each multicast group – two integers representing the routers state and level; the address and interface over which to reach the parent router and, if a core, an integer representing the level of the parent; a list of addresses of cores which will vary in size depending on the scheme used to disseminate core information and a single address of the current core; a list of the level, address and interface for each child directly attached to the router and an integer for each child to track keep-alive messages; and only two timers for generating retransmissions and to track keep-alive messages. Assuming four byte IP addresses and single byte integers for interface labels and for tracking levels, a core with five children and a list of five possible cores (a reasonable value under the core scheme in which each border router is an OCBT core) will therefore need only 66 bytes of storage and two timers for that group, not counting such things as pointers for the storage data structures. The state that the controller of an OCBT virtual router would need to maintain is similarly small. Instead of maintaining children based on their ID and interface, the controller would also need to maintain which border router was parent to the child. This would add only one byte, given a list of border router addresses at four bytes each but usable by all groups, that the single byte could be used to index. In addition, the parent interface needs the same index. The controller also needs to use the keep-alive timer to send and receive keep-alive messages over the ABR address, and needs to keep one byte for each border router to track missing keep-alive messages. Therefore, if there were five border routers (the same five that the lower-level routers used as a core list) and there were still five cores for the VR, this

would still total less than 100 bytes of storage for the multicast group. This state can be further reduced by maintaining child lists and the parent identity by interface only and omitting the child ID; this is certainly very effective if there is only one child per link, but requires some modification to the protocol to handle broadcast links with multiple group members. This modification would remove the child ID list and reduce the OCBT state for the example given to only 39 bytes and the VR state to 50 bytes, given the same number of VR children and border routers.

2.1.2 An Example of an OCBT Virtual Router

This section shows how a virtual router might operate for a single OCBT domain on a higher-level tree. Although HIP can support any multicast routing protocol running in a lowest-level domain, HIP uses OCBT at the higher levels so its operation is worth understanding. In Figure 2.3(a) the example first shows a receiver initiating the join for its domain and the controller determining the master router for the multicast group. Figure 2.3(b) shows how OCBT can create a tree between border routers to "redirect" a join request from a subordinate to an exit router. The sequence in Figures 2.4(a) and 2.4(b) shows how the virtual router acts as a normal OCBT router to join the multicast tree at the higher level.

In the example topography, routers D and E are the border routers of the OCBT domain and the virtual router consisting of the border routers and the other single-lettered routers. At some point in time, as shown in Figure 2.3(a), router A determines that it has members on some subnet that wish to receive a particular multicast group that has a given CP address associated with it. Router A initiates a normal level-zero OCBT join request for the designated CP. This request travels to router B, which consults its unicast routing table for the best next hop towards the center point. Though there are two equal paths, B chooses the next hop according to its routing table and happens to select the border router, D, as the next hop. In this case the border router is not yet part of the multicast tree. So far, this is all normal OCBT operation, but when the request reaches the border router, HIP causes a slightly different behavior. Because any router on a shared tree



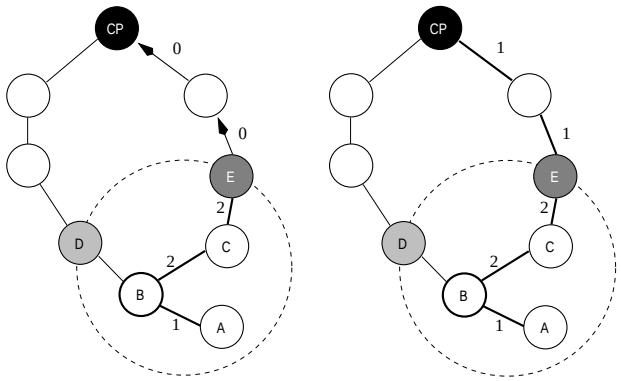
(a) Interior Join

(b) CP Redirect

Figure 2.3: Interior OCBT Virtual Router Operation

can have only one parent, and because HIP attempts to build shortest-path trees, the virtual router controller must determine which will be the exit router for that multicast group based on proximity to the groups designated center point. The controller sends a message requesting both the border router D and E's unicast distance to the given center point. The subordinate routers reply with that distance (if and only if its next hop to the center point is outside of the domain, otherwise it returns a message that it has no external path), at which time the controller determines that E is closer to the center point, and is thus chosen to be the exit router for the domain to that CP.

The border routers are now configured to build the shortest-path tree for the domain; however, the join request from A that started the process has reached the wrong router to reach the CP through the chosen exit router. This is not a problem because we use OCBT. In every OCBT virtual router, the subordinate routers act as a level-one cores while the exit router acts as a level-two core. Figure 2.3(b) shows how the router D simply sends a level-one join acknowledgment back to router A, instantiating a level-one branch from the subordinate router to A, then sends a level-two join request towards the exit router E to form a branch from itself to E so that it can deliver data on the level-one branch. As it happens, the route from the master router to the subordinate traverses



(a) Ext. Join (b) Ext. Reply

Figure 2.4: Exterior OCBT Virtual Router Operation

part of the level-one branch. Because OCBT breaks existing lover-level branches to allow formation of higher-level ones, the link between D and B becomes part of the level-two branch that is forming along the path from B to C to the exit router. Router A still receives data from B over the level-one link once the level-two branch is in place, which occurs when the exit router receives the request and sends the appropriate join acknowledgment back to B. At this point, D realizes that it has no children and does not want to receive this multicast group, so it quits from the tree resulting in the internal tree structures shown in Figure 2.4(a).

The virtual router still must join the external tree. This is straight forward; the exit router sends a normal OCBT level-zero join request to the CP; as shown in Figure 2.4(b), the CP replies with a level-one join acknowledgment that forms the branch back to the virtual router. If in the future, the subordinate router D receives a join request on an external interface, it need only re-send the internal level-two join request towards the exit router to rejoin the internal tree.

2.2 Hierarchical Routing and Center Point Dissemination

The previous section shows that HIP can be used to build a single level hierarchy, first by constructing virtual routers from domains and next building a shared tree, without discussing how to provide multiple levels of hierarchy or the details of how the center point location is distributed or even what the center point is. In HIP, it is a simple matter to provide multiple levels of hierarchy if desired or needed. Assume that a large organization runs it own network. Smaller parts of the organization each have their own subnetworks which run different multicast protocols internally. In order to route between the subnetworks, the organization has made each subnetwork its own virtual router and is routing between them internally. Now the organization wants to route multicast traffic to other organizations, requiring another level of hierarchy. The recursive application of the virtual router protocol solves this problem. The organization places its entire network into a single virtual router and places that virtual router on the shared tree between organizations. A three-level hierarchy is now in place without making any modifications other than at the border routers of the organization. This recursive application can be applied as many times as necessary or convenient. Distributing center points for higher-level shared trees requires some additional work. The following sections describe center points and their dissemination in detail and give an example of the recursive application of HIP and its CP mechanisms.

2.2.1 Virtual Center Points

The hierarchical nature of HIP dictates that a domain that contains a CP within itself must become a virtual CP for the shared tree at its level. The shared tree at each level will then have a CP with which to connect. To see that this is necessary, consider the arrival of a join message on the external interface of a border router destined for a particular internal router that is designated as the physical CP for a multicast group. Forwarding the request to join inward across the domain border and onto the inner shared tree would be a violation of the hierarchy. Therefore, the border routers must reply to the join message as if they were the CP. The ability to serve as a virtual CP is

already a necessary part of the encapsulation protocol that completely simulates a single router, as being a core is one of the possible router states. After acknowledging the external request, the border routers then can send a join message inwards towards the real CP, which may connect directly or may be received by some interior virtual CP. Border routers must be able to determine which center points are within their domain in order to recognize requests destined for an internal center point.

HIP's use of virtual center points is a major advantage when router failures are considered because each virtual router can be made up of several physical routers that can provide redundancy. A single CP is a likely point of failure in a shared-tree protocol, and though such protocols take this eventuality into consideration, the solutions to the problem can be expensive in terms of overhead and network traffic. Because the virtual router is made up of several border routers that work in concert to provide the same service, the failure of one or more of the border routers does not constitute failure of the virtual CP, and remaining border routers can maintain the functionality.

2.2.2 Center Point Location Distribution

A disadvantage of receiver initiated multicast protocols is that a receiver must be able to determine the location of the center point associated with the group address in order to join that group. There are four approaches to distributing this information [22]: advertisement at the application level; advertisement at the router level; a distributed directory supporting center point lookup; and algorithmic mapping. Each has advantages and disadvantages; application level advertisements can be protocol dependent, distributed lookups can introduce startup delay for receivers, router level advertisements require a significant change to the multicast model and algorithmic mapping can require significant network traffic to implement and can create very poorly shaped trees over many levels of hierarchy. While there is no best answer to CP location dissemination, HIP uses a combination of several approaches that minimizes network traffic, reduces the information that any individual router must have, and limits the delay before a member of the multicast group can start transmitting. HIP does not require that this particular method of CP location be used if other

methods are made popular. For example, the Multicast Address Set Claim [23] provides a method of distributing a mapping of multicast addresses to particular domain which act as center points and could be used with HIP without significant modification; however our methods allow for easy mixing of sender and receiver initiated protocols.

To help distribute center point locations effectively, HIP defines an additional multicast address within each higher-level domain. This address, known as the all-virtual-router (AVR) address, can be subscribed to by any virtual router that encapsulates a sender-initiated domain at the lowest level. Any domain that has internal AVR receivers joins the AVR address in the higher-level domain. Using the ABR and AVR addresses across the hierarchical domains, we can create an efficient means of distributing information about available multicast groups and their CP locations. To minimize the amount of traffic required to distribute the group address to center point location mapping, we combine two approaches. In the first and most conservative approach, an advertisement with the mapping is transmitted by the new CP (or by the creator of the multicast group) to the ABR address. One of the border routers receiving the advertisement caches it and then forwards it on its external ABR address. This process continues until the advertisement reaches the highest level, where it is either flooded across the entire level or delivered to some directory service available to the highest level routers and virtual routers. Each router is configured with the addresses of its border routers; when a receiver wishes to receive the multicast group, its router sends a join request towards any border router. The border router that receives the request acknowledges it and then commences locating the proper center point of the group. If the receiving group of border routers does not contain the CP information in their cache, the controller for that domain creates and forwards a CP location information request towards the higher levels on its external ABR address. As the border routers of each higher level receive the lookup request, they check their distributed cache and forward the information back to the requester or forward the request up. This process continues until the request reaches the top level or a level where some border router has the information cached. The border router at the level containing the information sends it down along the reverse path of

the request. The reply is cached by each successive set of border routers and forwarded along the reverse path until it arrives at the requesting virtual router. That virtual router is then free to join the tree using a process discussed in Section 2.2.3.

This approach limits the amount of network traffic needed, and only receivers requesting the information actually receive it. The information is also cached at each level, as it is reasonable to expect other nearby receivers may try to join the group, and having the information cached will reduce the number of requests traveling upwards in the hierarchy. There are two drawbacks to this approach, however. First, each virtual router must wait for CP information before joining the tree, increasing join latency for the domain and requiring that the border router cache any data being sent from internal members that are sending data. Second, a receiver must initiate the join, which is fine for the shared-tree protocol used at the higher level, it will not work for sender initiated domains (e.g. DVMRP or PIM-DM) that may be operating as a virtual router at the lowest level. These domains must receive an announcement that the group is available. Thus, a more widespread distribution of group information is required to accommodate these protocols.

In the second approach, information about the availability of a group and its center point is multicast to all border routers in the region. The CP does this by announcing it self on both the ABR and AVR addresses within its domain. One border router for the VR receiving such an advertisement on an internal interface also forwards it to both the external ABR and AVR addresses; if the advertisement is received on an external interface (as a result of being sent to the AVR address) it is forwarded to the internal AVR address, with the same elected border router address described above added to aid the joining process. The information travels up to the highest level via ABR transmissions; it travels across the highest level and to and into each domain as a result of the AVR transmissions and eventually reaches all domains in the region. A domain containing a sender initiated protocol will have the advertisement (which can simply consist of the first data of the session) flooded within it from one border router, and if the group is not pruned back entirely, a join request will be generated by the virtual router for the group. A border router from a receiver-

initiated group wishing to join can request the CP information via the ABR address and then commence the join process. If the request for the multicast group information results in a cache miss, the border routers can revert to the first method and forward the request upwards until the information is discovered. While this protocol is robust and addresses the need for sender-initiated protocols to be notified about group availability, it can be very wasteful in terms of excessive control traffic if there are few sender initiated domains.

In HIP, the method of CP information combines both approaches; receiver initiated lookup for areas with few receivers and without sender initiated domains, and multicast distribution for areas needing robust delivery. Only virtual routers containing a sender initiated group or a set of border routers that process too many requests for CP information need join the AVR address on its external interface. Any domain with an internal receiver on this group must become a receiver on the same address on the higher-level shared tree. This continues up to the highest level; advertisements for service arrive at and get sent across the highest level and down to each AVR receiver. Domains not subscribing to this group use the first request mechanism instead.

Limiting the area of a multicast via administrative *scoping* is very simple in HIP. The original advertisement announcing the availability of the group can also contain the name of the highest domain that can carry the multicast. When the advertisement reaches this level it is not forwarded to the higher level. Receivers outside the scoped domain will not be able to receive information about the group and hence will not be able to join. The information about a scoped group must be maintained at the highest scoped level until the group expires however; if the information were allowed to drop out of the cache there would not be any information about the group at a higher level to answer requests.

2.2.3 Building the Tree

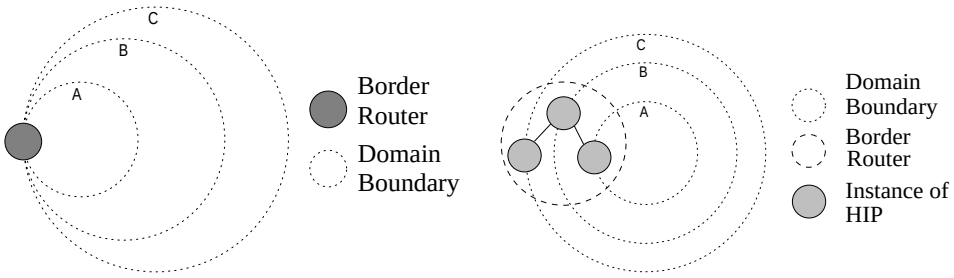
With a mechanism for CP information distribution in place, the process of building the tree can begin. For lowest-level domains using a source initiated internal protocol, this occurs when

the group advertisement is flooded throughout the domain from one elected border router and is not pruned completely back, indicating that some internal member wishes to receive the multicast. In a receiver initiated protocol, it starts when some internal router receives an IGMP [24] message from a subnet requesting the multicast. This physical router sends a join request to some border router of its domain and connects to it. If the border routers do not have the correct CP information, one retrieves it via the method described in Section 2.2.2. If the contacted border router turns out not to be the exit router for that group for the domain, it can form a connection to the exit router as shown in Section 2.1.2. The exit router then sends a join request for the group towards the appropriate center point.

This join message reaching the next higher set of border routers triggers that virtual router to join the forming tree itself. In this case the virtual router already has the information needed as to where to target its join message; it was forwarded and cached previously as a result of the internal router joining. If a virtual center point receives the request, it responds as a real CP would. The receiving border router then sends a join inward towards the actual CP. This join could pass through several other virtual CPs before reaching the actual CP; each responds as if it were the actual and then joins the group internally as well. No border router joins the internal multicast group unless it is necessary.

2.2.4 Border Routers for Multiple Domains

All of our explanation and examples of HIP have assumed that each domain has a single border router that operates for that domain only; that is, no single border router serves multiple domains on either side. In practice this need not be true. While HIP does require that a lower-level domain be enclosed by the higher-level domain, should a higher-level and some lower-level domains share a border router, a few special steps need to be taken to ensure that the protocol operates properly. Figure 2.5(a) shows a single border router that serves three separate domains, A, B and C.



(a) Physical Configuration (b) Software Configuration

Figure 2.5: Multiple domains sharing a common border router.

The border router runs multiple instances of the virtual router protocol, one for each domain for which it is a border. This is shown in Figure 2.5(b), in which each instance of the protocol serves one domain and all instances are organized into a stack, with the instance serving the lowest level domain on the bottom and each successive instance serving the next higher domain. This simulates each domain having its own border router that is connected directly and only to the next higher domain's border router. When traffic arrives at the bottom-most instance of the protocol that would normally be forwarded along to the next higher domain's border router it is instead simply passed up the protocol stack as appropriate. This allows as many domain boundaries to sit together as necessary. This method does require that messages from border routers be marked with which domain the border router is in, so when it arrives at another border router serving multiple domains, the message can be sent to the appropriate instance of the protocol.

2.3 An Example of HIP in Operation

While the actual ideas behind HIP are fairly simple, its recursive organization can make it difficult to envision it in action. This section presents an example showing, through a series of figures, how the protocol distributes CP information and constructs a shared tree. Figure 2.6 shows the example topography used throughout the example, and is followed by Figure 2.7, which shows

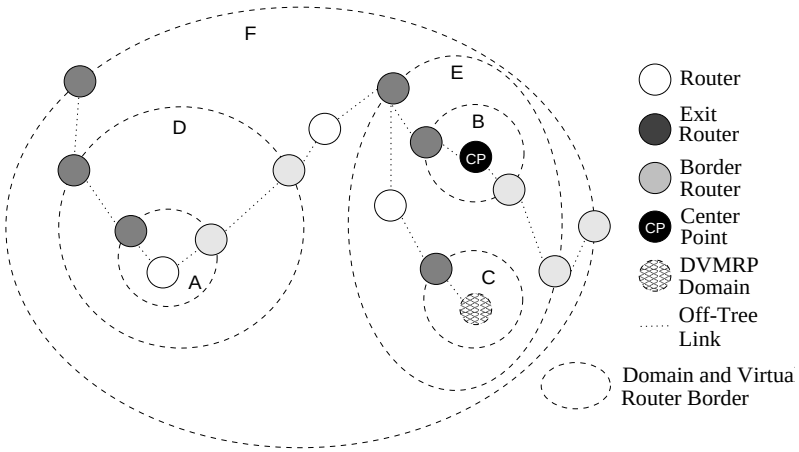


Figure 2.6: Example Topography

how the center point location is disseminated throughout the network. Once the center-point is known, individual domains are able to join the multicast tree. A sender-initiated domain's joining process is presented in Figure 2.8 and a receiver-initiated domains process of locating the center point and joining the description in Figure 2.9 and Figure 2.10.

Figure 2.6 shows the simple topography used. Virtual router A is an OCBT domain that does not subscribe to the all-virtual-routers address to receive CP information, while C contains a DVMRP domain that must be notified when a multicast group becomes available and therefore does subscribe to the AVR address. Virtual router B contains the physical CP for the group, making both E and F virtual center points for their levels. The exit routers are not aligned in any particular configuration initially; once they receive appropriate CP information they will switch to lie on the shortest path. In practice they need not even be identified until needed, however for illustration purposes they are selected and made visible.

The setup of the multicast group begins with the distribution of center point information. The new CP announces its availability on both the all-border-router and all-virtual-router addresses. The all-virtual-router announcement travels only within domain E to C as there are no subscribers to the AVR address in the higher-level domain F. The advertisement on ABR travels out of domain E to domain D, where it is unicast across the domain and forwards it to the exit router for the

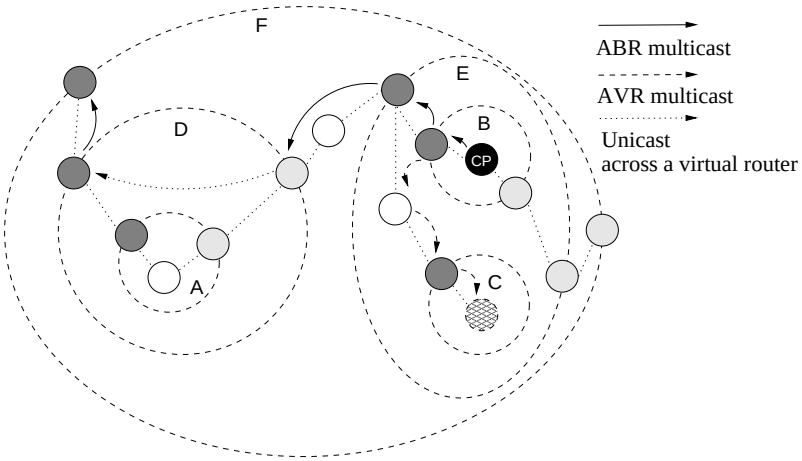


Figure 2.7: CP Location Dissemination

overall domain labeled F so that it can be forwarded towards the highest level its scoping permits. The route that the CP information follows is shown in Figure 2.7. Notice that domains A and D only forward and do not cache the CP information as they are not ABR receivers, but only hops on the shared tree at that level.

Once the CP advertisement reaches the DVMRP domain, the exit router of that domain (not illustrated) floods the advertisement through the domain. When the tree is not pruned back, the controller has the exit router send a join message towards the primary CP address received in the CP advertisement. This request travels first to the exit router of domain C, where it is acknowledged and forces the issuance of a join request for the domain. This higher-level request goes directly to the exit router for domain B. It does not need to be acknowledged by the border routers of domain E, as B is the virtual CP within E and E sees the request pass by on an internal interface. The join message is acknowledged back to C, and the exit router for B then joins the internal physical CP. This process is illustrated in Figure 2.8.

When the single router in the A domain receives an IGMP message from an attached subnet that desires the multicast, it selects a border router and joins to it for that multicast group. The controller for A must then locate the correct CP for the group as even though it traversed

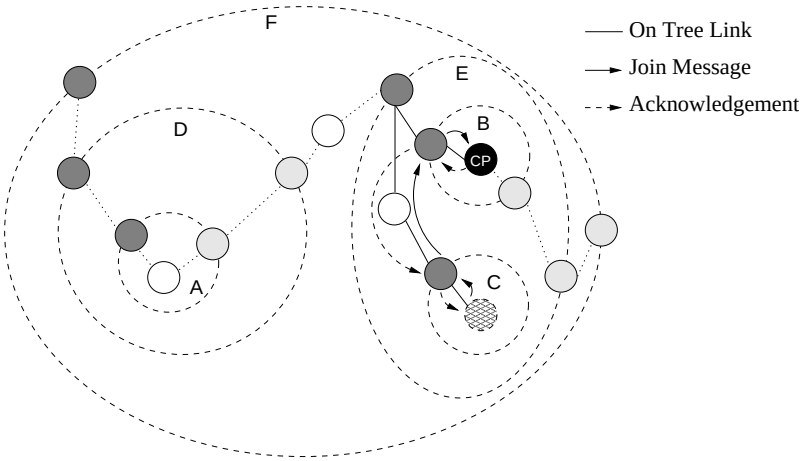


Figure 2.8: The DVMRP Domain Joins

the domain earlier, it did so as an encapsulated control packet and was not read or cached in A. Figure 2.9, shows how A sends a CP information request to the ABR group of its domain, D, which gets received at that domain's exit router. Because the original core advertisement passed through the D domain without getting read or cached, the controller for D must also forward the request to a higher level. The information is found when the request arrives at the F controller and is sent back down to D. There the controller consults the border routers confer and realize that the exit router must be switched to provide a shortest path to the indicated CP. The CP location is forwarded to A and the exit router for that group is switched.

At A, again, the controller realizes that the other border router must be the exit to provide the best exit point for the domain, and the role is transferred. The former exit router then redirects the initial join from the internal router by sending a level-2 request to the new exit router. This triggers the controller to have the new exit router to send a join request for domain A as shown in Figure 2.10. As in the join from the DVMRP domain, it gets acknowledged at the border router on the way out of domain D, then travels to domain E, where it encounters a border router that is already part of the tree that sends the final acknowledgment back to the D master router, completing the branch.

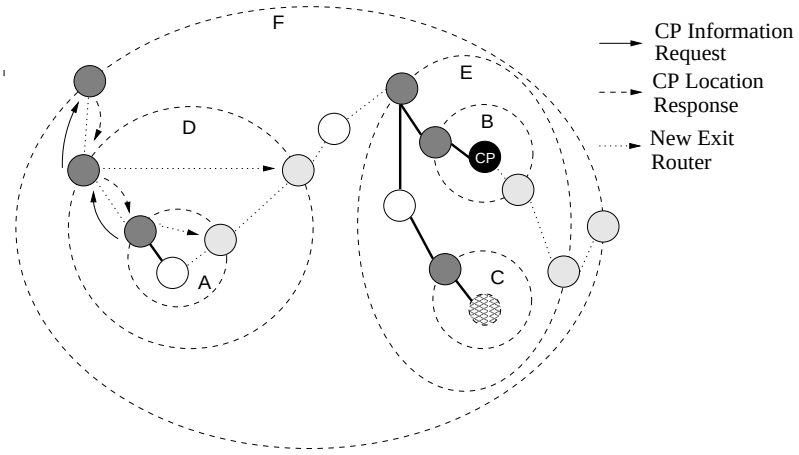


Figure 2.9: Receiver Initiated CP Information Request

2.4 Correctness of HIP

Proving that HIP is correct requires us to show two things; first, that OCBT always builds a multicast tree in bounded time without loops; and second, that the virtual router protocol correctly emulates the function of a single OCBT router. Recursive application of these proofs will then show HIP to be correct at all levels. First, OCBT has been previously proven to operate correctly; it always produces a multicast tree in finite time even under unstable unicast routing and link and router failure [47]. The proof relies on the fact that timers are used to generate retransmissions, so OCBT is always live, and that the use of logical-level labels guarantees that branches will form while preventing loops in the construction of the tree. It therefore remains to show that the virtual router protocol is never deadlocks and that it always correctly emulates the function of an OCBT router, even under internal link or router failure.

Theorem 1: Given a connected network, the virtual router protocol remains safe and live and always correctly simulates an OCBT router.

Proof: It is easy to show that the virtual router protocol does not reach deadlock as each message that requires a response has associated with it a timer. If the timer expires without a reply being received, the message is either resent or, after several failures, it is assumed that the

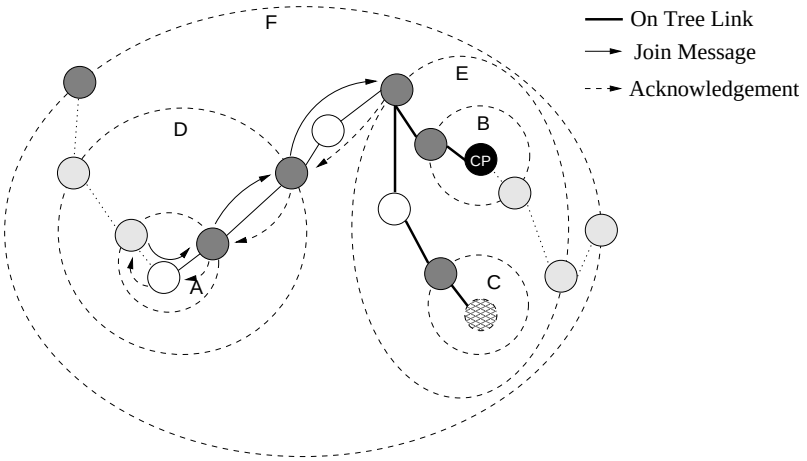


Figure 2.10: Receiver Initiated Join Process

intended recipient is unreachable and appropriate action taken. In either case, deadlock is avoided. To prove that a virtual router has the same functionality as a OCBT router only is slightly more complicated and requires that the responses, in terms of changes of internal state and transmission of control or data messages, be identical for both a virtual router and an OCBT router. This is also very simple, as the virtual router runs the same protocol and keeps the almost the same state as an actual router. Instead of tracking children by interface, the virtual router tracks them by border router and interface, and the subordinate routers keep state about data forwarding as directed by the controller. In addition, the routing table lookup of a router is replaced by the distributed lookup of the virtual router; since these transmissions and their replies go over reliable unicast or multicast channels the reply is assured as long as the routers are up. Therefore, this lookup will succeed and the virtual router can forward messages based on the distributed routing tables. When the pseudo code for OCBT and an OCBT virtual router is compared and state transition diagrams are prepared, they are identical. A virtual router therefore behaves the same as an actual router would in its place. □

Given that a virtual router behaves the same as an OCBT router under normal conditions, we now need to examine its performance under failure. There are three ways a link or router can fail

in a domain. The failure can occur such that either the ABR multicast tree or the data multicast tree is disconnected but paths exist between all border routers; a border router can fail; or a partition can separate the border routers into separate connected components. The following three lemmas show that HIP functions correctly in each of the three cases.

First we make some simple assumptions about the nature of the internal multicast routing protocol. First, we assume that if there exists some route between all border routers, the internal multicast routing protocol will rebuild the multicast tree in some finite time following a link or router failure. Second, if the domain is partitioned and there is no possible route between all border routers, we assume that the internal multicast protocol will build an ABR and data multicast tree within each partition, so that any border routers within the partition will be able to communicate, and that the tree will be rebuilt when the partitions merge. These are correct assumptions for all OCBT domains, as OCBT correctly builds a tree in finite time in any connected component following a failure or partition [47]. If, in fact, the internal multicast cannot support this, the virtual router protocol encapsulating that domain will need additional functionality to fulfill those requirements.

Lemma 1: In the event of an internal link or router failure such that all border routers have paths to each other, the virtual router protocol will continue to operate correctly.

Proof: In fact, in this case HIP needs to do nothing. The internal multicast protocol is able to reconstruct either or both the ABR and data tree as a path exists between all border routers, and the controller will be able to communicate and share data and control messages with all border routers. There may be internal routers that are not able to receive or send on those addresses, but that does not effect the function of the virtual router as a whole; it is still able to make appropriate state changes and to carry data across the virtual router for transmission on the higher-level tree. □

Lemma 2: In the event of a border router failure, the virtual router protocol will continue to operate correctly.

Proof: In this case, a single border router of a virtual router fails. This failure will be detected by the other border routers. If the controller is the router that failed, this will be detected

when the subordinates cease receiving keep-alive messages; if it is a subordinate router that fails, then the controller will detect the failure when the keep-alive messages go unanswered. If it is a subordinate router that goes down, the master router simply removes state about any children of the virtual router that were being served by the failed router, as if the external link of an OCBT router had failed. This is safe as the outgoing links are unreachable through the failed router and it is as if the domain simply has a different topology. If the controller is the router that has failed, it as if the domain has been partitioned in such a way that the controller is unreachable and the subordinate routers follow the same procedure as a network partition and the case is covered by Lemma 3. □

Lemma 3: In the event of a network partition, each partition will form its own virtual router, resulting in correct performance but a topology change.

Proof: In this final case, the virtual router is partitioned in such a way that some border routers are separated from each other, with one or more border routers part of each partition. Because the internal multicast protocol rebuilds the ABR tree within each partition and because the only exits out of the partition are through the border routers, each separate partition meets the criteria of and acts as its own virtual router. Therefore, each partition can appear as an OCBT router on the higher-level tree. The partition that does contain the controller simply removes state about the border routers it cannot reach and continues as in the case above. The partitions that do not contain the controller must elect a controller for that partition once the ABR multicast tree is constructed within the partition. This mechanism is very simple: each border router within the partitions sends keep-alive messages to the ABR address; as all BRs within a partition listen to this address, after several keep-alives have been sent each border router will have the addresses of all other BRs within the partition. One particular BR (the one with the lowest or highest IP address) will then take over as the controller and send an announcement of its status to the others within the partition; this announcement causes them to immediately drop all state about children that they have, after which they modify their state as directed by the controller. To the network outside the

VR, it will appear as if the VR has restarted, and new connections will be formed to the VR. When the partitions merge again, the ABR multicast tree between the two partitions reforms and each former partition receives the keep-alive messages of the other; at this point an election is held again for controller, with one particular router selected again by IP number or by pre-configuration if the set controller is within the partition. The losing controller sends no message but drops all routing state; the elected controller again announces its status to all other border routers. In this case, only the former subordinates of the losing controller need clear VR state and start anew. The these cases the topography of the higher-level tree is different now, with several disconnected virtual routers where there used to be a single one, but each virtual router will still function correctly. Because OCBT functions correctly regardless of the topography, HIP will still operate properly.

If a virtual center point is partitioned, it forms a new group of virtual routers as described above, but with one difference; only the partition containing the actual center point can respond to build the higher-level tree. Each part of the partition must therefore check via its unicast routing tables that the physical center point is reachable before accepting connections for it on the higher-level tree. The reason for this is clear – if there were several virtual center points accepting connections, a single tree would not be built at the higher level; instead, a forest of disconnected trees would be built. This may mean some partitions will be rejecting connection requests until the unicast routing has updated routes to the physical center point, but in time (if the partition lasts long enough) the join requests will start going to the correct partition. □

Theorem 2: Given failures inside a domain, the virtual router protocol will still simulate one (or more) OCBT routers.

Proof: Since the virtual router protocol functions correctly in each of the three possible failure conditions, HIP always correctly simulates an OCBT router, and since OCBT is correct, HIP will always be correct. □

Chapter 3

KHIP — Secure Hierarchical Multicast Routing

Security is expensive. In a computer network, providing security requires dedication of resources, such as bandwidth and processing time, that could otherwise be used to improve the overall performance of the network. Security is, however, necessary. As the value of the information traveling across the network increases, the expense of protecting it becomes less than the cost arising from disclosure or loss. Individuals are willing to tolerate higher delays to protect their credit information from thieves; companies are willing to pay the expense of providing a secure network to protect and assure delivery of proprietary information essential to their business; governments are willing to devote significant resources towards maintaining the secrecy of intelligence and military data. The challenge to a designer of network protocols is to provide a secure service in the most efficient manner possible.

Multicast routing protocols today lack effective security mechanisms that allow for privacy

or for safe transmission of proprietary information. This is due to the fact that because multicast involves a large number of participants, the identities of whom are not known across the entire group, it is fundamentally different than unicast and the mechanisms needed to secure multicast are correspondingly different [5]. The first problem is that of *authentication*, which requires that participants prove their identity before they are allowed to join the group and receive encryption keys for the session. While there are a number of relatively simple key exchange protocols that allow two participants exchange a shared key securely [45], corresponding protocols for arbitrarily sized multicast protocols require either a server or significantly more traffic to generate a common key. The difficulty in generating a key is compounded by the fact that a new key may have to be computed or distributed each time a member joins or leaves the group. The challenge is to distribute keys in a scalable way that supports re-keying when necessary.

The second problem is that of *authorization*, which implies that only authorized entities may use or alter the multicast routing tree of a given group. The fact that a multicast tree is a very efficient way to distribute data means there is a very efficient method of launching a denial of service attack against all members. If some malicious or mis-configured sender were to send data at a high rate to some multicast address, that data would be copied and forwarded over all branches of the tree to all receivers. By purposely saturating the multicast tree, an attacker can disrupt service to all receivers. This misuse is possible today as there is no mechanism for authorization to occur, and anyone can send to or receive from a particular multicast group. A multicast tree also provides an attacker with more opportunities to eavesdrop on the communication. Obviously, sender-initiated protocols are more vulnerable to this type of attack, because they initially broadcast all data to all possible receivers and trust those receivers to trim back the tree. They are also more vulnerable to eavesdropping, because they form a separate tree from each source, using more links in the network and providing more points for an eavesdropper to intercept data. Shared trees formed using a receiver-initiated protocol are still more vulnerable to eavesdropping than a unicast connection, though they span fewer network links than a source tree. An attacker, however, may be

able to alter the multicast routing to build a branch of the tree to some point where eavesdropping is possible. While encryption can protect against eavesdropping, there is the small risk of exposure through attacks against the encryption [8][57]. The purpose of authorization is therefore to limit the construction of tree branches to only those needed to connect authorized senders and receivers.

When authentication and authorization are not used, and any router in the tree is able of sending control messages that affect the multicast routing, a number of attacks are possible. This is particularly true of receiver-initiated protocols that rely on a shared tree for data delivery. The problem arises because an individual router never receives any assurance that the routers that are forwarding its join message towards whatever core, RP or root domain is being used to construct the multicast group. Instead, a corrupt router may choose to acknowledge the join message itself, without ever joining the tree. In this case the corrupt router would have sole access to the data being transmitted from the branch or could feed a particular receiver or subset of receivers on the branch below a false stream of data. A more active form of this attack, shown in Figure 3.1, could consist of several corrupted routers working together to isolate several multicast participants from the rest of the multicast tree by forwarding the join requests they receive towards a different core. In this figure, the corrupted routers have conspired together to isolate the leaf routers from the proper core of the multicast tree. These types of attacks do not necessarily isolate only a few routers; with the introduction of hierarchical multicast routing protocols that use shared trees to form a backbone between domains, entire domains may be targeted [48][33]. It is possible to imagine many other similar scenarios, all of which spring from the lack of trusted routing.

Other attacks against the multicast routing may not be designed to replace or limit access to the data packets moving across the tree. Instead, an attacker may want to deny multicast service over a wide area. In this case, a corrupted router could be used to introduce loops into the multicast tree, as shown in Figure 3.2. In this figure the corrupted router is joining a multicast core along two different paths; as the core accepts connection requests without regard to paths, a loop will be formed in the multicast tree. This would cause each data packets to traverse the loop as long as their

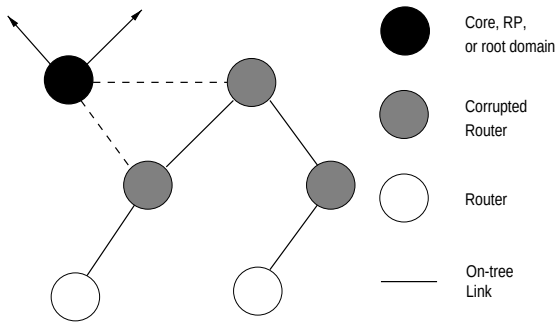


Figure 3.1: Establishment of a corrupted multicast tree

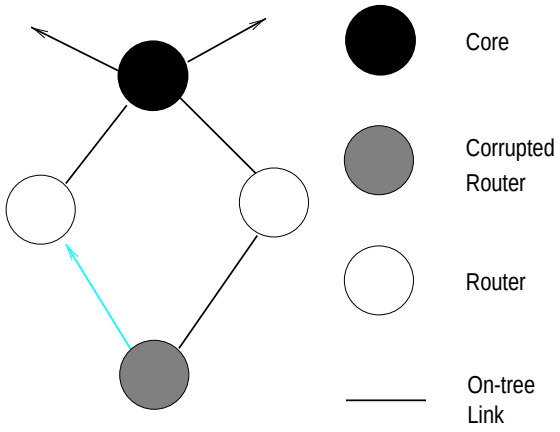


Figure 3.2: Corrupt router building a loop in the multicast tree

TTL allowed, with each active source adding to the traffic, and receivers receiving multiple copies of each packet. While this would congest the corrupted router as much as any other, an attacker who gained illicit access might not care. There are less malicious reasons for wanting to alter the structure of the tree, as well. An attacker may want to reduce the costs of their routing, and by changing the shape of the tree by forging control messages, could reroute traffic so that it traveled a path for which the attacker was not financially responsible, placing the burden on elsewhere.

To maintain security in the face of attacks, a secure multicast routing protocol must limit the construction of tree branches to those links required to connect authorized senders and receivers. This requires placing some trust in the routers of the network, and ensuring that the trusted routers are able to communicate securely with each other. Though compromise of a trusted router could

result in the same type of attacks described above, it should be easier to harden and monitor a relatively few trusted routers. The protocol should also limit the effects of flooding attacks, and should provide a scalable methods for distributing keys. The protocol must do this while maintaining an efficient multicast service that does not duplicate data packets over any link and minimizes the overhead required in terms of control traffic and router memory.

This chapter presents a solution that meets all the above criteria. Using techniques similar to those proposed by Gong and Shacham [28], we present simple extensions to HIP [48], called Keyed-HIP (KHIP) that creates a secure multicast routing service. KHIP provides security of multicast routing by adding several elements to the multicast model. First, KHIP creates an authentication service that is trusted to issue certificates to authenticated hosts and routers which meet the criteria for joining the group. These certificates are used in constructing the multicast tree. A router connecting to a parent higher in the tree (possibly through several untrusted routers) signs the control message and includes its certificate to prove to its parent that it is authorized to modify the tree routing; the parent includes its certificate in a signed reply so that the child may trust it has joined a secure tree. The tree that is built is divided into a number of sub-branches, each with its own key shared between the trusted members within that branch. Using this key, the trusted members can protect against flooding or replay attacks and efficiently distribute data encryption keys if desired.

Section 3.1 describes how the HIP tree is broken down into sub-branches, and provides detail of the messages used to create the KHIP tree, showing how HIP messages are made secure. Section 3.2 presents a formal analysis of the authentication mechanism using an extension to BAN logic [11] called GNY logic [26]. Section 3.3 provides some informal analysis of the effectiveness of different attacks against the routing.

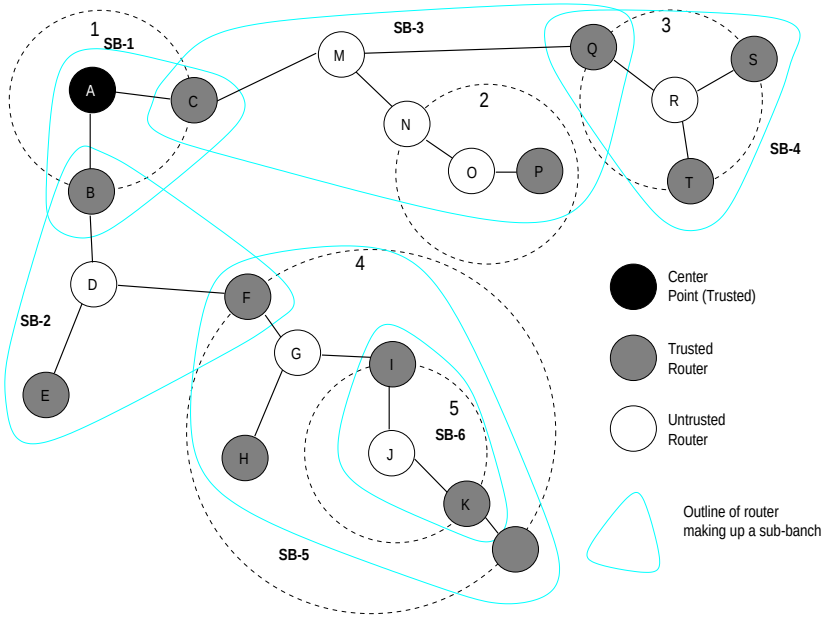


Figure 3.3: Secure sub-branches on the HIP tree

3.1 Keyed HIP

3.1.1 Overview

In extending HIP, the design goals are to provide scalable mechanisms for authentication and authorization, so that only privileged members are able to obtain the proper cryptographic credentials for retrieving keys and certificates that allow access to the multicast trees, and only those holding certificates can cause new branches to form in the tree for transmission or reception. We also wish to prevent or limit replay and flooding attacks. We assume that an attacker can be positioned along any link or control any non-trusted router so that they may drop, replay, delay, alter or issue any data or control packet. Additionally, we assume that an attacker may not be on a path used by the multicast tree and may send control messages to group members in an attempt to create branch of the tree to the attacker, either to receive data or to launch a denial of service attack via flooding. We do not require that clocks be synchronized tightly across the network. We do assume that some method of public key distribution and verification is possible. There are several

proposals for how to accomplish this [10], and any will suffice as long as it is secure. Finally, since HIP and OCBT use the existing unicast routing, we assume that some secure form of unicast routing is available, both for inter-domain and intra-domain routing. A number of proposals have been made for each protocol type [30][50][36][49]. We assume that secure unicast routing provides a reliable path between routers in the network.

Keyed HIP creates a hierarchy of sub-branches over all the branches of the multicast tree, which removes the need for a single shared key for the entire group. While a protocol that can compute a single key across all receivers may be used [55][58], this natural organization of KHIP provides a simple, efficient mechanism for distributing encryption keys. Each sub-branch shares a common key among its members for transmission across that sub-branch. Data packets are reprocessed at the root of the sub-branch for transmission on the next sub-branch towards the center point; similarly, children of a sub-branch that are roots for another sub-branch re-process that data before re-transmission to their children. Re-processing is made less expensive in terms of the amount of computation needed by encrypting the data in a random key and encrypting that key in the shared branch key. Re-processing consists of decrypting and re-encrypting only the random key and adding appropriate nonces for the new sub-branch. When new members join the tree they need only authenticate themselves to the root of the sub-branch they are joining, distributing the load of processing new members across many participants already in the tree. The nature of HIP, which relies on the existing border routers of multicast domains to serve as OCBT cores, provides a natural placement of trusted routers that act as the roots of sub-branches in the same location. A domain wishing to participate in a secure multicast session can therefore ensure its security and configure their border routers so that they posses a public/private key pair and are able to request a certificate from the authentication service. The authentication service is a hierarchy of trusted servers that maintain an access list for each multicast group and issues certificates, signed with a well known authentication server key, for trusted members to present as proof that they are eligible to join a particular group.

Figure 3.3 shows how a HIP tree might be broken down into sub-branches. Router A is the center point of the tree and as such must possess a valid certificate to prove to receivers joining that it is a valid center point. Figure 3.4 shows the structure of the secure tree. The border routers of domain 1 are also trusted, and a sub-branch, labeled *SB-1*, is formed with router A as the root and routers B and C as its secure children. Router C is the root of its own sub-branch, with child routers P and Q. Notice that a secure branch can traverse untrusted routers, and that the border routers of a domain do not need to be trusted in order to support internal trusted routers, as seen in domain two. However, it is recommended that a domain that is to contain a large number of members of the secure multicast group ensure that its border routers are secure, as this increases the scalability by lowering the number of children any one secure parent must service. Router Q supports its own sub-branch that consists of the other border routers of domain three. Router B is also root of a sub-branch, with children E and F. Domain four contains a sub-domain, numbered five, and the sub-branch *SB-5* rooted at F traverses the virtual trusted router, which contains its own sub-branch, labeled *SB-6*. Secure data traversing the virtual trusted router from F to L will be re-processed at router I for transmission across *SB-6* to router K, where they will be re-processed to appear as if it came directly from F. While this may seem strange, it is congruent with the operation of HIP, which requires a domain to act as a single router on the higher-level tree. In this case, domain five appears as a single router to both G and L, so the data that L receives must be the same as if the virtual router were a real one.

3.1.2 Building a Secure Multicast Tree

To describe the methods through which individual members request and receive certificates and then communicate with each other to build the multicast tree, we introduce the following notation, similar to that used by Gong, Needham and Yahalom [26]. The communicating entities consist of the authentication service AS, the HIP center point location service LP, the group initiator I, receivers of the multicast group (who on a shared tree are also senders) R, roots of the sub-branches

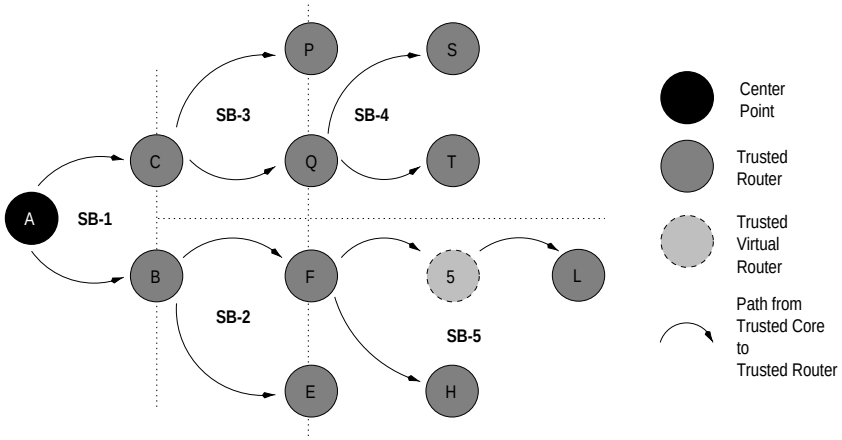


Figure 3.4: The shape of the secure KHIP tree

within the structure of the tree C, one of which will be the center point for the tree CP. Cores preside over sub-branches, B made up of some number of receivers. Any member of the group, M, has a public key, K_{+M} and the corresponding private key, K_{-M}, and uses these to prove its identity to the authentication service and other group members. Two entities, say A and B, can share a common key K_{AB}. The transmission of messages between entities is indicated with an arrow, $A \rightarrow B$, with a message that is encrypted being enclosed with braces with the key as a subscript, $\{message\}_{K_{AB}}$. We also make use of digital signatures, which consist of a cryptographic hash of the message encrypted with the private key of the signer. Digitally-signed messages are indicated in brackets, with the key being used to sign it as a subscript, $[message]_{K_{-M}}$.

Authentication service and certificates

The first necessary addition to the KHIP multicast model is an *authentication service*. This service maintains the list of who is allowed what access to specific multicast groups. The policies that it might enforce are beyond the scope of this work; however, it is easy to envision that the access lists might consist only of those entities who have paid for service or who are part of some governmental or commercial group that wish to share data securely. The authentication service issues *certificates*, credentials that verify the holders identity and specify what type of access the

holder is allowed to the multicast routing. The authentication service owns a well-known public key used to sign certificates and is assumed to be secure against compromise. This service might be a single server within a domain that regulates its internal multicast service, or it could be a robust hierarchy of world-wide servers, with lower-level trusted servers possessing keys signed by the highest-level authority. Such a model has already been proposed [10].

When a member wishes to become part of the secure multicast group, it must first secure a certificate for the group from the authentication service. It requests a certificate from the authentication service using its own public key to authenticate its identity. If authenticated and approved, it receives a certificate consisting of the member's IP address, its public key, the multicast group or range of groups that the member is authorized to use, what permissions the member has for that group, and a time stamp and lifetime for the certificate. In the GNY notation, a certificate for some member M appears as:

$$CERT_M = [IP_M, K_{+M}, MC, Perm, TS, Life]_{K_{-AS}}$$

The possible permissions are $Create$, $Join$ and $Destroy$. As the names imply, these allow a member to initiate a group with a particular address, subscribe to the group as a sender or receiver, and terminate a secure group. With modification to the routing protocol, it would also be possible to specify $Listen$ and $Send$ for finer granularity of access control. The time stamp and lifetime are used to expire the certificate, and should be reasonably short. Given that it is difficult to revoke issued certificates in a way that is scalable and can reliably reach all entities who could honor them, this will force a receiver to occasionally re-request a new certificate. If the access list has changed in the interim, it will be denied. A bad certificate will therefore never last more than the lifetime specified. This solution is deemed preferable as it is easier to expand the hierarchy of authentication services than to attempt to create a method of issuing certificate revocations.

Group creation

To start a secure multicast session, some initiator must request and receive a certificate with the appropriate permissions, then communicate the center point of the group to the location service. The initiator send its IP address, its public key and the multicast address and permissions desired, which in this case would include *create*, to the authentication service. The transmission from the initiator to the authentication service appears as:

$$1: \ I \ \rightarrow \ AS: \quad [IP_I, K_{+I}, MC, Perm]_{K_{-I}}$$

Upon receipt of this message, the authentication service retrieves the initiator's public key from the appropriate server and uses it to verify the IP address and public key contained in the message. It then checks the access list to see if the initiator is allowed to create the requested multicast groups. If these checks succeed, then the authorization service adds a time stamp and lifetime to the certificate, signs the certificate with the authentication service private key and returns it to to the initiator. The reply is then:

$$2: \ AS \ \rightarrow \ I: \quad CERT_I = [IP_I, K_{+I}, MC, Perm, TS, Life]_{K_{-AS}}$$

Now the initiator can send the group creation message to the location service so that the start of the group can be announced. This message includes the signed certificate, the center point for the group, the scope of the group (which specifies which domains the group should will cover) and a group lifetime, which should not exceed the lifetime of the certificate. Once the location service receives the creation request, it verifies the signature on the request and on the certificate it contains. Notice that the location service need not retrieve the initiator's public key as it is contained in the signed certificate. If the request passes these checks, the group center point is distributed as appropriate given the scope. The message sent from the initiator to the location service is:

$$3: \ I \ \rightarrow \ LS: \quad [Create, CERT_I, CP, scope, lifetime]_{K_{-I}}$$

As the center point will serve as the root of the multicast tree, it needs to obtain a proper certificate as well. This is necessary as even though it need not join the tree, it will need authenticate with those joining the tree. This occurs in the same manner as the initiator's certificate request,

though the permissions are for *Join* rather than *Create*. Though the center point can be a distributed entity consisting of a number of border routers of a domain operating together, they do operate as subordinates to one master router, and only that router need retrieve a key as it will be processing and replying to all messages. This exchange appears as:

$$4: \; CP \rightarrow AS: \quad [IP_{CP}, K_{+CP}, MC, Perm]_{K_{-CP}}$$

$$5: \; AS \rightarrow CP: \quad CERT_{CP} = [IP_{CP}, K_{+CP}, MC, Perm, TS, Life]_{K_{-AS}}$$

Host to router communication

The protocol described above details only the process of routers constructing the multicast tree; it does not describe the communication between hosts and routers. Today, hosts communicate with their designated multicast router using IGMP. For secure multicast, a secure version of IGMP needs to be developed [5]. A host could then authenticate with its router using the same four way authentication mechanism described below, first with an exchange to find the designated router and receive a nonce, then the message to the router with a host nonce and the rely bearing the key and sequence information. With this mechanism in place, the host to router connection will appear simply as the connection to leaves of the secure multicast tree.

Building the tree - securing OCBT

Once the group has been created and the center point has received its certificate, the process of building the tree can start. HIP uses OCBT to build the multicast tree. OCBT uses only four types of control messages in normal operation: a *join request (JR), join acknowledgment (JA), quit notice (QN)* and *flush (FL)*. The join request travels from receiver to core and sets up temporary state so that the join acknowledgment can traverse the same path back from core to receiver and instantiate a branch of the multicast tree. The other two messages are used to tear down branches of a tree, either following after a link or router failure, or when it is necessary to break a lower-level tree branch to allow a higher-level branch to form.

Since one design goal is to prevent branches from being built to unauthorized receivers, we add digital signatures to the join request and the join acknowledgment to ensure that the endpoints of a particular path along a branch are trusted receivers. Neither the flush message nor the quit notice need to be signed, as they do not result in the construction of a branch and also may be generated by untrusted routers on the path between to trusted routers in response to link or router failures. In order to prevent attackers from replaying join requests or acknowledgments, we add two more messages to the original protocol. Because a member of the group does not know the identity of the identity or the location of the trusted core that it will reach when sending a join request towards the center point, it instead finds and exchanges messages with the trusted core higher on the tree. These two messages, called a *core request (CR)* and *core acknowledgment (CA)*, are signed by each party and carry nonces that ensure that the messages are fresh. Using time stamps instead of nonces would lead to a simple replay attack in which an attacker could quickly replay the join request to a different core router, which would accept it as valid and send the join acknowledgment back, building a branch to the attacker.

The exchange of messages that leads to the formation of a branch between some receiver and some core is then as follows. First, the receiver router sends a core request message. This message travels hop-by-hop towards the center point. When it reaches some branch of the tree it is forwarded from child to parent until it reaches some trusted core. If it does not reach a branch of the tree it will eventually arrive at the center point. The message contains the receivers certificate, a random nonce and is signed by the receiver, and looks like:

$$6: R \rightarrow C: \quad [CR,\ CERT_R,\ N_R]_{K_R}$$

Once the trusted core or the center point receives the core request, the signature and certificate are checked and if passed, a core acknowledgment is sent directly to the initiating receiver. This acknowledgment contains the core's certificate, the nonce created by the receiver, a new nonce created by the core and is signed by the core. This transmission of the acknowledgment is:

$$7: C \rightarrow R: \quad [CA,\ CERT_C,\ N_R,\ N_C]_{K_C}$$

Once the receiver obtains the core acknowledgment, it forms and sends a join request. This join request carries the receiver's certificate and the nonce supplied by the core, and it is signed by the receiver. The join request is sent, hop-by-hop, towards the trusted core, and appears as:

$$8: R \rightarrow C: \quad [JR,\ CERT_R,\ N_C]_{K_{-R}}$$

The join request is not acknowledged by the first on-tree router it reaches. Instead, the join request is forwarded up the tree to the trusted core, which verifies the signature, nonce and certificate, then sends the join acknowledgment back down the tree, initiating the branch back to the receiver. When the receiver gets the acknowledgment, it verifies the nonce, certificate and signature to prevent replays of other join acknowledgments. The acknowledgment also contains some information encrypted in the receiver's public key; the branch key, K_B, which is used to send data to the multicast group as described below, a branch identification number, ID, which is unique to each trusted receiver serviced by that particular core, and a starting sequence number for data, SEQ. The ID and sequence number together make up nonces for each data packet sent by any member. This final acknowledgment is:

$$9: C \rightarrow R: \quad [JA,\ CERT_C,\ N_R,\ \{K_B, ID, SEQ\}_{K_{+R}}]_{K_{-C}}$$

3.1.3 Operation and Maintenance of the Secure Multicast Tree

Data flow

Each individual receiver that is part of the sub-branch under some trusted core is given a shared branch key, K_B, a unique identifier on that sub-branch, ID, and a starting sequence number. While it is possible to use other protocols for group key exchange [55][58], the unique nature of a KHIP tree provides an efficient method of doing so, which we consider here. Using KHIP's key exchange protocol, when a receiver wants to send data, it creates a random encryption key, K_{Rand} and encrypts the data with that key. It then creates a packet that consists of the encrypted data and a package of information encrypted in the branch key. This information consists of the random key used to encrypt the data, the sender's branch ID number, the next sequence number, and a

checksum of the encrypted data. The packet is then sent on to the branch. When other members of the branch receive the data, they decrypt the random key, sequence information and checksum. The ID and sequence information are used as nonces; each receiver keeps track of the sequence number from each different ID. Sequence numbers are used rather than random nonces to facilitate storage. If packets arrive in sequence and none are lost, then only one nonce need be kept for each sender. In the worst case, however, packet loss over the network or an attacker reordering packets can cause the required storage space to be increased. Each receiver need keep track only of sequence numbers in his sub-branch, limiting the number of receivers for which nonces need be kept. However, a router may have many individual hosts on a sub-net to service. In this case, one host on the sub-net can serve as a core for the other members on the same sub-net. This will result in duplicate data packets being sent over the sub-net, once from the router to appointed host and once from the host to other hosts, but will maintain the scalability of the protocol.

A branch member serving as a core that has parents or children on another sub-branch will re-process the packet before sending it on. It will decrypt the random key, verify the sequence number and checksum, then replace the sequence information with it's own sequence information for the new sub-branch, as if it were the originator. It will then re-encrypt the key, ID and sequence information and the checksum and send it out over the new sub-branch. This method consumes less processing time than would re-encrypting the entire packet, as the random key and sequence information can be much smaller than the data enclosed. A data transmission from a sender to its sub-branch would then be:

$$10a: \; R \rightarrow B: \quad \{Data\}_{K_{Rand}}, \; \{K_{Rand}, ID, SEQ, CS(\{Data\}_{K_{Rand}})\}_{K_B}$$

The checksum is intended to help prevent *cut-and-paste attacks*, in which the attacker replaces the encrypted data but leaves the other information intact. The attacker cannot replace the data at will without it being detected, assuming it can tell where the date ends and the key and sequence number starts, though it can replace it with something, including old meaningful data, that produces the same checksum. Though computationally expensive, the checksum should be a

cryptographic hash, or else the attacker will be able to replace the encrypted data with possibly random data that has the same checksum.

In some cases, it may be necessary for an end receiver to verify the identity of the original sender. In this case, the originator of the data can include his IP address with and digitally sign the data. Using the included IP address, a receiver can lookup the originator and retrieve his public key, then use that to verify the origin of the data. This looks like:

$$10b: \ R \rightarrow B: \quad \{[Data]_{K_{-R}}, IP_R\}_{K_{Rand}},$$
$$\{K_{Rand}, ID, SEQ, CS(\{Data, IP_R\}_{K_{Rand}})\}_{K_B}$$

If lookups of public keys are expensive and a sender is not sending a large amount of data, it may be worthwhile to include the certificate issued to access the multicast group with the data. By verifying the authentication server signature on the certificate, each receiver can then use the enclosed public key to verify the original sender's identity. This method of verifying the sender's identity appears as:

$$10c: \ R \rightarrow B: \quad \{[Data]_{K_{-R}}, CERT_R\}_{K_{Rand}},$$
$$\{K_{Rand}, ID, SEQ, CS(\{Data, CERT_R\}_{K_{Rand}})\}_{K_B}$$

Re-keying

Though there is no single encryption key for the entire multicast group, each sub-branch key will need to change branch keys K_B as receivers leave, either following their legitimate departure or following a link or router failure that removes a receiver from the tree. It will also be necessary to occasionally to restart the sequence numbers used as nonces. In these cases the core of the sub-branch creates a new branch key and starting sequence number, add those to the old branch key for use as a nonce, encrypts it with the public key of each authenticated member of the sub-branch, signs it and sends it multicast to all members. The structure of the re-keying message is:

$$11: \ C \rightarrow B: \quad [IP_{R_1}, \{K_{B_{New}}, K_{B_{Old}}, SEQ\}_{K_{+R_1}}]_{K_{-C}}, \ [IP_{R_2}, \{K_{B_{New}}, K_{B_{Old}}, SEQ\}_{K_{+R_2}}]_{K_{-C}},$$
$$\ldots$$

Following a key change, the old encryption key and nonce sequence remains valid for a short period to allow messages in transit that are encrypted in the old key to be accepted when they arrive.

Tree Maintenance

In HIP, when a router determines that a link to a child on the tree or the child itself has failed, it only needs remove information about the child from its routing state. When a child detects that a parent router or the link to the parent has failed, it needs to take action to assure that its children and itself can rejoin the tree as needed. Therefore, a router detecting a failure will send a flush message to all its children and removes state concerning them. Each non-core child receiving a flush message forwards it to all its children. This process results in the tree being removed from the point of failure down to the individual receivers or cores, who then have the responsibility of rejoining the tree. Other tree maintenance occurs when a router receives a higher-level join request, at which point the router sends a quit notice to its parent and becomes part of the higher-level branch that is forming. In this case the router maintains its children.

With KHIP, untrusted routers need to be able to respond properly to link and router failures but should not be able to cause branches of the tree to be formed to untrusted routers, including routers that were once part of the tree out of the necessity of being on the path to a trusted router, but no longer should be. This requires some small changes to the maintenance mechanism of OCBT. First, quit notices need to be forwarded up to the next trusted core in the tree, so that if a trusted receiver quits the tree, the core router can remove state about it and start the re-keying process. Second, an untrusted router that is forced to quit from its parent to join a higher-level branch that is forming must also send a flush message to destroy the tree below it. This is necessary because the trusted cores or receivers below that router have to authenticate themselves with the new trusted core and receive the branch key for their sub-branch.

Finally, to limit the effects of attack based on forging, replaying or failing to deliver control

messages and to detect expires certificates, we require that each sub-branch periodically be destroyed and re-constructed by the trusted core sending a flush message to all its children. Each receiver will then re-authenticate with the higher core. Each receiver should also keep a timer to ensure that they receive these flush messages periodically; if they do not, then they should quit from the tree and attempt to rejoin to make sure the core's certificate has not expired.

3.2 GNY Logic Expression of KHIP Authentication

For completeness, we present the authentication that takes place between the authentication service, core and receiver in terms of GNY logic. This is not a proof of security, but rather a more thorough method of presenting the initial assumptions about what each entity knows before and during each step of the protocol run. The presentation follows the structure of the examples presented in the original work [26].

The Message Exchange

We begin by rewriting the message exchange that takes place in the form favored in the GNY logic, after the protocol parser has added the "not-originated-here" markers. We use data transmission method marked 10a in Section 3.1.3 for this exchange.

$1 : AS \triangleleft *IP_C, *K_{+C}, *MC, *Perm, *\{H(IP_C, K_{+C}, MC, Perm)\}_{K_{-C}}$

$2 : C \triangleleft IP_C, K_{+C}, MC, Perm, *TS, *Life, *\{H(IP_C, K_{+C}, MC, Perm, TS, Life)\}_{K_{-AS}}$

$3 : AS \triangleleft *IP_R, *K_{+R}, *MC, *Perm, *\{H(IP_R, K_{+R}, MC, Perm)\}_{K_{-R}}$

$4 : R \triangleleft IP_R, K_{+R}, MC, Perm, *TS, *Life, *\{H(IP_R, K_{+R}, MC, Perm, TS, Life)\}_{K_{-AS}}$

$5 : C \triangleleft CR, *CERT_R, *N_R, *\{H(CR, CERT_R, N_R)\}_{K_{-R}}$

$6 : R \triangleleft CA, *CERT_C, *N_C, N_R, *\{H(CA, CERT_C, N_C, N_R)\}_{K_{-C}}$

$7 : C \triangleleft JR, CERT_R, N_C, *\{H(JR, CERT_R, N_C)\}_{K_{-R}}$

$8 : R \triangleleft JA, CERT_C, N_R, *\{K_B, ID, SEQ\}_{K_{+R}}, *\{H(JA, CERT_C, N_R, \{K_B, ID, SEQ\}_{K_{+R}})\}_{K_{-C}}$

$9 : R \triangleleft *\{Data\}_{K_{Rand}}, *\{K_{Rand}, ID, SEQ, F(\{Data\}_{K_{Rand}})\}_{K_B}$

$$10: R \triangleleft \ldots, IP_R, *\{K_{B_{New}}, K_{B_{Old}}, SEQ\}_{K_{+R}}, H(IP_R, \{K_{B_{New}}, K_{B_{Old}}, SEQ\}_{K_{+R}})_{K_{-C}}, \ldots$$

Assumptions

We next list the initial assumptions. The authentication service possesses a private and public key pair. It also possesses the public keys of other entities participating in the protocol, in this case limited to the core C and receiver R. The authentication services can also recognize requests for multicast certificates.

$$AS: \ AS \ni K_{+AS}, K_{-AS}, \ AS \mid\equiv \stackrel{K_{+C}}{\longmapsto} C, \ AS \mid\equiv \stackrel{K_{+R}}{\longmapsto} R, \ AS \mid\equiv \phi(IP_M, K_{+M}, MC, Perm)$$

The root of the sub-branch, referred to as the core here, also possesses a private/public key pair and a nonce, N_C, which it knows to be fresh. It possesses the public key of the authentication service, who it recognizes as having jurisdiction over the multicast group address, permissions and other entities public keys. The core also can recognize certificates when they are contained in a message. The core also has a clock which allows it to verify the freshness of the certificate time stamp and lifetime.

$$C: \ C \ni K_{+C}, K_{-C}, \ C \mid\equiv \sharp(N_C), \ C \mid\equiv \phi(N_C), \ C \ni K_{+AS}, \ C \mid\equiv \stackrel{K_{+AS}}{\longmapsto} AS$$

$$C \mid\equiv AS \Longrightarrow K_{+M}, Perm, MC, \ C \mid\equiv \phi(CERT_M)$$

The receiver has the same initial assumptions as the core, with the addition of recognizing the core as having jurisdiction over the branch key, K_B, and the sequence numbers and branch ID numbers for the sub-branch. It also can recognize its IP address and messages encrypted in its public key.

$$R: \ R \ni K_{+R}, K_{-R}, \ R \mid\equiv \sharp(N_R), \ R \mid\equiv \phi(N_R), \ R \ni K_{+AS}, \ R \mid\equiv \stackrel{K_{+AS}}{\longmapsto} AS,$$

$$R \mid\equiv AS \Longrightarrow K_{+M}, Perm, MC, \ R \mid\equiv \phi(CERT_M), R \mid\equiv C \Longrightarrow K_B, ID, SEQ$$

$$R \mid\equiv \phi(IP_R), \ R \mid\equiv \phi(\{X\}_{K_{+R}})$$

Common Operations

Each of the eight messages exchanged in the protocol run depends on a digital signature to ensure that it is not a forgery and that it has not be altered in transmission. In this section, we describe the operation carried out upon receipt of each message to check the signature. This will allow us to avoid repetitive applications of formulas in describing the protocol.

Several of the original GNY postulates reappear frequently in our analysis. Postulates T1, T2 and P1 [26] taken together state that a receiver knows things that it is sent. Postulate R5 states that a receiver who is able to recognize some bit stream and possesses that bit stream can recognize the hash of the bit stream. Postulate T6 states that a receiver can decrypt things encrypted with a public key if he possesses the corresponding private key. Postulate P2 states that a can be considered to possess the results of any computationally feasible product of things it possesses. Postulate I4 states that a receiver can infer that the owner of a private key sent a message that can be decrypted with the public key. Postulate I7 states that if a sender can be considered to have sent a concatenation of several but streams, it can be considered as having sent each one. Postulate J1 states that if a receiver gets a message from some entity it considers as having jurisdiction over the contents of the message and it believes that the entity believes the message, the receiver can believe the contents of the message as well. Finally, postulate F1 states that if a receiver knows part of a message to be fresh, it can consider the entire message fresh. Knowledge of these postulate will help in the following description of common operations of the authentication protocol.

When a signed message is received it can goes through operations corresponding to application of different postulates. First, by postulates T1, T2 and P1 the receiver possesses of each component of the message bases on the message's arrival. If the receiver does not already possess the sender's public key, as is true in messages five and six, it decrypts the hash that makes up the signature of the certificate and checks its validity by recomputing it and comparing the results. This corresponds to application of postulates R5, T6 and P1 to the hash of the certificate using the well known AS key, followed by application of postulate P2 to compute and verify the hash. If the hashes

match, the receiver then has the entities public key from within the certificate. This corresponds to application of I4 and I7 to show that the receiver believes that the AS signed the key, and, because the AS is assumed to have jurisdiction over the sender's public key, then by J1 the receiver believes the key to be correct. Now the receiver checks the time stamp, TS, and the lifetime, L. If they are fresh enough, he accepts the certificate as valid. This corresponds to use of F1 to show the entire message is fresh. Now that the receiver possesses the sender's key, it can use it to verify the entire message. First, it uses the sender's public key to decrypt the hash, corresponding to application of R5, T6 and P1, and computes the hash himself, corresponding to application of postulate P2. If it is correct, then the receiver believes that the sender is the actual originator, which corresponds to the application of postulates I4 and I7. Notice that if the message has been altered in transit, then the hash will not compute properly and the alteration will be detected.

The Protocol Run

Message 1: Applying T6 and I4, we see that the authentication service decrypts the hash and believes it to come from the core, C. From I4 and I7, the AS believes that the entire message came from the core. By the initial assumptions, it recognizes this as a certificate request and is able to issue a reply.

Message 2: The core receives the certificate. Following the method described above, it believes that it comes from the AS.

Message 3 and 4: The same process occurs for the certificate request from the receiver as happened for the core.

Message 5: The core receives the core request message, the receivers certificate, the nonce, N_R and the hash of the entire message. By assumption, the core recognizes the certificate, and, using the known AS key to decrypt and verify it, corresponding to the use of postulates T6, I4, and I7, believes that the AS issued the certificate. As the AS is recognized as an authority over the public key contained within, the core possesses K_{+R}, MC and $Perm$ and believes that the key

is suitable for use for communication with the receiver. This corresponds to use of postulate J1. It also believes that the receiver should be granted the multicast group and permissions contained within. Using the receiver's key, the core believes the message originated at the receiver and can reply to the message. Again, this corresponds to the application of postulates T6, I4 and I7. The core does not know anything about the freshness of the nonce, however.

Message 6: The receiver receives the core acknowledgment, the core's certificate, the core's nonce and its nonce. It determines the freshness of the message from it's nonce. This corresponds to the use of postulate F1 to determine the freshness of the message. It verifies the core's certificate using the known AS key as described above. This process is the same as using postulates T6, I4, and I7, and at the end we show that the receiver possesses the core's public key and its permissions for the group. Using the core's key, it verifies that the core was the originator of the message. By the returned nonce, it verifies the freshness of the message, corresponding to application of postulate F1. It is then able to send its join request to the core.

Message 7: The core receives the join request. Using the receiver's key, which it already possesses, it verifies the origin of the message, corresponding to the use of postulates T6, I4, and I7. It uses the nonce it sent the receiver, N_C, to verify the freshness of the request, corresponding to application of postulate F1. If the core has not cached the receivers permissions or key, it can re-verify them from the included certificate.

Message 8: Using the core's public key, the receiver verifies the origin of the message as described above. It then verifies the freshness using the core nonce. These operations follow use of postulates T6, I4, I7, and F1. At the end, the receiver is shown to possesses the branch key, and its branch key and the starting sequence number.

Message 9: When a data packet arrives at a receiver, it decrypts the random key and sequence information, corresponding to the use of postulate T3. Using the ID and sequence number it determines the freshness of the data packet, corresponding to use of postulate F1. If the datum is fresh, the receiver checks to make sure that the encrypted datum has not been altered by computing

the checksum of the encrypted datum, corresponding to application of postulate P2, and checks to verify they are the same. If they are not then the packet has been altered and is dropped. Otherwise, the encrypted datum can be decrypted using the known random key. This final step follows use of postulate T3.

Message 10: The re-keying message contains many different IP addresses and keys, but by our assumption, the receiver recognizes its own IP address and the message encrypted in its public key. Using the public key of the core which it knows, decrypts the hash and verifies it. These two step follow application of postulates T6 and R5. If it is correct, it believes it came from the core, which would be the same as application of postulate P5. It next uses its private key to decrypt the new branch key, old branch key and sequence number. By postulates T4 and P1 it possesses these things at the end of the decryption. If the old branch key is correct, it assumes the message is fresh and can start to use the new key. The assumption of the freshness of the message follows from postulate F1.

While GNY logic does not prove the security of a protocol, it is a valuable tool for explicitly making clear the assumptions and actions required of each participant in a protocol. Use of the logic made evident that the draft version of the KHIP authentication protocol was vulnerable to some replay attacks, as the protocol lacked mechanisms to guarantee freshness of certain messages. By adding mechanisms to guarantee freshness of the messages, we have limited the effectiveness of these replay attacks.

3.3 Denial of service attacks and attacks by untrusted routers

There are a number of denial of service attacks that are possible against members of a multicast group. The most potent type is a flooding attack, one that uses the natural efficiency of multicast to attempt to drown all receivers in a barrage of worthless data that is spread to all members of the group. Less potent types that use forged or replayed control messages or that simply do not process or forward control messages are also effective at denying service to individual receivers

or branches of the multicast tree. KHIP limits the effect of flooding attacks, first by limiting the spread of branches so that an attacker cannot easily access the tree to send to it, and second by verifying the encrypted sequence number and ID enclosed in each data packet. An attacker who happens to be on the path between a trusted core and receiver can attempt to flood the entire multicast tree, but since it is not in possession of the proper sequence information or branch key it cannot construct packets properly and the flooded data will be detected and dropped at each trusted receiver and at the core for that sub-branch. If an attacker were to try replaying old data, this too would be detected when the sequence number was seen as already having been received. Even if the attacker preserved old data from previous sessions it would be improbable that the core would have chosen the same random branch key to make the packets decrypt properly, as the branch keys changes regularly.

KHIP does not defend against other types of denial of service attacks, however. Untrusted routers that lie on the path between a trusted core and one or more trusted receivers can deny service to the receivers, and hence to any sub-branch that they may be the core for. In addition, some of these attacks may cause branches to be formed between the attacker and either the trusted core or trusted receiver. We consider these branches *malformed* as they do not span a path from trusted core to trusted receiver. In all cases these branches are transient as they will be removed when the core periodically flushes the tree to force receivers to re-authenticate. If the trusted receivers do not receive this periodic flush, they will quit the tree themselves. It is important to note that the malformed branches would have been part of the tree branch crossing the corrupt router anyway, so the corrupt router does not gain any additional information than it otherwise would have, and that this type of denial of service is no stronger than simply not forwarding control messages to build the tree in the first place. Malformed branches last only as long as the period allowed between core and receiver re-authentication, because the tree is destroyed and re-built at those times. The attacker also does not gain access the branch key with these attacks. Table 3.1 shows the effects of replaying, forging or dropping control messages.

	CR	CA	JR	JA	FL	QN	Data
Replay	No effect/ DOS - core	Detect	Detect	Detect	DOS/ Transient Malformed Branch C ◄— A	Transient Malformed Branch A ◄— R	Detect
Forge	Detect	Detect	Detect	Detect	DOS/ Transient Malformed Branch C ◄— A	Transient Malformed Branch A ◄— R	Detect
Drop	DOS- receiver	DOS- receiver	DOS- receiver	DOS/ Transient Malformed Branch C ◄— A	Transient Malformed Branch A ◄— R	Transient Malformed Branch C ◄— A	DOS

Table 3.1: Effects of on-tree untrusted router attacks.

In most cases, forgery or replay of control messages or data are detectable, because these messages are signed or encrypted in the branch shared key, cryptographic operations which the attacker cannot duplicate. These instances are marked "Detect" in the figure, as the receiver of a message can tell it has been altered when the signature does not match the message. There are also cases in which a router on the path can cause a denial of service attack by dropping control message or data packets. These cases are marked "DOS". A solution to corrupt routers purposely dropping packets would be to use a multi-path unicast routing protocol to determine alternate paths towards the center point, bypassing the misbehaving router.

In some instances the attacker can maintain a branch of the tree to themselves from either the core or receiver while denying service to one of the trusted participants. In those cases in which transient malformed branches exist, the table entry is labeled as such, with an arrow pointing from the child to the parent of the malformed branch. These cases occur as the untrusted routers on the path between trusted routers need to be able to issue flush and quit notices in response to link and router failures. Since these routers are not trusted, they cannot be issued certificates to use to sign the quit and flush control messages. A simple but costly solution to this is to require every

router in the network to be trusted, which is nice for scalability of data transmission but not for certificate distribution. Instead, we tolerate the possibility of these attacks as they are, in effect, just a combination of otherwise possible attacks. An untrusted router can listen to traffic that flows across it, and it can prevent formation of branches by not passing control messages. Attacks that allow the temporary formation of malformed branches are simply a combination of these two other attacks, though less potent as the attacker will only receive data from one direction.

As KHIP supports heterogeneous multicast routing protocols, those protocols are possible points of compromise, especially if they are not secure protocols. A possible solution is to secure and trust the entire domain and have the KHIP speaking border routers distribute the un-encrypted data to the trusted domain. This is clearly not always practical, however, and domains should use some secure multicast routing, either KHIP or some future secure protocol.

Chapter 4

Multicast Anonymity

The realization that some solution is needed for providing privacy and anonymity on a network is not new [15, 53, 43, 42, 16]. In many of the previous solutions, anonymity is provided by the cooperation of many hosts, whose identities are known to each other, working together in a group. In these schemes, the identity of the originator of a particular message is known to be one member of the group, but it is not possible to identify any specific individual. In this chapter we focus on providing anonymity in a fashion where cooperating participants are unknown to each other; for this we rely on properties of multicast communication. This work takes the approach that protocols should provide anonymous communication not only for an initiator; we develop protocols and methods for providing anonymous communication for a responder alone, and; for providing anonymous communication for both initiator and responder at once. It is not the case that only web surfers wish to remain anonymous—web servers may wish to serve pages anonymously, or both the surfer and the server may wish to exchange information without providing their identities to the other.

To provide a means of anonymous IP communication we introduce several new protocols and techniques:

- A new protocol for initiator anonymity, called Multicast Initiator Anonymity (MIA), that

utilizes IP multicast communication to hide the identity of the initiating party;

- A new protocol for responder anonymity, called Multicast Responder Anonymity (MRA) based on IP multicast communication and a technique that allows any previous protocol for initiator anonymity to provide responder anonymity;
- A technique for composing responder and initiator anonymous protocols so that two parties may communicate over the Internet based solely on aliases rather than IP addresses, called Zen routing.

In the next section we introduce our communication framework and provide an overview of all types of end-to-end anonymous Internet protocols. In Section 4.2 we introduce new protocols for providing initiator anonymity that utilizes the multicast routing available in IP. In Section 4.3 we define schemes for providing anonymity for scenarios where the responder's anonymity is maintained, and schemes were both initiator and responder maintain their anonymity from each other. We discuss the anonymity and security of these techniques and protocols in Section 4.4.

4.1 Anonymity

A common and simple solution for providing anonymity to the initiator of an Internet connection is to use a *proxy*, which is a single server that accepts connections from many initiators and forwards them on to the responders; all the responders ever learn is the proxy's address; thus, the initiator is anonymous to the server. For example, the Anonymizer [56] and the Lucent Personalized Web Assistant [1] provide anonymity using such methods. There is no anonymity on the path from initiator to the proxy however, and an eavesdropper on that path will be able to determine both the initiator and receiver by examining the packets in the stream.

A much more robust method of using proxies for certain defined services is onion routing [42], which is based upon the idea of mixes [16]. In onion routing, a series of proxies are used and the connection between each proxy is encrypted so that the contents of the communication

(which include the path it has traveled) are encrypted. In Crowds [43] the initiator forwards other encrypted communications and can include its own communications as if they were being forwarded; there is no way to tell which member of the crowd initiated any particular communication. Each member of the crowd thus trades the overhead of forwarding others communications in exchange for the anonymity of the crowd. There are other cryptographic solutions that provide the same level of anonymity as Crowds [53], though at a higher cost in terms of network traffic and processing.

Each of the above solutions addresses some definition of anonymity, but are not comprehensive. Using a single proxy can maintain the initiator's anonymity from the responder, but not from all possible eavesdroppers on the path, and the responder is not at all anonymous. In Crowds, the communications of initiators are indistinguishable from other initiators until an eavesdropper can gain access to a significant percentage of crowd members' networks: however, the responder is offered no such protection. Furthermore, the identity of the responder is revealed to every crowd member that the packet traverses. Thus, while these are solutions for a specific need for anonymity (initiator anonymity, in the examples given), what has been thus far lacking is an identification of all possible types of anonymous communications. We present the first such taxonomy here by considering the possible combinations of sender and receiver anonymity from each other and from an eavesdropper.

4.1.1 Anonymous Connection Scenarios

In any communication scenario, there are three possible participants; the initiator, the responder and and eavesdropper[1]. This leads to four different kinds of anonymity — the initiator's anonymity from the responder; the initiator's anonymity from the eavesdropper; the responder's anonymity from the initiator; and the responder's anonymity from the eavesdropper. An *anonymous communication scenario* can be defined by taking any combination of these four types of anonymity, so that there are sixteen different types of anonymous scenarios, including the one with no anonymity

[1] In fact, though there may be more than one eavesdropper we only consider the entire end-to-end scenario; thus we can ignore multiple eavesdroppers as each would hear the same traffic on such a channel.

at all, which represents the present scenario in IP protocols [2]. Table 4.1 shows all possible types of anonymous scenarios. The table is divided into four quadrants separated by the bold lines. Within each quadrant, the notation IkR indicates whether the initiator knows the identity of the responder, while RkI indicates the converse, whether the responder knows the identity of the initiator. This is repeated four times for to show whether the eavesdropper is allowed to know the identities of the responder and initiator, with similar notation; EkI and EkR indicate whether the eavesdropper knows the identity of the initiator or responder, respectively. Current Internet communication falls in the uppermost left hand box numbered 1, as the initiator and responder know each other and an eavesdropper can determine the identity of both. The scenario mentioned above in which a single proxy is used to hide the identity of the initiator falls in the box numbered 5, as the responder does not know the initiator, but an eavesdropper is able to determine both identities at some point in the channel.

Not all sixteen combinations are useful when considering end-to-end communications, however. As an initiator can eavesdrop on its own transmissions, it is meaningless for the eavesdropper know the responder but for the initiator not to. Similarly, the responder can eavesdrop on transmissions to it; therefore it is also meaningless for the eavesdropper to know the initiator but for the responder not to. The scenarios that do not maintain end-to-end anonymity are labeled "NA", for not anonymous. That leaves only eight possible different end-to-end anonymous communications scenarios to consider. Note that some of the scenarios that we do not consider are still useful though they do not meet our requirement of end-to-end anonymity. For example, the use of a single proxy to hide the initiator's address is not considered anonymous as an eavesdropper could determine the initiator's address before the transmission passed through the proxy, even though the end result of sender anonymity is maintained. Additionally, a government might choose to promote a scenario in which their agents, as the eavesdroppers, know the identities of parties that maintain their anonymity with respect to each other.

Out of the eight scenarios that maintain anonymity for an eavesdropper, only three are *truly*

			EkI = T		EkI = F	
			T IkR	F	T IkR	F
EkR = T	RkI = T	1	NA	NA	A	NA
		2				
	RkI = F	3	NA	NA	TA	NA
		4				
EkR = F	RkI = T	5	A	TA	A	A
		6				
	RkI = F	7	NA	NA	A	TA
		8				

Table 4.1: Anonymous Communication Scenarios

anonymous, that is they conceal the identity of one or both ends of the communications channel from the entity at the other end as well as the eavesdropper. We distinguish these three protocol by labeling them "TA". These protocols can form the basis for the other 5 anonymous protocols, labeled "A". If a truly anonymous protocol is instantiated between initiator and responder, they can easily establish a shared session key and then use the subsequently encrypted channel to divulge their identities to each other without fear of the eavesdropper gaining such information. The rest of this chapter therefore considers only the three truly anonymous scenarios, in which end-to-end anonymity is maintained, and represent initiator anonymity, responder anonymity, and concurrent initiator and responder anonymity.

For each scenario, there can be a number of ways to construct a protocol that implements the desired anonymous channel. For example, both Crowds [43] and Onion Routing [42] provide initiator anonymity from the responder and eavesdropper and thus fall in the box labeled 7. While both protocols use a series of proxies, they differ in the details of how the routing information is

maintained; in onion routing routing information is encrypted and included in the data, while in Crowds each host on the path in the crowd maintains state about the path. In addition, there exist cryptographic protocols (e.g., [53]) that provide the same anonymity but are more resistant to attacks involving collaborators.

4.2 Initiator Anonymity

Initiator anonymity hides the address of the originator of a transmission, and most previous work has been to provide this type of anonymity. The most common method of hiding the identity of an initiator is to use a single proxy that forwards connections for an entire organization; this is frequently done to hide the internal network addresses for security reasons more than to provide privacy. Indeed, by the definition of end-to-end anonymity, it is not anonymous, as an internal eavesdropper could determine the initiator's address. Additionally, given an eavesdropper who may be able to force a proxy to collaborate, the proxy provides a single point of attack for the eavesdropper. If the single proxy can be corrupted, then the identities of all initiators using it can be determined. To help reduce the chance that this type of attack will succeed, other schemes have been proposed and implemented that select a path through a subset of group of proxies.

Onion Routing [42] is a method that provides anonymity in exactly this manner. Each member of the protocol knows the addresses of all other members. The initiator begins by choosing a route through the other onion routers, constructing a path string denoting the route to be stored in each outgoing packet. Particular to onion routing is the layered encoding of the path string. For each onion router on the path, a *layer* of a connection setup packet is constructed consisting of the identity of the next onion router, the encryption key for communicating with the previous onion router, and the previous layer. Each layer is encrypted with the public key of the corresponding router. As the packet is forwarded through the path of onion routers,the layers are peeled off, until it reaches the last onion router in the path whose responsibility it is to forward all data directly to the responder. All data forwarded between neighboring onion routers are encrypted using the

shared key specified in each layer. All requests from the initiator are sent along the same path of onion routers. Replies are sent to the last onion router on the path, which in turns forwards the data along the reverse path of onion routers towards the initiator. As each onion router only knows the identity of the previous and next onion router on the path, or the initiator or responder if it is at the path endpoints, the chance that a corrupt router is able to identify the initiator is proportional to the number of onion router in the network.

The Crowds protocol [43] also requires that members of the crowd know each other's identities. A communication then starts with the initiator choosing a random *path id* for an outgoing packet. The initiator then forwards the packet to a random crowds member. Each crowd member receiving a packet then randomly decides whether to forward it on to the responder or to another random crowd member. Once the responder receives the packet, it returns a packet to the initiator through the last crowd member, whose identity was shown when it forwarded the packet. The return packet is then forwarded back to the initiator along the reverse path among Crowd members. Subsequent packets between the initiator and responder always follow the same path. While the identities of the crowds members are public knowledge, responders, eavesdroppers, and other crowds members never learn which particular crowd member is the initiator as in every case the initiator is a member of the crowd. It is therefore impossible to tell (short of timing attacks on all messages to a crowd member) if the initiator is sending its own message or forwarding one of another member. In this case, each member of the crowd gains anonymity at the cost of bandwidth in forwarding others communications.

For a crowd of size n with c malicious collaborating crowd members, if $n \geq (p_f/(p_f - 1/2)(c+1))$, where p_f is the probability of forwarding a packet to the destination, then the initiator's *probable innocence* is maintained against the collaborators. But Crowds relies on the assumption that IP addresses are a relatively expensive resource, and that an attacker might have difficulty in obtaining enough different IP addresses to mount an attack against the Crowd; in fact, this does not necessarily hold true. A reasonable powerful attacker might have little difficulty in obtaining an adequate

number of IP addresses, or might even attack the unicast routing in order to hijack an entire range of addresses. Later, we develop a scheme for proving uniqueness in a group that can be linked to an arbitrary resource, thus limiting the ability of an attacker to gather enough resources to mount an attack.

Each of these fully anonymous methods require a group of application-level proxies that are aware of the addresses of all other proxies. We present two new methods of maintaining initiator anonymity: one that does not require a proxy group at all, but instead uses only common functionality of IPv4; and another that maintains the use of a proxy group but removes the requirement that full group membership information is available to each proxy. Each of these protocols relies on *multicast* routing. Multicast is a service that provides many to many connectivity by building a distribution tree from each source to all receivers. A multicast *group* is the set of all sources and receivers and is assigned a single class D IP address. Sending to that address will result in delivery to all receivers of that group. An initiator can avoid using its own IP address for replies if it subscribes to a multicast group and has the responder send to the multicast address instead. As the initiator may be sending to the responder via unicast and receiving replies via multicast, it cannot use TCP connections. Instead, it uses unreliable UDP transmissions. This has the added benefit of allowing the initiator to forge the source address in the packets it sends. These *forged headers*, while not part of the IPv4 specification per se, are commonly used in many malicious denial-of-service attacks to hide the identity of an attacker who might be trying to flood a specific machine with requests or traffic. Since these packets are *spoofed*, with improper header information, tracing the source of these attacks is difficult. By combining multicast and UDP transmissions and by forging the header information in a packet we can construct an anonymous protocol using only these existing services.

4.2.1 Multicast Initiator Anonymity (MIA)

Multicast Initiator Anonymity is possibly the simplest protocol that retains the anonymity of the initiator from both the responder and an eavesdropper. It achieves this by trading anonymity

for bandwidth, using multicast communication to receive replies to forged requests.

The initiator, I, begins by subscribing to a multicast address, m, over which to receive replies.

Step 1: The initiator generates a 128-bit random number, id, that will be included as an identifier in each reply to distinguish it from other multicast traffic. The initiator transmits to the responder, R, using UDP packets with forged header information (denoted $F(I)$). The transmission includes the multicast address on which to reply and the 128-bit identifier.

$$F(I) \rightarrow R : m, id, \text{data} \tag{1}$$

Step 2: After receiving the transmission, the responder replies on the specified multicast address, prefacing each packet with the 128-bit number that the initiator selected.

$$R \rightarrow m : id, \text{reply-data} \tag{2}$$

The initiator can listen to the multicast traffic and drop any packets that are not the response it is waiting for; it can easily determine which those packets are by looking for the 128-bit identifier.

The initiator's anonymity is preserved from the initiator and from eavesdroppers as it never includes its actual IP address in any packet that is sent. The initiator forges the headers on the outgoing packets, replacing its actual IP address with another. In some cases it will be possible to choose the fake address at random. In many cases, however, the forged address will have to be selected from a more limited range, as many Internet service providers filter to prevent outgoing packets with external addresses from leaving their domain; this is done to prevent local users from originating denial of service attacks that depend on forged addresses. In such situations the initiator will have to choose a fake address that comes from within the domain doing the filtering; this does not identify the initiator directly, though it does gives a clue as to its location. The replies from the responder also do not include the initiator's address as they are sent to the multicast address, which could any number of recipients, depending on the multicast address used; the multicast address gives no clue as to the identity of any member of the group, as it is in essence a label for the group rather

than an actual address. While the initiator may have to listen to other multicast traffic to receive its reply, the extra cost of wasted bandwidth is the price it pays for anonymity.

Though this simple protocol provides initiator anonymity, there are several improvements that can be made to improve the reliability and anonymity. Since all transmissions are via UDP, packet loss can be a problem, since retransmissions are not automatically generated. To correct this, the initiator can use Forward Error Correction (FEC) codes [9] so that lost packets might be recoverable from packets that do arrive. To improve the privacy, of the protocol, encryption can be used to hide the contents of transmissions. A responder can have a public key that is available for potential initiators to look up, or the first set of messages sent from the initiator can request the responder's public key; thereafter the initiator can create a shared key, encrypt it with the responder's public key and send it to the responder for use in all further communications. Encrypting the contents of the packets can help remove clues as to the identity of the initiator that might be in the packet contents.

If anonymous communication becomes popular, a range of multicast addresses could be designated for anonymous communication. An initiator could subscribe to several of these, listen to the amount of traffic each was carrying and choose one that had enough receivers to ensure anonymity but not so many that it would be overwhelmed processing others traffic. If an initiator wanted to hide the fact it was even communicating anonymously, it could choose a multicast group that was carrying some non-anonymous multicast traffic and specify that address to the responder. The replies would then be mixed with the existing multicast traffic. Though the other, legitimate members of the group would be forced to process and discard the traffic from the responder, the initiator would gain the benefit of appearing a member of the legitimate group. Initiators could also make the eavesdroppers task harder by subscribing to several multicast addresses and varying the group to which the responder should reply. The traffic would then come across several groups and might be varied over time to achieve the same effect as frequency hopping radios.

4.3 Comprehensive Anonymity

Previous work has focused solely on providing anonymity for the initiator of a connection. In this section we discuss protocols and techniques for hiding the identity of a responder, and for providing completely anonymous communication in which both the initiator and responder are anonymous.

4.3.1 Responder Anonymity

Protocols providing initiator anonymity can be easier to design because the initiator has the opportunity to set up the return path leading back itself as the path to the responder is found. However, protocols providing responder anonymity must be designed to route packets to a responder without the opportunity to set up a return path ahead of time, since we must assume the responder does not know the which initiator will the communications.

We provide two solutions to this problem, each using a different technique to forward messages to an anonymous responder, which then allows the responder to set up a connection to the initiator using a protocol for initiator anonymity (See Section 4.2). The first technique, which we call Proxy for Responder Anonymity (PRA), utilizes a proxy, and the second technique, which we call Multicast Responder Anonymity (MRA), utilizes multicast routing.

Proxy for Responder Anonymity (PRA)

In PRA, responders rely on a publicly-advertised proxy to accept and relay datagrams. The link between the responder and the proxy remains anonymous. The protocol requires that the responder contact the proxy prior to receiving communications from any initiator. Figure 4.1 illustrates the protocol.

Step 1: The responder, R, sets up a connection to a known proxy, P, using a protocol that provides initiator anonymity, such as MIA, Crowds, or Onion Routing (See Section 4.2). The proxy maintains a public alias for the anonymous responder, without knowing the true identity of

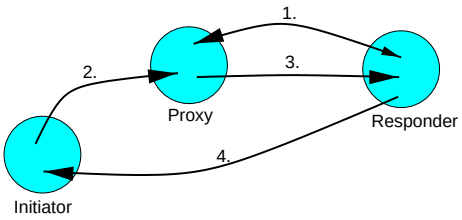


Figure 4.1: The PRA protocol.

the responder. Anonymous responders may wish to have more than just their alias advertised. For example, the proxy may also identify the type of service the responder is providing. The responder may also wish to generate a public/private key pair and use it to sign the alias (along with a time stamp) so that past correspondents can identify them from past sessions. The proxy need only be trusted to not engage in a denial-of-service attack by dropping requests, but does not have to be trusted to maintain the responder's anonymity.

$$R \rightarrow P : \text{(initiator anonymity set up from R to P)} \tag{1}$$

Step 2: Packets from an initiator are sent to the proxy, encapsulated with a header that specifies the alias of the responder. The proxy then forwards the packet to the responder via the anonymous protocol they share.

$$I \rightarrow P : A_r, \text{data} \tag{2}$$

Step 3: Once the proxy receives a datagram for a particular anonymous responder, the request is un-encapsulated and sent over the previously established anonymous channel between the proxy and responder.

$$P \rightarrow A_r : I, \text{data} \tag{3}$$

Step 4: Forwarding future communications through the proxy could introduce a bottleneck. Therefore, as a performance improvement, the responder could set up a connection to the initiator using a protocol for initiator anonymity, reversing its role in the protocol with the true

initiator. If only a brief reply to the initiator is needed, setting up an anonymous return connection may not be worth the overhead. Instead, the responder may choose to simply forge the return address of the packets containing the reply, or reply via the proxy.

$$A_r \rightarrow I : \text{(initiator anonymity set up from R to I)} \tag{4}$$

The PRA technique is similar to a method commonly used to load balance HTML requests over several web servers: a proxy accepts requests from a web browser on behalf of a group of servers, redirecting the request to one server in the group. Replies are sent directly to the web browser from the selected server, bypassing the proxy. The addresses of the servers are not advertised; only the address of the proxy. In this context, the web browser would be an initiator, and the selected server would be a responder. PRA builds on this method, hiding the identity of the responder from both the proxy and the initiator.

Multicast Responder Anonymity (MRA)

A proxy can be avoided altogether if the responder listens on a multicast address for new requests. An equivalent idea has been put forward in the past for broadcast networks [40]; multicasting is a natural mechanism for applying this technique to IP networks. Instead of advertising the proxy's address and alias, as in PRA, a multicast address m, an alias in the form of a random number id, and a public encryption key P_r, are advertised. The alias serves as a unique marker in case more than one anonymous responder chooses to listen on the same multicast address. The encryption key allows private communication over a broadcast medium. The choice of a multicast group may regulated to a section of the multicast address space shared by anonymous responders or a multicast address already in use by another user on the same network as the responder may be hijacked to hide the request.

Step 1: R chooses m, id, and a public key K_r and advertises these values on a public site (e.g., IRC channels, web sites, newsgroups, etc).

Step 2: The initiator I sends its message to the multicast group. This transmission reveals I's identity.

$$I \rightarrow m : id, \{\text{data}\}_{K_r} \tag{2}$$

Step 3: Once the responder receives the initiator's packet on the multicast address, it checks that the alias in the data portion of the packet is its own. A reply can be sent either with a forged source address in the packet's header if the transaction is brief, or the reply can be sent after a connection to the initiator is set up using an protocol for initiator anonymity, just as in PRA.

$$A_r \rightarrow I : \text{(initiator anonymity set up from R to I)} \tag{3}$$

The reply should be signed with the responder's private key to prove that the response was sent by the correct responder. As anyone may listen to a multicast group, it would otherwise be easy to masquerade as a particular responder. Note that while the identity of the responder may not be narrowed down further than the members of the multicast address, an immediate response by the responder may give away its identity, if eavesdroppers employ a timing attack.

The use of multicast in MRA creates more traffic than PRA. However, PRA requires the cooperation of a third party host, even though it need not be trusted to hide the identity of the responder.

4.3.2 Mutual Anonymity - Zen routing

In true anonymous communication, both the initiator and responder are completely anonymous. It is curious to consider the koan-like quality of the service provided by routing protocols that maintain anonymity for both initiators and responders, which we refer to as *zen routing*: the initiator does not know where packets are destined, the routers do not know where packets are destined, the responder does not know the origin of packets, and replies are routed back to an unknown location. Designing zen routing protocols is difficult for those reasons. However, it is enlightening to realize

that since we already have protocols for initiator anonymity and protocols for responder anonymity, in order to design a zen routing protocol, all we need do is compose a connection between the two correctly.

One interesting aspect of zen routing is that two anonymous communicators need some method of finding each other. For this it is possible to use a directory service. Anonymous communicators can use an initiator anonymous protocol to contact the directory and register with it, leaving information about what they might wish to communicate about, their anonymous contact information, and a public key. When an initiator looks up and contacts a responder, it can send a nonce. The responder can sign the nonce with its private key and return it to the initiator. This way the initiator can be sure that it is communicating with the entity that registered with the directory service. The responder may also send a nonce, and the initiator can sign it and return it with a public key. This will allow each of them to identify that they are communicating with the same entity between sessions, assuming that neither gives away their private key. They might also use the keys to distribute some shared secret between them and use that to verify their identities later.

Multicast Zen Routing

The simplest zen protocol is called Multicast Zen Routing (MZR). In this protocol, both the initiator and the receiver listen on a multicast address while sending their messages with spoofed headers. In essence, this is a combination of the simplest initiator protocol, MIA, and the simplest responder protocol, MRA, though the transmissions occur via spoofed multicast, instead of spoofed unicast. The initiator sends its request to the responder, including a multicast address and a unique ID. The responder uses this information to send a reply, again forging the multicast header. For a host, sending a multicast packet with forged headers is as easy as sending a similar unicast packet. However, depending on the multicast routing protocol used in the network, sending spoofed multicast packets may not be effective. This occurs because the multicast routing protocol may limit the choice of spoofed source addresses that are usable.

If the participating members are on such a network, then a proxy must be used instead. In this case, MZR becomes a combination of MIA and PRA. The initiator sends a datagram, made up of the sender's multicast information, the data, and the receiver's reply information, to the proxy. The proxy reads the sender's information from the datagram and uses it to forward the rest of the datagram to the responder. The responder uses the same mechanism to reply to the initiator, using the same or some other proxy. This method removes the dependency on spoofing multicast multicast packets.

4.4 Discussion

In this section we discuss the tradeoffs and degree of anonymity of the protocols presented in this chapter and show the results of an implementation of the simplest protocols that use multicast.

4.4.1 Evaluation of Anonymous Protocols

An anonymous protocol can be evaluated based on its resistance to attacks that attempt to defeat the anonymity of the protocol, and based on what information about hidden identities is revealed if the attack is successful. There are a number of different attacks that are possible against anonymous protocols that fall into three general categories. *Traffic analysis* consists of monitoring traffic and using the headers or contents of transmissions to determine the identities of the initiator or responder. Part of this attack might be a *timing attack*, which correlates the times network members send or receive messages to link them with each other. *Collaboration* occurs when some participant in the communication (be it an initiator, responder or proxy) does not act in a way defined by the protocol but instead uses its position to attempt to determine the identity of other participants. *Traceability* is an attack by which a powerful attacker, who is able to find some point along the communications path, attempts to follow the connection back through a series of routers or proxies to determine the endpoints. The attacker may do this by monitoring multiple network segments or by coercing proxies to become collaborators, if necessary by dispatching agents to force the proxy

operator to collaborate. If any of these attacks are successful, then the identities of some of the participants may be revealed, depending on the design of the protocol. In some cases, however, the protocol can be designed in a way that even successful attacks do not lead to a particular individual. Instead, they maintain a some *degree of anonymity* [43], which preserves anonymity to some extent. An example of this is Crowds [43], in which some attacks fail to locate which member sent or received a message; instead the attacker can only find out which crowd has the participant as a member.

An anonymous protocol can also be evaluated based on the various tradeoffs of each anonymous routing protocol design. An anonymous protocol can increase a participants overhead as compared to non-anonymous methods in several ways. The protocol might require significant processing power, especially if frequent encryption, decryption or verification of digital signatures were required. A participant might also be required to devote significant amounts of memory to preserving the state of the protocol; for example, a proxy such as that used in Crowds[43] might maintain routing state, a timer or other state for each connection it forwards. As the number of connections in the anonymous service grows, this overhead could become substantial. Finally, being a member of an anonymous protocol might require a substantial contribution of network bandwidth, especially if the protocol requires that the member receive transmissions destined for others, either for forwarding or to obscure the identity of individuals within the group. Frequently, the overhead required of one member may increase with the number of other members, limiting the scalability of the protocol. Typically, there are tradeoffs between the degree of anonymity provided and the overhead required, as participants process messages sent from or destined for others in exchange for a higher degree of anonymity in the event of an successful attack.

4.4.2 Multicast Routing, Forging, and Anonymity

The new protocols presented in this work depend on multicast routing to provide anonymity for the receiver of a transmission. Results of our implementation show that this works very well. In some protocols, the initiator's identity is hidden by the use of forged packet headers. While in

```
On host 1 (IP 10.1.1.1):
host1:> ./initiator -a i 10.2.2.2 10000
Spoof address set to:  196.164.167.123
Randomly generated reply info is:
Address:  234.26.109.6
Port:  36891
ID: 1709436268
Sent 48 bytes.
Received a reply of (Reply) apparently from 10.2.2.2

On host 2 (IP 10.2.2.2):
host2:> ./responder -a i 10000
Received request, apparently from:  196.164.167.123
Sent 37 bytes

tcpdump at host 2:
23:32:10.569080 196.164.167.123.59871 > HOST2.10000
23:32:12.675077 HOST2.10000 > 234.26.109.6.36891
```

Figure 4.2: Multicast Initiator Anonymity

theory this is an effective methods of hiding the sender's identity, in practice the anonymity they provide and the ability of some hosts to use them is limited.

Implementation

In order to test the performance of multicast routing as a method of providing anonymity we implemented several of the protocols we have proposed: MIA, MRA, PRA and Multicast Zen Routing. The tests were conducted between two hosts running FreeBSD [2] on the CAIRN network. CAIRN uses DVMRP as its multicast routing protocol.

Figure 4.2 shows the execution of Multicast Initiator Anonymity (MIA). Host 1 is the initiator, and the `-a i` flag indicates that the initiator's anonymity is to be preserved. The IP address `10.2.2.2` is that of the responder, and `10000` is the port the responder is listening to. Host 2 is the responder, and besides the correct anonymity flag, only needs a port number on which to listen.

Before host 1 can send its request, it needs to generate a fake source address, random multicast group, port and unique ID to use to receiving. It sends the request (an ASCII string

[2] Host information has been removed to maintain the authors anonymity.

```
On host 1 (IP 10.1.1.1):
host1:> ./initiator -a r -i 1122334455 233.33.33.33 10000
Sent 43 bytes.
Received a reply of:  Reply
It was apparently from:  39.12.151.215

On host 2 (IP 10.2.2.2):
host2:> ./responder -a r -i 1122334455 -m 233.33.33.33 10000
This is the multicast address I will listen on.
Address:  233.33.33.33
Port:  10000
ID: 1122334455
Received request, apparently from:  10.1.1.1
Sending reply to host:  10.1.1.1 port:45829
Sent 33 bytes.

tcpdump at host 1:
07:39:18.199243 host1.45829 > 233.33.33.33.10000
07:39:18.205154 39.12.151.215.63956 > host1.45829
```

Figure 4.3: Multicast Responder Anonymity

reading "Request") along with the reply information to the responder. The responder receives the request, prints the IP address it believes the request came from, and generates a multicast reply. Host 1 receives the reply (an ASCII string that reads "Reply") and prints the address it seems to have come from. Notice that the responder does not see the initiator's IP address. This is also shown in the output of *tcpdump*[3], which shows the receipt of the request, then the reply to the chosen multicast group.

Figure 4.3 shows the execution of Multicast Responder Anonymity (MRA). In this case the `-a r` flags indicate that the responder's anonymity is to be maintained. Responder anonymity requires more information be distributed before the protocol can be run. The initiator (at host 1) needs to know the multicast address the responder will be listening on, as well as the port and unique ID, which is specified with the `-i` flag. The responder at host 2 is similarly configured to listen on a particular multicast address with the `-m` flag.

The protocol is successful; the initiator never obtains the responder's IP address, nor is it available on the network. Instead it sees only the multicast address and the forged address

[3]tcpdump is a freely available tool for examining network transmissions and is available from: http://www-nrg.ee.lbl.gov/ftp.html

(39.12.151.215), used in the spoofed reply. The *tcpdump* trace recorded at the initiator shows the transaction.

We also implemented Proxy Responder Anonymity (using MIA to connect to the proxy), and Multicast Zen Routing. Because both PRA and MZR with a proxy show essentially the same thing as MIA and MRA, they are omitted. Multicast Zen Routing without a proxy was not successful on our network, due to the inability to spoof multicast packets, for reasons we detail below.

Forging Headers

Forging the headers of a datagram is perhaps the simplest method of hiding the identity of an anonymous participant. While this method prevents the use of reliable, connection-oriented protocols like TCP, its effectiveness has made it popular in a number of denial-of-service attacks that attempt to overwhelm a target machine with incoming packets [12]. By forging the source address of the packet, the attacker forces the target to go to the difficulty and expense of tracing the packets back through the network in order to stop the attack. Unfortunately, the malicious use of this method has made it more difficult to use for anonymous communications. In order to prevent users within the domain from launching such an attack, the domain may filter outgoing packets at its border, dropping those that have a source address that is outside the domain. This limits the ability of a participant in an anonymous protocol to choose an entirely random source address, and while they can use some other address from within the domain, it is easier to locate the source of a transmission. Using forged headers is very efficient, when possible, though it is subject to a traceback attack if there is a constant data stream to follow. It is also typically easy to detect the forged packets on the same local network of the originator by examining the MAC address of the transmission.

It is even more difficult to forge the source address of multicast packets. Some multicast routing protocols, including those commonly in use today like DVMRP [41] and PIM-DM [18], use the source address to ensure that the packet is routed correctly. When routers using these protocols receive a packet they check the source address against their routing table; if the path to that source

address is not across the interface from which the packet was received, the packet is dropped. This severely limits the choice of forged addresses an initiator can use, particularly as it is likely to be a leaf node on the multicast tree. In this case, the upstream router will only accept packets with an address from the same subnet as the originator, which does little to hide the originator's identity. This does not mean that it will never be a useful technique for anonymous communication, however. Other multicast routing protocols do not use this reverse patch check before forwarding the packet [21, 6]. These protocols instead use a single shared tree for all communication, and with these protocols it should be possible to send forged multicast packets, subject to the limits of administrative filtering. This is more true of CBT than PIM-SM. While CBT put packets on the tree without any checks, PIM-SM instead encapsulates them for transmission to the rendezvous point, where they are distributed down the shared tree. Even if the anonymous member were to send a spoofed multicast packet using PIM-SM, it would still be possible to see the address of the router performing the encapsulation and therefore obtain a good idea where to look for the originator. Someone wishing to send spoofed multicast over a tree formed this way must therefore forge not only the data packet, but the encapsulation from the first hop router as well. That also means they must be able to locate the rendezvous point, which may not be trivial, depending on how it is being selected.

Multicast routing

Multicast routing is an excellent method of hiding the identity of a receiver, though it has the disadvantage of not being fully deployed in the Internet. While this currently limits its use, the rapid growth of the Mbone [14, 20] means that multicast availability will be greater in the future. The very nature of multicast routing is what provides the initiator's anonymity. A multicast address is simply a label that gives no indication of who the receivers are. Currently, the only way to determine the receiver set of a multicast group, assuming they are not transmitting, is to trace the path from the source back to the receivers by examining the routing tables at each hop on the tree. This will not necessarily reveal the identity of a particular receiver, however. Since multicast

provides one-to-many communication, a traceback attack will yield an entire set of receivers. The composition of this set of receivers can vary. If they are all using the multicast group for anonymous communication, then, as in Crowds [43], no particular receiver will be identifiable. If the receiver is listening on some other multicast address that also carries non-anonymous traffic, it is possible that the receiver will be lost in that group of receivers, but this varies depending on the multicast routing protocol being used. Some protocols allow a specific source [54, 18] to be pruned from the tree and legitimate user of the group will prune out the messages destined to an anonymous hijacker. A traceback attack will, in this case, lead only to the anonymous receiver. For that reason, a receiver who is listening over this type of hijacked group should change groups periodically.

If a responder is listening on a multicast address and replying to an attacker, there is a unique attack that could give an attacker a good idea of where the responder is located. In IPv4, each unicast packet has a time-to-live (TTL) field that is decremented with each hop; the maximum value of the TTL is 255. When the TTL reaches zero the packet is dropped. This prevents packets from circulating the network endlessly in th event of routing loops. An attacker could send requests to the responder starting with a TTL of one and keep increasing it until the responder replies. At that point the attacker would know how many hops away the responder was. By repeating this from different points in the network, the attacker could effectively determine an approximate location for the responder. A responder listening on multicast should, therefore, only respond to requests with a large remaining TTL, and should vary the TTL of the packets it sends in reply so that the reverse attack is limited.

If a sender in an anonymous protocol is able to use spoofed multicast packets, a similar attack is possible. Multicast packets also have a TTL field, but it is used differently. In order to scope the area a multicast can reach, a multicast packet leaving a domain has its TTL decreased by 32; a packet that is sent with a TTL of less than 32 will not leave the domain [59]. Similarly, a multicast packet has its TTL decreased by 64 at the edge of a region and by 128 at the edge of a continent. By monitoring the packets coming from a particular sender at different points and

recording the TTL of the sender's packets, it may again be possible, over time, to determine the sender's location. This occurs because the attackers will see the TTL differently as it may cross different boundaries to reach the points where they are listening.

While using multicast to hide the identity of a receiver is very effective and simple, the trade-off is that anonymity provided comes at the cost of bandwidth. The anonymous receiver should listen on a multicast group with other receivers if it wishes to limit traceback attacks, so that if such an attack were to occur it would not be possible to differentiate one receiver out of the group of all receivers. This means that each receiver must listen to traffic destined for other receivers. While from a routing perspective this is inefficient, it may be worth the anonymity provided.

Chapter 5

Conclusions

We first described a new protocol for hierarchical multicast routing called HIP that joins heterogeneous routing domains using the Ordered Core Based Tree protocol. The method makes domains appear as a single virtual router on a higher-level shared tree using a simple protocol that organizes the border routers of the domain to simulate a single OCBT router. Any routing domain can be made a virtual router; all that is required is the ability to send multicast packets internally. Recursive creation of virtual routers yields a flexible hierarchy that is robust; has no strict ordering of levels; aligns easily with existing unicast domain boundaries; and always attempts to make the path to the center point of the tree as short as possible. The mechanisms for center-point location distribution are simple and effective in minimizing the amount of control traffic needed.

While maintaining the same functionality of other hierarchical multicast schemes, HIP approaches the problem in a very different manner. Routing between the domains is based solely on the unicast routing tables; the lower-level domains do not need to be explicitly named and no separate routing needs to occur to build paths to domains. This allows great flexibility in the addition or modification of domains in the region. Data traffic flows over a single tree, and while higher-level control messages may be encapsulated for transmission across a region, data packets are never encapsulated or duplicated. The tree itself is always built with an attempt at forming the average shortest path to the center point for the group. The ordering of the hierarchy is also

extremely general; except for the highest level domain, no domain needs to know what its place is in the hierarchy. While the bulk of the control traffic in HIP is for the distribution of proper core and best-path information, the amount of traffic is minimized as much as possible by only sending the information where it is requested or where it needs to be delivered for the proper operation of encapsulated protocols. The HIP protocol takes best advantage of the convergence of the unicast and multicast topographies and the development of high performance shared-tree protocols, while retaining the ability to inter-operate with existing domains and protocols.

We then described KHIP, an extension to HIP that provides secure, hierarchical multicast routing. Keyed HIP maintains the efficiency of multicast routing of HIP while providing authentication services and secure routing so that only authorized receivers may use the multicast tree and obtain keys for sending or receiving data. KHIP adds an authentication service that issues certificates to entities who are allowed access and who authenticate themselves with a known public key. These certificates are included in signed control messages to prove that the sender has the authority to alter the tree. The tree itself is divided into sub-branches, and messages within each sub-branch also carry nonces to prevent forgery or replay attacks that could build a branch of the tree to an unauthorized router. Each sub-branch can also use a shared key for data transmission, thus obviating the need for a single key shared across the entire tree. Data packets are re-processed for transmission between sub-branches. The amount of work needed for re-processing is minimized by encrypting the data in a random key and encrypting that random key with the shared sub-branch key. Re-processing is thus limited to re-encrypting the random key in the new branch key and adding new nonces for transmission in the new sub-branch. Untrusted routers can only eavesdrop on the encrypted data flow if they happen to lie on the path between two authorized entities. While some denial of service attacks by these untrusted routers are possible using unsigned control messages, they are no more effective than if the untrusted router was simply dropping control packets. Keyed HIP is the first secure, hierarchical multicast routing protocol. It meets the needs of security while providing delivery of data across many receivers.

Finally, we introduced new methods of providing anonymity in the Internet. For the first time, we have considered and shown methods for providing anonymity not only for the initiator of an anonymous connection, but for the responder as well. We also discuss the routing of packets between two simultaneously anonymous entities, calling this zen routing. We introduced the idea of using multicast routing as a method of providing anonymity for the receiver of a connection. This turns out to be very effective way of hiding a receiver, at the cost of the wasted bandwidth from listening and discarding traffic meant for other receivers. It is highly resistant to being traced back, however, as tracing the multicast tree does not lead to a single receiver but all the receivers of the multicast group, making identification of any particular one receiver difficult.

Though the technique has been used maliciously in denial-of-service attacks, we also introduce the idea of forging the source information in unicast headers to hide the identity of the sender of an anonymous communication. This is effective, though it can be subject to a traceback attack that would lead to the single sender. The use of forged headers is also sometimes limited administratively by some domains, and the use of forged headers in multicast packets can be severely limited by the multicast routing protocol that is being used in the network.

With these techniques, and with the use of proxies, we have developed anonymous communication protocols that protect the initiator (Multicast Initiator Anonymity - MIA), the responder (Multicast Responder Anonymity - MRA) and both the initiator and responder (Multicast Zen Routing - MZR). Implementation results for MIA and MRA show them to be effective in hiding the identities of the hosts that use them to communicate.

Bibliography

[1] Lucent Personalized Web Assistant. Available at `http://www.bell-labs.com/projects/lpwa`.

[2] F. Baker, editor. Requirements for IP version 4 routers. Request for Comments 1812, June 1995.

[3] A. Ballardie. Scalable Multicast Key Distribution. RFC 1949, May 1996.

[4] A. Ballardie, B. Cain, and Z. Zhang. Core Based Trees (CBT version 3) Multicast Routing. Internet-Draft, March 1998.

[5] A. Ballardie and J. Crowcroft. Multicast-Specific Security Threats and Counter-measures. In *Proceedings of the Symposium on Network and Distributed System Security*, pages 2–16, San Diego, CA, February 1995. IEEE Computer Society Press.

[6] A. Ballardie, P. Francis, and J. Crowcroft. Core Based Trees (CBT): An Architecture for Scalable Inter-Domain Multicast Routing. In *Proc. ACM SIGCOMM'93*, pages 85–95, October 1993.

[7] A.J. Ballardie. Core Based Trees (CBT version 2) Multicast Routing Protocol Specification. Internet draft, July 1997. Work in progress.

[8] E. Biham. How to Forge DES-Encrypted Messages in 2^{28} Steps. Technical report, Technion, 1996.

[9] R. Blahut. *Theory and Practice of Error Correction Codes.* Addison-Wesley, 1983.

[10] S. Boeyen, T. Howes, and P. Richard. Internet X.509 Public Key Infrastructure Operational Protocols - LDAPv2. RFC 2559, April 1999.

[11] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Trans. on Computer Systems*, 8(1):18–36, February 1990.

[12] CERT Advisory CA-98.01. 'Smurf' IP Denial-of-Service Attacks. http://www.cert.org/advisories/CA-98.01.smurf.html, January 1998.

[13] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, and B. Pinkas. Multicast Security: A Taxonomy and Efficient Construction. In *Proc. IEEE Infocom*, March 1999.

[14] S. Casner. Are you on the MBone? *IEEE Multimedia, Summer 94*, pages 76–79, 94.

[15] David Chaum. Blind signatures for untraceable payments. In *Proc. Crypto'82*, pages 199–203, 1982.

[16] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[17] S. Deering. *Multicast routing in a datagram internetwork*. PhD thesis, Stanford University, Palo Alto, California, December 1991.

[18] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, A. Helmy, and L. Wei. Protocol Independent Multicast Version 2, Dense Mode Specification. Internet draft, May 1997. Work in progess.

[19] S.E. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. An Arcitecture for Wide-Area Multicast Routing. In *Proc.of the ACM SIGCOMM94*, pages 126–135, London, UK, September 1994.

[20] H. Eriksson. MBone: the Multicast Backbone. *Communications of the ACM*, 37(8):54–60, August 1994.

[21] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol Independent Multicast-Sparse Mode (PIM-SM):Protocol Specification. Internet draft, September 1997. Work in progress.

[22] D. Estrin, M. Handley, A. Helmy, P. Huang, and D. Thaler. A Dynamic Bootstrap Mechanism for Rendezvous-based Multicast Routing. Technical report, University of Southern California, 1997. http://netweb.usc.edu/pim/.

[23] D. Estrin, M. Handley, S. Kumar, and D. Thaler. The Multicast Address Set Claim (MASC) Protocol. Internet-draft, Novermber 1997. Work in Progress.

[24] W. Fenner. Internet Group Management Protocol, Version 2. RFC 2236, November 1997.

[25] W. Ford. *Computer Communications Security*. Prentice Hall, 1994.

[26] L. Gong, R. Needham, and R. Yahalom. Reasoning about Belief in Cryptographic Protocols. In *Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy*, pages 234–48, May 1990.

[27] L. Gong and N. Shacham. Elements of Trusted Multicasting. In *Proceedings: 1994 International Conference on Network Protocols*, Boston, MA, October 1994. IEEE Computer Society Press.

[28] L. Gong and N. Shacham. Trade-offs in Routing Private Multicast Traffic. In *Proceedings of GLOBECOM '95*, pages p. 2124–8, Singapore, November 1995. IEEE Computer Society Press.

[29] M. Handley, J. Crowcroft, and I. Wakeman. Hierarchical Protocol Indepenent Multicast (HPIM). University College London, November 1995.

[30] R. Hauser, T. Przygienda, and G. Tsudik. Reducing the Cost of Security in Link-State Routing. In *Proceedings of the Symposium on Network and Distributed System Security*, pages 93–99, San Diego, CA, Feb 1997.

[31] F. Jordan and M. Medina. Secure Multicast Communications using a Key Distribution Center. In P. Viega and D. Khakar, editors, *Proceeding of IFIP TC6 International Conference on Information Networks and Data Communication.*, pages 367–380, Funchal, Portugal, April 1994. Elsevier Science B.V.

[32] S. Keshav and S. Paul. Centralized multicast. Technical report, Cornell University, April, 1998.

[33] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D. Estrin, and M. Handley. The MASC/BGMP Architecture for Inter-domain Multicast Routing. In *Proceedings of ACM SIGCOMM98*, pages 93–104, Vancouver, Canada, August 1998.

[34] S. Mittra. Iolus: A Framework for Scalable Secure Multicasting. In *In Proceedings of ACM SIGCOMM97*, Cannes, France, September 1997.

[35] J. Moy. Multicast Extensions to OSPF. Technical report, RFC 2117, March 1994.

[36] S. Murphy and M. Badger. Digital Signature Protection of the OSPF Routing Protocol. In *Proceedings of the Symposium on Network and Distributed System Security*, pages 93–102, San Diego, CA, Feb 1996.

[37] S.H. Ong and S.H. Goh. A Generic Multicast-key Determination Protocol. In *Proceedings of IEEE Singapore International Conference on Networks/International Conference on Information Engineering '93*, pages p. 518–22, Singapore, Sept 1993.

[38] M. Parsa and J.J. Garcia-Luna-Aceves. A protocol for scalable loop-free multicast routing. *IEEE Journal on Selected Areas in Communications*, 15(3):316–331, April 1997.

[39] M. Parsa, Q. Zhu, and J.J. Garcia-Luna-Aceves. An interative algorithm for delay-constrained minimum-cost multicasting. *ACM Transactions on Networking*, 1998.

[40] Andreas Pfitzmann and Michael Waidner. Networks without user observability. In *Computers & Security*, volume 6, pages 158–66, April 1987.

[41] T. Pusateri. Distance Vector Multicast Routing Protocol. Internet-Draft, March 1998. Work in Progress.

[42] Michael G. Reed, Paul F. Syverson, and David M. Goldschlag. Proxies for anonymous routing. In *12th Annual Computer Security Applications Conference*, pages 95–104. IEEE, December 1995.

[43] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. Technical Report 97–15, DIMACS, April 1997.

[44] Y. Rekhter and T. Li. A Border Gateway Protocol 4 (BGP-4). Technical report, RFC 1771, March 1995.

[45] B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., 2nd edition, 1996.

[46] C. Shields. Ordered Core Based Trees. Master's thesis, University of California, Santa Cruz, Santa Cruz, California, June 1996.

[47] C. Shields and J.J. Garcia-Luna-Aceves. The Ordered Core Based Tree Protocol. In *Proceedings of the IEEE INFOCOM97*, Kobe, Japan, April 1997. IEEE.

[48] C. Shields and J.J. Garcia-Luna-Aceves. Hierarchical Multicast Routing. In *Proc. Seventeenth Annual ACM SIGACT-SIGOPS Symposium on principles of distributed computing (PODC 98)*, Puerto Vallarta, Mexico, June 28–July 2 1998.

[49] B. Smith and J.J. Garcia-Luna-Aceves. Efficient Security Mechanisms for The Border Gateway Routing Protocol. *Computer Communications (Elsevier)*, 21(3):203–210, 1998.

[50] B. Smith, S. Murthy, and J.J. Garcia-Luna-Aceves. Securing Distance Vector Routing Protocols. In *Proc. Internet Society Symposium on Network and Distributed System Security*, San Diego, California, February 1997.

[51] D. Thaler, D. Estrin, and D. Meyer. Border Gateway Multicast Protocol (BGMP). Internet-draft, October 1997. Work in Progress.

[52] A. S. Thyagarajan and S. E. Deering. Hierarchical distance-vector multicast routing for the MBone. In *Proceedings of the ACM SIGCOMM95*, Cambridge, Massachusetts, September 1995.

[53] Michael Waidner and Birgit Pfitzmann. The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability. In *Eurocrypt '89*, 1989.

[54] D. Waitzman, C. Partridge, and S. Deering. Distance vector multicast routing protocol. Request for Comments 1075, November 1988.

[55] D. Wallner, E. Harder, and Ryan C. Agee. Key Management for Multicast: Issues and Architectures. Informational RFC, September 1998.

[56] Anonymizer web site. Available at `http://www.anonymizer.com`.

[57] M. Wiener. Efficient DES Key Search. Technical Report TR-244, School of Computer Science, Carleton University, Ottowa, Canada, May 1994.

[58] C. Wong, M. Gouda, and S. Lam. Secure Group Communications Using Key Graphs. In *Proceedings of the ACM SIGCOMM98*, pages 68–79, 1998.

[59] G. Wright and W. R. Stevens. *TCP/IP Illustrated Volume 2.* Addison-Wesley Publishing Company, 1995.