MULTOPS: a data structure for denial-of-service attack detection

by

Thomer M. Gil

tmgil@cs.vu.nl

Submitted to the Division of Mathematics and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science

at the

VRIJE UNIVERSITEIT

August 2000

© Thomer M. Gil, MM. All rights reserved.

The author hereby grants to MIT and Vrije Universiteit permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Author .....
Division of Mathematics and Computer Science
Vrije Universiteit
Aug 23, 2000

Certified by .....
M. Frans Kaashoek
Associate Professor of Electrical Engineering and Computer Science, Massachusetts Institute of Technology
Thesis Supervisor

Certified by .....
Massimiliano Poletto
Research Staff, Massachusetts Institute of Technology
Thesis Supervisor

Certified by .....
Andrew S. Tanenbaum
Professor of Computer Science, Vrije Universiteit
Thesis Supervisor

MULTOPS: a data structure for denial-of-service attack detection

by

Thomer M. Gil

Submitted to the Division of Mathematics and Computer Science
on Aug 23, 2000, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science

Abstract

A denial-of-service (DoS) attack is an attempt by a single person or a group of people to disrupt an online service. In a bandwidth attack, attackers clog links or routers by generating a traffic overload. This can have serious consequences to companies that rely on their online availability to do business. The ubiquity of tools to organize DoS attacks and the determination of some people to wreak havoc make for potential future problems. This thesis proposes a MUlti-Level Tree for Online Packet Statistics (MULTOPS): an attack-resistant data structure enabling routers to detect ongoing bandwidth attacks by searching for significant asymmetries between packet rates to and from different subnets. Statistics are kept in a tree that dynamically adapts its shape to (1) reflect changes in packet rates, and (2) avoid (maliciously intended) memory exhaustion. A MULTOPS is suitable to detect the type of bandwidth attack that occurred on a large scale in February 2000. To remain undetected, the attacker has to launch the attack from more sites which makes mounting the attack more difficult. This will hopefully discourage many attackers.

Thesis Supervisor: M. Frans Kaashoek
Title: Associate Professor of Electrical Engineering and Computer Science, Massachusetts Institute of Technology

Thesis Supervisor: Massimiliano Poletto
Title: Research Staff, Massachusetts Institute of Technology

Thesis Supervisor: Andrew S. Tanenbaum
Title: Professor of Computer Science, Vrije Universiteit

To Lisa

"At that time Michael, the great prince who protects your people, will arise. There will be a time of distress such as has not happened from the beginning of nations until then... Those who are wise will shine like the brightness of heavens... Many will go here and there to increase knowledge." (Daniel 12:1-4)

Contents

1 Introduction **9**

2 Overview of DoS attacks **11**

2.1 IP Spoofing 11
2.2 Distributed DoS attacks 12
2.3 Smurf: ICMP flood 13
2.4 Trinoo: UDP flood 14
2.5 SYN bombs 14
2.6 Stealth bombs 15
2.7 Bandwidth attack classification 16
2.8 Solutions 16

3 Design of MULTOPS **19**

3.1 Overview 19
3.2 Assumptions 19
3.3 Data structure 21
3.4 Algorithm 22
3.5 Rates 22
3.6 Unfolding 22
3.7 Folding 24
3.8 Memory exhaustion attacks 24

4 Implementation of MULTOPS **27**

4.1 Click 27
4.2 IPRateMonitor 28
4.3 Data structure 29
4.4 Algorithm 30
4.5 Unfolding 30
4.6 Folding 32

4.7 Memory exhaustion attacks 33
4.8 Measurements 35

5 Detecting bandwidth attacks with MULTOPS 39
5.1 Flood and D-Flood 39
5.2 Flood+ 40
5.3 Stealth 41
5.4 D-Stealth/Flashcrowd 41
5.5 SYN bomb 42

6 Future work 43

7 Conclusion 45

8 Acknowledgements 47

A Click config file for performance measurement 51

List of Figures

2-1 ICMP without (left) and with (right) IP spoofing 13
2-2 "Smurf" attack 14
2-3 SYN bomb 15
2-4 Bandwidth attack classification 16

3-1 Location of MULTOPS-equipped routers 20
3-2 MULTOPS 21
3-3 Algorithm for MULTOPS 23

4-1 Simple Click configuration 27
4-2 Click configuration with IPRateMonitor 28
4-3 Counter and Stats 29
4-4 Code executed for each packet 31
4-5 Memory use of IPRateMonitor as a result of folding 33
4-6 Relation between bandwidth and memory use 34
4-7 Packet rate as a function of memory limit 35
4-8 CPU cycles per packet as a function of memory limit 36

5-1 Example bandwidth attack setup 40

Chapter 1

Introduction

A denial-of-service (DoS) attack is a malicious attempt by a single person or a group of people to cripple an online service. This can have serious consequences for companies such as Amazon and eBay which rely on their online availability to do business. On February 9, 2000, Yahoo, eBay, Amazon.com, E*Trade, ZDNet, Buy.com, the FBI, and several other Web sites fell victim to DoS attacks ([1], [2], [3]) resulting in millions of dollars in damage. In May 2000 the same fate befell Slashdot.org [4].

Sophisticated tools to gain root access to other people's machines are freely available on the Internet ([5], [6]). These tools are easy to use, even for computer illiterates. Once a machine is cracked, it is turned into a "zombie" under control of one "master". The master is operated by the attacker [7]. At any specific time the attacker can instruct all its zombies to send bogus data to one particular destination. The resulting traffic can clog links, cause routers near the victim or the victim itself to fail under the load. The type of DoS attack that causes problems by generating an overload of traffic is known as a **bandwidth attack**. This thesis focuses on bandwidth attacks.

Several reasons underlie the absence of a solution against bandwidth attacks. Both IP and TCP were originally designed with a small, cooperative user group in mind. The inappropriateness of that mind set is now obvious. Both IP and TCP can be used as dangerous weapons quite easily. Since all Web traffic is TCP/IP based, attackers can release their malicious packets on the Internet without being conspicuous or easily traceable. It is the mass of all packets together that poses a threat rather than single characteristics of individual packets. A bandwidth attack solution is, therefore, more complex than a straightforward filter in a router.

A key problem to tackle when solving bandwidth attacks is **attack detection**. Detecting a bandwidth attack might be easy in the vicinity of the victim, but gets more difficult as the distance (i.e., hop count) to the victim increases. In addition, an attack detection mechanism must be able to establish the source(s) of the attack. Furthermore, any mechanism to detect bandwidth attacks must be robust, for it will be a likely target of attacks itself. In short, a bandwidth attack detection mechanism must be:

- **sensitive**: detect bandwidth attacks
- **accurate**: establish the source(s) of the attack
- **robust**: withstand attacks on the detection mechanism itself
- **inexpensive**: not induce an unacceptable per-packet overhead

TCP acknowledges the receipt of a packet by sending back a packet to the sender. The bandwidth attack detection mechanism proposed in this thesis exploits this property of TCP. In addition, TCP is an "**adaptive**" protocol: it reacts to the loss of acknowledgements by decreasing the sending rate. What follows is that the packet rate to a certain host or subnet should always be proportional to the packet rate from that host or subnet. A router observing a certain packet rate in one direction with a significantly lower packet rate in the opposite direction can suspect that the slower side is unable to cope with the traffic it receives and may, therefore, be under attack.

This thesis proposes a **MUlti-Level Tree for Online Packet Statistics (MULTOPS)**: an attack-resistant data structure enabling routers to detect ongoing bandwidth attacks by keeping track of packet rates to and from subnets. A MULTOPS can zoom in on subnets that behave conspicuously to gain more precise information and help determine the source(s) of the attack. The MULTOPS attack detection mechanism is a novel technique to detect bandwidth attacks that are mounted using unadaptive protocols such as UDP and ICMP. This covers all February 2000 attacks. In most cases, the MULTOPS bandwidth attack detection mechanism fails to detect attacks that are mounted using adaptive protocols such as TCP.

The rest of this thesis is organized as follows. Chapter 2 gives an overview of DoS attacks. Chapter 3 introduces the concepts behind MULTOPS bandwidth detection. Chapter 4 deals with the implementation of MULTOPS. Chapter 5 looks at how well a MULTOPS detects different bandwidth attacks. Chapter 6 gives recommendations for future research. Chapter 7 concludes this thesis.

Chapter 2

Overview of DoS attacks

DoS attacks can be divided into at least three types: (1) exploits of implementation bugs, (2) server resource attacks, and (3) server bandwidth attacks. DoS attacks that exploit implementation bugs are in principle relatively easy to solve by installing proper patches.

Attacks on server resources (memory, disk space, etc.) are more difficult to stop because they often exploit legitimate protocol features rather than simple bugs. Such attacks exploit features of applications or of protocols above the transport layer. Example attacks against pseudonym mail servers are generating exponential mail loops or maliciously creating a large number of pseudonym accounts in a short time [8]. These high-level DoS attacks must be handled by the application in question, since it is impossible for lower network or system layers to detect or counter the specific problem.

Attacks on server bandwidth are performed by congesting the victim's networks with (useless) traffic. Bugs in routers on the victim's network can cause the routers to crash, compounding the problem. Some attacks generate easily identifiable packets that can be filtered or rate-limited because they never occur in high volume during normal operations [9]. More subtly, bandwidth attacks may be caused by traffic that looks entirely normal except for its high volume [10]. Usually, bandwidth attacks require a group of attackers to cooperate so as to generate sufficient traffic.

The rest of this chapter is organized as follows. Sections 2.1 and 2.2 deal with some general DoS attack issues. Sections 2.3, 2.4, 2.5, and 2.6 describe specific DoS attacks. Section 2.7 classifies bandwidth attacks. Section 2.8 deals with some solutions that have been proposed so far.

2.1 IP Spoofing

IP spoofing is lying about one's own IP address. When writing to a raw socket—which requires root permission—a program can fill the header fields of an IP packet with whatever it wants. Since routing is done based on the IP destination address only, the IP source address can be anything. In some cases, attackers use one specific IP source address on all outgoing IP packets to make all

returning IP packets—and possibly ICMP messages—go to the unfortunate owner of that address. Attackers also use IP spoofing to hide their location on the network. Section 2.3 looks at an attack that uses IP spoofing to flood a victim.

Ingress filtering [11] and **egress filtering** [12] are performed by routers to effectively eliminate IP spoofing. Routers match the IP source address of each incoming packet against a fixed set of IP addresses. If no match is found, the packet is dropped. For example, a router at MIT will only route outgoing packets that have an IP source address from subnet 18.0.0.0/8. Although IP spoofing is a nice weapon for attackers to wield, it is in many cases no more than just that. As more attackers are involved in an attack, each operating from different networks, the need for IP spoofing becomes less. Ingress and egress filtering will make the life of an attacker more difficult, but they are far from being the panacea.

Stefan Savage et al. have devised a scheme called **IP Traceback** [13] that assists in tracking down attackers post-mortem. Their technique requires routers to probabilistically mark packets such that the receiving end can reconstruct the route that packets followed, provided that enough packets were sent. This technique looks very promising, but its main weakness is that it only assists in finding attackers; it provides no protection against bandwidth attacks. IP Traceback is primarily effective by being a deterrent.

2.2 Distributed DoS attacks

One single non-spoofing attacker that generates more traffic than the victim can handle is easily identifiable. The defense is to deploy a filter in a router on the victim's or—preferably—the attacker's network that blocks all packets from the attacker. If the attacker randomly spoofs IP addresses, then an ingress/egress filtering edge router on the attacker's network will stop most packets. That forces the attacker to only use IP addresses from her own network for spoofing. Confronted with this attacker, the victim is faced with a bigger problem. As soon as the victim knows from which network the attack is coming, a filter could be deployed that drops all traffic from that specific network. Even though this stops the attack, it also denies service to all other clients on the attacker's network. This solution is a little blunt, but still constitutes some progress because the rest of the world remains unaffected by this filter.

The attacker can get around this filter by launching an attack from different networks. The victim is now faced with a distributed DoS (DDoS) attack. If the attacker compromises 1 machine on N networks, then each machine needs to generate $1/N$ of the required bandwidth to flood the victim. If N is big (and, consequently, the amount of traffic generated from each network small), the victim can no longer label traffic from one network as malicious or benign based on its relative volume. Installing a filter that drops packets from all suspected networks means denying service to all users on all those networks and, therefore, defeats its own purpose.

The conclusion is that sophisticated attackers will try to use IP source addresses from many

different subnets. To do that, the attacker has to either find networks without ingress/egress filtering edge routers, or launch her attack from many different sites so that each attacking client can generate an inconspicuous amount of traffic. So far, attackers have not done any sophisticated IP spoofing. In many cases they use 0.0.0.0 as IP source address. The sad conclusion is that many attacks work only because ingress/egress filtering is inactive in many routers. However, it seems only a matter of time before more sophisticated attacks as described above will occur.

2.3 Smurf: ICMP flood

A "Smurf" attack [9] is carried out using ICMP. ICMP is a protocol used for sending control messages. ECHO REQUEST and ECHO REPLY are two such messages. UNIX's "ping" program, for example, uses ICMP to measure round-trip delays between two machines. The left side of Figure 2-1 exemplifies this. A sends an ECHO REQUEST message to B. B replies with an ECHO REPLY message. The notation "A $\rightarrow$ B" represents the IP source address and IP destination address in the IP packet that carries the ICMP message. B knows where to send its ECHO REPLY by looking at the IP source address in the arriving IP packet.

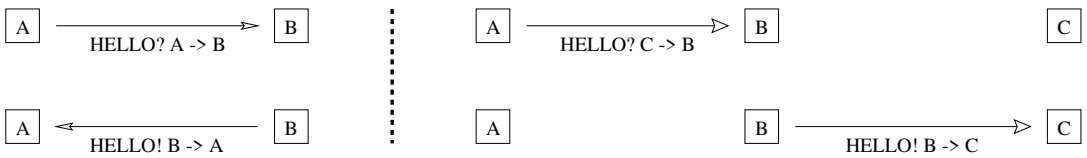

Figure 2-1: ICMP without (left) and with (right) IP spoofing

If A spoofs the IP source address, the situation as shown on the right side in Figure 2-1 occurs. A sends an ECHO REQUEST message to B, but spoofs the IP source address by using C, not A. As a result, C receives an ICMP ECHO REPLY from B, seemingly out of the blue. This does not pose an immediate threat to C because C can simply discard the message. The situation becomes harmful when A sends an ECHO REQUEST to a broadcast address. Each machine on the receiving network gets the ECHO REQUEST and each machine responds. If A spoofs the IP source address, an innocent party will receive all the ECHO REPLYs (see Figure 2-2). Consequently, links and routers to C might get clogged by all the traffic.

"Smurf" attacks work because of a bug in many ICMP implementations. A host may never send an ECHO REPLY in response to a broadcasted ECHO REQUEST. Unfortunately, many ICMP implementations fail to check the IP destination address of the incoming ECHO REQUEST.

IP spoofing is an absolute necessity for this attack to succeed. If the attacker fails to spoof the IP source address, the ECHO REPLY flood will come back to himself which is similar to shooting

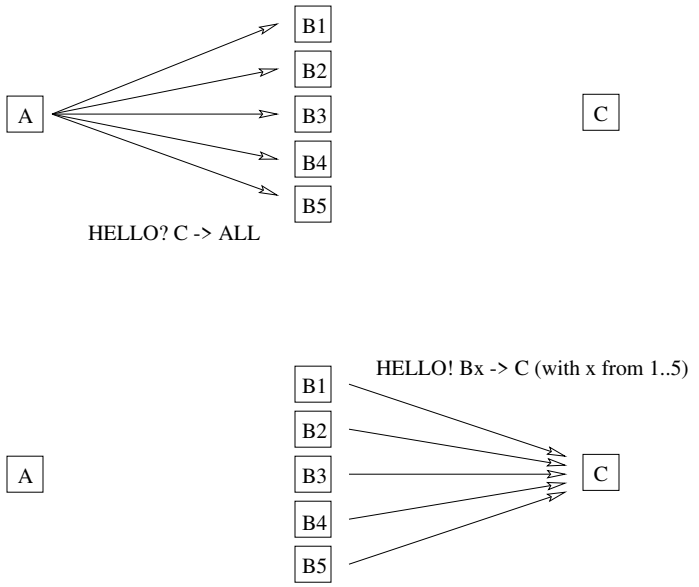

Figure 2-2: "Smurf" attack

into one's own foot.

2.4 Trinoo: UDP flood

Because of the logistics behind Trinoo attacks, they are far more sophisticated than "Smurf" attacks. After compromising a machine—a whole science of its own [6]—a small daemon is installed which waits for commands from a master: the compromised machine is turned into a zombie. Communication between the master and the zombies is often encrypted so as to complicate matters for network intrusion detectors. At any point in time, the master can instruct one or more zombies to start sending UDP packets to one destination.

2.5 SYN bombs

Normal establishment of a TCP connection requires three packets to be sent between the client and the server: (1) a client sends a SYN packet, (2) the server allocates a TCP control block and sends back a SYN/ACK packet, and (3) the server waits for an ACK to come back from the client. This is called a 3-way handshake. As long as the ACK from step 3 has not come back to the server, the connection is in half-open state. When no ACK comes back from the client at all, the connection remains in half-open state until TCP times out after a few minutes. When TCP times out, the

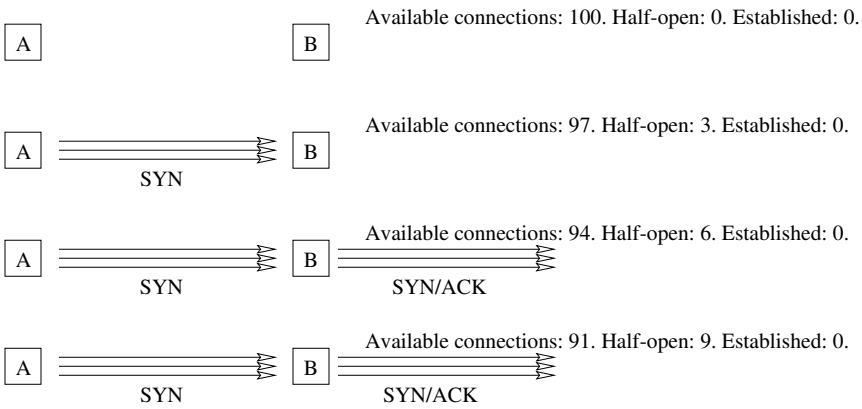

Figure 2-3: SYN bomb

allocated control block becomes available again.

A simple attack (see Figure 2-3) is for A to continuously send SYN packets in spoofed IP packets. The SYN/ACK packets will go to an innocent third party (who will drop them) and the required ACK packets never get sent by anyone. This will cause B to run out of TCP control blocks very fast with all available connections in half-open state. A server without any available TCP control blocks is unable to accept any more incoming TCP connections.

Several solutions have been proposed for solving SYN bombs: lowering the TCP timeout, increasing the number of TCP control blocks, SYN cookies [14] that eliminate the need to store information on half-open connections, and special firewalls that buffer SYN packets.

2.6 Stealth bombs

To remain filter-proof, the next generation of bandwidth attacks might simply generate a huge amount of normal TCP/IP traffic. A JavaScript running in a browser that pops up a few dozen windows each fetching a Web page from one server means certain death for that server if a few thousand people are willing to run this script in their browser simultaneously [10]. Such a script could easily spread by means of self-replicating e-mail viruses.

We make a distinction between a Stealth bomb and a **flashcrowd**. A flashcrowd is a group of benign clients that overload one Web server or link close to the Web server by causing too much traffic. A Web server displaying the results of the Super Bowl finals on a Web page is very likely to be subjected to such a scenario. A flashcrowd is similar to a SYN bomb in the sense that it can exhaust all the available TCP control blocks on the server. It differs from a SYN bomb in that the required 3-way handshake to establish a TCP connection progresses normally.

	not distributed		distributed
	no spoofing	**spoofing**	
adaptive	Stealth	*n.a.*	D-Stealth/Flashcrowd
unadaptive	Flood	Flood+	D-Flood

Figure 2-4: Bandwidth attack classification

2.7 Bandwidth attack classification

Many different types of bandwidth attacks exist. For later use, a convenient classification of bandwidth attacks is chosen; see the table in Figure 2-4. "+" indicates the use of spoofing; "D-" indicates a distributed attack. This classification uses three properties: protocol-type, distribution, and whether or not IP spoofing is involved. An adaptive protocol is one that adjusts its rate when packets get lost. TCP is an adaptive protocol. Examples of non-adaptive protocols are UDP and ICMP.

No distinction is made between a distributed attack with spoofing and a distributed attack without spoofing. One could also argue that a non-distributed attack with spoofing and a distributed attack are of the same type, too. From the victim's point of view this is true, but in chapter 5 we will show that, from a MULTOPS point of view, they are different.

SYN bombs are not bandwidth attacks, but rather server resource attacks. SYN bombs do not fit in this classification.

2.8 Solutions

Ingress/egress filtering and IP Traceback are methods to deal with attackers that use IP spoofing. Both are described in section 2.1.

A Cisco white paper [15] explains how to use a Cisco router to characterize an attack and how to reconfigure the router to minimize the negative effects. Unfortunately, in some February 2000 attacks, it was not a server that crashed under the load, but rather an upstream router. In some cases, the machine that was supposed to stop bandwidth attacks crashed itself [16].

The IETF is currently writing a draft that proposes a new ICMP message type: ICMP Traceback. These traceback messages should help solve bandwidth attacks. When forwarding packets, routers can, with a low probability (1/20000), generate a traceback message that is sent along to the destination. With enough traceback messages from enough routers along the path, the traffic source and path can be determined [17]. Unfortunately, this idea has several problems:

- Routers that support these traceback messages need to be widely deployed before the victim

can actually trace back packets to their source.

- Attackers can generate traceback messages, too. The IETF fails to describe a proper authentication mechanism for traceback messages. Authentication implies encryption. Encryption implies computation, which leads to a vulnerability for resource (CPU) attacks. Attackers could send many bogus traceback messages to a host that will, consequently, spend most of its time decrypting these messages.
- Routers assist the attacker by adding more packets to the flood.
- The victim of a bandwidth attack might not receive enough traceback messages because they might get dropped by overloaded routers.

Chapter 3

Design of MULTOPS

This chapter deals the design of MULTOPS. Chapter 4 deals with its implementation.

3.1 Overview

A MULTOPS is a data structure designed to keep track of aggregate packet rates to and from different subnets. MULTOPS should be deployed in routers (see Figure 3-1). A MULTOPS is a tree of nodes where higher layers in the tree contain aggregate packet rates to and from subnets of increasing size. The deepest level of the tree contains packet rates to and from single hosts. A MULTOPS is suitable for detecting bandwidth attacks by using a significant asymmetry in packet rates to and from a certain subnet as an indication for an attack on or from that subnet. This is based on the assumption that the TCP packet rate to a host or subnet is always proportional to the packet rate from that host or subnet.

The shape of the tree is determined by packet rates. If the packet rate to or from a subnet exceeds a certain threshold, a node is created to keep track of packet rates for subnets within that subnet. Similarly, a node is destroyed if the aggregate packet rate to and from the associated subnet falls below a certain threshold. This mechanism enables a MULTOPS to zoom in and out on subnets dynamically.

A MULTOPS is primarily effective as part of a mechanism to detect Flood and D-Flood attacks, i.e., attacks using an unadaptive protocol without IP spoofing. To successfully detect other attacks, collaboration between routers is required.

3.2 Assumptions

A few assumptions about bandwidth attacks are made in this thesis. These assumptions apply to all February 2000 attacks.

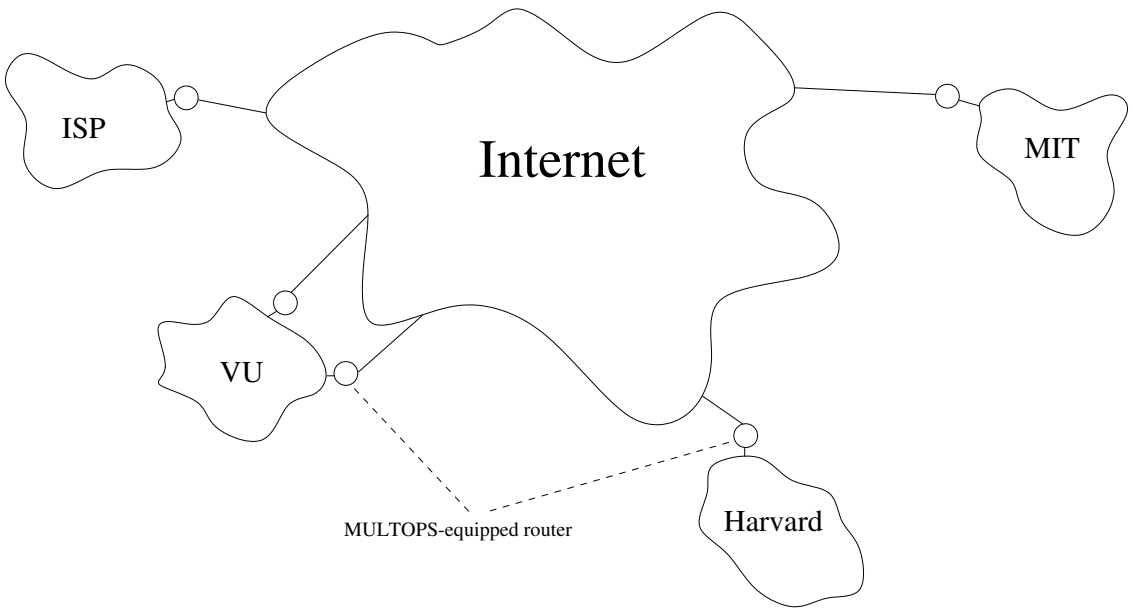

Figure 3-1: Location of MULTOPS-equipped routers

The attacker(s) and the victim are separated by at least one router. This assumption covers all February and May 2000 attacks.

In the normal case, packet rates between two hosts are symmetric. We say that packet rates between hosts A and B are symmetric when the packet rate from A to B is equal to the packet rate from B to A times some constant k. In other words: the packet rate to a host is proportional to the packet rate from that host. What follows is that packet rates between two subnets are also symmetric.

Although this assumption does not fully cover reality, it does apply to hosts that communicate with commonly used TCP implementations. This assumption also applies to hosts that are sending back and forth ICMP ECHO REQUEST/REPLY messages.

What follows from this assumption is that asymmetric packet rates are an indication of unusual behavior.

Routes through MULTOPS-equipped routers are symmetric and stable In this context, a symmetric route means that if packets going from A to B go through router R, then packets from B to A also go through R. If this is not true, then—obviously—a router cannot detect asymmetries in packet rates. Besides being symmetric, routes are assumed to be stable, i.e., routes do not change more than once every few minutes.

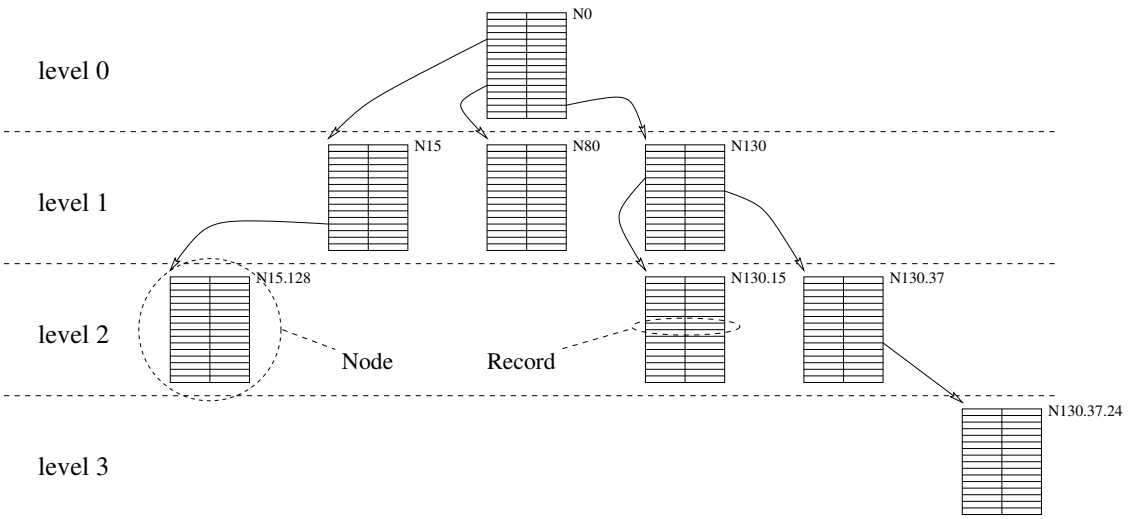

Figure 3-2: MULTOPS

3.3 Data structure

A MULTOPS is a tree of nodes. Every node has 256 records with 2 fields to keep track of aggregate packet rates to and from subnets within a certain subnet. The tree is at most 4 levels deep. Level 0 contains the root node only. The root node contains aggregate packet rates to and from 8-bit prefix subnets (0.0.0.0/8, 1.0.0.0/8, ..., 255.0.0.0/8). A node on level 1 keeps track of aggregate packet rates to and from 16-bit prefix subnets. A node in level 2 contains aggregate packet rates to and from 24-bit prefix subnets. Level 3 contains packet rates to and from single hosts. Figure 3-2 shows a MULTOPS. The node labeled "N130" keeps track of aggregate packet rates to and from subnets 130.0.0.0/16, 130.1.0.0/16, ..., 130.255.0.0/16. Node "N130.37.24" keeps track of packets rates to and from single hosts within subnet 130.37.24.0/24.

Nodes are created and destroyed in real time to reflect changes in packet rates. When the aggregate packet rate to or from subnet S exceeds a certain threshold, a child node is created under the relevant record to keep track of packet rates to and from all subnets within S. This "**unfolding**" can continue down to level 3 in the tree. When packet rates go down, relevant parts of the tree are destroyed ("**folding**") to minimize memory consumption and to avoid the overhead of updating packet rates that are not of interest. Folding and unfolding are the essential mechanisms in a MULTOPS.

3.4 Algorithm

A MULTOPS keeps track of packet rates to **and** from subnets. This requires a notion of direction. A router on the edge of subnet D can act in two different ways for each packet it is forwarding **to** D: (1) it always updates the numbers that keep track of packet rates **to** D, or (2) it always updates the numbers that keep track of packet rates **from** the source host (or source subnet). A router that uses method 1 is in **outside protection** mode. A router that uses method 2 is in **inside protection** mode. In either case, each packet causes at least 1 update (in the root node) and at most 4 updates (when the tree is unfolded to level 3 for that IP address). The complete algorithm is given in Figure 3-3. A router runs `per_packet_execute` for each packet it receives on one of its inputs. Unless specified otherwise, we assume that MULTOPS-equipped routers operate in inside protection mode.

3.5 Rates

Let $F(S)$ be the aggregate packet rate from subnet S and let $T(S)$ be the aggregate packet rate to S. R is defined as:

$$R(S) = \frac{T(S)}{F(S)}$$

In words: $R(S)$ is the ratio between the aggregate packet rate to subnet S and the aggregate packet rate from S, as measured by a router. In the normal case, R will be close to some constant k for all subnets, i.e., packet rates are symmetric. A router in inner-protection mode can suspect a bandwidth attack **from** one or more machines within subnet S if R is too small. It can suspect a bandwidth attack **on** one or more machines within subnet S if R is too big. These thresholds, which we refer to as R_{min} and R_{max} respectively, influence the router's sensitivity to detect bandwidth attacks: if R_{min} is too small or if R_{max} is too big, attacks might remain undetected. If R_{min} is too big or if R_{max} is too small, false positives might result.

3.6 Unfolding

A MULTOPS keeps track of packet rates on different aggregation levels. If $T(S)$ or $F(S)$ in a certain record r exceed threshold Q, the MULTOPS unfolds r. Record r is unfolded by creating a child node that keeps track of aggregate packet rates to and from subnets within S. In Figure 3-2, for example, the record with index 130 in the root node is unfolded into node N130. A record in a node in level 3 of the tree cannot be unfolded.

```
#define INSIDE_PROTECTION  0
#define OUTSIDE_PROTECTION 1
#define TO   0
#define FROM 1

extern int threshold;   // Threshold for unfolding
extern int mode;        // INSIDE_PROTECTION or OUTSIDE_PROTECTION
extern Subnet S;        // S is some subnet close to this router
extern Node *root;       // root of MULTOPS

per_packet_execute(IPpacket *packet)
{
  if(mode == INSIDE_PROTECTION) {
    if(packet is heading for S)
      update_rate(FROM, packet->source_addr);
    else
      update_rate(TO, packet->dest_addr);
  } else { // i.e., mode == OUTSIDE_PROTECTION
    if(packet is heading for S)
      update_rate(TO, packet->dest_addr);
    else
      update_rate(FROM, packet->source_addr);
  }
  if(folding necessary)
    fold();
}

update_rate(int direction, IPaddress ip_address)
{
  Node *node = root;
  Record *record;

  for(i = 0; i <= 3; i++) {
    int n = ip_address[i]; // i.e., i-th byte from ip_address
    record = node->record[n]; // i.e., n-th record from node
    update rate[direction] in record // i.e., either TO or FROM rate updated
    if(rate[direction] > threshold)
      create child node;
    if(no child node) // descend?
      break;
    else
      node = record->child_node;
  }
}
```

Figure 3-3: Algorithm for MULTOPS

3.7 Folding

The reverse of unfolding is folding. Folding a record means deleting the whole subtree under that record. We say that the record is **eligible** for folding if both packet rates to and from subnet S are less than Q. The primary motivation behind folding is to constrain memory use and to avoid (maliciously intended) memory exhaustion. Several issues around folding need to be addressed:

- How often should records in a MULTOPS be folded?

  Compacting can be made: (1) packet-triggered, (2) time-triggered, or (3) memory-triggered.

  Packet-triggered folding means folding the MULTOPS for every N packets that pass by. This means that the folding frequency increases when the traffic rate goes up. Since some computational effort is associated with folding, this is a bad idea because a router should have its resources free for routing, not for folding when packet rates go up.

  Time-triggered folding means folding the MULTOPS every N ms. Obviously, time-triggered folding does not have the problem that packet-triggered folding has. Choosing a correct value for N is tricky, though. Choosing N too low is bad because folding might not be necessary every N ms and as a result, routing slows down unnecessarily. Choosing N too high is dangerous because a router may run out of memory before N ms have passed.

  Memory-triggered folding means folding the MULTOPS when its memory use hits a certain threshold. A variation is to make the folding frequency increase as the MULTOPS memory use nears the threshold.

- Which records are folded and which are not?

  Obviously, folding eligible records before other records is best. Finding eligible records can be expensive, though. A depth-first search through the whole tree is unattractive when the tree has many nodes. Compacting should be quick because packets keep pouring in and may get dropped if folding takes too long.

  If no record in the tree is eligible but folding must be done to avoid memory exhaustion, then the following strategies can be followed: (1) fold records with symmetric rates before records with asymmetric rates, or (2) fold records with low rates before records with high rates.

  Section 4.6 deals with these issues in the context of the MULTOPS implementation.

3.8 Memory exhaustion attacks

An attacker might try to launch a memory exhaustion attack on a MULTOPS-equipped router by causing the MULTOPS to branch profusely. The two opposing forces in such an attack are the attacker sending packets on one side, and the MULTOPS folding parts of the tree on the other

side. The attacker causes nodes to be created, folding causes them to be destroyed. Since a node for subnet S is destroyed when the aggregate packet rates to and from S are both less than Q, the attacker will try to generate a bandwidth higher than Q for as many different subnets as possible. Section 4.7 deals with memory exhaustion attacks in a quantitative context.

Chapter 4

Implementation of MULTOPS

4.1 Click

A MULTOPS is implemented as a Click [18] element. Click is a modular software router architecture developed at the MIT Laboratory for Computer Science. A Click router is an interconnected collection of modules called elements. Each element performs a simple, straightforward task such as communicating with devices, queueing packets, and implementing a dropping policy. A Click router is configured by feeding it a file written in a language designed to describe the interconnection between different elements. Click runs under Linux as a user program or as a kernel module.

Figure 4-1 shows a simple Click configuration with 5 elements. Each rectangle is an element; the arrows represent connections between elements. The arrows indicate the direction of the packet flow between elements. Elements can have 0 or more inputs and outputs. `FromDevice`, for example, has no inputs and 1 output; `Split` has 1 input and 2 outputs.

Incoming packets are grabbed from a network card (eth0) by `FromDevice`. `FromDevice` has one argument (`eth0`) that tells it what device to grab packets from. Depending on its specification, an element can have zero, one, or any number of arguments. Every packet flows through `Counter` which simply keeps track of the number of packets by increasing a counter by 1 for each packet

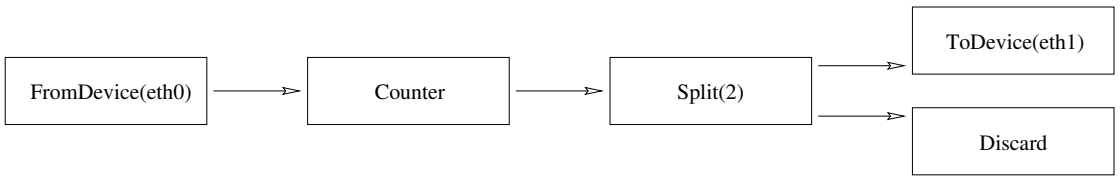

Figure 4-1: Simple Click configuration

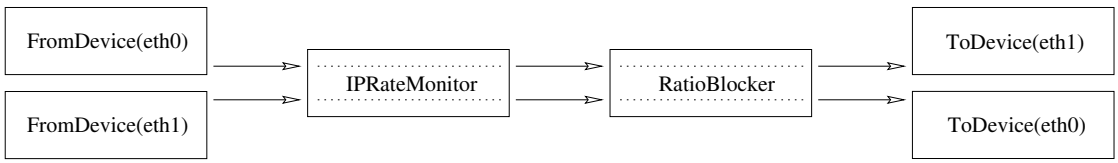

Figure 4-2: Click configuration with IPRateMonitor

that comes by. `Split` creates a copy of each packet and pushes one copy on each of its outputs. Note that the number of copies `Split` makes (and the number of outputs it has) is determined by the argument passed to it. `Split` pushes one copy to `Discard` which simply drops each packet on the floor. The other copy goes to `ToDevice`, which hands the packet over to a network card (eth1) where it gets sent out.

When Click runs as a kernel module, it allows user-level programs to query some elements. Interaction between user-level programs and Click goes through the `/proc` file system. For example, the command `cat /proc/click/Counter/count` prints the number of packets `Counter` has seen so far. Certain elements allow their behavior to be influenced in real time by writing to a `/proc` file. For example, the count of `Counter` can be reset to zero by executing `echo 0 > /proc/click/Counter/reset`.

Click elements can write "annotations" on a packet. An annotation is a per-packet piece of information that exists as long as a packet resides in Click. Annotations enable elements to pass information to each other.

More information on Click is available at `http://pdos.lcs.mit.edu/click`.

4.2 IPRateMonitor

A MULTOPS is implemented as a Click element called `IPRateMonitor`. Figure 4-2 shows a simple Click configuration with an `IPRateMonitor`. This Click configuration sends all packets that come in on eth0 to eth1 and vice versa. Every packet is led through `IPRateMonitor` and `RatioBlocker`. Collaboration between `IPRateMonitor` and `RatioBlocker` is achieved through two annotations. Before `IPRateMonitor` pushes a packet p on one of its outputs, it annotates p with the packet rates that p itself is a part of.

The `RatioBlocker` element implements **bandwidth attack protection**. `RatioBlocker` calculates R from both annotations and drops the packet if R violates either R_{min} or R_{max}. Otherwise it pushes the packet on one of its outputs. Thus, `RatioBlocker` distinguishes between packets that are part of a symmetric flow and packets that are part of an asymmetric flow.

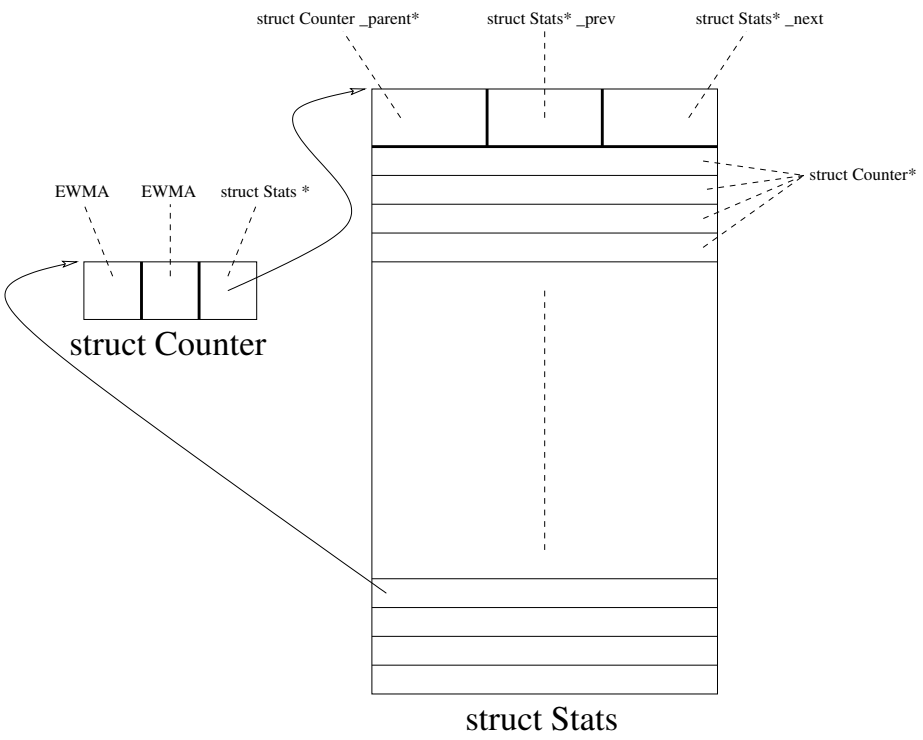

Figure 4-3: Counter and Stats

4.3 Data structure

`IPRateMonitor` is implemented using two `structs`: `Counter` and `Stats`.

```
struct Counter {
    EWMA rev_rate;
    EWMA fwd_rate;
    Stats *next_level;
};

struct Stats {
    Counter *_parent;
    Stats *_prev, *_next;
    Counter* counter[256];
};

Stats *_base;
```

Class `EWMA` implements an exponentially weighted moving average. `EWMA`s are used to keep track of rates. `Counter` is the C++ equivalent of a MULTOPS record. In addition to 2 `EWMA`s, `Counter` contains a pointer to a child node (`next_level`) which is `NULL` if the record is not unfolded and has a non-`NULL` value if the record is unfolded. In that case, `next_level` points to the child node. `Stats` is the C++ equivalent of a MULTOPS node. It contains 256 `Counter *` pointers and some additional pointers. `_base` is a pointer to the root node. Figure 4-3 shows a schematic representation of both data structures.

4.4 Algorithm

Figure 4-4 shows the simplified code that is executed for each packet in a router that is operating in inside protection mode. When a packet is ready, Click wakes up `IPRateMonitor` by calling `handle_packet`. `handle_packet` calls `update` with either `p`'s source address or destination address, depending on the input that the packet arrived on. After `update`, it pushes the packet on the corresponding output.

`update` starts with a `for`-loop in which it uses `i` to extract a specific byte from `address`. This byte (`x`) is used to get `s->counter[x]` (i.e., the `x`-th record in node `s`). If `s->counter[x]` is `NULL`, a `Counter` is allocated and assigned to `c`. In any case, either `c->fwd_rate` or `c->rev_rate` is updated. Finally, it checks for a deeper `Stats` by looking at `c->next_level`. If non-`NULL` (i.e., `c` is unfolded), another iteration is done for `c = c->next_level`. Otherwise the loop is broken.

After execution of the `for`-loop, `c` points to the deepest available (i.e., unfolded) `Counter` for this `address`. Packet `p` is annotated with `c->fwd_rate` and `c->rev_rate`. After that, `c` is unfolded if either `c->fwd_rate` or `c->rev_rate` is higher than `_thresh`. `_thresh` is the C++-equivalent of Q. As long as both rates stay below `_thresh`, the `Counter` remains folded. The `i<3` part in the `if`-statement prevents the deepest level from being unfolded.

4.5 Unfolding

In an experiment, an early implementation of MULTOPS using memory-triggered folding was subjected to a bandwidth attack that involved IP source addresses from many different subnets. The MULTOPS had an imposed memory limit of M_{max}. Results show that the memory use of the MULTOPS fluctuates extremely within a small time scale. This behavior occurs regardless of the size of M_{max}, provided that the attack forces the MULTOPS to allocate more memory than M_{max}. The fluctuations are the result of the following sequence of events: if no records are eligible for folding, the process divides Q by 2 and tries again. After several iterations of not finding any records to delete, Q becomes so low that nearly every record in the tree suddenly becomes eligible for folding; the tree collapses. After that, the tree expands quickly due to incoming packets,

```
IPRateMonitor::handle_packet(int input_port, Packet *p)
{
    if(input_port == 0)
        update(p->src_address, p, true);
    else
        update(p->dst_address, p, false);
    output(input_port).push(p); // push packet on output input_port.
}

IPRateMonitor::update(char address[4], Packet *p, bool forward)
{
    struct Stats *s = _base;
    struct Counter *c;

    // Descend into MULTOPS tree while updating rates.
    for(int i=0; i<4; i++) {
        char x = address[i];
        if(!(c = s->counter[x]))
            c = make_counter(s->counter, x)

        if(forward)
            c->fwd_rate.update();
        else
            c->rev_rate.update();

        if(!c->next_level)
            break;
        s = c->next_level;
    }

    annotate_packet(p, c->fwd_rate, c->rev_rate);

    // Unfold if necessary
    if((c->fwd_rate >= _thresh || c->rev_rate >= _thresh) && (i < 3)) {
        c->next_level = new Stats();
        c->next_level->_parent = c;
    }
}
```

Figure 4-4: Code executed for each packet

violates the memory limit again, and the cycle starts anew. These extreme fluctuations occur for less aggressive methods of lowering Q, too.

The solution to this problem is to never unfold a record if that would violate the memory limit. This policy is a variation of memory-triggered folding and is called the "no-grow" policy. `IPRateMonitor` follows the no-grow policy. This introduces a problem, though: the tree can no longer unfold any record after it has reached its memory limit. This condition is called "no-grow lockup". The next section looks at possible solutions against no-grow lockup.

4.6 Folding

In `IPRateMonitor`, the no-grow policy is combined with time-triggered folding. Thus, no memory is allocated if that would violate the imposed memory limit and the MULTOPS is folded every N ms whatever its state is. If a no-grow lockup occurs, it lasts no longer than N ms. Furthermore, there is no risk of choosing N too large because memory will never run out. $N = 1000$ ms in `IPRateMonitor`.

The nodes in the tree are kept in a linked list, hence `_prev` and `_next` in each `struct Stats`. During folding, the process traverses this list of nodes in search of eligible records (i.e., records for which $T(S)$ and $F(S)$ are both lower than Q). In an initial implementation, the linked list was ordered such that the most recently updated nodes were in the front of the list and the longest untouched nodes were in the back of the list. This strategy is based on the assumption that the longest untouched nodes were most likely to contain records eligible for folding and could, therefore, be found in the back of the list. Keeping the list ordered in this fashion required an update in the linked list for each packet. The total overhead induced by keeping the list ordered was bigger than the time gained during folding. For that reason, the current implementation of `IPRateMonitor` simply appends new nodes at the end of the (unordered) list. Compacting traverses the linked list and inspects each node on records eligible for folding. If no records are eligible for folding, then Q is decreased by 5% and searching starts again.

To avoid heavy fluctuations in memory use (as described in section 4.5), folding stops when a certain fraction f of allocated memory has been freed. To avoid that nodes in front of the list are more likely to get folded than nodes at the back of the list, the process randomly traverses the list in forward or backward direction. When MULTOPS memory use is at its imposed maximum and suddenly no more packets arrive, its memory use will decrease by a fraction f every N ms. The graph in Figure 4-5 shows the drop in memory use after 1000 attackers from 100 subnets cease their bandwidth attack. The y-axis shows memory use in bytes and the x-axis shows time in seconds. At $t = 5$ the attack starts, at $t = 10$ the attack stops. $N = 1000$ and $f = 10$, i.e., every second 10% of memory used by `IPRateMonitor` is freed.

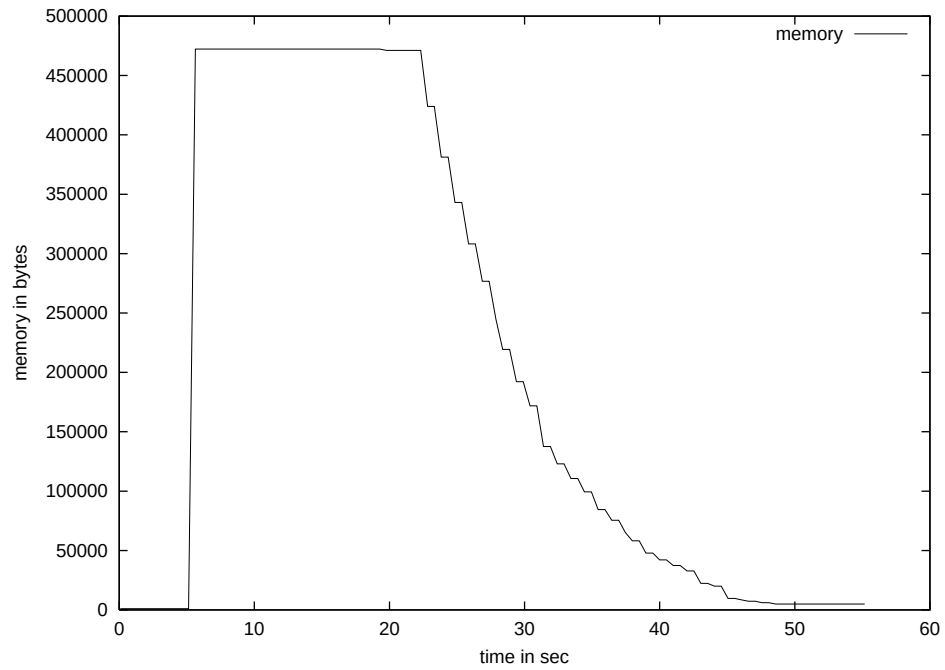

Figure 4-5: Memory use of IPRateMonitor as a result of folding

4.7 Memory exhaustion attacks

Section 3.8 sketches a scenario in which an attacker tries to run a MULTOPS-equipped router out of memory by forcing the MULTOPS to branch profusely. This section explains how `IPRateMonitor` deals with such attacks.

To keep memory consumption as low as possible, `IPRateMonitor` only allocates records (`Counter structs`) and nodes (`Stats structs`) when necessary. Implementing a `Stats` as an array of `Counter *` pointers as opposed to an array of `Counter` reduces the size of `Stats` from 7180 bytes to 1040 bytes. Although keeping the data structure small is desirable for many different reasons, it only stretches the time before a router runs out of memory. It does not prevent memory exhaustion.

Since nodes require more memory than records, the most effective way to run `IPRateMonitor` out of memory is by forcing it to allocate many nodes. Assume that the attacker has infinite resources and is not constrained by any ingress/egress filtering. The attacker runs the following program:

```
for(l=0; l<=255; l++)
  for(k=0; k<=255; k++)
    for(j=0; j<=255; j++)
      for(i=0; i<=255; i++) {
```

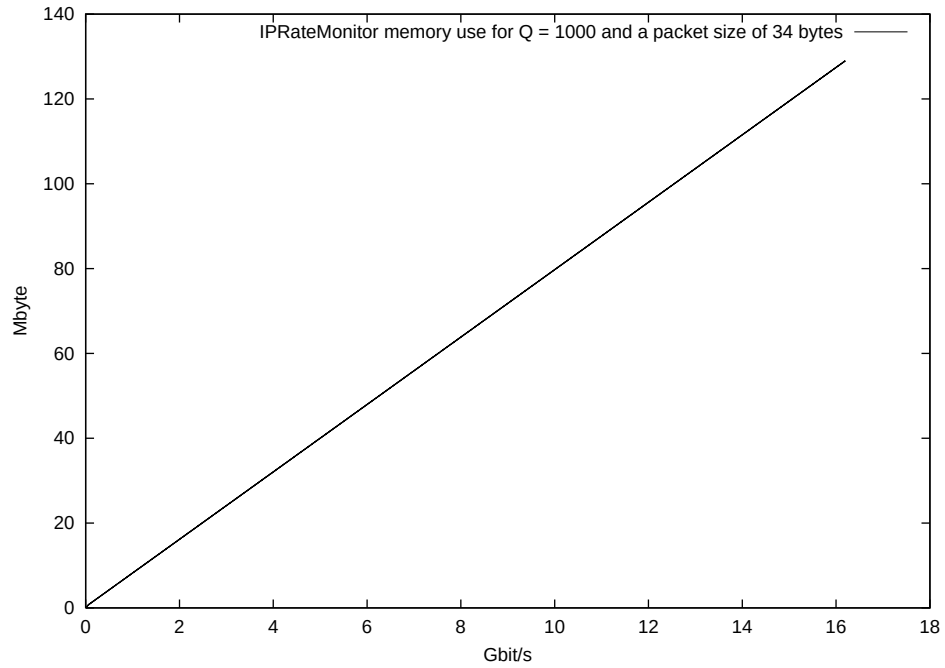

Figure 4-6: Relation between bandwidth and memory use

```
    fork(program sending packets with source address i.j.k.l at rate Q+1);
    if(--max_addresses == 0)
      exit(0);
  }
```

Each iteration of the innermost loop starts a program that keeps sending IP packets with IP source address $i.j.k.l$ at rate of $Q + 1$ packets per second through the attacked router. After forking off `max_addresses` programs, the loop stops. This algorithm most effectively maximizes the number of nodes in the MULTOPS. The amount of memory (M) the MULTOPS allocates in the attacked router, as function of i, j, k, and l, is:

$$M = n + \begin{cases} l = k = j = 0 & : & c_0(i) & = (4r + 3n)(i + 1) \\ l = k = 0 & : & c_1(j) & = c_0(256) + (3r + 2n)(256(j - 1) + (i + 1)) \\ l = 1 & : & c_2(k) & = c_1(256) + (2r + n)(256^2(k - 1) + 256j + (i + 1)) \\ & : & c_3(l) & = c_2(256) + r(256^3(l - 1) + 256^2k + 256j + (i + 1)) \end{cases}$$

where n is the size of a `Stats` (1040 bytes) and r is the size of a `Counter` (28 bytes).

The graph in Figure 4-6 shows that an attacker, running the above algorithm, needs to generate spoofed packets at a bandwidth of roughly 16 Gbit/s to make `IPRateMonitor` allocate 128MB of memory, provided that the network has the physical capability to carry this traffic to the target

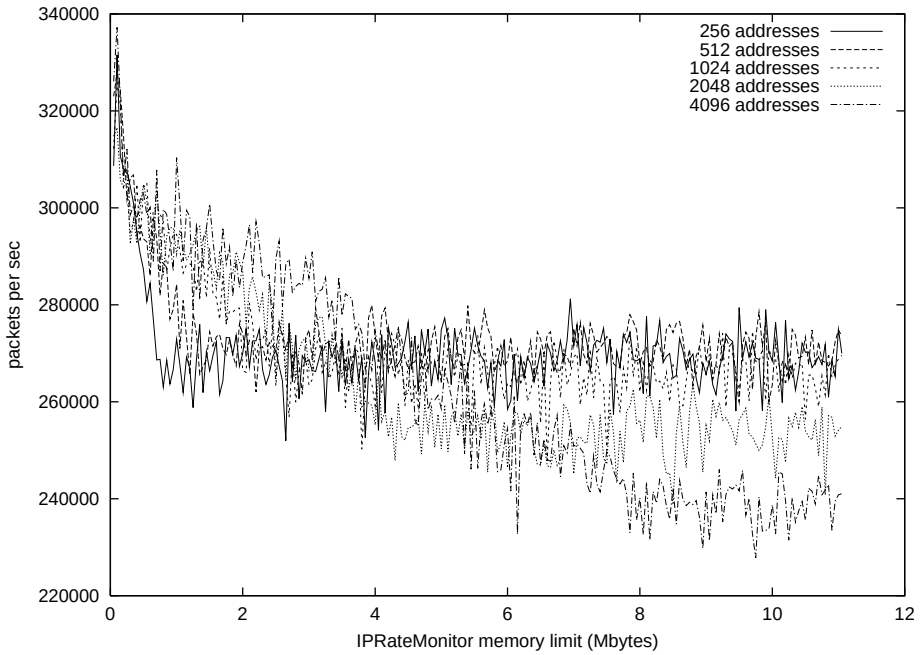

Figure 4-7: Packet rate as a function of memory limit

router. The graph is plotted for a packet size of 34 bytes and for $Q = 1000$. Given a maximum available bandwidth and a memory limit in a router, Q can be set to a value that ensures memory never to run out. It is safe to conclude that it is impossible to run `IPRateMonitor` out of memory.

4.8 Measurements

To measure performance of `IPRateMonitor`, a simple Click configuration was run in a Linux kernel 2.2.16 on an off-the-shelf PC (Pentium III, 700Mhz, 256 KB cache, 256 MB memory) that sends packets through an `IPRateMonitor` element that uses the packets' IP source address to index in the MULTOPS. Bogus UDP/IP packets were generated by Click itself to avoid interaction with network cards. Appendix A shows the Click configuration that was used for these measurements. `RoundRobinUDPIPEncap` is initialized with a set of IP addresses that it traverses in round-robin fashion to use as IP source address on UDP/IP packets it generates. This simulates an IP spoofing attacker. `IPRateMonitor` is initialized with different values for `MEM_LIMIT`, and always with $Q = 0$, i.e., maximum unfolding.

The graph in Figure 4-7 shows the number of packets that `IPRateMonitor` can handle as a function of its imposed memory limit. The graph shows 5 lines, each representing the number of different IP source addresses on the packets flowing through `IPRateMonitor`. The actual addresses

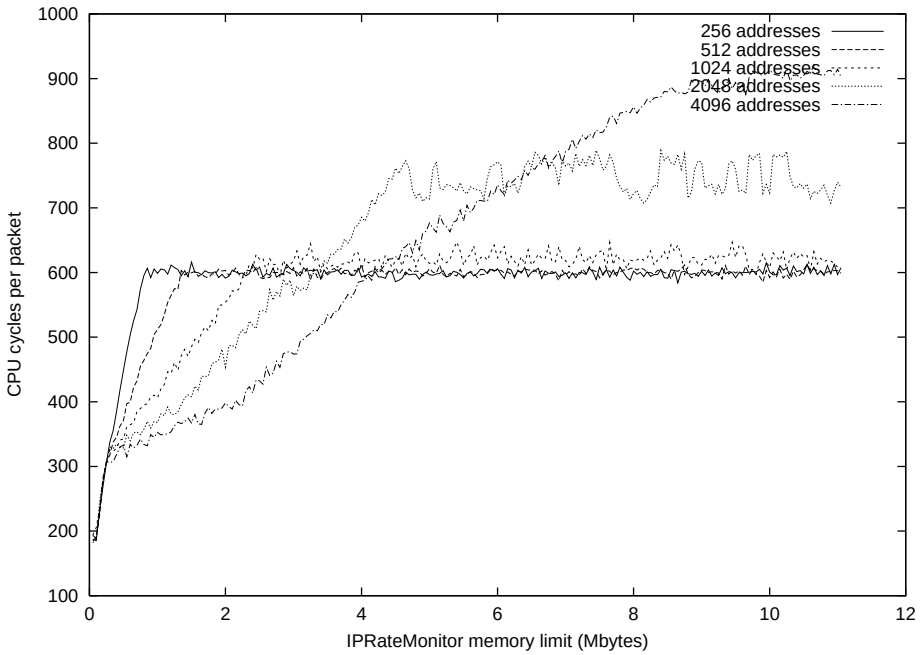

Figure 4-8: CPU cycles per packet as a function of memory limit

used on the UDP/IP packets are determined by the algorithm in section 4.7. For example, the line labeled "4096 addresses" shows that `IPRateMonitor` can handle roughly 240,000 packets per second when it has 10 MB of memory at its disposal. The packets come from the same 4096 different subnets as packets that are generated by the algorithm in section 4.7 with an initial value of 4096 for `max_addresses`.

From Figure 4-7 it is clear that `IPRateMonitor` can handle more packets when its available memory is less. A small MULTOPS fits in cache entirely and is, therefore, fast. As the amount of available memory grows, so does the size of the MULTOPS, which makes it too big to fit in cache entirely and cache misses result. The performance of `IPRateMonitor` for 256, 512, and 1024 addresses is roughly the same (270.000 packets/sec). In these cases, the tree is small enough to fit in cache entirely. For 2048 and 4096 addresses, we observe that rates keep dropping up to a certain, stable point. In both cases, the tree grows past the capacity of the cache. As a result, its performance drops proportional to the amount of memory used by `IPRateMonitor`.

The graph in Figure 4-8 shows the number of CPU cycles that `IPRateMonitor` consumes per packet as a function of its imposed memory limit. The different lines have the same meaning as in Figure 4-7. An `IPRateMonitor` consumes more CPU cycles when it is given more memory. Most likely, most of these cycles are spent waiting for a memory fetch after a cache miss.

`IPRateMonitor`'s performance is better when it has little memory at its disposal. Unfortunately,

its ability to unfold records and, therefore, to precisely determine the source(s) of the attack, is also limited. Improving the speed of `IPRateMonitor` can be achieved by making the tree consume less memory. Section 6 proposes a method to achieve this.

Chapter 5

Detecting bandwidth attacks with MULTOPS

We explore the ability of a MULTOPS to detect bandwidth attacks using the attack classification presented in section 2.7.

Consider the setup in Figure 5-1. S is a Web server, A is a single attacker or a group of attackers on different networks—this will be specified per example. R is a router. The fat lines represent links with a higher capacity than the link that is represented by the thin line. R has some sort of packet dropping mechanism based on MULTOPS. C is a benign client or a group of benign clients on different networks.

5.1 Flood and D-Flood

In Flood and D-Flood bandwidth attacks, the attacker sends packets using an unadaptive protocol such as UDP or ICMP. The attacker does not spoof IP addresses.

Suppose that A starts her bandwidth attack by sending ICMP ECHO REQUEST packets to S at a rate that saturates the link between R and S. As a result, most packets coming back from S get dropped in R; C's browser freezes. R observes an asymmetry in the packet rates from A and to A, and an asymmetry in the packet rates from C and to C. Consequently, R concludes that both A and C are attackers and, therefore, drops their packets. In reaction to that, TCP in C backs down because it sees no ACKs coming from S. Because C is no longer sending packets, symmetry is re-established and R stops blocking packets from C. A, on the other hand, keeps sending packets and, with that, maintains the asymmetry.

If A is a distributed attacker launching a D-Flood from N different networks, then each client generates $1/N$ of the required bandwidth. R will observe asymmetries in the packet rates to and from each of those N networks. Consequently, packets from those networks are dropped. Conclusion: a MULTOPS is very well suited for stopping Flood and D-Flood bandwidth attacks.

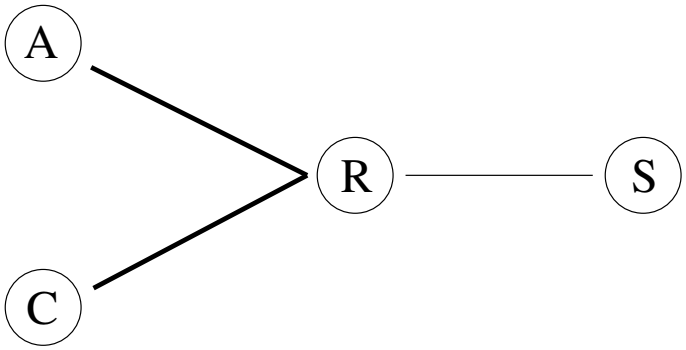

Figure 5-1: Example bandwidth attack setup

However, many factors play a role in detecting D-Flood attacks: the value of N, how much benign traffic there is from those N networks, and the value of R_{min}. If N is high, the relative volume of malicious traffic from that network is small, and, therefore, the asymmetry harder to detect. Similarly, if a relatively large share of traffic from a subnet is benign, it is hard to detect a maliciously caused asymmetry. The closer R_{min} is to 1, the more sensitive the router is to asymmetries, but the higher the chance for false positives.

5.2 Flood+

In Flood+ bandwidth attacks, the attacker sends packets using an unadaptive protocol such as UDP or ICMP. The attacker spoofs IP addresses, which is only possible if she is not behind an ingress/egress filtering edge router.

The attacker can launch a successful Flood+ bandwidth attack if she uses C's address for spoofing. R observes an asymmetry from (what R thinks is) C and subsequently drops all packets coming from C, including packets that are really sent by C.

A router in outer-protection mode deployed on the edge of A's network will detect the attack on S because that router observes the asymmetry between packets going to S (many) and packets coming back from S (few or none at all). Conclusion: a MULTOPS-equipped router can only stop Flood+ attacks if deployed on (or close to) the attacker's network. More generally, a distributed approach, i.e., having MULTOPS-equipped routers spread out over the entire Internet, increases the ability of the system as a whole to detect distributed bandwidth attacks that use IP spoofing.

5.3 Stealth

In Stealth bandwidth attacks, the attacker sends packets using an adaptive protocol; most likely TCP. The design of MULTOPS makes it unsuitable to detect this type of attack.

An attacker can open many normal TCP connections to a server and let each connection download a big file. If the resulting traffic constitutes an overload, then all TCP connections will slow down, including benign TCP connections to that server. Deploying MULTOPS-equipped routers on the attacker's network does not stop the attack either, since this attack causes no asymmetry.

There are several ways to adapt a system with a MULTOPS to make it more suitable for detecting Stealth bandwidth attacks.

- Rate-limit certain clients when total bandwidth nears threshold.

  Instead of looking at ratios only, a MULTOPS could easily keep track of the total bandwidth going through the router. As soon as the total bandwidth exceeds a certain threshold, the router can rate-limit clients that consume most bandwidth, the clients that have most connections, the clients that show most asymmetric behavior, or even based on the payload of packets. The aggressiveness of this rate-limiting can grow as the total bandwidth nears the maximum bandwidth. This strategy is similar to RED [19] and turns MULTOPS into a dropping policy based on packet rate (a)symmetry.

- Create a server API.

  A router could ask a server how it is doing. In the event that the server lets the router know it cannot cope with all the traffic it receives, the router can rate-limit or completely drop traffic to that server. Of course, communication between routers and servers needs to go over a different link than the one under attack.

- Communication between routers.

  Whether or not a suspected victim is really under attack can be determined with greater precision if MULTOPS-equipped routers can send (parts of) their MULTOPS to other routers who compare this data to the contents of their own MULTOPS. This introduces several problems, though. Communication cannot be done through flooded links, and attackers can forge this data and send it to routers, too. Out-of-band communication provides a solution to both problems.

5.4 D-Stealth/Flashcrowd

In D-Stealth bandwidth attacks, the attacker sends packets using an adaptive protocol; most likely TCP. The attacker does not spoof IP addresses. Because a MULTOPS searches for asymmetries

to detect attacks, it is not suitable to detect attacks that are mounted using an adaptive protocols such as TCP.

Unlike a Stealth attack, a MULTOPS-equipped router deployed on the victim's network cannot detect the attack, since traffic flows are symmetric and, therefore, inconspicuous. A Flashcrowd is similar to a D-Stealth attack, but more distributed than a D-Stealth attack.

5.5 SYN bomb

SYN bombs can be stopped by a router because each SYN requires a SYN/ACK to be sent back, which is the kind of symmetry a MULTOPS expects. If an attacker keeps sending SYN packets without any SYN/ACKs coming back, the attack becomes similar to a Flood. The capability of a MULTOPS to detect a SYN bomb strongly depends on the values of R_{min} and R_{max}, though. A short explosion of SYN packets may be enough to run a server out of TCP control blocks, but may be too short for the MULTOPS to detect the asymmetry.

Chapter 6

Future work

We suggest that the following issues are further explored.

Make unfolding R-triggered, not Q-triggered. Currently, a record unfolds if the packet rate exceeds threshold Q. A single packet rate is not an indication for an attack, though; asymmetry is. Thus, if packet rates are asymmetric, the relevant records should be unfolded. Care has to be taken, though, that this change does not make the MULTOPS more vulnerable to memory exhaustion attacks.

Look into application-dependent and level-dependent thresholds. By making Q, R_{min}, and R_{max} application dependent, a router's overall sensitivity to attacks can be improved. In addition, since average packet rates tend to be lower in deeper levels of the tree, making thresholds level-dependent is an obvious improvement.

Handle delayed ACKs correctly. TCP waits a short time between receiving a packet and sending an ACK. If, in that period, one or more packets come in, then TCP acknowledges only this last packet. This invalidates the assumption that traffic rates are symmetric. This problem can be solved by setting R_{min} and R_{max} such that the MULTOPS accepts asymmetric connections. Unfortunately this will also severely degrade the sensitivity to attacks.

Look into different folding strategies. Currently, the whole tree is searched for eligible records. Searching stops when a certain fraction f of memory is freed. If the attacker manages to blow up the tree faster than folding can reduce it, then memory exhaustion might result. Fraction f and threshold Q should increase as the total memory use gets closer to its imposed maximum to make folding more aggressive. Another (additional) method is to make the frequency of time-triggered folding increase as the total memory use gets closer to its imposed limit.

Do more on Stealth bombs. Stealth bombs are a MULTOPS' Achilles heel. Section 5.3 proposes several ideas to make MULTOPS better in detecting Stealth bombs, but more research is needed.

Improve performance. To improve cache performance, level 0 and level 1 of the tree can be combined into a two dimensional array where the row is chosen based on the first byte of an IP address and the column based on the second byte of an IP address.

IPv6. With small adaptations, `IPRateMonitor` can be made suitable for IPv6. Its memory requirements are different, though.

Asymmetric routes. If routes are not symmetric, then routers need to exchange information to find asymmetries. For example, if a network has one router for all incoming traffic and one router for all outgoing traffic, then these routers need to exchange data to detect asymmetries.

Chapter 7

Conclusion

This thesis proposes a bandwidth attack detection mechanism wherein routers monitor traffic and consider a subnet to be under attack when the packet rate coming from a subnet is disproportional to the packet rate going to that subnet. A MUlti-Level Tree for Online Packet Statistics (MULTOPS) is a data structure suitable for keeping track of packet rates to and from different subnets on different aggregation levels. A MULTOPS can be used to detect incoming or outgoing bandwidth attacks.

A MULTOPS is suitable for detecting bandwidth attacks that are mounted using unadaptive protocols such as UDP and ICMP. This covers all February 2000 attacks. To our knowledge, no such detection mechanism has been proposed yet. In some cases a MULTOPS is suitable to detect bandwidth attacks that are mounted using adaptive protocols such as TCP. Depending on the attack, a MULTOPS can point out the sources of the attack. It is exceedingly difficult to run a MULTOPS-equipped router out of memory.

Measurements show that the performance of a MULTOPS is primarily influenced by the size of the cache and the numbers of IP source addresses involved in the attack.

Chapter 8

Acknowledgements

First and foremost, I wish to thank Max Poletto for his enduring patience and his constructive advice. Thank you, Frans Kaashoek, for your critical hieroglyphs and for giving me this tremendous opportunity. I wish to thank Andy Tanenbaum for always gently nudging me in the right direction. Thank you for your guidance, your efforts, your unflagging support, and your long and frequent e-mails. I wish to thank my father, Jossi Gil, for being a knowledgeable and critical listener. Thanks for believing in me, even when I did not. Furthermore, I thank Chuck Blake, Mama (Deborah Günzburger), Jinyang Li, David Mazières, Marcel van den Broecke, Robert Morris, David Karger, and all members of the MIT/LCS/PDOS group for their helpful input. Many ideas that are presented in this thesis were drained from the brains of Benjie Chen, Eddie Kohler, and Max Poletto. Thank you for your ideas and for the things you taught me.

Ellen, Klarke, and Yonathan: thank you for being my trinity during the past few months and years. You made all the difference. Thank you, Hannah, for being here. Thank you, Lotte and Floortje, for your friendship and for your wake-up calls in the morning.

Last, but certainly not least, I say murp to Lovely Lady "Twiga" Lisa, who fills me with happiness, joy, and pleasure. You turned Cambridge, Massachusetts into a place where true miracles happen. נס גדול היה פה. Thank you.

> Then he prayed, "... give me success today, and show kindness to my master Abraham. See, I am standing beside this spring, and the daughters of the townspeople are coming out to draw water. May it be that when I say to a girl, 'Please let down your jar that I may have a drink,' and she says, 'Drink, and I'll water your camels too'—let her be the one you have chosen for your servant Isaac. By this I will know that you have shown kindness to my master."
>
> Before he had finished praying, Rebekah came out with her jar on her shoulder. (Genesis 24:12-15)

Bibliography

[1] CNN. Cyber-attacks batter Web heavyweights, February 2000. Available at `http://www.cnn.com/2000/TECH/computing/02/09/cyber.attacks.01/index.html`.

[2] CNN. 'Immense' network assault takes down Yahoo, February 2000. Available at `http://www.cnn.com/2000/TECH/computing/02/08/yahoo.assault.idg/index.html`.

[3] Netscape. Leading Web sites under attack, February 2000. Available at `http://technews.netscape.com/news/0-1007-200-1545348.html`.

[4] Netscape. Slashdot struck by denial-of-service attacks, May 2000. Available at `http://technews.netscape.com/news/0-1003-200-1889595.html`.

[5] Packetstorm. Packetstorm, 2000. Available at `http://packetstorm.securify.com/`.

[6] Lance Spitzner. The Tools and Methodologies of the Script Kiddie. Know Your Enemy, 2000. Available at `http://www.enteract.com/~lspitz/enemy.html`.

[7] Dave Dittrich. The DoS Project's 'trinoo' distributed denial of service attack tool. Technical report, University of Washington, 2000. Available at `http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt`.

[8] David Mazières and M. Frans Kaashoek. The design, implementation and operation of an email pseudonym server. *Proceedings of the 5th ACM Conference on Computer and Communications Security*, 1998.

[9] CERT Coordination Center. CERT Advisory CA-98.01 "smurf" IP Denial-of-Service Attacks, 1998. Available at `http://www.cert.org/advisories/CA-98.01.smurf.html`.

[10] The electrohippies collective. Client-side Distributed Denial-of-Service, 2000. Available at `http://www.gn.apc.org/pmhp/ehippies/files/op1.pdf`.

[11] P. Ferguson et. al. RFC 2267. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoof-

ing. Technical report, The Internet Society, 1998. Available at http://sunsite.cnlab-switch.ch/ftp/doc/standard/rfc/22xx/2267.

[12] SANS Institute. Egress filtering v 0.2, 2000. Available at http://www.sans.org/y2k/egress.htm.

[13] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical Network Support for IP Traceback. Technical report, Department of Computer Science and Engineering, University of Washington, 2000.

[14] Thamer Al-Herbish. Secure Unix Programming FAQ, 1999. Available at http://www.whitefang.com/sup/secure-faq.html.

[15] Cisco Systems. Characterizing and Tracing Packet Floods Using Cisco Routers. Available at http://www.cisco.com/warp/public/707/22.html.

[16] Rob Malda. The Slashdot DDoS. What Happened?, 2000. Available at http://slashdot.org/articles/00/05/17/1318233.shtml.

[17] Bellovin. ICMP Traceback Messages. Technical report, AT&T, 2000. Available at http://www.ietf.org/internet-drafts/draft-bellovin-itrace-00.txt.

[18] Robert Morris, Eddie Kohler, John Jannotti, and M. Frans Kaashoek. The Click modular router. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP '99)*, pages 217–231, Kiawah Island, South Carolina, December 1999.

[19] Sally Floyd, Kevin Fall, and Kinh Tieu. Estimating Arrival Rates from the RED Packet Drop History. Available at http://www.aciri.org/floyd/papers/red-dropping.ps.

Appendix A

Click config file for performance measurement

```
is :: InfiniteSource(<00000000111111112222222233333333444444445555>,
      5000000, 1);
udpgen :: RoundRobinUDPIPEncap(
0.0.0.0 1234 1.0.0.2 1234 1,
1.0.0.0 1234 1.0.0.2 1234 1,
2.0.0.0 1234 1.0.0.2 1234 1,
...
254.0.0.0 1234 1.0.0.2 1234 1,
255.0.0.0 1234 1.0.0.2 1234 1,
0.1.0.0 1234 1.0.0.2 1234 1,
1.1.0.0 1234 1.0.0.2 1234 1,
2.1.0.0 1234 1.0.0.2 1234 1,
...
254.1.0.0 1234 1.0.0.2 1234 1,
255.1.0.0 1234 1.0.0.2 1234 1,
... etc...
);
is -> udpgen ->
CycleCount(0) ->
rm :: IPRateMonitor(PACKETS, 0, 1, 0, <MEM_LIMIT>) ->
CycleCount(1) ->
st :: StoreCycles(0,1) ->
co :: Counter -> Discard;
```