Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2007-035 June 21, 2007

Stateful Anycast for DDoS Mitigation

Richard E. Hansen

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Stateful Anycast for DDoS Mitigation

by

Richard E. Hansen
S.B. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2006

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

June 2007

© 2007 Massachusetts Institute of Technology. All rights reserved.

Author .....
Department of Electrical Engineering and Computer Science
May 28, 2007

Certified by .....
Karen R. Sollins
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Theses

Stateful Anycast for DDoS Mitigation

by

Richard E. Hansen

Submitted to the Department of Electrical Engineering and Computer Science
on May 28, 2007 in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Distributed denial-of-service (DDoS) attacks can easily cripple victim hosts or networks, yet effective defenses remain elusive. Normal anycast can be used to force the diffusion of attack traffic over a group of several hosts to increase the difficulty of saturating resources at or near any one of the hosts. However, because a packet sent to the anycast group may be delivered to any member, anycast does not support protocols that require a group member to maintain state (such as TCP). This makes anycast impractical for most applications of interest.

This document describes the design of *Stateful Anycast*, a conceptual anycast-like network service based on IP anycast. Stateful Anycast is designed to support stateful sessions without losing anycast's ability to defend against DDoS attacks. Stateful Anycast employs a set of anycasted proxies to direct packets to the proper stateholder. These proxies provide DDoS protection by dropping a session's packets upon group member request. Stateful Anycast is incrementally deployable and can scale to support many groups.

Thesis Supervisor: Karen R. Sollins
Title: Principal Research Scientist

Contents

1 Introduction **11**

1.1 Two Types of DDoS Attacks 12
1.2 Assumed Attacker Abilities 13
1.3 How DDoS Attacks are Possible 13
1.3.1 Portable Stateholder Handles 13
1.3.2 Permanent Stateholder Handles 14
1.4 Why Attackers Launch DDoS Attacks 14
1.5 Defending against DDoS Attacks 15
1.6 Anycast as a DDoS Defense 17
1.6.1 Anycast Defined 17
1.6.2 How Anycast Helps with DDoS 18
1.6.3 Anycast's Big Limitation 19
1.7 Improving Upon Anycast: *Stateful Anycast* 19
1.8 Organizational Overview 20

2 Background and Related Work **21**

2.1 BGP: Global Unicast Routing on the Internet 21
2.2 IP Anycast 23
2.2.1 Deploying an IP Anycast Group 23
2.2.2 IP Anycast Usage Today 25
2.2.3 Characteristics of IP Anycast 25
2.2.4 Limitations of IP Anycast 27
2.3 Proxy IP Anycast Service (PIAS) 28
2.3.1 Properties of PIAS 30
2.3.2 Scalability of PIAS 32
2.4 Internet Indirection Infrastructure (*i*3) 33
2.5 Distributed Hash Tables and Structured Peer-to-Peer Overlay Networks 34
2.6 Other Related Work 35
2.6.1 Anycast 35
2.6.2 DDoS Mitigation 35

3 Stateful Anycast Design **37**
3.1 Design Objectives . . . 37
3.1.1 Primary Design Objectives . . . 37
3.1.2 Secondary Design Objectives . . . 38
3.2 Design Overview . . . 38
3.3 Examples with TCP . . . 39
3.3.1 Example with Caching . . . 45
3.4 Deploying the Stateful Anycast Service . . . 48
3.5 Deploying an Anycast Group . . . 48
3.6 Packet Processing Algorithms . . . 49
3.6.1 Delivering Packets to the Anycast Proxies . . . 49
3.6.2 Processing Packets at the Anycast Proxies . . . 49
3.6.3 Receiving Packets at the Group Member . . . 50
3.6.4 Sending Reply Packets . . . 50
3.6.5 Reacting to Attacks . . . 51
3.7 Determining Whether a Packet is Part of a Stateful Session . . . 52
3.8 Choosing a Recipient Group Member for Non-Stateful Packets . . . 53
3.9 Session Identifiers . . . 53
3.9.1 Motivation for using session IDs . . . 54
3.9.2 Extracting the session ID . . . 54
3.10 Session Identifier Directory Service . . . 55
3.10.1 Directory Service Functions . . . 55
3.10.2 Directory Service Mapping Data Structure . . . 62
3.10.3 Caching Directory Service Query Results . . . 63
3.10.4 Managing Directory Service Entries . . . 64
3.10.5 Directory Service Implementation . . . 64
3.11 Encapsulating Packets . . . 65

4 Design Evaluation **67**
4.1 Support for stateful sessions . . . 67
4.2 DDoS Mitigation . . . 68
4.2.1 Attacks on group members . . . 69
4.2.2 Attacks on the directory service . . . 70
4.2.3 Attacks on the proxies . . . 71
4.2.4 Other attacks . . . 72
4.3 Scalability . . . 72
4.3.1 Scalability in the number of groups . . . 72
4.3.2 Scalability in the number of group members per group . . . 74
4.3.3 Scalability in group membership dynamics . . . 74
4.3.4 Scalability in the number of anycast proxies . . . 75
4.3.5 Scalability in the rate of change in anycast proxy availability . . . 75
4.3.6 Scalability in the number of stateful sessions . . . 75
4.3.7 Scalability in session dynamics . . . 76

4.4 Deployability 77
4.5 Protocol Generality 78
4.6 Performance and Efficiency 79

5 Stateful Anycast Software Design **83**
5.1 Background: Virtual Network Interfaces 83
5.2 Group Member Software 84
5.3 Anycast Proxy Software 85
5.3.1 Remote Client to Anycast Proxy: Virtual Interface Approach . . 85
5.3.2 Remote Client to Anycast Proxy: Packet Capture Library Approach 88
5.4 Testing Stateful Anycast 90

6 Conclusion **93**
6.1 Future Work 95

List of Figures

2.1 Example AS topology illustrating the various AS types 22
2.2 Example AS topology with IP anycast group members 24
2.3 Illustration of anycast catchments 26
2.4 Packet delivery in PIAS 29

3.1 TCP three-way handshake and TCP states 40
3.2 Example TCP session with Stateful Anycast (non-caching) 41
3.3 Packet contents for TCP example session (without caching), step #1 . 41
3.4 Packet contents for TCP example session (without caching), step #2 . 42
3.5 Packet contents for TCP example session (without caching), step #3 . 43
3.6 Packet contents for TCP example session (without caching), step #4 . 43
3.7 Packet contents for TCP example session (without caching), step #5 . 44
3.8 Packet contents for TCP example session (without caching), step #6 . 44
3.9 Packet contents for TCP example session (without caching), step #7 . 45
3.10 Example TCP session with Stateful Anycast (with caching) 46
3.11 Packet contents for TCP example session (with caching), step #1 . . . 47
3.12 Packet contents for TCP example session (with caching), step #2 . . . 47
3.13 Directory service query using **query**() 56
3.14 Directory service query using **queryAndForward**() 57
3.15 Out-of-order packet delivery with **queryAndForward**() 58
3.16 Directory service query with **queryAndReturn**() 59
3.17 Out-of-order packet delivery with **queryAndReturn**() 60
3.18 Sample directory service mapping 62
3.19 Illustration of an encapsulated incoming TCP SYN packet 66

4.1 Scaling Stateful Anycast to support a large number of groups 73

5.1 Virtual network interface packet paths 84
5.2 Group member software architecture 86
5.3 Anycast proxy incoming packet path when using the virtual interface approach 87
5.4 Anycast proxy incoming packet path when using the packet capture library approach 88

5.5 Testing Stateful Anycast using a virtualized anycast implemented with tunneling 91

Chapter 1

Introduction

Internet distributed denial of service (DDoS) attacks have been receiving much attention due to the disruption they cause and the lack of an easy defense. DDoS attacks overwhelm their targets by saturating the data links leading to the target or by completely consuming some other computational resource. With the target overwhelmed, legitimate users can no longer access any of the target's services. The attacks are launched from a *botnet*, a collection of remotely-controlled computers (referred to as *zombies*) distributed throughout the Internet. These zombies have typically been compromised with automated scripts designed to quickly grow the size of the attacker's botnet to thousands or possibly millions of machines. Each machine may not be capable of sourcing a significant amount of traffic, but when the capabilities of many zombie machines are combined, a formidable weapon is created.

A DDoS attack can take down a host or network for weeks, wreaking serious economic damage to an on-line business. One DDoS extortion case in 2003 was documented in a May 2005 article in CSO Magazine [17]. DDoS attacks also have the potential to take out key infrastructure, as witnessed by a couple of major attemps to take down the Internet's root DNS servers [24, 32, 42, 73]. DDoS attacks even disrupted Akamai—a content distribution network often used by companies to protect themselves against DDoS attacks—in 2004 [72].

Despite years of research, an effective, practical, and general solution to the DDoS problem has remained elusive. Many of the proposed solutions are specific to a certain type of attack or protocol, or they require extensive modifications to the network architecture or infrastructure. Stateful Anycast is an attempt at providing a partial solution to the DDoS problem that is effective, practical, and general. Stateful Anycast is not a complete solution to the DDoS threat, as it does not handle the problem of detecting attacks; its primary purpose is to squelch an attack once it has been discovered.

1.1 Two Types of DDoS Attacks

Before diving into how and why DDoS attacks happen, it is important to recognize that DDoS attacks come in many flavors. Mirkovic and Reiher describe a comprehensive taxonomy of DDoS attacks in [45], but this document will use a simpler classification and divide DDoS attacks into two simple types:

1. *End-node attacks*: These attacks consume some resource in the targeted end node. When a resource is consumed, the end node is unable to service the legitimate traffic it receives. These attacks can consume storage used to hold state (e.g., TCP SYN floods [21]), processing cycles (e.g., repeated cryptographic key exchange), battery power, and so on.

2. *Network resource attacks*: These attacks consume some resource in the network leading to the end-node. When a network resource is consumed, the network is unable to deliver some fraction of the legitimate traffic to the end node. The canonical example is flooding the link that leads to a strategic end node with random[1] traffic. The end node may be perfectly capable of processing all of the offered load, but, because the network is the bottleneck, the end node never receives the good traffic to process.

Categorizing DDoS attacks in this way is useful because of how each type of attack is mitigated. Attacks on the end node involve local defenses, such as an increase in processing power or storage. While implementing local defenses is not always easy, it is generally possible and—more importantly—in the victim's control.

Attacks on the network, on the other hand, pose a greater challenge. A DDoS attack on the network can only be mitigated upstream of the consumed resource, often moving the only feasible points of defense out of the direct control of the victim. Even when upstream network operators are willing to implement defenses on behalf of the victim, technical challenges remain. The end node has all the insight into whether a packet might be legitimate or malicious, yet it is the network that must decide whether to drop or forward the packet. Imbuing the network with the ability to identify attack traffic without any of the contextual knowledge held by the end node remains a challenging problem, although there has been some work in this area [30, 63].

Stateful Anycast is designed to address the greater challenge of network resource attacks. Because of this focus, and to simplify discourse, the phrase "DDoS attacks" will be used throughout the remainder of this document to refer to DDoS attacks on the network rather than its more general meaning.

[1] An attacker could add more teeth to the attack by choosing to flood with traffic that also consumes an end-node resource, but this is not necessary if the attacker is successful in saturating links and preventing legitimate traffic from reaching the end node in the first place.

1.2 Assumed Attacker Abilities

Throughout this document, attackers are assumed to be able to amass a well-distributed botnet capable of disabling a limited number of selected targets by saturating the network links leading to each target. Each zombie can source arbitrary packets, including packets with spoofed source addresses. The attacker's botnet is not large enough to disable all members of an anycast group.

1.3 How DDoS Attacks are Possible

Currently, there is no easy way to stop DDoS attacks from happening—the design of the Internet makes it so. There is no general way for a receiver to tell the network not to deliver a particular packet to it. The Internet was designed to be transparent to the end hosts: in theory, packets going in come back out at their destination, unchecked and unmodified. This end-to-end architecture, defended by Saltzer, Reed, and Clark [67], has some important benefits. First, the network can be constructed using simple, inexpensive devices. Second, the network's obliviousness to the meaning or intent of a packet allows it to support novel, unforeseen applications. Unfortunately, obliviousness also means that the network cannot be asked to decide which packets should and should not be delivered, so everything arrives at the recipient.

The fundamental challenge in mitigating DDoS attacks stems from the need of every participant in a stateful session to direct packets toward the session's stateholder (or holders). Participants must be given a *handle* on the stateholder—an IP address of a web server, for example—that is then given to the network. The network uses these handles to locate and route packets to the stateholder.

The ability to send to a specific stateholder can be abused if the network is not designed appropriately. Specifically, DDoS attacks are possible when stateholder handles are both portable and permanent. Unfortunately, both of these conditions are true in the current design of the Internet's basic protocols. Elimination of either the portability or the permanence of handles will limit the success of DDoS attacks.

1.3.1 Portable Stateholder Handles

Portable stateholder handles are handles that can be used from anywhere in the network to reach a specific stateholder. Non-portable handles are valid only at certain locations or regions in the network, limiting their usability to one or a small fraction of the hosts. In the Internet, handle portability can be safely assumed: IP addresses globally identify a single host, except for the anycast, multicast, and broadcast addresses that are rarely—if ever—used in a stateful conversation.

With portable stateholder handles, an attacker can amplify an attack by instructing an arbitrary number of distributed, colluding hosts to send packets to the victim. By limiting the usability of a handle to one or a few hosts, a malicious, distributed load can be diffused among a group of identical stateholders in a manner similar to anycast.

Each sender would be given a handle to a different stateholder upon session initiation, and because each handle is unusable by others, the malicious senders would be unable to focus their resources on disrupting a specific stateholder.

Traffic Validation Architecture, the capabilities-based DDoS mitigation approach proposed by Yang, et al. [75], is one proposed mechanism for making stateholder handles non-portable. IP anycast, discussed in section 2.2, also makes handles non-portable by restricting communication with a particular group member to the region around the group member.

There are a couple of negative side-effects to making stateholder handles non-portable, however. First, support for sender mobility is lost. If a legitimate host moves to a different region of the network, the handle may no longer be usable and communication can be disrupted. Second, if handle usability is tied to topology, changes in topology due to events such as failed links can invalidate handles. To continue a session disrupted by an invalidated handle, the sender must be able to detect that the handle is unusable and acquire a new handle. This may be a challenging task depending on the design of the mechanisms that cause the non-portability. To avoid this issue, Stateful Anycast chooses to mitigate DDoS attacks by making handles non-permanent.

1.3.2 Permanent Stateholder Handles

Permanent stateholder handles are handles that, once acquired, can be used for all time to reach a specific stateholder. The unicast IP address is the most immediate example of a permanent handle. Once an attacker knows a stateholder's IP address, the only way to remove the attacker's ability to send packets to the stateholder is to relocate to a different part of the network and get a different IP address.

If handles are made temporary or cancelable, stateholders can get reprieve from sessions that have turned malicious. Any attempts at obtaining new handles to replace revoked ones can be diffused over a group of identical stateholders in order to prevent the load from overcoming any one stateholder. This is the approach Stateful Anycast takes.

Session invalidation must be enforced by the network far away from the stateholder. If enforcement happens near the stateholder, an attacker may still be able to inflict significant damage. The closer the enforcement is to the stateholder, the more malicious traffic aggregates. The more traffic aggregates, the more likely the attacker will saturate the network resources leading to the stateholder and disrupt legitimate users.

1.4 Why Attackers Launch DDoS Attacks

The motives behind DDoS attacks vary widely. The Internet is a public network and is popular among people of various cultures, beliefs, ages, incomes, social classes, citizenships, levels of education, and so on. Of course, this diversity brings conflicts of interest and motives for attack.

Compounding this, targets are becoming increasingly lucrative due to the ever-tightening integration of the Internet with society. As testimony to this, early DDoS attacks were largely motivated by adolescent machismo: attackers wanted to earn recognition, assert superiority, and exact revenge; other early attacks were proof-of-concept [44]. More recently, the motives for DDoS have expanded and now include extortion [52] and competitive advantage [51].

DDoS attacks are common because they are both easy to launch and effective. Launching a DDoS attack is a straightforward task, and is well-documented in [13]. To summarize, an attacker creates "zombies" by compromising a large collection of hosts, instructs those zombies to join the botnet by contacting the command-and-control node, then issues the order to attack. Sophisticated software to automate the whole process—including source code released under the GPL open-source license—is readily available. Compromising hosts is straightforward due to the transparency of the Internet combined with shared vulnerabilities among many hosts. Controlling zombies is straightforward because hosts on the Internet are typically high-powered general-purpose computing platforms (due to the end-to-end nature of the Internet). Compromised hosts are not quickly cleaned up because each machine is usually owned or operated by a different individual, making it difficult to contact those who would be responsible for the clean-up. Thus, zombies are easy to recruit, easy to control, and can remain in the botnet ranks for a long time.

In addition to lack of direct control over defenses mentioned in section 1.1, DDoS attacks are effective because the task of filling up a link with garbage traffic is easily parallelizable among many machines. It does not matter how little traffic each node can source as long as the total amount of traffic is enough to saturate the target link. In fact, many nodes (each sourcing little traffic) can be more effective than a few nodes (each sourcing a lot of traffic) due to the increased challenge in the typically manual process of identifying and implementing filters for each attacking node.

The one mitigating factor is the visibility of the attacks. Victims know when they are under attack, giving them opportunity to respond by implementing technological countermeasures and contacting law enforcement. In addition, launching an attack can tip off the owners of zombie machines and encourage them to fix any security vulnerabilities.[2] Finally, visibility encourages security researchers from academia, government, and industry to develop defensive mechanisms.

1.5 Defending against DDoS Attacks

A good DDoS defense has the following properties:

1. *Minimal impact on legitimate users.* The end-user performance obtained when

[2]However, residential users—the stereotypical zombie owners due to a lack of trained network security personnel in the home—rarely have monetary incentives to repair a compromised machine. Few residential Internet users are charged bandwidth usage fees by their ISPs—service fees are by-and-large flat-rate.

the DDoS defense mechanism is deployed should not be significantly degraded from the performance without the DDoS defense. DDoS attacks are much more rare than legitimate sessions, and will be even more rare with an effective defense. Even a small performance degradation can result in the cure being worse than the disease.

In addition, the defensive mechanism should have very few false positives. Filtering by IP address may not be good enough—a legitimate user and a zombie might use the same IP address because they are both on the same machine, behind the same NAT device, using the same proxy, or they were assigned the same address at different times.

2. *The defensive mechanisms are not themselves subject to attack.* For example, if a defensive mechanism requires manual intervention when under attack, attackers can still inflict damage in the form of increased labor costs.

3. *Support for arbitrary applications.* Defenses that only interoperate with specific applications increase the cost of switching to newer, better applications. Generality is the key to preventing the ossification of protocols and the stunting of innovation.

4. *Malicious traffic is dropped early.* Bystanders are frequently caught in DDoS attacks targeted at someone else because they are topologically close to the intended victim. Dropping malicious traffic near the source minimizes collateral damage and reduces the network management costs imposed by malicious traffic.

5. *Inexpensive to deploy, maintain, and scale.* Forcing a victim to spend large amounts of money on defenses is a form of denial-of-service attack.

Upon the launch of a large DDoS attack, a common defensive reaction often comes from the victim's ISP: "Lacking more advanced infrastructure/tools, most ISPs primarily mitigate attacks by filtering all traffic to the victim," said Arbor Networks in a September 2006 news release [12]. This defense obviously prevents legitimate users from reaching the service, but it also prevents the DDoS attack from disabling the ISP's other customers. From the perspective of the ISP, dropping malicious traffic early is the most important property when there is a lot of attack traffic because it minimizes collateral damage. The victim is sacrificed early to save the ISP's other customers simply because the ISP neither has the ability to distinguish between good and bad traffic nor the capacity to let the traffic flow without impacting other customers.

One of the victim's possible responses is to scale the network surrounding the unreachable node to handle the load. This can be done by simply purchasing more capacity from the ISP, by moving the node to a larger ISP, or by obtaining service from multiple ISPs. This defensive response minimally impacts users, is not subject to attack, and supports arbitrary applications, but fails to drop attack traffic early and can be expensive to deploy. The cost of upgrading the network depends on the amount

of traffic that must be supported. According to Arbor Networks, "ISPs now regularly report attacks beyond than [*sic*] capacity of core backbone circuits in the 10-20Gbps range" [12]. It is difficult to purchase network service capable of sinking this amount of traffic in a single location, especially at prices within reach of small or medium businesses.

Instead, many at-risk organizations elect distributed approaches that both simplify the problem of scaling up connectivity as well as enable cost savings from statistical aggregation with others. Companies such as Akamai and Prolexic Technologies sell services that enable their customers to sink significant amounts of traffic on demand. These companies have partnerships with numerous ISPs to increase the total amount of traffic they can sink. They distribute devices—caching web proxies in the case of Akamai and proprietary packet filters in the case of Prolexic—to process the load close to the source. The cost of deploying and maintaining the extensive infrastructure is amortized over several customers, making their services inexpensive enough to be a worthwhile investment for at-risk organizations. However, these services are not general: they either support a limited set of applications (such as static web content) or do sophisticated higher-layer processing to identify which packets are good and which are bad without input from the ultimate destination.

Anycast is another distributed approach to scale connectivity. It is discussed in detail in the following section and in section 2.2. Stateful Anycast, summarized in section 1.7, is an attempt to improve upon the limitations of anycast, and is designed to have all of the properties of a good DDoS defense listed above.

In the long run, the best way to defend against DDoS attacks is to either reduce attack effectiveness through architectural changes in the Internet or increase the difficulty of launching attacks through better end-node security.

1.6 Anycast as a DDoS Defense

Anycast is a packet delivery service that can be utilized to distribute load among a group of identically-configured machines. Anycast is unique in that the network itself chooses which group member will be the recipient of a sender's packet, outside the control of the sender or receiver. This gives anycast an interesting security advantage: distributed attacking nodes are unable to target a specific group member.

1.6.1 Anycast Defined

Anycast is perhaps easiest to understand when defined by contrasting it with other routing services: unicast is one to one, broadcast is one to all, multicast is one to all of a defined subset, and anycast is one to *any* of a defined subset. This definition is incomplete, however, as it allows some subtle but significant variations in what can be considered an anycast routing service. For the purposes of this document, the general notion of anycast is defined as a routing service with the following two properties:

1. packets sent to a group of hosts are delivered to only one member of the group (with best-effort reliability), and

2. the routing infrastructure—not the sender—selects which group member will receive a packet sent to the group.

Note that this definition of anycast makes no guarantees about which group member will receive a packet. For instance, a packet could be delivered to the current closest member (for some definition of "closest"), or it could be delivered to a pseudorandomly selected member. A sequence of packets sent by a single host could all be delivered to the same member, or they could be delivered to several different members. The actual behavior of an implemented anycast service depends on the topology of the network and the implementation details of the underlying routing protocols. For example, IP anycast generally selects a packet's recipient based on topological closeness to the sender. For a full discussion on how IP anycast works on the Internet, see section 2.2.

This definition is also silent on the configuration of each host in the group. While it is typical for each host to be configured as identically as possible, this is not a requirement. Configuring each group member identically can provide an illusion of a single, well-connected host—a desirable property when the goal is to handle large traffic loads. Because handling large traffic loads is an important goal for any DDoS defense, the rest of this document assumes that all group members are configured to behave identically when interacting with the outside world.

These two properties have their advantages and disadvantages; the more significant of each are discussed below.

1.6.2 How Anycast Helps with DDoS

Anycast's most significant benefit (from a DDoS mitigation perspective) is its ability to distribute packets to group members in a way that is outside the control of the sender. Should a malicious collection of nodes decide to send a flood of packets to an anycast address, the packets can be diffused among the group members to balance the load and prevent any one group member from being taken down. The number of malicious packets received per group member depends only on the topology of the network and the routing protocol employed by the network—not on any information supplied by the sender.

With careful analysis of the network's topology and route selection algorithms, and with reasonable estimates for the quantity, location, and capabilities of malicious sources, it is possible to engineer the distribution of attack traffic among the group members. As long as it is easy to place group members at strategic locations in the network, anycast can be an effective way to absorb the heavy load of a DDoS attack.

By distributing load to multiple group members, anycast provides a way to provide a service that appears to be running on a single, powerful end node with a high-bandwidth connection to the network. All that is needed is to configure the group

members to provide the same behavior to the rest of the network. With all group members offering the same service, members become indiscernible and an illusion of a single node is formed. With this configuration, upgrading the network is no longer the only way to defend against network resource DDoS attacks: a similar effect can be achieved by adding additional nodes to the anycast group.

1.6.3 Anycast's Big Limitation

The most significant disadvantage of anycast is the lack of support for stateful conversations. In a multiple-packet conversation, the two conversing nodes typically must hold some private information about the state of the conversation for the duration of the conversation. If the packet recipient changes mid-conversation, the new recipient will not have the same state as the original recipient and will therefore be unable to continue the conversation.

When a stateful session is initiated with an anycast group, the state resides on a single group member. If future packets are not delivered to that particular group member, the session is unable to continue.

1.7 Improving Upon Anycast: *Stateful Anycast*

Stateful Anycast adds support for stateful sessions to regular anycast. All incoming packets belonging to the same stateful session are delivered to the same group member. This is done with a two-stage approach: First, packets are delivered to an anycasted set of middleboxes called *anycast proxies*. Next, the proxies forward the packets to the appropriate group members. For each incoming packet, the receiving anycast proxy determines which session the packet belongs to, determines which group member is the stateholder for that session, and forwards the packet.

A packet is matched to a session by examining an embedded *session identifier*. Session identifiers are required to be present in all packets produced by a stateful protocol. Most existing stateful protocols already have these identifiers in every packet—TCP's session identifier is the quadruple composed of the source and destination address and port.

A session is matched to a stateholder by looking up the session identifier in a distributed database of identifier-to-stateholder mappings. If the mapping does not exist, or if the identifier is mapped to a reject instruction, the packet is dropped. Group members are responsible for maintaining their own mapping entries.

The use of a mapping database allows group members to defend against DDoS attacks by revoking malicious sessions. Identifying malicious senders is easiest at the group members because they have the greatest ability to understand a sender's intent. In addition, putting the responsibility of revoking malicious sessions on the group members agrees with the end-to-end arguments and allows Stateful Anycast to remain relatively simple and general.

While the decision to revoke is made by the group members, revocation is enforced at the anycast proxies. Since the anycast proxies form an anycast group and IP anycast generally selects the closest member to receive a packet, enforcement at the proxies means that malicious traffic is dropped close to the sender.

The two-stage indirection approach allows Stateful Anycast to scale to support a large number of groups, group members, and proxies. Stateful Anycast can benefit from statistical aggregation, allowing many groups to share the cost of deploying and maintaining the anycast proxies. In addition, Stateful Anycast can provide Akamai-like improved performance during times of peace thanks to the ability to connect remote users with the closest non-busy group member.

Altogether, Stateful Anycast is an effective, practical, and general DDoS mitigation mechanism.

1.8 Organizational Overview

The rest of this document is organized as follows: Chapter 2 provides background information and descriptions of related work. Chapter 3 details the design of Stateful Anycast. Chapter 4 provides an evaluation of the design of Stateful Anycast and comments on its ability to mitigate DDoS attacks. Chapter 5 discusses the design of the currently in-development software to use on anycast proxies and group members. Chapter 6 concludes with some final remarks and suggestions for future work.

Chapter 2

Background and Related Work

This chapter provides background related to Stateful Anycast. Sections 2.1 through 2.5 discuss technologies employed by Stateful Ancyast. Section 2.6 lists some previous work in anycast and DDoS mitigation.

2.1 BGP: Global Unicast Routing on the Internet

IP anycast works by taking advantage of existing unicast routing protocols. To understand how IP anycast packets are delivered to a particular group member, it is helpful to understand how unicast packets are routed through the Internet.

On a global scale, routes are selected using the Border Gateway Protocol (BGP) [65]. BGP divides the Internet into independently operated networks called Autonomous Systems (ASes). BGP controls the routing of packets among these ASes, but leaves the task of routing packets within an AS to the operator of the AS. The separation between inter- and intra-AS routing protocols aids scalability and allows each AS operator to design and operate the local network according to local needs. The topology connecting ASes can be arbitrary. AS operators agree to connect with each other according to their needs and economic interests.

ASes can be classified into different types. Peterson and Davie [57] classify ASes into three broad categories based on their connectivity and their willingness to transport traffic from one neighboring AS to another:

- *Stub ASes* are only connected to one other AS. The neighboring AS is typically an Internet service provider (ISP) selling connectivity to the stub AS operator. Any packets in the stub AS that are not destined to a host in the stub AS are passed to its single neighbor. These ASes are uncommon because the hosts in a stub AS can simply be a part of the neighbor's AS and use a subset of its advertised address space instead. Becoming a part of an existing AS is easier to initially set up, plus it does not consume extra resources in routers around the Internet (a significant benefit given BGP's difficulty in supporting many ASes).

- *Multihomed ASes* have multiple neighboring ASes but do not allow packets to pass through from one neighbor to another. Multihoming provides multiple BGP-level paths between the hosts inside the AS and hosts outside the AS. Multihomed ASes are often operated by companies that desire redundant connectivity in order to improve the reliability and performance of the services they provide [11].

  Creating a multihomed AS to achieve rich connectivity is disfavored because the same effect can usually be achieved without creating another AS: The multihomed network operator can use a block of IP addresses assigned by the network's primary ISP but ask its other ISP(s) to also advertise that block.

- *Transit ASes* are connected to multiple ASes and provide packet transit from one neighboring AS to another. ASes of this type are usually operated by larger ISPs.

For an illustration of these three, see figure 2.1.

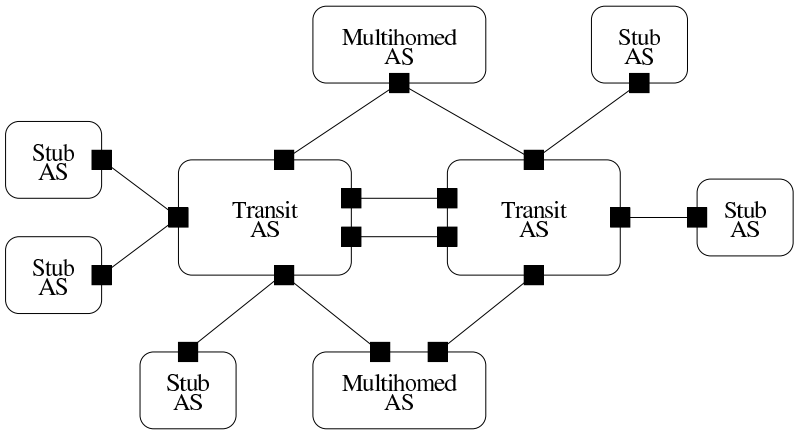

Figure 2.1: Example Autonomous System topology illustrating the various AS types. The black squares represent BGP border routers. The lines between the border routers represent inter-AS links.

A router that connects an AS with a neighbor AS is called a *border router*. The black squares in figure 2.1 represent border routers in the example AS topology. Border routers can be thought of as having two sides: an outward side that leads to the neighboring AS's border router and an inward side that leads to the hosts and routers in the AS.

In addition to its packet forwarding duties, a border router sends messages to its neighboring AS to advertise the block(s) of IP addresses it is willing to accept. These advertisements are repeated from AS to AS, allowing each AS throughout the Internet to discover which neighbors can be used to deliver a packet to an advertised destination.

Accompanying a BGP advertisement is a list of ASes traversed by the advertisement. Each AS adds itself to this list when it relays the advertisement to a neighbor.

When an AS receives advertisements for the same IP address block from multiple neighbors, it can refer to this list to help it make an informed decision about which route is "best". "Best" is not determined entirely by topology or other technological matters; instead, the economics of network interconnection play a dominant role in choosing which neighbor an AS will use for forwarding packets it cannot deliver itself. However, because the cost of delivering a packet roughly correlates with the number of hops, it is reasonable to assume that the path selected by BGP is approximately the shortest in the topological sense (i.e., the fewest number of hops).

BGP is designed to be resilient to topology changes. Whenever the topology changes (due to, e.g., a link failure), update messages are propagated to the ASes affected by the change. Routers receive these updates and recalculate the paths used to deliver packets. Thus, two packets sent from a given host A to a different host B could traverse different paths depending on the dynamics inside the network. The topology of the Internet today is generally stable due to the prevalence of fixed, non-mobile infrastructure. Thus, over a window of a few seconds, it is improbable that packets between two given hosts will traverse significantly different paths.

As links become more ephemeral due to improvements in wireless and mobile technologies, and as routing decisions incorporate rapidly-changing dynamics such as congestion [15], route stability will likely decrease. For unicast, lower route stability can mean increased jitter and out-of-order packet delivery, which can negatively impact real-time Internet applications. As explained in the next section, lower route stability also harms IP anycast's ability to support stateful sessions.

2.2 IP Anycast

IP anycast is the Internet's anycast service. Proposed in 1993 by Partridge, Mendez, and Milliken [53], IP anycast makes clever use of BGP to provide an anycast service without modifying existing protocols.

BGP allows an IP anycast group to appear to be a single multicast host within a multihomed AS. Indeed, there are many similarities between an anycast group and a host in a multihomed AS: both use a single IP address, and the rest of the Internet sees multiple possible routes to the IP address. The only difference is that with the host in a multihomed AS, all of the routes to an advertised address lead to the same node; with IP anycast, each route leads to a different member of the IP anycast group.

2.2.1 Deploying an IP Anycast Group

Because IP anycast groups are similar to multihomed ASes, IP anycast groups are formed in a similar manner to multihomed ASes. The group operator first identifies the desired points of attachment (one for each group member), enters agreements with the AS operators at those points of attachment, and connects border routers to these points. Each of these border routers is configured to advertise the same block of IP addresses. However, unlike the border routers in a multihomed AS, the inward sides of the anycast

group's border routers are not attached together via an intra-AS network. Instead, the inward side of each router is attached to one or more anycast group members. Each group member is typically configured identically to provide the illusion of a single node inside a multihomed AS. The border router–group member pairs are like a simple stub AS that has been cloned and distributed. See figure 2.2 for an illustration.

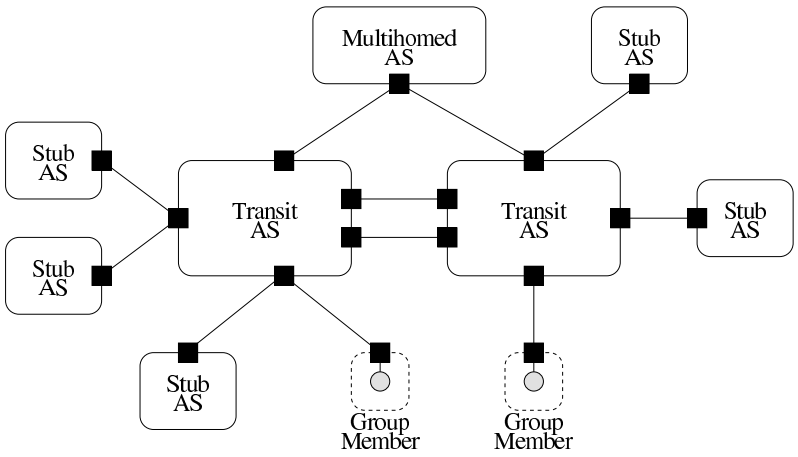

Figure 2.2: Example Autonomous System topology with IP anycast group members. Both anycast group members use the same AS identifier and advertise reachability to the same block of IP addresses, preventing the other ASes in the topology from distinguishing the anycast group from a single unicast host in a multihomed AS. Compare to figure 2.1.

Deploying a new IP anycast group member can be nontrivial. There are two challenges that must be overcome by the anycast group operator. First, a good attachment point must be located. Choosing the right adjacent network can make a significant performance difference [11]. Poor placement of the group member could prevent it from capturing a good fraction of the overall traffic load, while good placement can result in a relatively even traffic distribution among group members. Second, an agreement has to be reached with the operator of the adjacent AS. Typically, the agreement is based on a customer–provider relationship, but ISP customers do not typically have their own AS and IP address block. The customers that do are either large or have uncommon requirements, prompting specially negotiated pricing [19]. Accommodating a customer that has its own AS number and runs its own BGP is a straightforward but manual process involving some router configuration changes. Thus, getting Internet service for a new group member is not as easy as signing up for generic residential- or commercial-class Internet service.

While the discussion so far has focused on IP anycast groups deployed on a topologically-broad scale using BGP, it is also possible to run an anycast group entirely within an AS. This eliminates the need to enter into agreements with multiple

network operators while still providing many of the same benefits, especially if traffic is predominantly local. However, flexibility is reduced, so it may not be suitable for evenly distributing load if there is a large number of widely distributed senders. For an in-depth treatment of the issues involved in deploying an anycast group on the Internet, see [5].

2.2.2 IP Anycast Usage Today

While a full survey of the usage of IP anycast is difficult due to the challenge of distinguishing between an anycast group and a host in a multihomed network, there are a few known notable uses of IP anycast today. These uses include: replicating root DNS servers to improve reliability and performance [7, 8, 27], sinking meaningless DNS queries before they reach the public DNS infrastructure [6], and simplifying the process of locating special hosts called *6to4 gateways* that are used to aid in the transition from IPv4 to IPv6 [20, 29]. Note that all of these are stateless services.

Replicating the root DNS servers is partially done in response to the DDoS threat [26]. In October 2002, a DDoS attack was launched that disabled several of the root DNS servers [24, 73]. A much smaller attack took place in February of 2007 that disrupted two of the root servers [32, 42]. While neither attack caused a noticeable degradation in overall DNS performance, DDoS attacks have prompted accelerated infrastructure upgrades [73]. The primary mode of upgrade is to deploy additional root DNS servers via anycast, largely due to a technical issue that limits the number of root DNS IP addresses to thirteen.

2.2.3 Characteristics of IP Anycast

IP anycast's characteristics are strongly influenced by its use of BGP. One effect of using BGP is that IP anycast shares the same address space as IP unicast. The routing infrastructure is unable to differentiate between an IP unicast host and an IP anycast group, so there is no requirement to set aside a part of the IP address space for use by IP anycast groups. An anycast group can use whatever unicast IP address it has available. It is therefore possible to convert a single-node service to an anycasted multi-node service without changing the IP address used to reach the service. While there is no longer a strong need to operate a service on a particular IP address due to the prevalence of the domain name system (DNS), this may be useful for when DNS is unavailable or for legacy applications that use hard-coded or manually-configured IP addresses.

More significantly, the use of BGP to implement anycast results in packets being delivered to the topologically closest group member. As with multihomed networks, routers on the Internet hear the advertisements for the block of IP addresses and locally decide which route is best. The route to an anycast group member is decided under the belief that the advertisements indicate alternate paths to the same network, when in reality the different advertisements correspond to physically distinct devices.

The particular anycast member selected by a router is typically the topologically closest member, subject to the economically-motivated constraints that network operators impose on routing decisions.

Because routes are generally determined by topological closeness, an anycast group member will sink the traffic sourced in the topological region surrounding it. Using the terminology of Abley and Lindqvist [5], the Internet is divided into *catchments*, each of which "drains" to a particular anycast host. See figure 2.3 for an illustration.

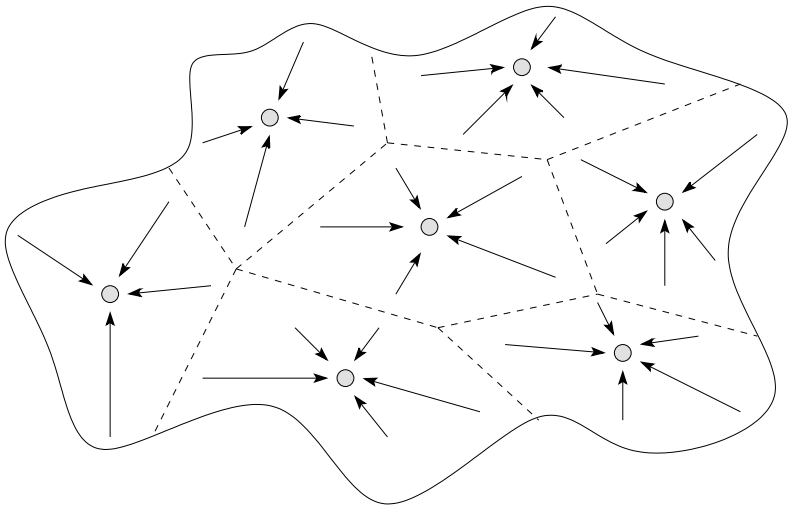

Figure 2.3: Illustration of anycast catchments. The blob represents the entire Internet. Grey circles represent anycast group members. Catchment boundaries are illustrated with dotted lines. Traffic sourced by a host inside a catchment will be delivered to the anycast group member in the catchment. The location of catchment boundaries is determined by routers deciding which group member is "best".

One consequence of routing based on topological closeness is short-term consistency in the recipient of packets sent to the group. The only way the recipient group member will change is if the topology changes in a way that causes a catchment boundary to move underneath the sender. Because the topology of the Internet changes slowly, catchment boundaries change slowly. Thus, in the short term, a sender's sequence of packets will likely be delivered to the same anycast group member. Possible topology-changing events include the addition or removal of network links, the addition or removal of anycast group members, or sender mobility. The amount of time a sender can expect to reach the same group member with high probability depends on how rapidly the network changes and where the changes are relative to the sender.

There is considerable debate over whether routing is stable enough to deploy stateful services over IP anycast [38]. Route selection is generally stable thanks to a largely fixed, wired infrastructure. However, there are at least three trends that will

likely reduce route consistency: the widespread adoption of routing algorithms that base decisions on rapidly-changing factors such as congestion or instantaneous load rather than slowly-changing factors such as topology [15], an increase in the use of dynamically established links made possible by wireless networking technologies, and the ubiquitization of mobile devices and protocols. As these changes occur, IP anycast's ability to support stateful sessions will diminish due to a lower probability of delivering all of a session's packets to the same group member.

Another consequence of routing based on topological closeness is providers have a coarse, non-dynamic means of adjusting the distribution of traffic per anycast group member. Through the choice of attachment point, the group operator can influence the size and location of the catchments and therefore the approximate distribution of traffic per anycast host. For example, an overloaded group member can have its catchment reduced by placing other group members nearby. However, operators are unable to fine-tune the load distribution—especially in response to dynamic conditions—so they should carefully identify the most strategic attachment points based on current and expected loads.

Yet another consequence of routing based on topological closeness is that the network does the minimal amount of work necessary to deliver a packet to a group member. This is important for suppressing DDoS attacks because traffic is terminated near the source (assuming a good number of well-distributed group members). This is also beneficial for normal operation, as it can result in improved latency, jitter, bandwidth, and reliability between the sender and the service provided by the anycast group.

2.2.4 Limitations of IP Anycast

In addition to the lack of support for stateful sessions discussed in section 1.6.3, IP anycast has limitations stemming from the details of its implementation:

- *Security:* Because IP anycast is implemented using BGP, IP anycast inherits the same well-known vulnerabilities present in BGP [50]. Fortunately, it can benefit from the same defensive mechanisms [28, 36, 68]. One possible attack involves an adversary advertising the block of IP addresses used by the anycast service. This would cause some of the traffic to be diverted to a destination of the adversary's choosing. Authenticating BGP advertisements from neighboring routers can reduce this particular risk.

- *Availability:* To ensure the continued availability of an anycasted service, care must be taken to ensure that a group member's route is withdrawn when that group member is incapable of providing the service. If this does not happen, hosts in the catchment served by the group member will be unable to access the service.

  If the route is successfully withdrawn, hosts previously served by the disabled group member will be automatically directed toward an alternate group member once the BGP route selection process converges. However, BGP can take a long time to converge on routes [37], so anycast packets may be dropped for

several minutes after the withdrawal. To minimize the likelihood of needing to issue a BGP route withdrawal, redundant group members should be deployed at each network attachment point. Under normal operation, packets arriving at the attachment point would be routed to any one of the redundant group members. If a group member fails, incoming anycast packets would be routed to an alternate group member at the attachment point, preventing the need to withdraw the BGP route. Abley [8] describes one particular redundant topology designed to maximize uptime.

- *Primitive group member selection:* Group member selection is based on topological closeness, not on metrics such as workload, latency, available bandwidth, packet loss, etc. Thus, the network cannot automatically balance the traffic load among the group members for optimal performance.

- *Scalability:* A major issue hindering the adoption of IP anycast is difficulty scaling in the number of anycast groups. An anycast group looks just like a multihomed network to the routing infrastructure, and the Internet's difficulty in supporting numerous multihomed networks is well known [4, 16, 31, 76]. Routes to the anycast group members cannot be aggregated with routes to other networks, so each anycast group requires its own unique routing table entry on every router in the Internet. Thus, the size of a router's forwarding table scales linearly with the number of IP anycast groups [43].

  To minimize the amount of very expensive, very fast memory used to store forwarding tables in routers, network operators typically ignore advertisements for small blocks of IP addresses. By ignoring small blocks of IP addresses, network operators exclude all but the largest players. Thus, in order to obtain reachability throughout the Internet, an IP anycast group operator must obtain a large block of IP addresses even if only one IP address is required for the service. These large blocks consume a significant portion of the scarce IPv4 address space remaining.

Three proposals described later in this chapter—PIAS (section 2.3), *i*3 (section 2.4), and GIA (section 2.2)—address some of these problems.

2.3 Proxy IP Anycast Service (PIAS)

PIAS [14] is an IP anycast service designed to scale in the number of anycast groups. PIAS is the foundation for Stateful Anycast, so a good deal of discussion is warranted.

The core idea behind PIAS is to break the packet delivery process into two stages. The first stage uses "native" IP anycast to deliver a packet to an *anycast proxy*. The proxy then finishes the job by tunneling the packet to a target host via unicast. See figure 2.4 for an illustration.

Each PIAS anycast proxy is a member of a normal IP anycast group. Because they are IP anycast group members, the proxies advertise a common block of IP addresses to

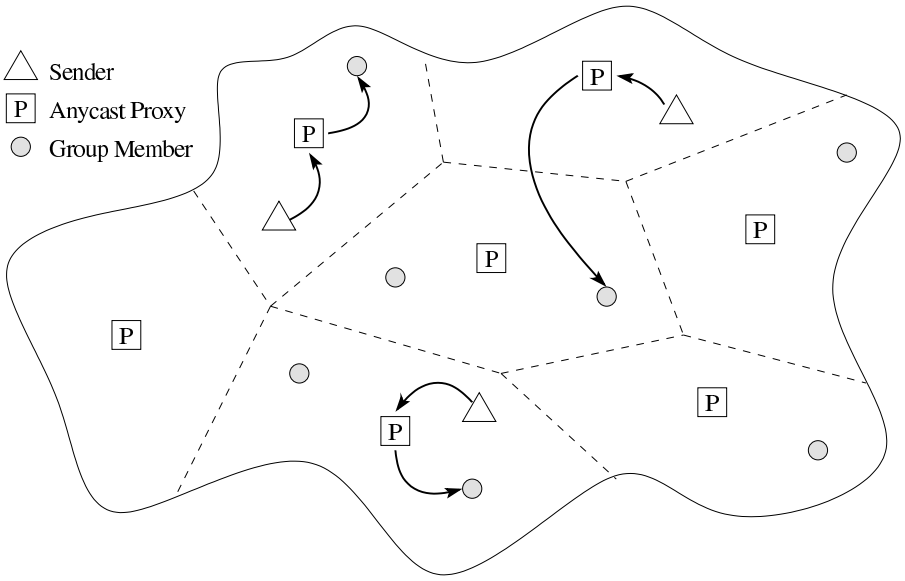

Figure 2.4: Packet delivery in PIAS. There are two distinct packet delivery stages. In the first stage, packets from a sender are delivered to the PIAS anycast proxy (which is a normal IP anycast group member) in the sender's catchment. The anycast proxy then chooses a PIAS group member and forwards the packet to the group member. The chosen group member can be located anywhere in the Internet.

the rest of the Internet via BGP. Each anycast group served by the proxies is assigned an IP address from this block. When a packet arrives at the anycast proxy in the sender's catchment, the proxy uses the destination IP address to identify the corresponding anycast group, selects a member from that group, and tunnels the packet to the group member's unicast address by encapsulating the anycast packet inside a unicast packet. The group member then unwraps the encapsulated packet and injects it into the kernel's routing subsystem to be processed. Return packets follow the reverse path: group members tunnel return packets to the proxies where they are unwrapped and sent to the recipient.

The collection of anycast proxies that announces the same block would be operated by a single organization. Group members in PIAS register with the anycast proxies when they start up to let the proxies know where they can be reached. To prevent malicious users from siphoning a portion of the anycast traffic, there must be a relationship of trust between the operators of the proxies and the group members. This precludes random individuals from contributing to the pool of anycast proxies without coordination and mutual trust; instead, a single organization—an *anycast service provider*—would deploy and manage the proxies. It also precludes random individuals from establishing a PIAS anycast group without obtaining prior permission from the anycast service provider. While these are unfortunate limitations, deployment is still feasible because of PIAS's marketable features, some of which are described below.

2.3.1 Properties of PIAS

This indirection approach has several advantages over regular IP anycast, all of which are inherited by Stateful Anycast:

- *Scalable in the number of groups:* A single route in the Internet's routing fabric is enough to support an arbitrary number of groups. Routers only need to know a route to an anycast proxy in order to support all of the groups served by the proxy. No longer do routers need to know a route per group.

- *Keeps group members isolated from the routing infrastructure:* With normal IP anycast, the routing infrastructure must keep track of all of the group members. It also has to recalculate routes when a group member is withdrawn or added. While these are not major costs, the anycast proxies in PIAS take over the responsibility. Routers on the Internet only need to deal with the anycast proxies, which, if configured well, can be made quite reliable to reduce the frequency of route announcements or withdrawals. Shielding group members also provides some operational secrecy for group operators.

- *Efficient use of the IP address space:* Rather than each group having its own large block of IP addresses regardless of the number of addresses actually used (see section 2.2), all groups share a single block of IP addresses. This allows for the dense packing of IP anycast addresses.

- *Advanced routing:* The anycast proxies can choose a recipient group member based on metrics other than topological proximity. Such metrics can include latency, packet loss, or even workload.

- *Easy group management:* Group members can easily relocate without complicated reconfiguration. This allows group operators to easily move group members to new Internet providers as they quest for ideal locality and service price. All that is needed to deploy a new group member is a basic Internet connection combined with software to tunnel packets to and from the anycast proxies. Group members could even operate behind a NAT [22] device, assuming the tunneling protocol is NAT-traversable.

- *Amortized costs:* Adding a new anycast proxy benefits all PIAS anycast groups. Together, a moderate number of groups can justify a large deployment of anycast proxies. More proxies means better routing efficiency and a better ability to drop unwanted traffic early.

- *Few modifications necessary:* There is no need to modify client software, router software, or routing protocols to deploy PIAS. However, group members do need to run software to initiate an association with the proxies and tunnel packets to and from the proxies.

- *Incrementally deployable:* Anycast proxies, groups supported by the proxies, and group members can all be added one at a time. Small deployments still provide improved performance and protection, so there is no need for heavy initial investments.

- *Support for advanced features:* Because the anycast proxies are "inside" the network, they provide a great location for adding advanced features to the network. For example, Firebreak [25] drops unwanted traffic early by simply placing firewall rules in the anycast proxies.

  This property is the primary reason why PIAS was selected as the scalable basis for Stateful Anycast. Anycast proxies provide the ideal location for the placement of session lookup and revocation enforcement services in order to construct an anycast service that supports stateful conversations in a DDoS-resistant way.

PIAS's approach also has a few disadvantages, also inherited by Stateful Anycast:

- *Slight routing inefficiency:* Just like all indirection architectures, packets suffer from triangle routing. Packets must first travel from the source to the anycast proxy, then travel from the proxy to the destination. This extra stop between the source and destination can lengthen the path traveled by the packet. Longer paths often mean higher latency, higher latency variability, and a greater likelihood of encountering congestion. However, because the anycast proxies are selected by

BGP, an anycast proxy receiving a packet is likely to be the closest proxy to the sender. Thus, the anycast proxy should be close to the direct source-to-destination forwarding path, resulting in a minimal packet detour.

- *Lower reliability:* Anycast proxies are another point of failure. However, with careful design, they may be able to actually improve reliability by quickly detecting a group member failure and routing packets to alternate members.
- *Managed by a single organization:* As discussed above, the anycast proxies must be managed by a single organization that offers the service to others.
- *Group member modification:* Because group members receive packets from the anycast proxies via a tunnel, they must have software to wrap and unwrap tunneled packets. However, application server software running on group members should not need modification.
- *Security by unknown IP address:* To prevent a targeted attack on a group member, the group member's unicast IP address must be kept secret. However, if a group member's IP address is exposed, that group member can be removed from the group. In addition, because group members utilize commodity Internet connections, an exposed group member can be quickly relocated to a different attachment point with a different IP address.

As an interesting aside, PIAS can be used to provide a scalable alternative to AS multihoming. This is not surprising given the fact that an IP anycast group appears to be a single host in a multihomed AS and PIAS is designed to provide a scalable alternative to IP anycast. To construct a multihomed network, a PIAS group operator configures the group members to unload the unwrapped encapsulated packets onto a local network. A reply to an incoming packet follows the reverse path: it is routed through the local network back to the group member that received the incoming packet, then encapsulated and sent to the anycast proxy that originally forwarded the incoming packet. Sending a reply packet via the same path as the incoming packet is not necessary for correctness, but should provide the best performance since the incoming path is likely to be the shortest.

2.3.2 Scalability of PIAS

PIAS moves the challenges of managing a large number of groups, a large number of group members, and group membership dynamics off of the Internet's routing infrastructure and onto the anycast proxies where they can be addressed in a scalable way. PIAS isn't just shifting the problem to a new location; the challenges become manageable by moving them to the proxies. There are many valid approaches for dealing with size and dynamics in a scalable way, and PIAS chooses one particular method described in [14]. For discussion on how Stateful Anycast scales, see section 4.3.

2.4 Internet Indirection Infrastructure ($i3$)

Internet Indirection Infrastructure ($i3$) [69] is an overlay routing service designed to generalize the idea of indirection in routing. Rather than a single level of network-controlled indirection as in PIAS, $i3$ allows for multiple levels of recipient-controlled indirection. As a result of the indirection, $i3$ easily supports host mobility, multicast, and anycast.

Senders do not address packets to the topological location of the intended recipient; rather, packets are addressed to a recipient-determined rendezvous point in the overlay network. Recipients establish the rendezvous point by inserting a *trigger* into the overlay network. In the most basic form of the abstraction, this trigger is a pair consisting of an overlay network rendezvous point identifier and the recipient's unicast IP address. The identifiers are m-bit binary strings with no inherent meaning. If the recipient's address changes due to host mobility, it can simply update the trigger.

When a sender injects a packet into the $i3$ overlay network destined to a certain identifier, the overlay attempts to find any triggers corresponding to the destination identifier in the packet. If one is found, the network takes the packet and forwards it to the recipient indicated in the trigger. If multiple triggers are found, a copy of the packet is sent to each recipient, providing a very simple multicast service.

To support anycast, the identifier bits are divided into two groups. The first k bits of the identifier can be thought of as the anycast group identifier, while the lower-order $(m-k)$ bits can be thought of as the group member identifier. A trigger matches an identifier if the first k bits match and there is no other trigger with a longer prefix match. To send a packet to a member of the anycast group, the sender ensures that the first k bits of the identifier in the packet match the first k bits of the identifiers used by the anycast group members. The remaining $(m-k)$ bits of the identifier can be random. When $i3$ receives the packet, it chooses the group member based on the trigger that has the longest prefix match of the $(m-k)$ bits. It is not clear what happens when multiple unique triggers have equal-length prefix matches.

Note that this is not quite anycast as defined in section 1.6.1. The selection of a group member is not determined by the routing infrastructure. Instead, the group member is determined using the identifier provided by the sender, making the choice of recipient group member completely controllable by the sender. An attacker can exploit this fact to target and take out one group member at a time through packet flooding. Even worse, if the attack traffic causes a group member to be withdrawn, all attack traffic will be automatically diverted to an alternate group member. Thus, anycast in $i3$ is unsuitable for mitigating DDoS attacks without an additional mechanism.

Fortunately, $i3$ can be easily modified to support a true anycast service through simple changes to the syntax and semantics of triggers. One possibility would be to create a new kind of trigger that maps an identifier to a *set* of destination addresses (and/or other identifiers). When an identifier matches the trigger (completely), $i3$ would choose one destination from the set and forward the packet to the chosen destination as usual. The particular choice could be made randomly or according to some metric such

as group member workload or the estimated topological distance to a group member.

While *i*3's anycast service alone may not be suitable for mitigating DDoS attacks, *i*3's unicast service can be combined with PIAS to provide support for stateful sessions. In Stateful Anycast, the anycast proxies use an *i*3-like overlay network to direct packets that are a part of a stateful session to the appropriate stateholder. This arrangement is explained further in section 3.10.5.

*i*3 can be implemented on top of any suitable overlay network. The authors chose to build *i*3 on top of Chord [70], a structured peer-to-peer overlay network that implements a distributed hash table (DHT). This type of overlay network, described below, was chosen for its excellent scalability and reliability properties.

2.5 Distributed Hash Tables and Structured Peer-to-Peer Overlay Networks

A distributed hash table (DHT) is simply a way to store key–value pairs in a distributed, scalable manner. Stateful Anycast relies on a DHT to store mappings between a session identifier (the key) and the session's stateholder (the value) in a scalable, robust manner (see section 3.10.5).

DHTs support three basic operations:

- *put*(*key*, *value*) puts the value *value* into the hash table at the position identified by *key*. If there is already a value associated with *key*, the value is replaced with *value*.
- *value* ← *get*(*key*) fetches the value stored in the hash table at the position identified by *key*.
- *remove*(*key*) removes the value stored in the hash table at the position identified by *key*.

Each node participating in the DHT is responsible for a fraction of the key space. When a *put*, *get*, or *remove* request is made, the request is routed through a structured peer-to-peer overlay network formed by the nodes participating in the DHT. The request is eventually delivered to the node that is responsible for the fraction of the keyspace containing the key.

Efficient overlay network routing algorithms give DHTs their scalability and performance. There are several well-known algorithms, including CAN [64], Chord [70], Kademlia [41], Pastry [66], and Tapestry [77]. All of these key-based routing algorithms trade *diameter* (the maximum number of hops to reach the destination) for *degree* (the number of neighbors each node has). A higher-diameter network typically has lower performance, while a higher-degree network may have higher management overhead and may not scale as well. A typical overlay network has degree $O(\log n)$ and diameter $O(\log n)$.

2.6 Other Related Work

This section provides an incomplete list of other anycast and DDoS work.

2.6.1 Anycast

Global IP-Anycast (GIA) [34] achieves scalable anycast by mapping an anycast address to a default unicast address so that routes to rarely used anycast addresses need not consume router state (routers simply forward the packet to the unicast address). For popular anycast groups, routes to nearby members are sought out and cached. GIA requires all routers to be modified.

IAS [74] is a proposal to improve the scalability of anycast. All groups are divided into sets by taking a hash of the anycast address modulo the desired number of sets. Each router is assigned one of the sets and is required to know how to route to all of the anycast groups in that set. If a router receives a packet destined to a group outside of its assigned set, the packet is forwarded to a nearby router that is assigned the set. IAS requires modification to every router and requires anycast addresses to be distinguishable from unicast addresses.

Szymaniak, et al. [71] propose an anycast mechanism based on mobile-IP [55] that supports stateful sessions. However, this mechanism reveals the unicast IP address of its group members (the "care-of" address) and is therefore not anycast as defined in section 1.6.1. Contact with a group member is initiated through the home agent—a single node that can be attacked to disable the entire group.

Akamai [10] is a company that offers content distribution services. Akamai directs a client to a nearby web cache by returning the web cache's unicast IP address in the DNS query response. While Akamai's service is not anycast as defined in section 1.6.1 (senders can control which web proxy receives packets because the proxies' unicast IP addresses are revealed), the service can be thought of as anycast at a higher layer. Akamai's business model is very similar to the business model envisioned for a Stateful Anycast service provider.

2.6.2 DDoS Mitigation

For additional references, see the survey of network-based DDoS defense mechanisms conducted by Peng, et al. [54].

Prolexic Technologies [63] is a company that specializes in DDoS mitigation. Prolexic uses proprietary filters deployed throughout its network to eliminate malicious traffic before it is delivered to the customer. Prolexic takes advantage of statistical aggregation to reduce the per-customer cost of maintaining a network that is capable of identifying and sinking large malicious loads.

Traffic Validation Architecture (TVA) [75] is a proposal that uses capabilities [39] to verify a sender's authorization to send packets. The capabilities are non-portable because they are specific to the routers along the path from sender to receiver. The

capabilities also expire, allowing a receiver to revoke a malicious session by refusing to renew the sender's capability. Unfortunately, the non-portability causes invalidation of capabilities upon sender mobility or other topology dynamics. TVA also requires a non-trivial deployment of TVA-aware routers before DDoS attacks are substantially mitigated.

Firebreak [25] is a proposed distributed firewall system with the ability to drop traffic close to the sender. As both Firebreak and Stateful Anycast are built on top of PIAS, the two can be combined to provide the best of both systems.

dFence [40] is a proposed network-based DoS mitigation mechanism, envisioned to be deployed by ISPs as a value-added service for their customers. When an attack is detected, routing updates are sent to redirect all of a victim's incoming and outgoing traffic through special middleboxes. These middleboxes classify packets as legitimate or malicious and drop accordingly. Besides a tight coupling with existing protocols and the inability to obtain end-node assistance in classifying traffic, dFence is vulnerable to a flooding attack that can take out a specific middlebox. By using the same session identifier in every malicious packet, an attacker can direct all of the botnet's malicious traffic to a single middlebox, causing its links to saturate and preventing legitimate traffic from reaching the destination.

Chapter 3

Stateful Anycast Design

This chapter describes the technical design of Stateful Anycast. Section 3.1 starts off with a list of the objectives Stateful Anycast was designed to meet. Section 3.2 provides a high-level overview of the design, followed by a couple of example packet exchanges in section 3.3 to help the reader form a mental picture of the whole system. Sections 3.4 and 3.5 discuss how Stateful Anycast is deployed. The remaining sections discuss specific algorithms and procedures.

3.1 Design Objectives

Stateful Anycast was designed to meet several objectives. The objectives are broken into two types: primary and secondary. Primary objectives are required for effectiveness. Secondary objectives are strongly desired, but not critical.

3.1.1 Primary Design Objectives

1. *Support for stateful sessions:* As discussed in section 1.6.2, normal anycast is already an effective way to defend against DDoS attacks when using stateless protocols. With stateful protocols, normal anycast is no longer a reliable option. For Stateful Anycast to provide any benefit over normal anycast, it must be able to direct all packets belonging to the same session to the group member participating in the session.

2. *DDoS mitigation:* Stateful Anycast is primarily motivated by the need to mitigate DDoS attacks. As such, it should have the properties mentioned in section 1.5.

   Mitigating DDoS attacks while supporting stateful sessions is non-trivial because giving the sender a handle on a particular node restores some of the control that was originally taken away by anycast. Normal anycast strips the ability to target a particular group member from the sender and causes attack traffic to naturally diffuse among group members. Adding support for stateful sessions allows the sender to source an arbitrary number of packets to the same

group member, opening up the opportunity to to knock out individual group members. For Stateful Anycast to be an effective means of mitigating DDoS attacks, the sender must not have unchecked control over the choice of packet recipient. Thus, the recipient must be able to somehow revoke sessions when they turn malicious.

3. *Scalable:* Stateful Anycast must be able to scale in the number of groups, in the number of group members (per group and total), and in group membership dynamics.

3.1.2 Secondary Design Objectives

4. *Easily deployable on the Internet:* While creating a mechanism usable only on some hypothetical network architecture may still have value, the economic and social significance of the Internet makes deployability on the Internet an important goal.

   Specifically, Stateful Anycast should be as backwards-compatible as possible without compromising DDoS defense capabilities. Any required client, router, or protocol modifications would limit its value.

   Ideally, Stateful Anycast would either be deployable by uncoordinated participants (like a peer-to-peer network) or provide a strong business case for a commercially-supported deployment. Either way, it should be incrementally deployable and have low setup and operational costs.

5. *Protocol generality:* To help prevent layer abstraction violations and the ossification of protocols, Stateful Anycast should ideally support arbitrary new stateful and stateless protocols without requiring modifications to its software or protocols. Barring that, Stateful Anycast should at least support the major protocols and accommodate new protocols through simple software changes.

6. *Performance and Efficiency:* Stateful Anycast should route packets in the most direct manner possible. Ideally, the path an anycast packet takes to a group member would be the same as the path a unicast packet would take to the group member.

   An added bonus would be the ability to select a group member (for a new session or for sessionless packets) based on advanced metrics such as group member workload, packet loss, and latency.

3.2 Design Overview

Stateful Anycast employs an indirection approach based on PIAS [14] to match a packet to its stateholder. An anycast packet is first delivered to an *anycast proxy* using normal anycast. The proxy then matches the packet to the proper recipient group member and forwards it using regular unicast.

If the packet is part of a stateless protocol, any group member is chosen. The choice of group member could be made randomly or based on an operator-chosen metric such as topological closeness, latency, average packet loss, available bandwidth, or group member workload.

If the packet is part of a bootstrapping stateful session such that no corresponding state has been established in any group member yet (for example, a TCP [61] SYN packet), the packet is treated as belonging to a stateless protocol and sent to any group member.

If the packet is part of an established stateful session, the anycast proxy must determine which group member is participating in the session. This is done by requiring a *session identifier* in every packet and resolving that session identifier to the participating group member using a directory service, discussed in section 3.10. The group members are responsible for inserting, updating, and removing directory service mappings as appropriate.

The directory service can be used in three different ways. First, an anycast proxy can submit a query consisting of a session ID and wait for the reply containing the recipient group member's unicast address before forwarding the anycast packet. However, to prevent the need to queue packets at the anycast proxies while directory service queries are executing, the whole anycast packet can be included as part of the query. The directory service can either deliver the packet itself once the group member has been resolved, or return the packet back to the anycast proxy as part of the resolution results. Each of the three methods has its advantages and disadvantages, as described in section 3.10. Directory service query results are cached at the anycast proxies to improve delivery performance and to help prevent the directory service from being vulnerable to a DDoS attack.

The directory service is implemented as a structured peer-to-peer overlay network in a manner similar to *i*3 (see section 2.4). The mappings between a session ID and its associated group member are stored in a distributed hash table implemented on top of the overlay network. The mappings are similar in concept to *i*3's triggers. A query is routed through the overlay network until it reaches the node that is storing the matching mapping. The resolution result is then returned to the querying anycast proxy.

When delivering an anycast packet to a group member, the packet is encapsulated inside a unicast packet. This allows the packet to be delivered without modification, plus it provides a means for including useful metadata and cryptographic services (see section 3.11).

3.3 Examples with TCP

To help motivate the packet processing algorithms detailed later, this section presents two examples of what happens when a host initiates a stateful session with a Stateful Anycast group member. The protocol used in the examples is Transmission Control Protocol (TCP) [61]. These examples discuss *what* happens, but not *why*. Justifications

will be presented in later sections.

Both of these examples include simplifications and omissions to aid in understanding the general process of establishing a session. Later sections present the complete details.

Starting a TCP session involves three packet exchanges, called the TCP three-way handshake. This exchange is illustrated in figure 3.1 along with each host's TCP state (as described in section 3.2 of [61]). First, the initiator (the client) sends a TCP SYN packet to a host that is listening for new connection requests (the server). The server replies with a TCP SYN/ACK packet—a packet that acknowledges the client's SYN and contains a SYN of its own. Once the client acknowledges the server's SYN with a TCP ACK packet, the session is established and either host may start sending data to the other side.

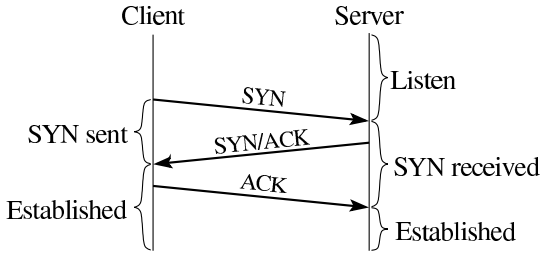

Figure 3.1: TCP three-way handshake and TCP states. Arrows represent packet journeys and time increases in the downward direction.

When Stateful Anycast is introduced, the process of starting a TCP session becomes considerably more complicated. Figure 3.2 shows the start of a TCP session (handshake plus a data packet) in Stateful Anycast when no caching is employed at the anycast proxies. In this particular example, the second packet reaches a second anycast proxy, illustrating how Stateful Anycast can keep a TCP session running when the receiving anycast proxy changes mid-session. The circled numbers correspond to the following steps:

1. Anycast proxy #1 receives a TCP SYN packet from the client. Because a TCP SYN indicates the start of a new stateful session, the proxy treats the packet as belonging to a stateless protocol. The proxy chooses a recipient group member, concatenates the packet with its own unicast IP address, and forwards the bundle to the chosen group member. Figure 3.3 shows the contents of the arriving and departing packets. See sections 3.6.1 and 3.6.2 for more information.

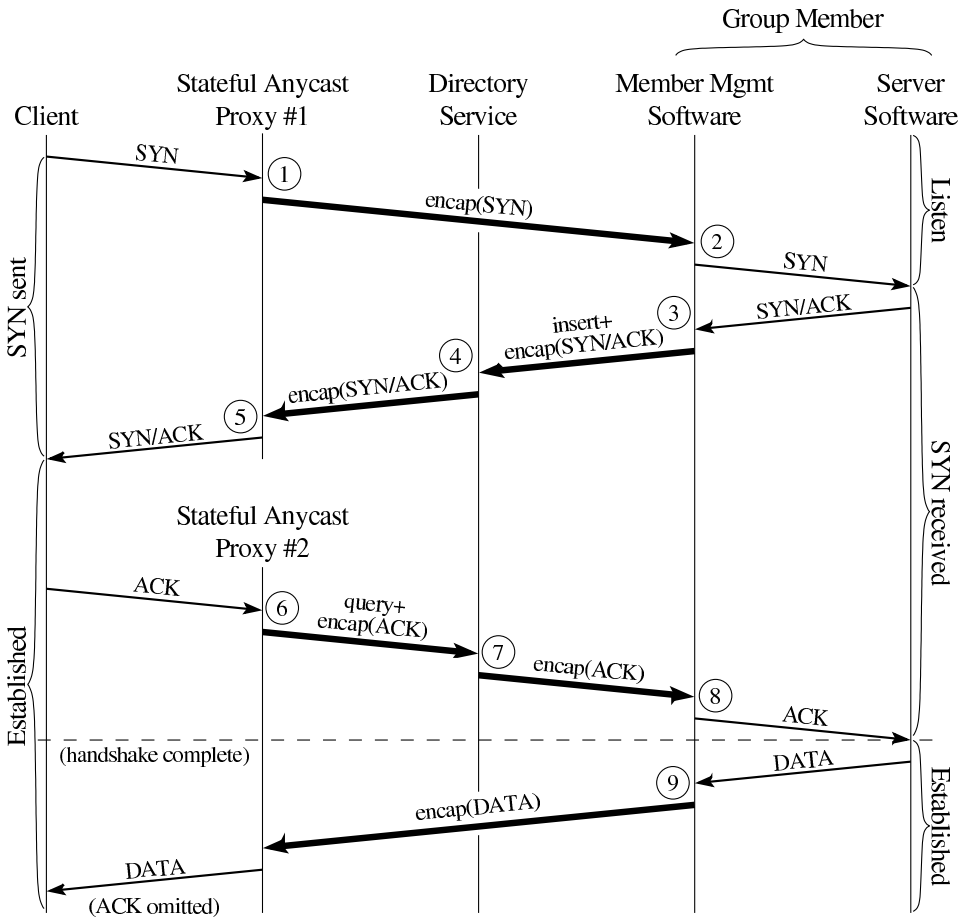

Figure 3.2: The start of an example TCP session with a non-caching version of Stateful Anycast. In this sequence, the client's second packet reaches a second anycast proxy. Bold arrows indicate packets containing another packet inside. The circled numbers indicate interesting steps in the process; these steps are described in the text.

Incoming packet:

To:	group
From:	client
Data:	TCP SYN

Outgoing packet:

To:	member
From:	proxy1
Data:	*To:* group *From:* client *Data:* TCP SYN
	arrived at proxy proxy1

Figure 3.3: Packet contents for TCP example session (without caching), step #1: TCP SYN packet entering anycast proxy #1 and encapsulated SYN packet to be sent to the group member.

2. The group member's Stateful Anycast software receives the encapsulated anycast packet, records that the packet came in via proxy #1, unwraps the anycast packet, and passes the unwrapped packet to the kernel networking subsystem via a virtual network interface. Figure 3.4 shows the contents of the arriving and departing packets. See section 3.6.3 for more information.

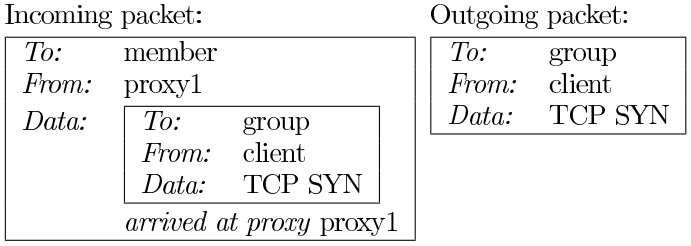

Figure 3.4: Packet contents for TCP example session (without caching), step #2: Encapsulated TCP SYN packet arriving at the group member and the unwrapped TCP SYN packet that is passed on to the kernel networking subsystem. Compare with figure 3.3.

3. The group member's Stateful Anycast software receives the reply TCP SYN/ACK through the virtual network interface. The software reads the session identifier in the packet and forms a packet to tell the directory service to map the session identifier to the group member's unicast address with a limited mapping lifetime. The software also encapsulates the SYN/ACK packet and adds it to the directory service packet along with instructions to deliver the encapsulated packet to proxy #1 (the anycast proxy recorded in step #2) once the directory service mapping has been made. The TCP SYN/ACK accompanies the directory service message in order to guarantee that the mapping has been made before the client replies to the SYN/ACK. Figure 3.5 shows the contents of the arriving and departing packets. See the description of the **insert**() functions in section 3.10.1 for more information.

Incoming packet:

To:	client
From:	group
Data:	TCP SYN/ACK

Outgoing packet:

To:	directoryService
From:	member
Data:	*Map* sessionID *to* member *with lifetime* = 10 seconds. *Deliver the following to* proxy1:

To:	client
From:	group
Data:	TCP SYN/ACK

Figure 3.5: Packet contents for TCP example session (without caching), step #3: TCP SYN/ACK generated at the group member and the encapsulated SYN/ACK to be sent to the directory service.

4. The directory service receives the packet from the group member, inserts a mapping between the session ID and group member IP address, and forwards the encapsulated packet to proxy #1. Figure 3.6 shows the contents of the arriving and departing packets.

Incoming packet:

To:	directoryService
From:	member
Data:	*Map* sessionID *to* member *with lifetime* = 10 seconds. *Deliver the following to* proxy1:

To:	client
From:	group
Data:	TCP SYN/ACK

Outgoing packet:

To:	proxy1
From:	directoryService
Data:	

To:	client
From:	group
Data:	TCP SYN/ACK

Figure 3.6: Packet contents for TCP example session (without caching), step #4: The encapsulated TCP SYN/ACK packets that enter and leave the directory service.

5. Proxy #1 receives the encapsulated packet, unwraps the embedded SYN/ACK packet, and forwards the unwrapped packet to the client. Figure 3.7 shows the contents of the arriving and departing packets.

Incoming packet:

To:	proxy1
From:	directoryService
Data:	*To:* client *From:* group *Data:* TCP SYN/ACK

Outgoing packet:

To:	client
From:	group
Data:	TCP SYN/ACK

Figure 3.7: Packet contents for TCP example session (without caching), step #5: The encapsulated TCP SYN/ACK packet arrives at the proxy, is unwrapped, and sent to the client.

6. Proxy #2 receives the ACK packet from the client and recognizes this packet as part of a stateful protocol. The proxy forms a directory service query packet. Included in this packet is the session ID, the proxy's unicast address, and the ACK packet. Figure 3.8 shows the contents of the arriving and departing packets.

Incoming packet:

To:	client
From:	group
Data:	TCP ACK

Outgoing packet:

To:	directoryService
From:	proxy2
Data:	*Query for* sessionID *Deliver on resolution:* *To:* client *From:* group *Data:* TCP ACK *arrived at proxy* proxy2

Figure 3.8: Packet contents for TCP example session (without caching), step #6: The client's ACK packet arrives at a different proxy and is sent to the directory service as a query.

7. The directory service receives the query, resolves the group member (based on the mapping that was inserted in step #4), and forwards the encapsulated anycast packet to the group member. Because no caching is employed, there is no need to return the query results to the anycast proxy. Figure 3.9 shows the contents of the arriving and departing packets.

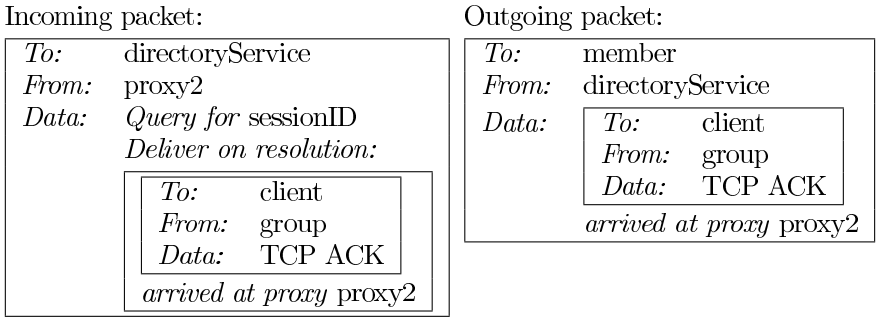

Figure 3.9: Packet contents for TCP example session (without caching), step #7: The directory service query is processed and the ACK packet relayed to the group member.

8. The group member's Stateful Anycast software receives the encapsulated packet, records that the anycast packet came in via proxy #2, unwraps the ACK packet, and passes the unwrapped packet to the kernel networking subsystem via the virtual network interface. This step is identical to step #2.

9. This particular outgoing packet is delivered via proxy #2 because the last incoming session packet came in via proxy #2.

3.3.1 Example with Caching

Without caching, the anycast proxies never deliver packets directly to the group members because they never know which group member should receive a particular stateful packet. Instead, incoming packets are always handed off to the directory service for delivery. Not only could this slow down Stateful Anycast, this could potentially expose the directory service to various types of DDoS attacks. The following is an example of Stateful Anycast with caching. Figure 3.10 illustrates the packet exchange when caching is involved. Several key differences in the way packets are processed exist between the non-caching and caching version of Stateful Anycast. The circled numbers identify interesting packet processing steps and correspond to the following descriptions:

1. Sending the SYN/ACK is almost the same as in the non-caching version of Stateful Anycast. The only difference is that the encapsulated packet is accompanied by caching information to minimize the need to query the directory service. Note that a directory service mapping is still inserted in case the client's second packet—the ACK packet—arrives at a different proxy. Figure 3.11 shows the contents of the arriving and departing packets.

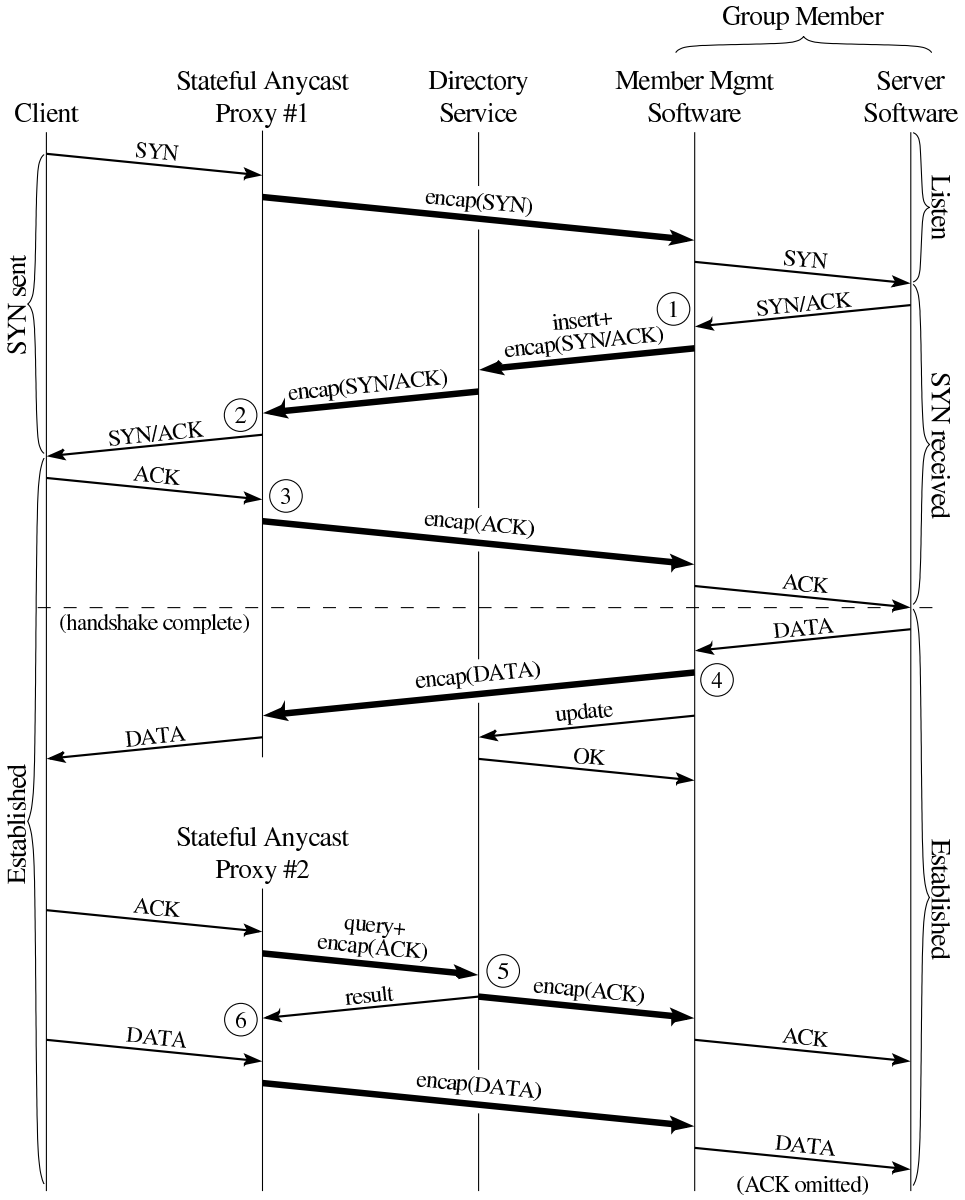

Figure 3.10: TCP three-way handshake with a caching version of Stateful Anycast. In this example, the receiving anycast proxy changes between the sender's second and third packets. The directory service is queried only when a packet arrives at a different anycast proxy. Compare with figure 3.2.

Incoming packet:

To:	client
From:	group
Data:	TCP SYN/ACK

Outgoing packet:

To:	directoryService
From:	member
Data:	*Map* sessionID *to* member, *entry lifetime* = 10 seconds, *cache lifetime the shorter of:* 10 seconds *or* 10 packets. *Deliver the following to* proxy1 *along with caching info:*

To:	client
From:	group
Data:	TCP SYN/ACK

Figure 3.11: Packet contents for TCP example session (with caching), step #1: The encapsulated TCP SYN/ACK is accompanied by mapping instructions for the directory service as well as caching information for the proxy to help it avoid the need to query the directory service. Compare with figure 3.5.

2. The proxy receives the encapsulated packet from the directory service, inserts the mapping into its cache, and forwards the SYN/ACK to the client. Figure 3.12 shows the contents of the arriving and departing packets.

Incoming packet:

To:	proxy1
From:	directoryService
Data:	*Cache* sessionID *to* member, *lifetime the shorter of:* 10 seconds *or* 10 packets. *Deliver the following packet:*

To:	client
From:	group
Data:	TCP SYN/ACK

Outgoing packet:

To:	client
From:	group
Data:	TCP SYN/ACK

Figure 3.12: Packet contents for TCP example session (with caching), step #2: The incoming encapsulated packet is accompanied by cache instructions to help the proxy deliver future packets without needing to query the directory service.

3. The proxy receives the ACK from the client, recognizes the packet as part of a stateful session, extracts the session ID, and looks up the session ID in its cache. Because the cache contains the mapping, the proxy can encapsulate and forward the packet directly to the group member.

4. The group member includes cache refresh instructions with the outgoing data packet. The group member also separately refreshes the directory service mapping in case future packets arrive via a different proxy.

5. With caching, it is useful to return the directory service lookup results to the proxy. Along with the session ID to group member mapping, the results contain information on how long the mapping should be cached at the proxy (set by the group member in step #1).

6. The proxy caches the results such that all future packets can be delivered directly to the group member.

3.4 Deploying the Stateful Anycast Service

Deploying Stateful Anycast is similar to deploying PIAS (see section 2.3). First, the operator of the Stateful Anycast service acquires a block of IP addresses to advertise via BGP. Each Stateful Anycast group will be assigned one or more of the IP addresses from the block. The group of proxies themselves should be assigned one of the addresses from the block to make it easier for a group member to find its closest proxy. Next, the anycast service operator deploys the anycast proxies. As anycast proxies are also normal IP anycast group members, deployment is identical to deploying an IP anycast group (see section 2.2). Next, the Stateful Anycast software is deployed on the proxies. This software contains the basic processing rules (see section 3.6) as well as software that implements a distributed directory service for resolving session identifiers to their corresponding group members (see section 3.10).

Multiple proxies should be deployed at each attachment point to maximize service availability. For more information, see the discussion on the limitations of IP anycast in section 2.2.

3.5 Deploying an Anycast Group

To add a new group, the anycast service provider assigns the group one or more IP addresses. The group operator deploys its group members as it sees fit. For optimal performance, the group members should be placed at strategic locations just like normal IP anycast group members would. However, placing Stateful Anycast group members is much easier than placing normal IP anycast group members. Stateful Anycast group members need only be a simple host with a simple connection to the Internet because packets arrive and leave using unicast. The group members can even reside behind a NAT [22] box without affecting the services provided by the group member—as long as the proxy to group member tunneling protocol is NAT-traversable.

A group operator can deploy as many group members as it sees fit. The number of group members is independent of the number of anycast proxies. A group operator could choose to deploy only a single group member in order to take advantage of the filtering and session revocation capabilities of the anycast proxies. Or the operator could deploy many more group members than proxies, which might be necessary if the group members have low bandwidth connections or are computationally limited.

For extra flexibility, the group operator could choose to implement its own directory service. The anycast proxies would then query the group's directory service in order to route packets. Or, the group could also choose to have the proxies treat all packets as stateless. This could be used to implement a scalable multihoming solution by configuring each group member to offload incoming packets to a local network.

Group members run special software that performs many tasks. For one, it authenticates the group member to the anycast proxies to prevent unauthorized hosts from intercepting incoming traffic and sourcing outgoing traffic. It also provides the tunnel end-point for unwrapping incoming packets and wrapping outgoing packets. Finally, it provides management functions such as the propagation of workload information.

3.6 Packet Processing Algorithms

This section describes the procedures for processing packets as they travel from source hosts to the group members. It also describes the reverse path—what happens when a group member sends a reply packet back to the host.

3.6.1 Delivering Packets to the Anycast Proxies

IP anycast is used to route packets to the anycast proxies. Clients address packets to the anycast group's IP address and the Internet's existing routing infrastructure forwards them to the anycast proxies. Clients need no special modification or software—they simply treat the anycast group as if it was a normal Internet host.

Anycast proxies must be configured to listen for and accept packets destined to the whole block of advertised IP addresses and then pass the packets to the Stateful Anycast proxy software. How this is implemented depends on the lower-layer networking technology used and the design of the networking subsystem of the proxy's operating system. An implementation using Ethernet and packet capture libraries is described in section 5.3.2.

3.6.2 Processing Packets at the Anycast Proxies

When a packet arrives at an anycast proxy, the proxy must determine which group member should receive the packet. This process requires a handful of steps. Several of the steps are only briefly explained here; full details can be found by following references to the appropriate section.

1. Identify the group that will receive the packet. Each group is assigned an address, so the anycast proxy only needs to examine the destination address in order to discover which group should receive the packet.

2. Fetch and process the group's configuration details. Each group might choose different features, which may or may not relate to the core functionality of Stateful

Anycast. For example, a group could choose to turn off support for stateful sessions and have only basic anycast service. (If the group does not want support for stateful sessions, the proxy skips to step #7.) Or, a group may desire the proxies to enforce specific packet filter rules, as is done in Firebreak [25].

3. Determine whether the packet is part of a stateful session (see section 3.7).

4. For packets belonging to a stateless session, pick any group member (see section 3.8) and jump to step #7.

5. For packets that are part of a stateful session, extract the session ID from the packet (see section 3.9).

6. Check the local session ID resolution cache to see if the session ID is known to the proxy. If it is, decrement the entry lifetime counter. (See section 3.10.)

7. Encapsulate the anycast packet: concatenate the packet with metadata (including the unicast address of the anycast proxy that received the packet) and session ID (if the packet is part of a session). See section 3.11 for more details.

8. If the particular group member recipient is unknown, pass the encapsulated packet to the group-specific directory service as a query (see section 3.10). The directory service takes responsibility for delivering the packet to the appropriate group member and returns the resolution result to the originating anycast proxy to be cached.

   Otherwise, send the encapsulated packet directly to the group member.

3.6.3 Receiving Packets at the Group Member

Group members run software that listens for the incoming encapsulated packets. Once an encapsulated packet comes in, the software processes any metadata accompanying the encapsulated packet (see section 3.11), unwraps the packet, and hands the packet to the appropriate network application process for further processing.

One particular piece of metadata—the address of the anycast proxy that originally received the incoming anycast packet—is recorded to enable reply packets to be sent through the same anycast proxy that received the incoming packet. While this is not strictly required for correct operation, using the same proxy for reply packets has several benefits, discussed in the following section.

How the packet is handed to the appropriate network application process depends on the design of the application software. Section 5.2 discusses an implementation that provides backwards compatibility with legacy networking application software.

3.6.4 Sending Reply Packets

Outbound packets follow the reverse path (with an exception for packets that go through the directory service—see the **insert**() functions in section 3.10.1). Once the outbound

packet is handed to the group member software, the software encapsulates the outbound packet and its accompanying metadata inside a packet addressed to an anycast proxy.

Outbound packets are sent back through a proxy for two reasons: to bypass ingress filters [23] and to efficiently send instructions and feedback to the proxies.

While the group member can send the encapsulated outbound packet to any proxy, group members should choose the remote host's local anycast proxy for sending outbound packets. The biggest reason is to pre-cache the anycast proxy with the session's mapping in order to eliminate the need to query the directory service. Another reason is routing efficiency (the remote host's proxy is close to the remote host). Finding the remote host's local proxy is done by noting which proxy received the remote host's latest incoming packet.

Once the encapsulated packet arrives at the proxy, the proxy processes the metadata and sends the outbound packet. The network's routing infrastructure then delivers the outbound packet to the destination.

3.6.5 Reacting to Attacks

Stateful Anycast proxies make no attempt to detect attacks. A proxy's job is to simply forward packets as fast as possible, not to analyze the sender's intent. Instead, the anycast proxies rely on the group members to detect whether a particular session has become malicious.

Once a group member has determined that a session is malicious, it can revoke the session by contacting the directory service and mapping the session ID to a special value that instructs Stateful Anycast to drop the packet. This new mapping will affect proxies that do not have a mapping to the victim group member cached. Any proxies that still have a cache entry that maps the session ID to the victim will continue to forward malicious packets until the cache entry expires or the proxy is contacted by the victim group member and instructed to change the cache entry destination to "drop".

Session revocation assumes that the group member is able to contact the directory service and proxies while under attack. If it is not able to send outgoing packets, possible actions are to use an out-of-band communication mechanism, wait for the directory service mapping to expire, or rely on a dead-man switch—a monitoring service that withdraws all of the victim host's directory service mappings once it detects that the victim host is no longer responding to status queries.

The process receiving the packets on the group member machine is in the best position to gauge whether a session has become malicious. It has the most information about the remote peer's behavior and how well it matches appropriate or expected behavior. However, the group member software may also have useful information about packets that is not be passed to the receiving process. For example, the group member software knows which proxy received each packet, but this information is not passed on if the receiving process is a legacy software application. Packets belonging to the same session that arrive via many different proxies may indicate a distributed attack on the group member. Collecting valuable information from all possible sources and

deciding whether a session has turned malicious would likely be the most difficult task when deploying a DDoS-resistant anycasted service.

3.7 Determining Whether a Packet is Part of a Stateful Session

For an anycast proxy to determine whether a packet is part of a stateful session, the proxy must have a partial understanding of the syntax and semantics of the protocols used by the groups it supports. Thanks to the IETF's publicly-accessible RFCs, this is a simple matter for most protocols. For example, TCP [61], the ubiquitous Internet transport protocol, is connection oriented and therefore stateful. Any IP packet with a value of 6 in the "Protocol" field [60] is a TCP packet and therefore known to be part of a stateful protocol (with an exception; see the next paragraph). User Datagram Protocol (UDP) [58], another common transport protocol, is connectionless and therefore not inherently stateful. However, stateful higher-layer protocols, such as multimedia streaming protocols, may use UDP as a foundation. Thus, the proxy must examine the UDP payload to discover whether the packet is part of a higher-layer stateful protocol session.

Packets belonging to a bootstrapping stateful protocol, such as TCP SYN packets, are treated as belonging to a stateless protocol. Any group member can receive these packets without disrupting the protocol's operation because no state has been established in a group member (yet).

Session statefulness can be relative to the role in a packet exchange. A session could be stateful to both participants, or to only one participant. A session is stateful to a participant if the participant must keep state between messages to ensure correct operation. TCP is stateful to both participants because both ends must keep track of the state of the connection. A DNS [46, 47] lookup is stateful to the query issuer because the issuer waits for a response to its query packet. From the perspective of the responder, the DNS lookup session is stateless because there is nothing about the session that needs to be remembered after the reply packet has been sent. Therefore, proxies need to be aware of the role being taken by the group member in addition to the nature of the protocol embodied in a packet. Throughout this document, a session is considered to be stateful when it is stateful from the perspective of a group member.

The information needed to determine whether a packet is part of a stateful protocol is not always available to the anycast proxy. For example, encryption can completely obscure the contents of a packet. IPsec [35] can hide everything except for the encrypting host's IP address, the group's anycast IP address, and the fact that the payload is an IPsec-encrypted packet. As another example, a protocol may rely on historical context to interpret a packet's data. In other words, the contents of a previously sent packet may be needed to extract a session ID, and the two packets might not arrive at the same anycast proxy.

If an anycast proxy is unable to determine whether the packet is part of a stateful

protocol, the proxy could either treat the packet as belonging to a stateless protocol and simply deliver to any group member, or use the limited information it has to attempt to identify the stateholder (see section 3.9). The choice can be hard-coded or expressed in group configuration information. Ideally, groups will choose to use only protocols that expose the relevant information to the anycast proxies in every packet, making this issue irrelevant.

3.8 Choosing a Recipient Group Member for Non-Stateful Packets

For packets that either belong to a non-stateful session or to a bootstrapping stateful session, anycast proxies can arbitrarily choose any group member. Choosing a group member based on a metric such as topological closeness, latency, or workload may result in a significant performance improvement from the perspective of the remote host. An appropriate algorithm for choosing a group member depends highly on local circumstances, such as the anycast group size, group membership dynamics, availability of network and group member performance measurement algorithms, the needs of the group, and so on. The choice of group member selection algorithm does not need to apply to every group; each group can use a different algorithm.

Akamai has effectively solved the problem of assigning an arbitrary client to a "good" server in a scalable way. Algorithms similar to those employed by Akamai can provide scalable group member assignments that result in good performance as seen by remote users—perhaps even better than unicast thanks to proximity to the user.

A simple way to choose a group member is to randomly choose from the group members that are in the anycast proxy's catchment. Proxies can become aware of the group members in their catchments by asking the group members to periodically connect to the anycast IP address reserved for the group of proxies. Extra mechanisms would be required to handle the cases of no group members in a catchment and a non-even distribution of group members among the proxies. This approach can be extremely effective if the group members are collocated with the anycast proxies.

There may be a significant advantage in choosing an algorithm that consistently routes packets from a particular source to a particular group member. This consistency can aid in providing proper behavior when proxies are unable to definitively identify the proper recipient of an incoming packet due to a failure in the directory service or the inability to extract a session identifier from a packet.

3.9 Session Identifiers

Session identifiers are the focal point of Stateful Anycast's support for stateful sessions. Stateful Anycast requires that every packet contain a globally-unique session identifier. This section discusses the reasons for choosing session IDs and how they are extracted from packets.

3.9.1 Motivation for using session IDs

For packets that are part of a stateful session, anycast proxies must determine which group member is holding the session's state. To do this, the packet must contain information that somehow indicates the identity of the group member. There are two possible ways to do this: (1) require the sender to embed a *group member* identifier inside every stateful packet, or (2) require the sender to embed a *session* identifier inside every stateful packet and then use the session identifier to determine the group member.

Having an embedded group member indicator in every packet allows the anycast proxies to quickly determine which group member should receive the packet. All the proxies would need to do is extract the group member indicator, look up the unicast IP address of the corresponding group member, and forward the packet. Unfortunately, most protocols do not have an effective way to embed this information. In addition, requiring an embedded group member indicator would require a change in sender software, which is contrary to objective #4 (easy deployability). While this is only a secondary objective, there is a more significant problem: Stateful Anycast would not meet design objective #2 (DDoS mitigation). If the indicator was the only piece of information used by the anycast proxies to determine the recipient group member, a malicious session could not be revoked because there is no way to distinguish between different sessions at the proxy. Essentially, the group member indicator would be an irrevocable handle on a particular group member.

Instead of embedding a group member indicator inside every packet, Stateful Anycast requires some sort of session identifier to be embedded in every incoming packet. Fortunately, this information is already present in most stateful protocol packets, eliminating the need to require sender software or protocol modification. In order to support multiple concurrent sessions between hosts, protocols typically embed some information in every packet that helps the receiving host demultiplex the various sessions. This information, when combined with data from the lower-layer protocols such as IP addresses, yields a globally unique session identifier. For example, TCP uses a combination of source IP address, destination IP address, source port, and destination port to uniquely identify the session a particular TCP packet is in. This set of data is globally unique per session, as no two peers will use the same source and destination port pair for distinct sessions. UDP also has source and destination port fields, which are typically used in a manner similar to TCP ports (packets belonging to the same session will have matching IP addresses and ports).

3.9.2 Extracting the session ID

For many protocols in use on the Internet, extracting the session identifier is straightforward. Most sessions are contained inside a single TCP session, and TCP exposes the session identifier in the form of the following tuple: {*client IP address*, *client TCP port*, *server IP address*, *server TCP port*}. Protocols that use UDP usually follow the same convention: packets belonging to the same session all have the same client and server IP

addresses and port numbers. Protocols with randomly generated session identifiers or randomly generated per-session host identifiers are ideally suited for Stateful Anycast, assuming the identifiers are exposed in every packet. Such identifiers are present in *i*3 in the form of private triggers and may be present in future host identity or mobility protocols.

It may not always be possible to extract the session identifier from a stateful packet. Three possible causes are encryption, aggregating multiple lower-layer sessions into a single higher-layer session, and sessions that are identified using context not available in the packet.

Encryption can prevent a proxy from extracting a session identifier if the session identifier is part of the encrypted data. For groups that require the use of protocols that encrypt the session identifier, the decryption keys must be shared with the anycast proxies.

A high-level session composed of multiple low-level sessions can also prevent a proxy from directing packets to the right group member. As an example, imagine an anycasted web server for a retail business. Rather than storing the user's shopping cart data in a user-held cookie or specially formed URL, the storefront software stores the user's shopping cart data in the web server. To participate in the session, the user sends an identifier (such as a username) to the server with every HTTP query. The server uses the identifier to fetch the session state. Because each HTTP query can use a different TCP session, the anycast proxies must be able to direct all of the component TCP sessions to the same web server. Unfortunately, a higher-layer session identifier can not be included in a TCP SYN packet, making this task impossible. While it is easy to work around this particular problem by using cookies, specially formed URLs, and/or a common database shared among all group members, it illustrates how higher-layer sessions that use multiple lower-layer sessions can be difficult to support.

A session might only be identifiable with information not present in the received packet. The required information might only be available in a previously-sent packet or some other historical context unavailable to the proxy.

3.10 Session Identifier Directory Service

The directory service is responsible for matching a session identifier to the group member holding the session. Mappings between a session identifier and a group member are stored in the directory service and are queried by proxies to determine which group member should receive a particular stateful packet.

3.10.1 Directory Service Functions

The directory service exports several functions to the anycast proxies and group members. There are three basic functions: query, insert, and remove.

Query Functions

There are three query function variations, each of which has advantages and disadvantages. The following are descriptions of the directory service query functions:

- *result* ← **query**(*sessionID*) instructs the directory service to find a stateholder mapping for the session identified by *sessionID*. The results are delivered back to the querying host, which uses the results to deliver an incoming anycast packet. See figure 3.13 for an illustration of the process.

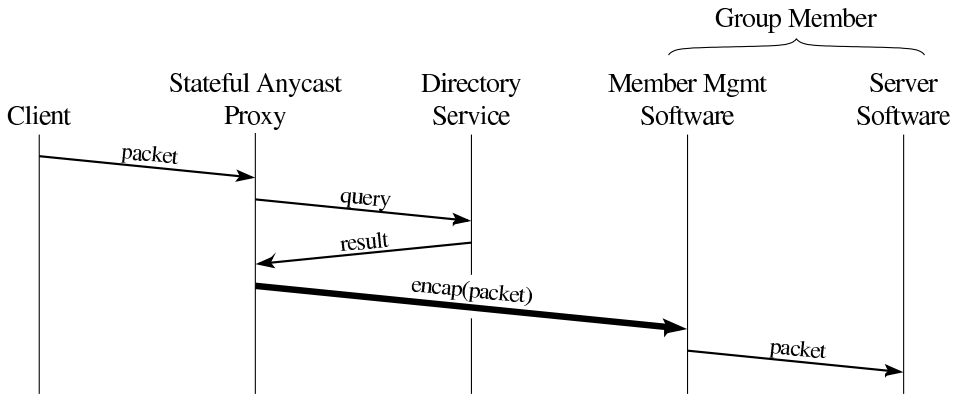

Figure 3.13: Directory service query and packet delivery using **query**().

If there is no mapping associated with the session identifier, the directory service returns a configurable (per-group) default destination in the results. This destination could be a special value that instructs the anycast proxies to drop the packet or treat it as if it was part of a stateless protocol. This returned result is cached just like a successful query result.

Advantages of this function (relative to the other two query functions): (1) If multiple same-session packets arrive at the proxy, only one query needs to be issued for the collection of packets. (2) The collection of packets can be delivered in the order they arrived at the anycast proxy. (3) The query's packet size is small, minimizing the burden on the directory service.

Disadvantages of this function: (1) anycast proxies must queue incoming packets as they wait for replies from the directory service. (2) Waiting for a reply from the directory service can increase packet delay. Asking the directory service to deliver the packet once the query is resolved can improve performance.

- *result* ← **queryAndForward**(*sessionID*, *packet*) instructs the directory service to find a mapping for the session identified by *sessionID* and forward the packet *packet* to the group member identified in the mapping. The results are returned to the query issuer to be cached. This is the query method used in the TCP examples in section 3.3. See figure 3.14 for an illustration of the process.

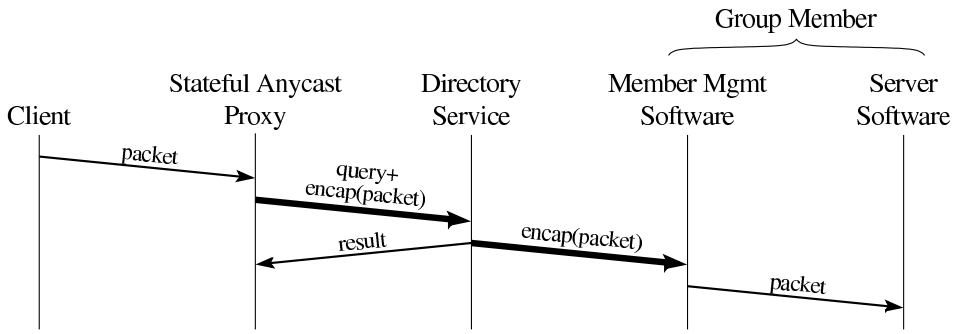

Figure 3.14: Directory service query and packet delivery using **queryAndForward**().

Advantages of this function: (1) Anycast proxies do not need to queue packets as they wait for query results. (2) An individual packet's delay is minimized as the packet is forwarded the instant the destination host is discovered.

Disadvantages of this function if an anycast proxy does not wish to queue packets: (1) Packets sent to the directory service are large, imposing an increased burden on the directory service infrastructure. (2) When multiple packets for the same session arrive before the first query results return, the same session ID query is issued multiple times. This repeated work can be a significant problem if each directory service query is burdensome on the infrastructure. (3) The combined delay of all of the incoming packets can be high, especially if the directory service takes a significant amount of time in processing a query. To minimize delay, it would be better to queue the incoming packets that arrive shortly before the session ID query results are returned. (4) This method can result in significant out-of-order packet delivery, especially if directory service queries take a long time to return. If a burst of an unknown session's packets suddenly arrives at an anycast proxy, many of the packets that are sent through the directory service can be delivered to the group member after packets that arrive at the anycast proxy shortly after the query results are returned. Figure 3.15 shows an illustration. The number of out-of-order packets depends on the incoming packet rate and the relative difference between the time it takes to route packets through the directory service and the time it takes to send packets directly to the group member. While the Internet's best effort delivery makes no promises about in-order packet delivery, significant out-of-order delivery can disrupt protocols such as TCP.

Problems that arise with this method may be reduced by queueing packets at the anycast proxies.

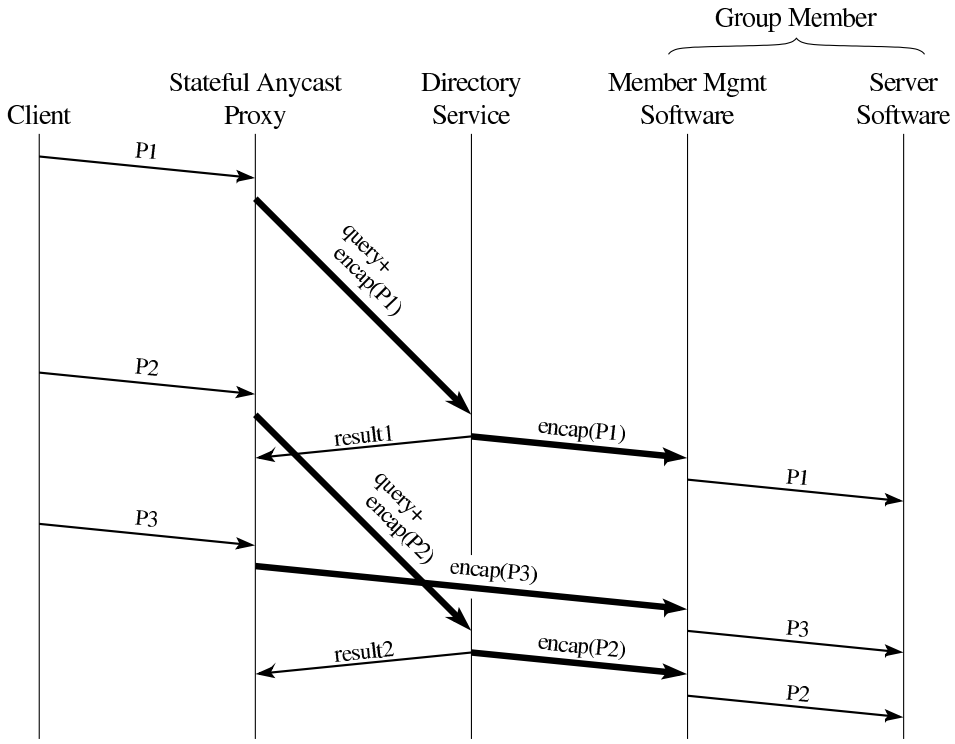

Figure 3.15: Out-of-order packet delivery with **queryAndForward**(). Each of the three packets belongs to the same session. Because the directory service takes a long time to process a query, a packet that arrives at the proxy right after results are returned is delivered to the group member before a packet that arrives at the proxy just before the results are returned.

- (*result*, *packet*) ← **queryAndReturn**(*sessionID*, *packet*) instructs the directory service to find a mapping for the session identified by *sessionID* and return the packet *packet* to the anycast proxy along with the results. See figure 3.16 for an illustration of the process. This is a compromise between the above two methods: queries are returned to the anycast proxy before the packet is sent to the group member to minimize out-of-order delivery, and the packets are passed to the directory service to eliminate the need to queue packets at anycast proxies.

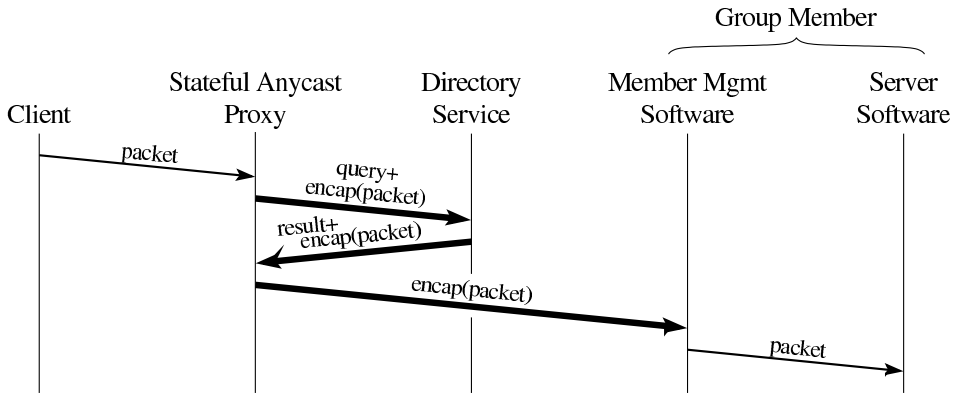

Figure 3.16: Directory service query and packet delivery using **queryAndReturn**().

Advantages of this function: (1) Anycast proxies do not need to queue packets as they wait for query results. (2) Because the anycast proxies receive the query results before forwarding any packets, packets are more likely to be delivered in order.

Disadvantages of this function: The first three disadvantages are the same as in **queryAndForward**(): (1) Packets sent to the directory service are large, imposing an increased burden on the directory service infrastructure. (2) When multiple packets for the same session arrive before the first query results return, the same session ID query is issued multiple times. This repeated work can be a significant problem if each directory service query is burdensome on the infrastructure. (3) The combined delay of all of the incoming packets can be high, especially if the directory service takes a significant amount of time in processing a query. To minimize delay, it would be better to queue the incoming packets that arrive shortly before the session ID query results are returned. (4) This method can still result in out-of-order packet delivery if repeated queries for the same session ID take a different amounts of time to process. See figure 3.17 for an illustration. The likelihood of this happening depends on how the directory service is implemented.

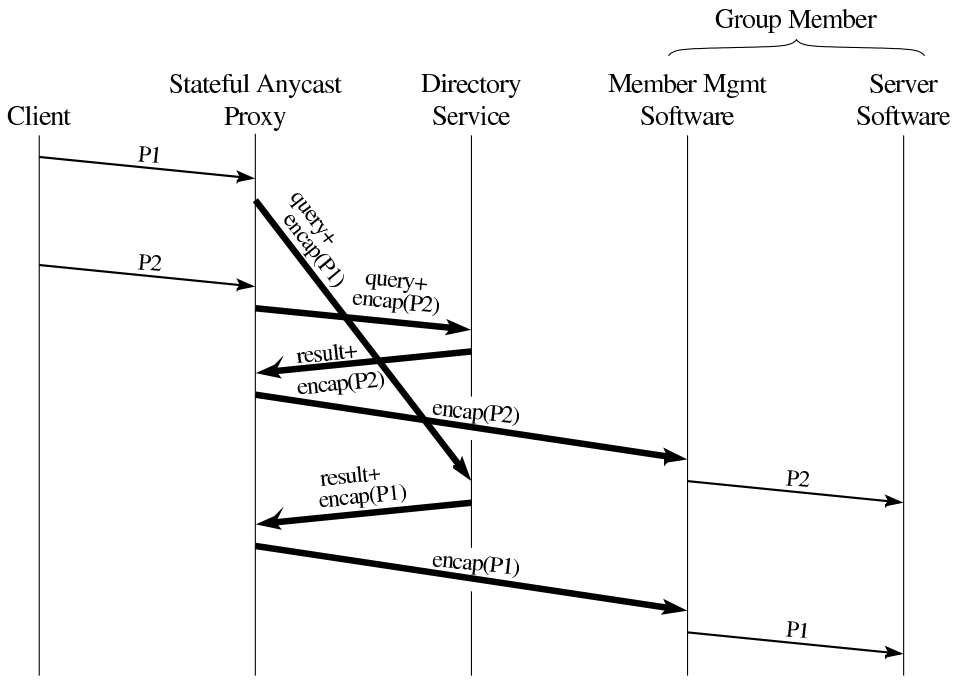

Figure 3.17: Out-of-order packet delivery with **queryAndReturn**(). Both packets belong to the same session. Because the directory service takes a longer time to process the first query than the second, the second packet is delivered to the group member first.

The appropriate circumstances to use each method is left to future experimentation. Directory service and anycast proxy software can easily support all three methods, making experimentation easy.

Insert Functions

There are also three insert function variations, similar to the three query function variations. The following are descriptions of the directory service insert functions:

- *status* ← **insert**(*sessionID*, *groupMember*, *mappingLifetime*, *cacheLifetime*) instructs the directory service to insert a mapping between *sessionID* and *groupMember* with mapping entry lifetime settings as specified in *mappingLifetime* and cache lifetime settings as specified in *cacheLifetime*. When the mapping has been inserted, a success or failure code is returned. If a mapping already exists for the session ID, the mapping is replaced, allowing **insert**() to be used to update existing mappings.

  This function is used by the group members when a session has been established. Outgoing packets that evoke a response must wait for a successful directory service mapping insertion or else the reply packet may reach the directory service before the mapping has been inserted, resulting in a failed lookup. Thus, just as in **query**(), this method requires the group member software to queue outbound packets that would evoke a response in the remote host until the insertion operation returns successfully. However, queueing is not likely to pose a problem because the number of outbound packets at the beginning of a session is usually small, especially for congestion-controlled transport protocols.

- *status* ← **insertAndForward**(*sessionID*, *groupMember*, *mappingLifetime*, *cacheLifetime*, *packet*, *proxy*) is similar to **insert**(), but also instructs the directory service to forward *packet* to *proxy* when the mapping has been inserted.

  This function may provide lower packet delay over **insert**() and it does not require packet queueing. However, just as with **queryAndForward**(), multiple outbound packets may result in excess directory service work and out-of-order packet delivery. However, a directory service insert of this sort is likely to take place only after the start of a session, and congestion-controlled transport protocols such as TCP will not have many packets to send at first.

- (*status*, *packet*) ← **insertAndReturn**(*sessionID*, *groupMember*, *mappingLifetime*, *cacheLifetime*, *packet*) is similar to **insert**(), but also instructs the directory service to return *packet* along with the success or failure code.

  This function is a compromise between **insert**() and **insertAndForward**(). With this function, group members can insert a directory service mapping without queueing outgoing packets and while minimizing the probability of out-of-order packet delivery. Because the number of outgoing packets is typically not high at the start of a session, **insertAndForward**() will likely be favored over this function due to this function's greater packet delay.

Remove Function

Finally, there is one remove function:

- *status* ← **remove**(*sessionID*) instructs the directory service to remove the mapping for the session identified by *sessionID* and return a success or failure code. If there is no mapping with that session identifier, the directory service simply returns a success code.

  This function can be invoked when a session has completed or when a group member wishes to revoke a session.

3.10.2 Directory Service Mapping Data Structure

A directory service mapping has four basic parts: the session identifier being mapped, the destination group member, when the mapping should expire, and how long a mapping should be cached in an anycast proxy.

The destination group member field usually contains the unicast address of the group member, but it can also hold special values to indicate alternate processing. One special value instructs proxies to simply drop packets that are a part of the session. This value is employed when a group member wishes to have a session negatively cached at the proxies, perhaps because the session has become malicious. Another special value instructs proxies to treat packets belonging to the session as stateless.

The directory service mapping lifetime field is an expiration date on the mapping. This value is used only by the directory service to purge stale mappings. This expiration date provides two benefits. First, it keeps the directory service clear of stale mappings that group members might fail to remove. Second, it provides an automated session revocation mechanism in case a group member is unable to remove the mapping due to an attack. It is therefore important to keep the lifetime as short as possible without requiring an impractical number of directory service updates to extend the lifetime.

The cache lifetime information tells a proxy how long it should cache the results of a directory service query. There are two parts to the cache lifetime: a byte limit and a time limit. These two are explained below.

Figure 3.18 shows an sample mapping entry.

Session ID:	sessionID
Group Member:	groupMemberIP
Mapping Expiration:	May 28, 2007
	at 14:00:00 EDT
Cache Lifetime:	*The shorter of:*
	10 seconds *or* 1×10^6 bytes

Figure 3.18: Sample directory service mapping.

3.10.3 Caching Directory Service Query Results

Caching is a critical part of Stateful Anycast. Caching improves performance by giving anycast proxies the ability to route packets directly to the recipient group member. Packet delays and network load can be significantly reduced by avoiding multi-hop treks through an overlay network.

More importantly, caching can protect the directory service from DDoS attacks by reducing the service's workload while under attack. For example, without caching, a flood of packets with the same session identifier will cause multiple identical lookups. If lookups for the same session are handled by the same directory service node, the packet flood may disable the directory service node. If that directory service node is also an anycast proxy, hosts in that anycast proxy's catchment will be unable to reach any anycast groups. With caching, the directory service's workload is reduced to an occasional lookup per anycast proxy. For more discussion on possible DDoS attacks on the directory service, see section 4.2.

Cache lifetime is controlled by two variables. The first is a time limit that indicates the maximum amount of time a cache entry should be considered valid. This guarantees a level of freshness and, like the directory service mapping lifetime, provides an automatic revocation mechanism in case the group member is somehow unable to instruct the anycast proxy to void the cache entry. The second cache lifetime variable is a byte limit that indicates the number of bytes that can be received before the cache entry is considered invalid. Each time a session packet arrives at an anycast proxy, the byte limit stored in the cache entry is decremented by the size of the packet. Once this count reaches zero, future incoming packets are dropped. Together, the cache expiration time and byte limit allow the group member to enforce a per-proxy bandwidth limit.

If there are many anycast proxies, a distributed attacker can stay below the per-proxy bandwidth limit and still source a substantial amount of traffic. To combat this, group members can take advantage of the fact that a valid session's traffic will typically enter through only one proxy thanks to the stability of BGP routes on the Internet. A group member can have a very limited cache lifetime in its directory service mapping but give the proxy used by the session an extended lifetime by directly sending cache updates to the proxy. These updates can either have a long lifetime and be sent occasionally or have a short lifetime and be sent frequently to keep the byte count from reaching zero. The updates can be piggybacked on outgoing packets to keep the overhead down. If the session turns malicious and tries to flood the group member from distributed sources, the traffic from other proxies will be very limited. If the session stays legitimate but the session traffic shifts to an alternate proxy, the session may be forced to slow down until the group member can update the new proxy's cache.

When a proxy's cache is full, it must start replacing cache entries. Proxies should implement a cache replacement algorithm that minimizes the amount of directory service queries. Sessions with a low packet rate should be purged first. With potentially malicious sessions, calculating the expected packet rate is not easy. An attacker could either source more packets than expected or attempt to fill the cache with rarely used

cache entries to force legitimate sessions to invoke frequent directory service lookups. To prevent such attacks, proxies should base expected packet rate on recent history and discount new session entries.

Caching is helpful only when packets belonging to the same session are likely to arrive at the same small set of proxies. If each packet of a session arrives at a different proxy, caching provides no benefit. On the Internet, BGP route stability ensures that packets will arrive at the same proxy unless there is some dynamic event such as remote host movement or a link failure. Because of this stability, caching is likely to be very effective on the Internet.

3.10.4 Managing Directory Service Entries

Inserting, updating, and removing directory service mappings is the responsibility of the group members. This frees the proxies from tracking session states and gives the most control to the group members, making Stateful Anycast more in-line with the end-to-end arguments [67].

Compared to the anycast proxies, group members have more session knowledge, so giving group members the ability to set session mappings provides the greatest amount of flexibility. For example, a session may be composed of multiple sub-sessions, and only the group member may be aware of the association between the sub-sessions. Also, with control over mapping in their hands, group members are able to migrate sessions to other group members by transferring a session's state and updating the corresponding directory service mapping.

Group members should wait to insert a mapping until after the session is fully established. Fully establishing a session usually requires at least one round trip. Waiting for the round trip makes it harder for an attacker to establish a session using a spoofed source address, which in turn makes it harder for the attacker to consume state in the directory service.

3.10.5 Directory Service Implementation

So far, discussion about the directory service has been quite general, leaving implementation details largely unstated. However, the choice of directory service implementation can significantly impact Stateful Anycast's ability to mitigate DDoS attacks. Directory service queries and mapping inserts are triggered by the actions of untrusted hosts, so these functions must be resilient to DDoS attack.

To minimize vulnerability, the directory service is implemented on a structured peer-to-peer overlay network. The mappings are stored in a distributed hash table implemented by the overlay network. In some ways, the directory service is similar to *i*3: mappings are like triggers in that they store indirection instructions, and packets are routed through the directory service. However, there are a few key differences: mappings have cache metadata not present in *i*3 triggers, and the contents of a mapping are returned to a querying host to be cached.

Each overlay node is responsible for a subset of the session identifier space. Mappings in a node's subset are stored at the node, and queries are resolved at the node. Each directory service message (which could be a query, an insert, or a remove) is routed through the overlay network until it reaches the node responsible for the session ID contained in the message. Repeated queries for the same session identifier will be routed to the same node, so caching should be employed at anycast proxies to prevent DDoS attacks on directory service nodes. In addition, a directory service node's session identifier subset should be kept secret to prevent an attacker from directing a large collection of unique queries to the same node. This can be accomplished by passing each session identifier through a secret one-way function and using the output as the distributed hash table key.

The anycast proxies comprise the overlay network nodes. Because anycast proxies are likely to be stable members of the overlay network, overhead from maintenance traffic will be low even if the chosen routing algorithm results in high-degree nodes. Thus, it is advantageous to choose a routing algorithm that minimizes network diameter because this also minimizes the amount of work that must be done to resolve a query or to insert, update, or remove a mapping. A simple routing algorithm is to configure each overlay node to be aware of every other node such that queries or other messages can be sent directly to the destination node in a single hop ($O(n)$ degree, $O(1)$ diameter).

3.11 Encapsulating Packets

Incoming anycast packets and outbound reply packets are passed between the group members, directory service, and anycast proxies by taking the complete, unmodified packet data and placing it in the payload of a unicast packet. The source IP address, destination IP address, and all other IP headers of the original packet are left unmodified. This encapsulation provides the greatest protocol support and does not require the workarounds needed for network address translation [22]. Packet encapsulation is also referred to as *tunneling*, where the host that encapsulates the packets and the host that unwraps the encapsulated packets are the two ends of the tunnel.

Encapsulation also provides an easy way to include in-band metadata. This data is included alongside the embedded packet. Figure 3.19 contains an illustration of an encapsulated incoming TCP SYN packet.

The metadata can be anything that is helpful to the party receiving the encapsulated packets. Helpful metadata might include:

- Cryptographic data. This can include a digital signature to ensure data integrity and to verify packet authenticity or information needed to negotiate encryption keys.
- A timestamp used to calculate the packet's delay between the two ends of the tunnel.
- A sequence number to identify packet loss between the two ends of the tunnel.

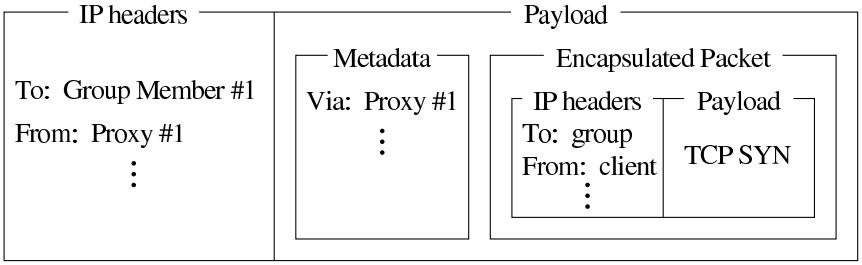

Figure 3.19: Illustration of an encapsulated incoming TCP SYN packet.

- Reverse-path performance information. Statistics regarding the delays and losses experienced by packets sent from a proxy to a group member can be reflected back to the proxy to help the proxy choose the best group member for new sessions.
- Incoming proxy identity. A group member needs to know which proxy received an incoming anycast packet to help direct replies and to help identify distributed attacks. This information is not in the encapsulating packet's source address if the packet is routed through the directory service.
- Cache information. A group member may wish to ask a proxy to change a cache entry by extending the lifetime, altering the destination, purging the entry, and so on. An anycast proxy may wish to inform a group member of hit/miss statistics or a cache entry's remaining lifetime.
- Group member workload information. A group member may wish to inform a proxy of its current workload to help the proxy balance load among group members. A proxy may wish to tell a group member about the average group workload to help the group member identify whether its workload is unusually high.
- Acknowledgments of requested actions. Proxies should confirm requests to update cache entries.

Chapter 4

Design Evaluation

This chapter discusses how well Stateful Anycast meets the design objectives outlined in section 3.1.

4.1 Support for stateful sessions

Stateful Anycast provides remote users with a handle on a particular group member. So long as group members use a supported protocol, Stateful Anycast guarantees best-effort delivery to the proper stateholder. Delivery is not conditioned on route stability as in normal anycast.

The support for stateful sessions does not need to be placed in the proxies. Matching a packet to a group member could be the sole responsibility of the group. For example, group members could run their own private directory service. Indeed, pushing responsibility for stateful sessions to the group would be more in-line with the End-to-End Arguments [67]. Because of this, a Stateful Anycast deployment should give groups the option to handle stateful sessions on their own. However, there are at least three reasons for implementing stateful support at the proxies. First, it simplifies the deployment of a new anycast group. Second, it allows for the efficient reuse of functionality. Finally, it allows for unwanted traffic to be dropped earlier.

Some argue that regular IP anycast is good enough to support stateful sessions [38]. There is a large class of applications that are adequately served by IP anycast, namely applications whose sessions are short-lived relative to topology changes or whose sessions easily recover from a sudden loss of server-side state. Typical web browsing is one such application. An HTTP page or image download happens relatively quickly compared to the current rate of changes in routing or from mobility. In addition, most web session state is stored in the web client in the form of cookies or specially-formed URLs. Stateful Anycast is most suitable for applications with long-lived sessions that do not easily recover from a sudden loss of server-side state. Such applications often have real-time demands or require substantial server-side state. Examples include streaming media and complex multiplayer games. As mobility, wireless communications, and

advanced routing become more prevalent, applications will become increasingly long-lived relative to topology changes, making stateful support important for a broader range of applications.

4.2 DDoS Mitigation

Anycast diffuses DDoS attack traffic among multiple hosts, and IP anycast chooses the destination hosts based on topological closeness. Attack traffic is therefore sunk early rather than after a cross-network journey. This minimizes the work done by the network during a DDoS attack.

Stateful Anycast adds to IP anycast by providing remote users a handle on a group member. However, this allows malicious users to end-run anycast's natural closest-member diffusion. To counter this while still providing legitimate users with a handle on a stateholder, Stateful Anycast allows group members to revoke a sender's ability to direct traffic to a specific host. Once a session is revoked, attack traffic reverts to being diffused among distributed traffic sinks: the traffic is either dropped at the proxies or processed at various group members. In essense, Stateful Anycast embeds authorization for a specific destination inside the network.

However, Stateful Anycast depends on an attacker's inability to get an irrevocable handle on a group member or anycast proxy. Specifically, attackers must not be able to determine the unicast IP addresses of proxies or group members. Once this information is exposed, the proxy or group member is constantly at risk of attack. This is arguably the weakest link in Stateful Anycast, as an attacker observing the unicast traffic between proxies and group members could become a considerable threat. However, unlike Akamai's service and the mobile-IP–based anycast service proposed by Szymaniak, et al. [71], the Stateful Anycast design prevents the wholesale gathering of unicast addresses.

There are two possible ways to mitigate an attack on an exposed proxy or group member, neither of which is ideal. First, the exposed node could simply move to an alternate attachment point. This is not an easy task, especially for a proxy. Depending on how quickly an attacker can discover the new location, moving a proxy or group member can easily escalate into a costly game of hide–and–seek. Alternatively, filters can be deployed in the nearby routers. These filters would drop all traffic destined to the exposed node's unicast IP address except for Stateful Anycast traffic (encapsulated packets, directory service queries, etc.). This method is most effective when the filter rules are installed close to the traffic source, but that is not easy with a well-distributed attacker. Distributing filters to nearby routers may be enough to substantially weaken an attacker.

Besides discovering the unicast IP addresses, there are numerous other possible attacks. The remainder of this section informally lists many of the interesting attacks along with their possible defenses. The analysis assumes that Stateful Anycast is deployed on a network with stable routes (such as the Internet) and that the directory

service is implemented on a structured peer-to-peer overlay network. It also assumes that the adversary is capable of forming arbitrary packets (including spoofing the source address) and sending enough total traffic to saturate the links leading to a proxy or group member. The attacker's power is limited, however: the amount of traffic sourced in any proxy's catchment is not enough to significantly impact legitimate sessions even if all of the catchment's traffic is directed to a single group member node.

4.2.1 Attacks on group members

- **Shared session ID**

  The attack: An attacker establishes a session with a group member. Once the directory service mapping is added, the session identifier is shared with the distributed malicious sources. The attacker then instructs the sources to flood the group using packets containing the shared session ID.

  Effects: All of the attack traffic is delivered to a single group member.

  Defenses: The group member revokes the session by changing the mapping destination to "drop" in the directory service and in anycast proxy caches.

  Required capabilities: The victim must be able to detect that it is under attack and be able to contact the directory service and proxies while under attack.

- **Repeated shared session ID**

  The attack: An attacker repeatedly opens a new session, shares the session ID, and instructs the distributed sources to flood using the session ID. The new sessions are always started from the same node. The attacker attempts to start a new session before the victim can react to the old session.

  Effects: If the proxies consistently direct a given source's new sessions to the same group member, the victim may be continually subject to all of the attack traffic. Even if a group member revokes the session being used to attack, the attacker migrates to an alternate session ID.

  Defenses: Proxies can assign new sessions based on workload to make sure that a new session is not given to a group member that is under attack. In addition, group members can refuse sessions from hosts that have attacked in the recent past.

  Required capabilities: The victim must be able to detect that it is under attack and revoke sessions while under attack. To temporarily refuse sessions from a source that has attacked in the recent past, the group member must be able to determine that the new attack sessions are coming from the same source.

- **Shared collection of session IDs**

  The attack: A single malicious source opens as many concurrent sessions as possible. The source then gives each distributed attacker a different session ID to flood.

  Effects: If the anycast proxies consistently direct a given source's new sessions to the same group member, the group member will be flooded with all of the

attack traffic. The amount of per-session traffic may be low, but the total amount of traffic will be high.

Defenses: Once the victim detects that it is under attack, it identifies the malicious sessions by examining the traffic per session initiator (rather than the traffic per session). All of the sessions that were started by the same malicious node are then revoked.

Required capabilities: The victim must be able to detect that it is under attack and revoke sessions while under attack. It must also be able to identify the true originator of a session. Identifying the originator may be difficult when packets can be spoofed, but new sessions could require at least one round trip to make it difficult for an attacker to start a session when spoofing.

4.2.2 Attacks on the directory service

- **Random, bogus session IDs**

  The attack: Malicious senders source many small packets with random, bogus session IDs.

  Effects: Every packet becomes a query. If query-and-forward is used, the query packets are larger than the incoming packets due to encapsulation and query overhead. If the packets are small enough, the overhead could be relatively large and become a significant load amplifier. For example, a 28-byte IPv4 UDP packet becomes at least a 48-byte IPv4 packet (probably much larger). These query packets are handed to the directory service peer-to-peer overlay network where they are forwarded to various directory service nodes, many of which are likely to be topologically distant. If the diameter of the directory service overlay network is greater than one, the packets can end up bouncing back and forth across the Internet. While the attack traffic is still diffused, the burden placed on the network could be heavy enough to impair the directory service and cause some collateral damage.

  Defenses: It is particularly difficult to defend against this attack. Two possible approaches:

  1. The proxies could perform normal lookups rather than query-and-forward lookups in an attempt to reduce traffic, but normal queries require substantial overhead of their own in the form of reply packets, bringing their defensive utility into question.
  2. The directory service could temporarily increase mapping redundancy so that proxies are less likely to require a query to a distant directory service node. Increasing redundancy comes with a cost: extra traffic will be required to insert, update, and remove the redundant mappings. Automated redundancy tuning would be required to maintain the minimal directory service load under varying attack loads. The effectiveness of increased redundancy depends on the relative rate of directory service changes.

Required capabilities: To automatically tune the mapping redundancy, directory service nodes would need to be able to gather load statistics, calculate the replication sweet spot, and coordinate the change to a different level of replication.

- **Rapid session setup**

  The attack: Malicious sources rapidly establish as many sessions as possible.

  Effects: The bogus sessions consume directory service state.

  Defenses: Sessions that never become active after they are established should be terminated after a timeout period or when the group's overall state consumption is high. In addition, if the first several packets of a session are likely to arrive at the same proxy (due to topology and routing stability), a group member could consider delaying the insertion of a mapping into the directory service until the session sees sufficient activity. At first, the mapping would only be in the initial proxy's cache, and the group member would rely on packets arriving at that same proxy until the mapping is inserted into the directory service. This defense mechanism would disrupt legitimate sessions if the legitimate sender's packets arrive at a different proxy early in the session.

  While this attack seems straightforward to defend against, it is a significant problem due to a security limitation in TCP. One incoming SYN packet is all that is required to cause a TCP server to establish session state. This exposes the server to a resource-consuming DoS attack via a SYN flood [21]. When only a single packet is required to establish state, a flood of state-generating packets can be created with minimal effort. In addition, the attacker's location can be masked by spoofed source addresses, making it difficult to filter the attack traffic. In Stateful Anycast, each TCP SYN would also result in a directory service insert.

  SYN cookies [18] were developed to prevent SYN flood attacks in direct client-to-server TCP sessions. With SYN cookies, servers do not establish state until the client echoes a server-generated cookie. This mechanism makes a successful attack significantly more difficult to launch by requiring the attacker to expose a return address and process three packets (two outbound and one inbound). The additional processing slows down the attack while the exposed return address makes it easier to filter malicious sources. Proxies can be made SYN cookie–aware to prevent TCP SYN flood attacks in Stateful Anycast.

4.2.3 Attacks on the proxies

- **DDoS the last hop before a proxy**

  The attack: The attacker discovers the unicast address of the last router before an anycast proxy (by using the traceroute utility from a particular machine, for example) and directs a flood of traffic to that address.

  Effects: Links leading to the router are saturated with malicious traffic. The nearby anycast proxy is rendered unreachable due to its proximity to the unreachable router.

Defenses: To help prevent an attacker from discovering the unicast addresses of routers near anycast proxies, the routers should not send ICMP time exceeded messages [59] in response to packets with a time-to-live field equal to zero. If a unicast address of a router near a proxy is discovered, one recourse is to deploy filters in all of the routers in the surrounding region. These filters would block all traffic addressed to the victim router except for necessary management traffic.

Required capabilities: Deploying filters in routers requires filter support in the routers, possible coordination with other network operators, and the ability to distinguish valid management traffic from spoofed management traffic.

4.2.4 Other attacks

One class of attacks not mentioned above is insider attacks. Stateful Anycast assumes a trusted relationship between the anycast proxies and the group members. While the insider threat is important—especially considering the possibility of an attacker compromising a group member and using it to collect information and launch attacks—an analysis is not provided here.

Many other attacks not mentioned above can be easily thwarted through appropriate use of existing Stateful Anycast mechanisms. Some attacks, however, may require additional mechanisms not provided by Stateful Anycast. Due to the anycast proxies' advantageous location between the two communicating ends, as well as Stateful Anycast's broad protocol support, many other defensive technologies can easily complement Stateful Anycast.

4.3 Scalability

This section discusses how well Stateful Anycast would scale on a number of dimensions.

4.3.1 Scalability in the number of groups

Unlike IP anycast, Stateful Anycast can support a large number of anycast groups. Stateful Anycast inherits this scalability from PIAS, which scales by moving the burden of supporting multiple groups off of BGP routers and on to the anycast proxies where they can be effectively managed.

The anycast proxies take the scaling burden off of the BGP routers by requiring a single routing table entry for an arbitrary number of groups, assuming the proxies only advertise a single block of IP addresses. No matter how many IP addresses are in the advertised block, only one routing table entry is consumed. Routers only need to know how to reach one anycast proxy in order to reach every group supported by the anycast proxy. With regular IP anycast, routers must know paths to members from every group—every group consumes a routing table entry.

While the proxies relieve the routing infrastructure from the burden of supporting many groups, they must support the groups themselves. This is an easily dividable prob-

lem because not all proxies need to know about every group (the groups are independent of each other). The task of managing groups can be handled in a divide-and-conquer manner by multiple physical machines, each capable of handling a subset. As the number of groups grows to exceed the management capabilities of an anycast proxy, the anycast service operator adds another physical machine to each network attachment point and reallocates group management responsibilities. The router connecting the proxies to the rest of the Internet is then configured to forward incoming packets to the relevant machines by entering static routes or by using a local routing protocol such as OSPF [49].

As an example (illustrated in figure 4.1), suppose an anycast service provider wishes to support 8192 groups occupying the IP address block 172.16.0.0/19[1]. If a physical anycast proxy machine is only capable of handling 5000 groups, two proxies can be placed behind the attachment-point router. Proxy #1 would handle the 4096 groups in the block 172.16.0.0/20 (the lower half of 172.16.0.0/19) while proxy #2 would handle the 4096 groups in the block 172.16.16.0/20 (the upper half of 172.16.0.0/19). The router would direct only the packets destined to an address in 172.16.0.0/20 to proxy #1 while packets destined to 172.16.16.0/20 would only go to proxy #2.

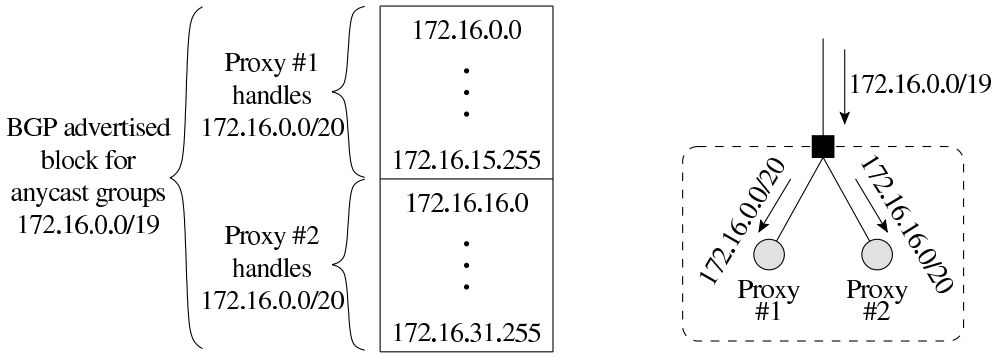

Figure 4.1: Scaling Stateful Anycast to support a large number of groups. A large block of IP addresses is advertised to the Internet but each proxy handles a fraction.

Responsibility for groups does not have to be divided evenly; an anycast service provider can arbitrarily divide the groups among the proxies (to give more groups to newer hardware, for example).

The burden a group imposes on an anycast proxy may be reduced by increasing the number of proxies that are responsible for the group. This is done by deploying proxies at new network attachment points. The new proxies add additional catchments,

[1]172.16.0.0/19 refers to the block of IPv4 addresses that have the same first 19 bits as the IP address 172.16.0.0 (172.16.0.0 through 172.16.31.255 inclusive). Because IPv4 addresses contain 32 bits, the number of IP addresses in a /19 block is $2^{32-19} = 2^{13} = 8192$. The ⟨address⟩/⟨bits⟩ notation is called *CIDR notation*.

which reduces the size of other catchments. Smaller catchments should result in fewer sessions and less overall work for each proxy, allowing them to support more groups.

4.3.2 Scalability in the number of group members per group

If anycast proxies only forwarded packets belonging to established stateful sessions, they would never need to maintain long-term state about the location and status of group members. Instead, the directory service would inform the proxies about destination group members on an as-needed basis. However, to forward stateless packets or packets belonging to a bootstrapping stateful session, proxies must be aware of the location of a good group member (where "good" is defined by operator-specified metrics such as workload, latency, packet loss, topological closeness, available bandwidth, etc.). Thus, each proxy must be made aware of at least one available group member in each group.

Determining which group members should be known to a particular anycast proxy is an implementation choice. To provide the most information for choosing a good group member, each proxy should be made aware of every group member. However, this approach does not scale as the number of proxies and group members increases. To manage large deployments, proxies should be made aware of a subset of the group members. By manually configuring each group member to associate itself with specific proxies, a group operator can manually tune application responsiveness and load distribution. One automated method is to have each group member register with its closest proxy (by using the group of proxies' anycast address). This information could then be shared with proxies in neighboring catchments to ensure that each proxy has a good number of group members from which to choose. Sophisticated distributed algorithms for automatically associating group members with proxies to maximize some desired performance metric is a possible subject for future research.

If the desired group member–proxy assignment still results in too many group members per proxy, the number of proxies must be increased. This can be done in the two ways mentioned previously: add more proxies to busy attachment points and divide responsibility for the groups among the proxies, or deploy proxies to new attachment points.

4.3.3 Scalability in group membership dynamics

The load a group imposes on the proxies includes the work needed to associate or disassociate group members with proxies as the group members come on-line or go off-line. The more frequently the group membership changes, the greater the computational and communications needs of the proxies. Frequent membership change thus reduces the number of group members (and by extension the number of groups) that can be supported by a proxy. The burden of frequent membership changes is managed in the same way as too many groups or too many group members: add more proxies to busy attachment points or install proxies at new attachment points.

4.3.4 Scalability in the number of anycast proxies

BGP routers only need to be aware of a path to a single anycast proxy. While routers may receive advertisements from multiple proxies, they can safely ignore the non-optimal choices. Deploying a large number of anycast proxies on the Internet does not impose any larger of a state-keeping or message-passing burden on the routers than a single anycast proxy would.

However, because the directory service is implemented on the proxies, the structured peer-to-peer overlay used to create the directory service must be able to support all of the proxies. Structured peer-to-peer overlay networks are capable of growing to millions of nodes, but there is a trade-off. Minimizing the number of hops required to reach other nodes requires each node to keep track of many other nodes. While the state required to keep track of millions of nodes is modest with modern hardware, the traffic required to update node adjacency state is substantial when nodes frequently join and leave. If anycast proxies are designed for high availability, the join and leave frequency should be very low. This would permit an overlay configuration that requires one or a few hops to reach any other node and would increase the performance of the directory service.

4.3.5 Scalability in the rate of change in anycast proxy availability

Unfortunately, Stateful Anycast does not handle frequent changes in anycast proxy availability very well. A proxy going up or down has two effects: routes must be recalculated and the directory service has to reallocate which nodes have responsibility for which parts of the session ID space.

If an anycast proxy is added or removed, the routing infrastructure must recalculate routes. If there is only one proxy at an attachment point and its availability changes, routers throughout the proxy's catchment must recalculate paths. With BGP's slow convergence, frequent proxy availability changes can render groups unreachable an unacceptable amount of the time. To minimize service disruption, care must be taken to minimize downtime. One approach is to deploy multiple proxies at every attachment point. If one of the proxies goes down or comes up, route calculation only occurs at the attachment point's router. The local route adjustment is nearly instantaneous and has minimal impact on service availability.

Proxy availability also affects the directory service. Each time a proxy goes down or comes up, responsibility for the directory service state must be reallocated among the proxies. If the overlay is well-connected to minimize the number of hops required to reach another node, proxy availability changes can result in significant management traffic.

4.3.6 Scalability in the number of stateful sessions

Mappings for established stateful sessions must be stored in the directory service. For good performance and to protect against DDoS attacks, the mappings should also be

cached at anycast proxies. Since directory service nodes are also anycast proxies, this means that each proxy stores about twice the number of mappings as the number of sessions it supports. Supporting directory service data redundancy for failover or performance reasons requires this number to be even higher.

If each proxy supports thousands of group members and each group member has thousands of concurrent sessions, tens of millions to hundreds of millions of mappings may need to be stored and managed at each anycast proxy. This state-keeping burden requires carefully chosen data structures and a substantial amount of memory per proxy in order to provide reasonable packet forwarding performance.

If the number of stateful sessions becomes too numerous for the proxies to manage, there are at least three ways to relieve the burden. Two of the ways are the same as handling too many groups or group members: add more proxies to busy attachment points or install proxies at new attachment points. The additional proxies result in fewer mappings per proxy.

The third approach is to move the directory service onto the group members themselves. Each group would implement its own distributed directory service that would support the group's members. The directory service software could be easily integrated into the software that already resides on each group member. The group members already must support some state per session; pushing the directory service onto the groups would simply require a bit more state per session. One disadvantage to this approach is the extra hop required to resolve a query. Another is that it may be easier for an attacker to disrupt a group due to a limited distribution of directory service state: A bombardment of traffic with bogus session identifiers would not be stopped at the proxies—possibly disastrous for a small group.

The amount of traffic overhead to maintain cache entries is minor if there is adequate memory and if group members include caching information in outbound reply packets. A proxy should only need to query the directory service if a session moves.

4.3.7 Scalability in session dynamics

As sessions start and end, the directory service and proxy caches must be updated. Updating the directory service first requires routing the insert or removal request through the overlay network to the directory service node that is responsible for the session ID. Since the responsible node could be anywhere on the Internet, this could be an expensive operation in terms of network effort. The operation will be even more expensive if multiple directory service nodes must be contacted due to redundancy requirements. However, as long as the update packets are infrequent relative to session packets, the overhead will not be significant.

The challenge is to ensure that the update packets are rare compared to session packets. This is especially important if an attacker is maliciously starting and stopping sessions as rapidly as possible. One approach is to rely on IP anycast's consistent delivery to a particular proxy and delay inserting the mapping into the directory service until the remote end has done a sufficient amount of work. This could be after a given

number of packets or after the remote end has solved a CAPTCHA [9] or computational puzzle [33].

The mapping also has to be inserted into the local data structure in both the responsible directory service node and in the anycast proxies that receive the session's packets. As discussed above, the number of mappings each proxy must store can be very large, requiring careful data structure design to accommodate frequent updates.

If session updates are simply too frequent for the proxy hardware to handle, the workload can be managed by throwing more hardware at the problem: either add more proxies to busy attachment points or install proxies at new attachment points. With more proxies, each proxy will process fewer updates.

4.4 Deployability

Deploying Stateful Anycast is just like deploying any other IP anycast group. The careful planning and negotiating involved in installing a proxy at an new attachment point is not easy, but cost is amortized across all of the groups and the benefits are shared.

Operational costs are somewhat higher than running a simple anycast group. Anycast proxy hardware needs to be maintained in addition to the group member hardware. Network connectivity is paid for twice—once by the anycast service provider and once by the by the group operator. However, these extra costs are also largely shared among all of the groups.

Almost everything is incrementally deployable. Proxies, groups, and group members can all be added one at a time. Even a small deployment can considerably increase the amount of malicious traffic that can be sunk. Developing the Stateful Anycast software and acquiring a large block of IP addresses are the only major deployment tasks that cannot be done incrementally.

Deploying a Stateful Anycast group is much simpler than deploying an IP anycast group. All that is required to attach a group member is an ordinary Internet connection. For optimal performance, the group members should be topologically close to the proxies, but if load distribution is all that is required, placement is not very important.

Stateful Anycast unfortunately does not support uncoordinated deployment. Due to the trust required between the group members and the anycast proxies, individuals cannot independently contribute their own anycast proxies. In addition, group operators must obtain permission from the Stateful Anycast service provider before they can deploy the group members.

Although Stateful Anycast does not support uncoordinated deployment, there is some incentive for deploying a Stateful Anycast service in the form of a business opportunity. The amortized cost of deploying and running the proxies combined with the cost of operating the group members can be significantly lower than deploying a normal IP anycast group. The hard work of developing software to support stateful sessions, locating good attachment points, negotiating contracts with network operators,

acquiring a large enough block of IP addresses to prevent filtering by other network operators, and deploying BGP routers only needs to be done once for all groups. In addition, the anycast proxies can serve as a platform for deploying complementary services such as simple multihoming, high performance content distribution, managed security services, and reliable virtual private networks.

4.5 Protocol Generality

While Stateful Anycast was designed to be as general as possible, it does not support all protocols. This section discusses some of the limitations and requirements imposed on protocols by Stateful Anycast.

The primary requirement is that a session identifier must be present in every packet. Proxies cannot determine the correct session if there is no identifier in a packet.

In addition, Stateful Anycast proxies must be able to extract the session identifier without knowing any session state. Because a session's packets can arrive at any proxy, proxies are not guaranteed to have any historical data about the session. This includes encryption keys that have been negotiated at the start of the session: proxies will be unable to identify the session if the identifier is part of the encrypted data. Mechanisms to distribute session state to proxies would likely suffer from scalability limitations and DDoS vulnerabilities.

New protocols are not automatically supported by anycast proxies. The proxies must be programmed to decode the protocols used by the groups they support. If a group wishes to use a new protocol—or a new version of an old protocol—the Stateful Anycast operator must update the proxy software to enable the extraction of the session identifier. While this is not a particularly challenging task, this lack of generality may discourage the adoption of newer protocols and instead encourage stagnation of protocol usage.

High-layer stateful sessions that are composed of multiple lower-layer sessions pose a challenge to Stateful Anycast. Proxies must be able to direct all lower-layer sessions to the same group member. Support for these kinds of protocols is largely a topic for future research, but an initial analysis suggests a couple of possible approaches. One possible approach is to simply limit support to those higher-layer protocols that include the higher-layer session identifier in every packet. Proxies would then be programmed to search for a higher-layer session identifier in the payload of a lower-layer protocol. This will exclude some important protocols, such as the File Transfer Protocol (FTP) [62]. FTP requires multiple TCP sessions—one for control and one for each file transfer—yet does not include a session identifier in the payload of each TCP packet.

An alternate approach for supporting higher-layer protocols whose sessions are composed of multiple lower-layer sessions is to require group members to insert mappings for each lower-layer session. The higher-layer protocol must allow a group member to either discover the lower-layer session identifiers before the lower-layer sessions begin or choose each lower-layer session identifier. The proxies would be required to treat even

the start-of-session packets as potentially belonging to an existing stateful session: If they resolve, send them to the proper group member. If not, send them to any group member.

For maximal protection against DoS attacks, protocols used by group members should require a round trip before any state is established in the group member. This prevents SYN-flood–like attacks and enables group members to easily filter sources that are trying to rapidly establish sessions (see the "rapid session setup" attack described in section 4.2 for more details). If cookies are used to require a round trip before establishing state, the cookie generation and verification algorithms must be the same for all group members. In addition, proxies should be able to distinguish return cookies from other types of session data so that they know to forward the return cookie to any group member rather than issue a directory service query (which would fail).

Even with the above restrictions, most common protocols can be used with Stateful Anycast without modification. Any stateful protocol that uses a single TCP session is easily supported (as long as a lower-layer encryption protocol such as IPsec is not used) by using the source and destination ports and IP addresses as the session identifier. Stateful protocols that use UDP can be supported in a similar way. FTP can be supported by instructing the proxies to treat TCP SYN packets as possibly belonging to a stateful session and inserting an additional mapping for the data session when a PASV or PORT command is issued.

Protocol support would be improved by adding a generic session ID or flow ID field to the IP headers. Senders would choose a unique value for this field for each session to help anycast proxies and other middleboxes route packets to the proper destination.

Network architectures that route packets based on an ephemeral, recipient-chosen identifier would also make Stateful Anycast protocol-independent. *i*3, with its support for private triggers, is one example of such an architecture. Host identity protocols such as HIP [48] could also be used, assuming a mechanism exists for redirecting a session to a newly generated (and revocable) identity.

4.6 Performance and Efficiency

Stateful Anycast imposes a packet performance penalty from a few different sources: longer packet traversal paths, processing delay at the anycast proxy, directory service delays, and encapsulation and metadata overhead. However, Stateful Anycast may provide better overall performance when compared to traditional single-host communications. Under certain conditions, Stateful Anycast may even outperform normal IP anycast. This section discusses how Stateful Anycast might affect packet forwarding performance.

A packet must stop at an anycast proxy before it can complete its journey to a group member. This waypoint results in a longer traversal path compared to the direct path to the group member.[2] Longer paths often imply lower path bandwidth due to

[2]An exception is when routing through a waypoint allows a packet to traverse a short path that is

a greater number of possible bottlenecks, higher delay and jitter from sitting in more queues, and greater packet losses from contending for queue space more often. However, because IP anycast generally selects the topologically closest anycast proxy, the two-segment forwarding path should not diverge significantly from the direct forwarding path. Thus, users should see a minimal performance penalty resulting from the longer paths.

Once a packet reaches an anycast proxy, the packet must be processed. Processing delay is highly dependent on trade-offs between cost and performance made in the implementation of an anycast proxy. The impact of processing delay is a possible topic for future study.

Delays caused by directory service queries could have a significant impact. An overlay network's topology is generally independent of the underlying network's topology, sometimes causing long traversal paths between adjacent overlay nodes. Directory service queries may need to traverse great distances before they reach the resolving directory service node. Minimizing the average number of overlay hops required to reach a destination node is important for minimizing the negative performance impact of the directory service. Each overlay node can be configured to be aware of every other overlay node, resulting in single-hop packet delivery. Even with a proxy population of a few hundred thousand, the state-keeping burden of this configuration should be easily manageable. However, with frequent proxy membership changes, the traffic required to maintain this topology state would be very high. Designing the proxies for high reliability should keep topology maintenance overhead small relative to the performance savings provided by single-hop delivery.

Even a single hop in the directory service overlay network can be very long relative to the distance between the sender and the group member. Thus, caching directory service query results is critical for obtaining performance comparable to normal IP anycast. With adequate cache sizes, caching should result in directory service queries only when network topology dynamics causes an incoming packet to arrive at a different proxy.

Encapsulation and metadata overhead provides additional performance penalties. With small packets—common in real-time applications such as multiplayer games—the overhead can easily reach 100%. Per-packet overhead for encapsulation inside a UDP packet is 28 bytes (20 for IP headers, 8 for UDP headers) plus whatever else is required for digital signatures, proxy identification, group member load, cache update instructions, etc. This overhead can significantly cut goodput over bandwidth-limited links. For large packets, the overhead is not as significant. However, if the size of the original packet is close to the maximum transmission unit of a link, the extra overhead will cause packet fragmentation. The excess packets will hurt goodput over packet-rate–limited links.

Even with all of the above performance limitations, Stateful Anycast may provide a significant application performance improvement compared to deployment at a sin-

otherwise unreachable due to routing policies imposed by network operators.

gle location. Like Akamai's content distribution services, Stateful Anycast can direct packets to nearby idle group members. The short trips to nearby servers reduces packet delay and the intelligently balanced load eliminates hotspots, resulting in improved application responsiveness. Regular IP anycast also reduces packet delay, but is unable to intelligently balance load among group members. This could give Stateful Anycast a performance advantage over regular IP anycast when used with processing-intensive applications.

Chapter 5

Stateful Anycast Software Design

Software for the evaluation of Stateful Anycast is currently being developed. This chapter highlights some of the key features of the software design.

Section 5.2 discusses the group member software and details how packets are passed between the group member software and the Internet application running on the group member. Section 5.3 discusses the anycast proxy software and details how packets for all groups are captured and forwarded to the group members. Section 5.4 discusses how Stateful Anycast can be evaluated without deploying an anycast group. However, before diving into the design of the software, a brief diversion will be made in order to provide background on virtual network interfaces, an operating platform feature utilized in the Stateful Anycast software.

5.1 Background: Virtual Network Interfaces

The idea behind virtual interfaces is simple: when the operating system routes a packet to a virtual network interface, the outgoing packet is passed to a separate userspace program rather than to a hardware device. Packets can also follow the reverse path: The userspace program associated with the virtual interface generates an incoming packet and hands it to the virtual interface driver, which injects the packet into the operating system's networking subsystem to be routed appropriately.

The userspace program associated with a virtual interface emulates a network comprised of zero or more hosts. There are many valuable uses for network emulation: Network simulation programs such as ns-2 [3] emulate a collection of nodes to evaluate distributed algorithms in a mixed simulated/real environment. Virtual Private Network (VPN) software—and other tunneling programs such as the Stateful Anycast group member software—emulate the hosts on the other side of a tunnel by coordinating with the remote tunnel endpoint. The userspace program receives and processes outgoing (from the kernel's perspective) packets and may generate incoming packets.

Figure 5.1 illustrates in greater detail the process of sending and receiving packets through a virtual interface on a Linux machine.

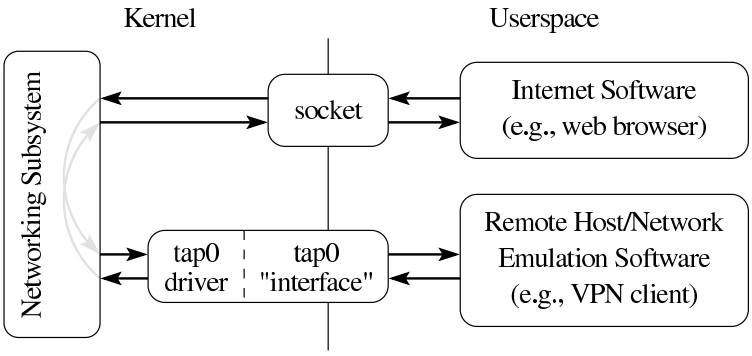

Figure 5.1: Virtual network interface packet paths: Outgoing packets are sourced by some Internet program (a web browser, a web server, etc.) and passed to the kernel via the Berkeley sockets interface (API) [1]. The kernel then passes the packet to its networking subsystem, which, due to configured routing rules, routes the packet to the virtual network interface. The kernel then passes the packet to the virtual network interface driver, where it is made available to a userspace program that is emulating a network (such as a VPN client).

5.2 Group Member Software

Hosts must run specialized software in order to participate in a Stateful Anycast group. This software is responsible for numerous tasks, such as relaying packets, authenticating with the rest of the group and with the proxies, inserting and removing directory service mappings, and detecting when a session has become malicious. The design of the first version of the group member software, intended to be a proof-of-concept, contains only the features necessary to demonstrate the effectiveness of Stateful Anycast.

The group member software's most basic function is to relay packets between the anycast proxies and the server software running on the group member node. How this function is implemented determines the high-level software architecture. Encapsulating and decapsulating packets is architecturally straightforward, as is sending packets to and receiving packets from the anycast proxies. The greater design challenge lies in how packets are passed to and from the server software.

There is a range of approaches to handling the server–group member connection. One extreme is tight integration, where the group member software is also the server software. With this approach, passing an incoming packet to the server software becomes passing the payload memory location to a function. This approach can yield high performance, resulting in a better end-user experience. More importantly, it simplifies the process of combining security information from both the group member and server roles of the software.

The other extreme is complete separation: the group member software abstracts away the anycast proxy communication and provides an ordinary network service to

the server software. All communication between the group member software and the server software would pass through the operating system's kernel as normal IP packets. This strong modularity improves flexibility and provides backwards compatibility with existing server software. However, the abstraction prevents the group member and server software from sharing information that would be valuable in making security decisions.

The group member software design takes the middle ground: Complete separation except for a process to monitor the health of the protected server software. Monitoring processes for new server applications can be implemented as plug-ins to the group member software. These plug-ins should evaluate server and session health by watching server log output, for example. An illustration of the high-level architecture is provided in figure 5.2.

This chosen approach is well-suited for evaluating Stateful Anycast's effectiveness, as software development time is kept at a minimum. Existing server software can be reused to provide a realistic application to run on top of Stateful Anycast, and a manual or scripted monitoring process should suffice for testing and evaluation purposes.

5.3 Anycast Proxy Software

The anycast proxy software's most basic function is to relay packets between the remote users and the group members. Encapsulating/decapsulating packets and communicating with the group members is architecturally straightforward; the design challenge lies in capturing incoming packets from remote users as well as sending outgoing packets to the remote users.

A first-pass approach to capturing incoming packets is to configure the network interface with each and every anycast IP address; the anycast proxy software would then bind to every anycast IP address upon initialization. This approach is somewhat messy (the IP addresses must be assigned to the interface before the anycast proxy software is started) and may not scale to a large number of anycast addresses. In addition, care must be taken to ensure that other applications do not bind to the anycast IP addresses, otherwise they might intercept the packets that should be sent to the group members.

Using one of two alternate approaches—routing to a virtual interface or using a packet capture library—are preferred, as they require less system configuration and do a better job at isolating packets from the rest of the system. These two approaches are discussed in more detail below. Each approach has its advantages and disadvantages, and neither is clearly superior. The packet capture library approach has been implemented; the virtual interface approach may be implemented at a later time.

5.3.1 Remote Client to Anycast Proxy: Virtual Interface Approach

The first approach, illustrated in figure 5.3, is to route all incoming anycast packets to a virtual interface. The anycast proxy software is attached to the "other side" of the virtual interface so that it can read and process the incoming anycast packets.

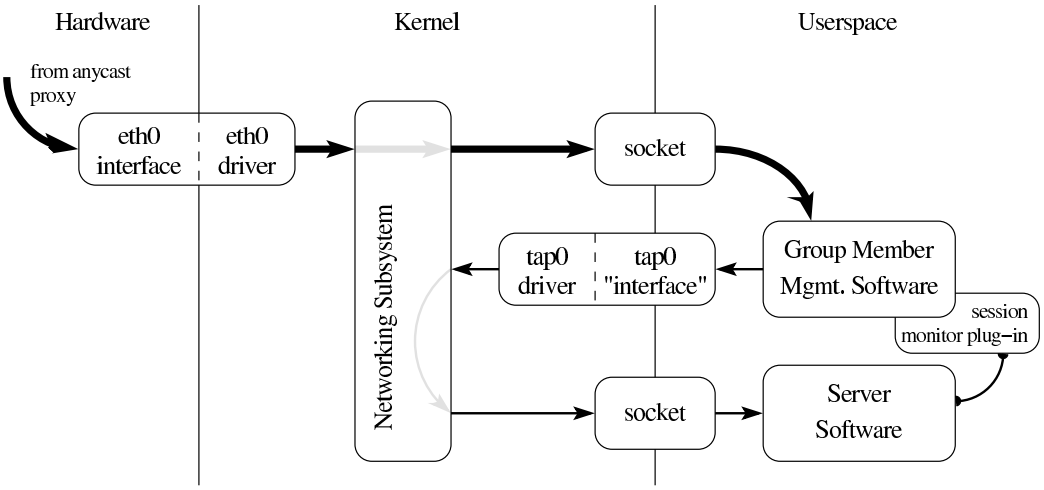

Figure 5.2: Group member software architecture. Compare with figure 5.1. Bold arrows represent paths taken by encapsulating packets, while the thin arrows represent paths taken by unwrapped packets. An incoming encapsulating packet first arrives at the hardware network interface, where it is picked up by the interface driver and passed to the kernel's networking subsystem. The kernel networking subsystem examines the destination IP address (which is the host's unicast IP address) and UDP port number of the encapsulating packet and determines that it should be routed to the group member management software via a socket previously established by the management software. After processing any metadata, the management software extracts the encapsulated packet and passes it to the virtual interface driver. The virtual interface driver places the packet back in the kernel's networking subsystem, making it appear as if the encapsulated packet just arrived at a second hardware network interface. The networking subsystem examines the destination IP address (which is now the anycast group IP address) and TCP or UDP port number of the encapsulated packet and determines that it should be routed to the server software via a socket previously established by the server software. The server receives and processes the packet as normal. Reply packets follow the reverse path. Meanwhile, a plug-in monitors the health of the server and stands ready to revoke malicious sessions.

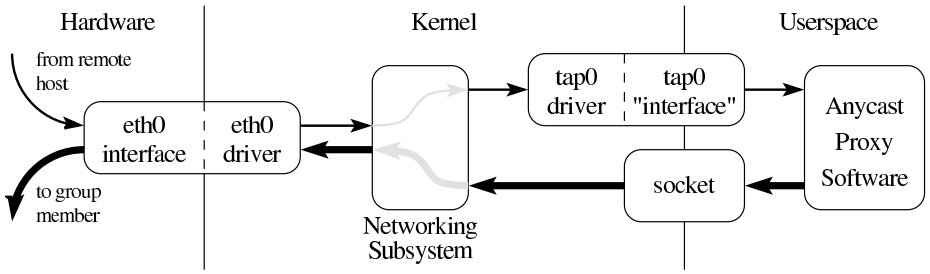

Figure 5.3: Anycast proxy incoming packet path when using the virtual interface approach. Incoming anycast packets arrive at the hardware interface and are passed to the kernel's networking subsystem. Due to preconfigured routing rules, the kernel forwards the packets via the virtual interface, where they are picked up by the anycast proxy software. Once the packets are encapsulated, they are sent out using the standard Berkeley sockets API.

For this approach to work, the upstream router and anycast proxy require configuration, but nothing as involved as assigning each anycast IP address to an interface. First, the upstream router must be configured to forward all traffic destined to an anycast group IP address to the unicast address of the anycast proxy. Second, the anycast proxy must have IP forwarding enabled in the kernel. Third, the anycast proxy must have a route in its routing table that says to send all traffic destined to an anycast IP address out the virtual interface.

The detailed processing steps for an incoming packet are as follows:

1. When an incoming anycast packet arrives at the upstream router, the router sends an ARP query to determine the lower-layer MAC (hardware) address associated with the anycast proxy's unicast IP address.

2. The anycast proxy's operating system receives the ARP request, recognizes that the query IP address matches the IP address assigned to its hardware interface, and responds by sending back an ARP reply containing the MAC address of its hardware interface.

3. The upstream router caches the ARP reply and forwards the incoming anycast packet to the anycast proxy by addressing the link-layer packet to the proxy's MAC address.

4. The anycast proxy's operating system receives the packet and recognizes that it is not destined for local delivery (the destination IP address does not match any of the IP addresses assigned to its interfaces). Because it is not destined for local delivery, the operating system decides to forward the packet.

5. The operating system identifies the best-matching route entry in its routing table and sends the packet out the interface identified in the routing table entry. If the

routing table is configured properly, the outgoing interface should be the virtual interface that leads to the anycast proxy software.

6. The anycast proxy software receives the anycast packet from the virtual interface device driver, processes it appropriately, and forwards it to the proper group member.

When compared with the packet capture library approach, the main advantage is that packets follow the operating system's normal processing path. This enables the use of system-provided packet filtering, simplifies troubleshooting (as normal network troubleshooting tools such as packet sniffers can be easily employed), and possibly provides superior performance. The main disadvantage is that programmatically configuring the virtual interface and manipulating the routing table is difficult to do in a platform-independent way.

Outgoing packets can be sent by writing their contents to the virtual interface driver.

5.3.2 Remote Client to Anycast Proxy: Packet Capture Library Approach

The second approach, illustrated in figure 5.4, is to use a portable packet capture library (such as libpcap [2]) to grab a copy of all packets destined to an anycast group address that arrive at the physical interface. Because the anycast proxy software receives a *copy* of each incoming packet, it is important that the original packets be silently dropped. If the originals are routed back out the system or if ICMP [59] messages are generated in response to incoming anycast packets, communication can be disrupted.

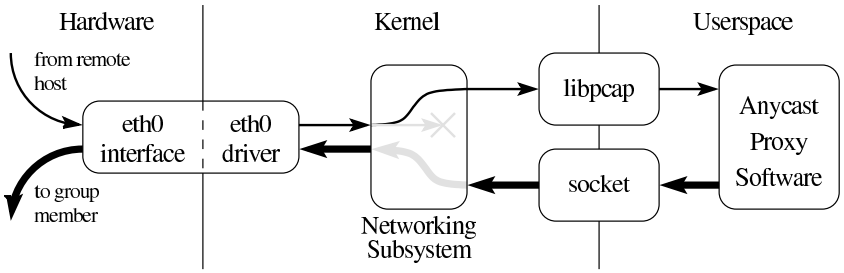

Figure 5.4: Anycast proxy incoming packet path when using the packet capture library approach. Incoming anycast packets arrive at the hardware interface and are passed to the kernel's networking subsystem. A copy of the packets are given to the packet capture library while the originals are dropped.

Getting the operating system to drop the original copy of the anycast packet can be tricky. The simplest way to implement the packet capture approach is to configure the upstream router to forward all anycast packets to the unicast address of the anycast

proxy (just as in the virtual interface approach), turn off forwarding, and use libpcap in non-promiscuous mode to grab a copy of all anycast packets. With this configuration, the host recognizes that the anycast packets are not destined to the local host, so they are simply discarded. However, if forwarding is turned on, the kernel will attempt to forward the packet. If there is no known route to the destination (possibly due to a *reject* route added to the routing table), the host will bounce back an ICMP *destination unreachable* message. If there is a route to the destination but the next hop is back out the interface the packet came in on (which is likely due to a default route match), the host will send an ICMP *redirect* message to the upstream router. If there is a route out a different interface, the operating system will simply pass the packet on to someone else. All of these will likely disrupt communication.

Alternate packet capture method to improve robustness

To improve robustness in the face of an accidental enabling of packet forwarding, an alternative packet capture tactic can be used. This alternate method is more complicated to implement, but results in the kernel dropping the original copy of each incoming anycast packet with certainty.

For this alternate method, the upstream router is configured to believe that all of the anycast IP addresses can be directly reached via the same interface as the anycast proxy (as opposed to being configured to send the packet via the anycast proxy's unicast address). The packet capture library in the anycast proxy is configured to place the interface in *promiscuous mode* (all packets received on the interface—not just those with a destination MAC address that matches—are passed to the kernel). Finally, the anycast proxy software randomly generates a MAC address to be used in all outgoing packets (discussed below).

When an anycast packet arrives at the upstream router, the processing steps are as follows:

1. The router issues an ARP query on the downstream interface to determine the MAC address of the hardware interface configured for the destination anycast IP address.

2. The anycast proxy's operating system receives the ARP query, but ignores it since none of the interfaces are configured to use any of the anycast IP addresses.

3. libpcap grabs a copy of the ARP query and passes it to the anycast proxy software.

4. The anycast proxy software notices that the query is for an anycast address and forms an ARP reply packet containing the randomly-generated MAC address. The reply ARP packet is passed to libpcap, which injects the raw frame into the hardware interface's outgoing queue.

5. The upstream router receives the ARP reply containing the randomly-generated MAC address and forwards the incoming anycast packet to that MAC address.

6. The anycast proxy's operating system receives the anycast packet, even though the destination MAC address does not match the hardware interface's MAC address. This is because the interface was placed in promiscuous mode by libpcap. The operating system simply ignores the anycast packet because the destination MAC address does not match

7. libpcap grabs a copy of the anycast packet and sends it to the anycast proxy software, where it is processed as usual.

The key to this method is to ensure that the MAC address used by the upstream router to forward the anycast packet does not match the MAC address of the anycast proxy's hardware interface. With the differing MAC addresses, the incoming anycast packets will certainly be dropped by the operating system, yet libpcap will still obtain a copy.

The packet capture library approach is certainly more complicated to implement, but it is simpler for the system administrator. Routing tables do not need to be modified, nor do virtual interfaces need configuration—everything is handled by the anycast proxy software. In addition, libpcap can be used on many platforms. Unfortunately, libpcap's design trades packet delay for high throughput and low overhead, making it less suitable for Internet applications that are sensitive to delays or jitter.

Outgoing packets can be sent by injecting raw frames into the hardware interface's outgoing queue using libpcap.

5.4 Testing Stateful Anycast

There are a couple of straightforward ways to test the Stateful Anycast software and design without going to the trouble and expense of deploying an actual anycast group. One method is to use the network emulation capabilities in a network simulator such as ns-2 [3]. Several virtual machines plus the network emulator could be bridged together, with the network emulator moderating communication between the hosts on a scripted or programmatic basis.

The other approach is to deploy nodes on a network research platform such as PlanetLab [56]. Because these nodes are not anycasted, anycast will have to be emulated using tunneling. The test clients will run a proxy selector application that, for every packet, selects the unicast IP address of an anycast proxy from a preconfigured list, encapsulates the packet, and forwards it to the proxy. The proxy runs software to unwrap the encapsulated packet and pass it to the anycast proxy software. An illustration of the process is presented in figure 5.5. Return packets take the reverse path.

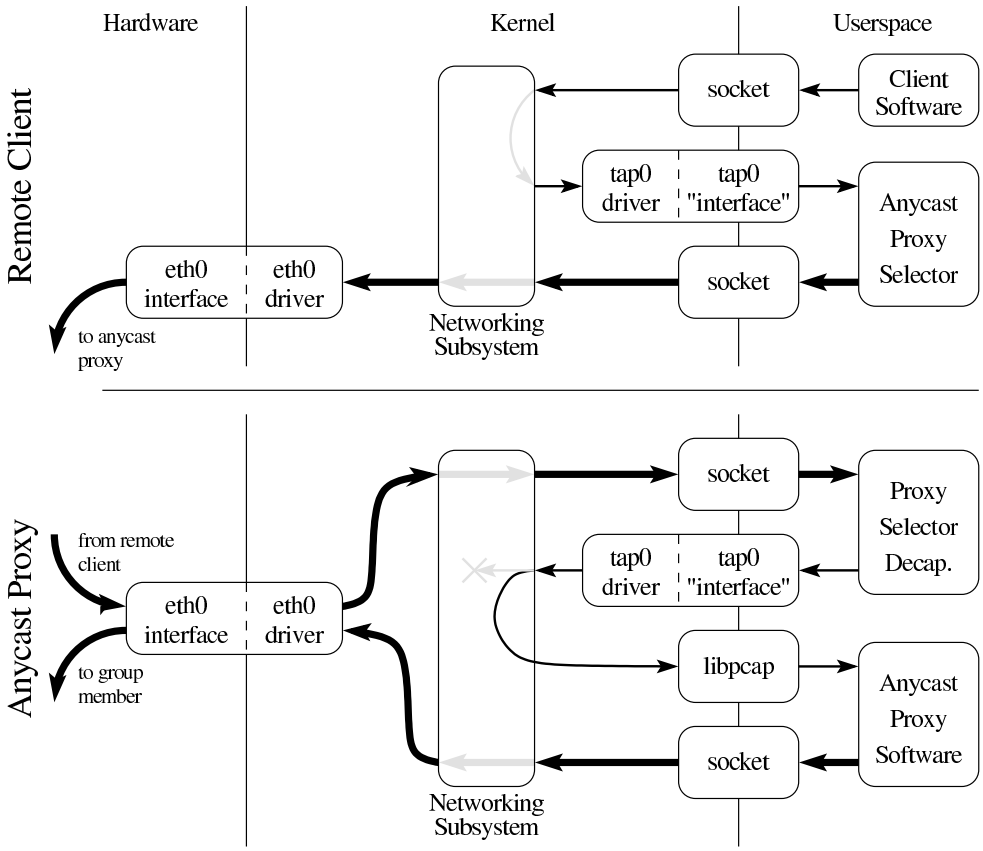

Figure 5.5: Testing Stateful Anycast using a virtualized anycast implemented with tunneling. Applications running on the test client bind to a virtual interface that is connected to an anycast proxy selector program. When the client sends a packet, the proxy selector chooses an anycast proxy, encapsulates the packet, then sends the packet out the hardware interface. When the encapsulated packet arrives at the anycast proxy, extra software there decapsulates the packet and passes it to the anycast proxy software.

Chapter 6

Conclusion

Distributed Denial-of-Service attacks have been plaguing the Internet for some time, with no end in sight. Several mitigation techniques have been proposed, but none have emerged as a clear leader. No existing Internet DDoS mitigation mechanism has been shown to be effective, practical, and general. Stateful Anycast is an attempt at filling the void.

The toughest type of DDoS attack to defend against is one where the attacker has a well-distributed botnet capable of sourcing enough traffic to overwhelm the links leading to a selected node. With this kind of attacker, defenses at the victim are useless because legitimate traffic is suppressed before it even reaches the victim. An effective solution must be "in" the network, where it can drop traffic before damage is done.

Anycast has proven to be effective at handling these flooding attacks, as witnessed by failed attempts at the critical DNS infrastructure [32]. Unfortunately, its effectiveness is limited to stateless communications. Anycast makes no guarantees about which group member will receive a packet that is addressed to an anycast group. Senders are unable to hold a reliable conversation if the anycast group member is obligated to remember state between packets.

To hold a stateful conversation, the sender must have some way of instructing the network to deliver data to the stateholder—a so-called *handle* on the stateholder. The simplest example is an address given by the sender to the network. The network uses the address to find and route packets to the stateholder. With stateless protocols, the sender does not need a handle on a specific node—the sender can describe a particular desired service and the network can arbitrarily choose the recipient from the set of nodes that provide the service.

If stateholders are unable to invalidate handles, attackers have a permanent ability to direct packets toward a particular stateholder. If the handles are not tied to a specific sender, the handle can be shared with fellow attackers to launch a distributed attack against the stateholder. Therefore, to protect stateful services from DDoS attacks, handles on stateholders must either be revocable or unusable by other nodes.

Making handles unusable by other nodes removes support for host mobility. For

the handle to be non-portable, its validity must be tied to something outside the sender's control, namely the topology between the sender and the stateholder. Unfortunately, this means that a legitimate node can have its handle invalidated by moving to a different region of the network.

Rather than making handles unusable by other nodes, Stateful Anycast is designed to give stateholders the power to revoke handles. By revoking a malicious handle, an attacker is prevented from focusing all of its resources onto a single stateholder (assuming there is more than one member of the Stateful Anycast group). The best the attacker can do is repeatedly attempt to establish new sessions or mimic a flash crowd, both of which will cause Stateful Anycast to distribute the malicious load among the group members. As long as there are enough group members with enough bandwidth to collectively absorb the malicious load without disruption, the attacker will fail.

Revocation is enforced by a distributed collection of anycasted middleboxes called *anycast proxies*. These proxies are the gateways to the stateholders, and have the primary responsibility of matching an incoming packet with the proper stateholder. Matching a packet to the proper stateholder involves querying a distributed database that is updated by the stateholders as sessions are established, concluded, or declared malicious. Database query results are cached to improve performance and limit DoS vulnerability.

Anycasting the proxies serves two purposes. First, attackers are prevented from singling out and disabling any one of the proxies. Second, because the Internet's routing protocols will generally select the anycast group member that is closest to the sender, the proxies are well-positioned to drop malicious traffic early.

The approach taken by Stateful Anycast provides several features:

- **Generality:** Stateful Anycast can support a wide range of stateful protocols. As long as a proxy is able to extract the session identifier from a packet, it can route the packet to the proper stateholder or drop it if the stateholder desires. In addition, the group members—not the anycast proxies—are responsible for detecting attacks. This allows them to change behavior without causing false alarms.

- **Practicality:** Stateful Anycast can be incrementally deployed, and benefits are realized even with small deployments. Stateful Anycast scales gracefully, and can provide benefits beyond DDoS protection (a Stateful Anycast group can provide Akakmai-like performance improvements). There is no need for multiple parties to coordinate deployment or even agree upon protocols—any organization can unilaterally decide to become a Stateful Anycast service provider. The only third-party dependency is on Internet Service Providers to supply straightforward Internet access for the anycast proxies.

- **Effectiveness:** Stateful Anycast lets the destination stateholder—the node with the most information regarding the intent of a sender—decide which packets to

drop, yet packets are dropped close to the source before significant damage is done.

One drawback of Stateful Anycast's design is that an attack must necessarily reach its victim before it can be detected and shut down. Fortunately, Stateful Anycast can be easily combined with other defense techniques to provide a comprehensive network security system. Some mechanisms, such as packet filters, could be deployed on the anycast proxies to take advantage of their strategic location. A powerful defensive tactic may be to rely on other in-network defensive techniques to take care of the most obvious attacks and let the group members use their intimate knowledge of "proper" behavior to clean up the rest.

6.1 Future Work

The next step is the completion of an initial version of the anycast proxy and group member software. Once a basic version is complete, evaluation can begin on Stateful Anycast's efficiency and effectiveness under realistic scenarios. Reliance on the directory service is perhaps the most risky aspect of Stateful Anycast's design, so a thorough evaluation of its performance under stress is perhaps the most important future result. Other aspects to evaluate include path stretch, tolerance to rapid topology changes, revocation response time, packet forwarding rates on commodity hardware, directory service latency, and so on.

Once Stateful Anycast has demonstrated adequate performance in a real-world scenario, a further step would be the maturation of the software into a deployable product. This would include the development of scalable algorithms for distributing new sessions among group members in a way that maximizes end-user performance, tuning the directory service routing algorithms to achieve maximal uptime in an adversarial environment, increasing the cache lookup and packet forwarding rates in the anycast proxies, and implementing DDoS attack detection plug-ins to monitor server health.

Taking a long-term perspective, a valuable exercise would be to design a hypothetical network architecture that includes packet flows as a network primitive. With routing based on packet flows, Stateful Anycast could forward or drop traffic in a protocol-agnostic manner.

Bibliography

[1] IEEE Std 1003.1. Also known as POSIX.1 and ISO/IEC 9945.

[2] libpcap. Available on-line at `http://www.tcpdump.org/`. WinPcap is a Windows port of libpcap and is available on-line at `http://www.winpcap.org/`.

[3] The network simulator: ns-2. Available on-line at `http://www.isi.edu/nsnam/ns/`.

[4] J. Abley, B. Black, and V. Gill. Goals for IPv6 Site-Multihoming Architectures. RFC 3582 (Informational), August 2003.

[5] J. Abley and K. Lindqvist. Operation of Anycast Services. RFC 4786 (Best Current Practice), December 2006.

[6] J. Abley and W. Maton. AS112 Nameserver Operations (draft-jabley-as112-ops-00). Internet-Draft Version 00, IETF, June 2006. Work in progress.

[7] Joe Abley. Hierarchical Anycast for Global Service Distribution. Technical Note ISC-TN-2003-1, ISC, 2003.

[8] Joe Abley. A Software Approach to Distributing Requests for DNS Service using GNU Zebra, ISC BIND 9 and FreeBSD. Technical Note ISC-TN-2004-1, ISC, 2004.

[9] Louis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: Using Hard AI Problems for Security. In *Advances in Cryptology – EUROCRPYT 2003: International Conference on the Theory and Applications of Cryptographic Techniques*, volume 2656 of *Lecture Notes in Computer Science*, pages 294–311, May 4–8, 2003.

[10] Akamai Technologies, Inc. `http://www.akamai.com/`.

[11] Aditya Akella, Bruce Maggs, Srinivasan Seshan, Anees Shaikh, and Ramesh Sitaraman. A measurement-based analysis of multihoming. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 353–364, New York, NY, USA, 2003. ACM Press.

[12] Arbor Networks. Internet service providers (ISP) report emergence of massive (10-20Gbps) attacks over the last 12 months. News release, September 12, 2006. Available on-line at `http://www.arbornetworks.com/news_detail.php?id=719` (accessed January 27, 2007).

[13] Paul Bächer, Thorsten Holz, Markus Kötter, and Georg Wicherski. Know your enemy: Tracking botnets. The Honeynet Project and Research Alliance, March 2005. Available on-line at `http://www.honeynet.org/papers/bots/` (accessed September 1, 2006).

[14] Hitesh Ballani and Paul Francis. Towards a global IP anycast service. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 301–312, New York, NY, USA, 2005. ACM Press.

[15] Anindya Basu, Alvin Lin, and Sharad Ramanathan. Routing using potentials: a dynamic traffic-aware routing algorithm. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 37–48, New York, NY, USA, 2003. ACM Press.

[16] T. Bates and Y. Rekhter. Scalable Support for Multi-homed Multi-provider Connectivity. RFC 2260 (Informational), January 1998.

[17] Scott Berinato. How a bookmaker and a whiz kid took on an extortionist—and won. *CSO Magazine*, May 2005. Also available on-line at `http://www.csoonline.com/read/050105/extortion.html` (accessed September 1, 2006).

[18] D. J. Bernstein. SYN cookies, September 1996.

[19] Robert Beverly. Private correspondence, August 2006.

[20] B. Carpenter and K. Moore. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056 (Proposed Standard), February 2001.

[21] CERT. TCP SYN Flooding and IP Spoofing Attacks. CERT Advisory CA-1996-21, CERT, September 1996.

[22] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631 (Informational), May 1994. Obsoleted by RFC 3022.

[23] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice), May 2000. Updated by RFC 3704.

[24] Dennis Fisher. Internet survives massive ddos attack. *eWEEK.com*, October 2002. Available on-line at `http://www.eweek.com/article2/0,1759,1653752,00.asp` (accessed September 1, 2006).

[25] Paul Francis. Firebreak: An IP perimeter defense architecture. Available on-line at `http://www.cs.cornell.edu/People/francis/firebreak/hotnets-firebreak-v7.pdf` (last accessed August 23, 2006).

[26] M. Handley, E. Rescorla, and IAB. Internet Denial-of-Service Considerations. RFC 4732 (Informational), December 2006.

[27] T. Hardie. Distributing Authoritative Name Servers via Shared Unicast Addresses. RFC 3258 (Informational), April 2002.

[28] A. Heffernan. Protection of BGP Sessions via the TCP MD5 Signature Option. RFC 2385 (Proposed Standard), August 1998.

[29] C. Huitema. An Anycast Prefix for 6to4 Relay Routers. RFC 3068 (Proposed Standard), June 2001.

[30] Alefiya Hussain, John Heidemann, and Christos Papadopoulos. A framework for classifying denial of service attacks. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 99–110, New York, NY, USA, 2003. ACM Press.

[31] G. Huston. Commentary on Inter-Domain Routing in the Internet. RFC 3221 (Informational), December 2001.

[32] Internet Corporation for Assigned Names and Numbers. Factsheet: Root server attack on 6 February 2007. Announcement, March 1, 2007. Available on-line at `http://www.icann.org/announcements/factsheet-dns-attack-08mar07_v1.1.pdf` (accessed May 27, 2007).

[33] Ari Juels and John Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the 1999 ISOC Network and Distributed System Security Symposium*, February 5, 1999.

[34] Dina Katabi and John Wroclawski. A framework for scalable global IP-anycast (GIA). *SIGCOMM Comput. Commun. Rev.*, 31(2 supplement):186–219, 2001.

[35] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005.

[36] Stephen Kent, Charles Lynn, and Karen Seo. Secure Border Gateway Protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, April 2000.

[37] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed internet routing convergence. In *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 175–187, New York, NY, USA, 2000. ACM Press.

[38] Matt Levine, Barrett Lyon, and Todd Underwood. Operational experience with TCP and Anycast. Presentation and Q&A given at NANOG 37, June 2006.

[39] Henry M. Levy. *Capability-Based Computer Systems*. Digital Press, 12 Crosby Dr, Bedford, MA 01730, 1984. Out of print, but available on-line.

[40] Ajay Mahimkar, Jasraj Dange, Vitaly Shmatikov, Harrick Vin, and Yin Zhang. dFence: Transparent network-based denial of service mitigation. In *NSDI '07: Proceedings of the 4th USENIX Symposium on Networked Systems Design and Implementation*, pages 327–340, April 13, 2007.

[41] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.

[42] Robert McMillan. Hackers slow Internet root servers with attack. *Computerworld*, February 2007. Available on-line at `http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9010619` (accessed February 12, 2007).

[43] D. Meyer and K. Patel. BGP-4 Protocol Analysis. RFC 4274 (Informational), January 2006.

[44] Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher. *Understanding Denial of Service*. Radia Perlman Series in Computer Networking and Security. Prentice Hall PTR, December 2004. Also available on-line at `http://www.phptr.com/articles/article.asp?p=386163` (accessed September 1, 2006).

[45] Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, 2004.

[46] P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592.

[47] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 2137, 2845, 3425, 3658, 4035, 4033.

[48] R. Moskowitz and P. Nikander. Host Identity Protocol (HIP) Architecture. RFC 4423 (Informational), May 2006.

[49] J. Moy. OSPF Version 2. RFC 2328 (Standard), April 1998.

[50] S. Murphy. BGP Security Vulnerabilities Analysis. RFC 4272 (Informational), January 2006.

[51] U.S. Department of Justice. Michigan man arrested for using New Jersey juvenile to launch destructive "DDOS for hire" computer attacks on competitors. Press release, March 2005. Also available on-line at `http://www.usdoj.gov/usao/nj/press/files/arab0318_r.htm` (accessed September 1, 2006).

[52] Denise Pappalardo and Ellen Messmer. Extortion via DDoS on the rise. *NetworkWorld.com*, May 2005. Available on-line at `http://www.networkworld.com/news/2005/051605-ddos-extortion.html` (accessed September 1, 2006).

[53] C. Partridge, T. Mendez, and W. Milliken. Host Anycasting Service. RFC 1546 (Informational), November 1993.

[54] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Comput. Surv.*, 39(1):3, 2007.

[55] C. Perkins. IP Mobility Support for IPv4. RFC 3344 (Proposed Standard), August 2002. Updated by RFC 4721.

[56] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the Internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003. Also see `https://www.planet-lab.org/`.

[57] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*, page 311. Morgan Kaufmann Publishers, San Francisco, CA, USA, third edition, 2003.

[58] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[59] J. Postel. Internet Control Message Protocol. RFC 792 (Standard), September 1981. Updated by RFC 950.

[60] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981. Updated by RFC 1349.

[61] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFC 3168.

[62] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (Standard), October 1985. Updated by RFCs 2228, 2640, 2773.

[63] Prolexic Technologies. `http://www.prolexic.com/`.

[64] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[65] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), January 2006.

[66] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

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

[68] K. Seo, R. Watro, D. Kong, and S. Kent. Certificate Policy (CP) for the Internet IP Address and AS Number (PKI). Internet-Draft Version 00, IETF, October 2006. Work in progress.

[69] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. *IEEE/ACM Trans. Netw.*, 12(2):205–218, 2004.

[70] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM Press.

[71] Michal Szymaniak, Guillaume Pierre, and Maarten van Steen. Versatile anycasting with mobile ipv6. In *AAA-IDEA '06: Proceedings of the 2nd international workshop on Advanced architectures and algorithms for internet delivery and applications*, page 2, New York, NY, USA, 2006. ACM Press.

[72] Jaikumar Vijayan. Akamai now says it was targeted by DDoS attack. *Computerworld*, June 2004. Also available on-line at `http://www.computerworld.com/securitytopics/security/story/0,10801,93862,00.html` (accessed September 1, 2006).

[73] Paul Vixie, Gerry Sneeringer, and Mark Schleifer. Events of 21-Oct-2002. Technical report, ISC/UMD/Cogent, November 2002. Available on-line at `http://www.isc.org/ops/f-root/october21.txt` (accessed February 12, 2007).

[74] Chi-Jen Wu, Ren-Hung Hwang, and Jan-Ming Ho. A scalable overlay framework for internet anycasting service. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 193–197, New York, NY, USA, 2007. ACM Press.

[75] Xiaowei Yang, David Wetherall, and Thomas Anderson. A DoS-limiting network architecture. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 241–252, New York, NY, USA, 2005. ACM Press.

[76] J. Yu. Scalable Routing Design Principles. RFC 2791 (Informational), July 2000.

[77] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, Berkeley, CA, USA, 2001.