MULTOPS: a data-structure for bandwidth attack detection

Thomer M. Gil and Massimiliano Poletto
Vrije Universiteit, Amsterdam, The Netherlands
and
M.I.T., Cambridge, MA, USA
{thomer,maxp}@lcs.mit.edu

Abstract

A denial-of-service bandwidth attack is an attempt to disrupt an online service by generating a traffic overload that clogs links or causes routers near the victim to crash. We propose a heuristic and a data-structure that network devices (such as routers) can use to detect (and eliminate) such attacks. With our method, each network device maintains a data-structure, *MULTOPS*, that monitors certain traffic characteristics. MULTOPS (MUlti-Level Tree for Online Packet Statistics) is a tree of nodes that contains packet rate statistics for subnet prefixes at different aggregation levels. The tree expands and contracts within a fixed memory budget.

A network device using MULTOPS detects ongoing bandwidth attacks by the significant, disproportional difference between packet rates going to and coming from the victim or the attacker. MULTOPS-equipped routing software running on an off-the-shelf 700 Mhz Pentium III PC can process up to 340,000 packets per second.

1 Introduction

A bandwidth attack is an attempt to disrupt an online service by generating a traffic overload that clogs links or causes routers near the victim to crash. This can have serious consequences for Web companies which rely on their online availability to do business. This paper introduces a data-structure that routers and network monitors can use to collect packet rate statistics for subnet prefixes at different aggregation levels. These statistics can be used to detect bandwidth attacks using a simple heuristic: a significant, disproportional difference between the packet rate going to and coming from a host or subnet. This heuristic is based on the assumption that, during normal operations on the Internet, the packet rate of traffic going in one direction is proportional to the packet rate of traffic going in the opposite direction. Although this assumption does not hold in some cases, it is a close approximation to reality.

Bandwidth attacks are typically distributed attacks. An attacker uses tools to gain root access to machines on the Internet [Pac00, Spi00]. Once a machine is cracked, it is turned into a "zombie." The attacker instructs the zombies to send bogus data to one particular destination [Dit00]. The resulting traffic can clog links, cause routers near the victim or the victim itself to fail under the load.

One major reason underlies the absence of a simple solution against bandwidth attacks: attackers can release high volumes of normal-looking packets on the Internet without being conspicuous or easily traceable. It is the mass of all packets together directed at one victim that poses a threat, rather than any characteristics of the individual packets. A dropping policy in routers based on per-packet characteristics will, therefore, not work.

It is relatively easy, but rather useless, to detect a bandwidth attack in the vicinity of the victim: by measuring the traffic load on a link or in a router, the exceptionally high volume of packets can be detected. Unfortunately for the victim, determining that it is under attack will not make the packets go away. Harm has already been done by the time the malicious packets reach (the vicinity of) the victim. A bandwidth attack should, therefore, be detected close to the attacker rather than close to the victim so that malicious packets can be stopped before they can cause any harm.

This paper proposes a *MUlti-Level Tree for On-*

line Packet Statistics (MULTOPS). MULTOPS enables routers or network monitors to detect ongoing bandwidth attacks. A handful of attackers that blast packets to a victim without any (or disproportionally fewer) packets coming back will be identified as malicious by MULTOPS. Large attacks that occurred in February 2000 [CNN00a, CNN00b, Net00] displayed these disproportional packet flows. Routers (or network monitors) using MULTOPS could have been used to stop (or detect) those attacks.

MULTOPS is a tree of nodes that contains packet rate statistics for subnet prefixes at different aggregation levels. It dynamically adapts its shape to (1) reflect changes in packet rates, and (2) avoid (maliciously intended) memory exhaustion.

Depending on their setup and depending on their location on the network, MULTOPS-equipped routers or network monitors may fail to detect a bandwidth attack that is mounted by attackers that randomize IP source addresses on malicious packets. In a different setup, MULTOPS-equipped routers may cause "collateral damage" by dropping legitimate packets with an IP destination address that MULTOPS identified as being under attack.

MULTOPS fails to detect attacks that deploy a large number of proportional flows to cripple a victim. (Proportional flows are flows in which the packet rate in one direction is proportional to the packet rate in the opposite direction.) For example, many attackers could open FTP or HTTP connections to one victim and download—preferably large—files over these connections, thereby overloading the victim. Even though the packet rates between the attackers and the victim are relatively low (because the victim cannot handle all the parallel downloads), they are proportional and, therefore, undetectable by MULTOPS. However, to successfully mount such an undetectable bandwidth attack, attackers need to be numerous, geographically distributed, and well organized. This makes it more difficult to mount an undetectable attack.

MULTOPS has been implemented in a software router and was tested with simulated attacks. Results are encouraging: attacks are stopped and legitimate traffic continues in a normal fashion, even with a large number of participating attackers. An off-the-shelf 700 Mhz Pentium III PC, running MULTOPS-equipped routing software, routes between 240,000 to 340,000 packets per second, depending primarily on the resources available to MULTOPS.

The rest of this paper is organized as follows. Section 2 takes a look at related work, Section 3 looks at different types of bandwidth attacks, Section 4 explains the design of MULTOPS, Section 5 looks at the details of the MULTOPS implementation, Section 6 deals with measurements, Section 7 discusses the details of some (unresolved) issues, and Section 8 concludes this paper.

2 Related work

Most of the techniques proposed so far for protection against denial-of-service attacks can be used in conjunction with MULTOPS. We quickly review the major techniques and how MULTOPS can augment them.

Ingress/egress filtering is a technique performed by routers to effectively eliminate IP spoofing [ea00, Ins00]—lying about one's own IP address in the header of outgoing IP packets. To stop spoofed IP packets, edge routers match the IP source address of each outgoing packet against a fixed set of known IP address prefixes. If no match is found, the packet is dropped. Another possible technique is for a router to only send off a packet from interface i if a potential reply to this packet is, according to the router's routing tables, expected to arrive on interface i. If not, the packet is dropped. Even though these techniques are simple and effective remedies against IP spoofing, unfortunately many routers are not configured to deploy these techniques and they are not complete solutions. However, MULTOPS benefits from them because IP spoofing hurts MULTOPS' ability to detect attacks (see also Section 7.1).

IP Traceback assists in tracking down attackers post-mortem [SWKA00, SP01, DFS01]. This technique requires routers to, with a low probability, mark packets such that the receiving end can reconstruct the route that packets followed, provided enough packets were sent. A similar technique is ICMP Traceback [Bel00]. When forwarding packets, routers can, with a low probability (1/20,000), generate an ICMP 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. The main advantage of these techniques is that it assists

in finding attackers. It does not stop them.

All the traceback approaches have serious deployment and operational challenges. A sufficient number of routers need to support traceback before it is effective. Attackers can generate traceback messages too, so some form of authentication of traceback messages is necessary. The victim of a bandwidth attack might also not receive enough traceback messages because they might get dropped by overloaded routers. In addition, if an attack is very distributed, there may not be enough traceback information to find the attackers.

A number of routers provide information about packets that can be used to implement the same detection heuristic that MULTOPS is using. Cisco routers, for example, support RMON [Cisb] and Netflow [Cisa]. Unfortunately, both RMON and Netflow data is expensive to process off-line. RMON copies complete packets to a port for off-line analysis—this slows down the router's normal operation. Netflow keeps a table with 45-byte entries for every flow, which can be queried by and transferred to an external analysis program. Netflow provides no protection against attackers that might blow up the table. In the worst case, RMON and Netflow can magnify an attack. MULTOPS is intended to be integrated into a router or a monitoring device for on-line analysis. MULTOPS also runs in a fixed-size memory footprint so that attackers cannot run a MULTOPS device out of memory.

Stone [Sto99] proposes CenterTrack, an overlay network that consists of IP tunnels which can be used to selectively reroute packets from routers on a network to special "tracking" routers. This architecture can be used to analyze traffic for signs of a bandwidth attack, and optionally drop traffic that seems suspicious. MULTOPS could probably be used as a component of CenterTrack to help routers determine whether a bandwidth attack is occurring and what IP addresses are involved.

Bellovin [Bel01] discusses aggregate congestion control and pushback. The central idea is to identify "aggregates"—subsets of traffic defined by some characteristic, such as a particular destination address—that may be involved in the bandwidth attack. Pushback is a cooperative mechanism in which routers can ask adjacent routers to block an aggregate upstream. MULTOPS could be viewed as a data-structure for efficiently tracking the aggregate defined by IP addresses for which traffic flow is asymmetric.

Intrusion detection system such as Bro [Pax99] try to detect attacks by monitoring network links over which the attacker's traffic transits. Armed with (statistical) knowledge about normal behavior of different applications and protocols, these systems detect anomalies in traffic patterns and report a wide range of attack types. Although similar to MULTOPS in that it monitors traffic, the primary difference is that these systems do not attempt to stop attacks.

3 Bandwidth attacks

The common denominator of all bandwidth attacks is the desire to cripple someone else's infrastructure by generating a traffic overload. Bandwidth attacks vary, among other things, in the protocol being used to mount the attack. In addition, attackers can use IP spoofing. As mentioned above, IP spoofing is lying about one's own IP address.

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 forged 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 7.1 discusses how IP spoofing affects MULTOPS' ability to detect (the source(s) of) attacks.

An attacker can forge an ICMP packet with a spoofed IP source address and launch a "Smurf" attack [CC98]: he sends this *one* forged ICMP packet to a broadcast address and *all* the receivers respond with a reply to the spoofed IP address (the victim). (A solution is to never reply to ICMP packets that are sent on a broadcast address, or to let routers filter such packets [ea81, ea00].) In a "Fraggle" attack, an attacker instructs many zombies to send UDP packets to one victim. Both Smurf and Fraggle attacks can be detected by MULTOPS because in both cases the packet rate to the victim exceeds the packet rate coming back from the victim in a disproportional manner.

There are several types of attack that use TCP. The best known is "SYN Flooding" [CC96]. Several so-

lutions have been proposed for solving SYN Floods: lowering the TCP time-out, increasing the number of TCP control blocks, SYN cookies [AH99] that eliminate the need to store information on half-open connections, and special firewalls that buffer SYN packets. Although a SYN Flood is actually a resource attack, it is similar to a bandwidth attack because of the flood of SYN packets.

Another attack works by generating a huge amount of normal traffic by, for example, running a JavaScript program in a browser that pops up a few dozen windows each fetching a Web page from one server. This may constitute a problem if a few thousand people are willing to run this script in their browser simultaneously [ec00]. Such a script could easily spread by means of self-replicating e-mail viruses. (This phenomenon can also occur without it being an attack.)

As mentioned in Section 1, attacks that cripple a victim by sending or receiving a high volume of traffic using proportional flows may go unnoticed by MULTOPS.

4 MULTOPS design

4.1 Overview

MULTOPS uses disproportional packet rates to or from hosts and subnets as a heuristic to detect (and potentially stop) attacks. To collect these statistics, a tree-shaped data-structure keeps track of packet rates to and from those subsets of the IP address space that display disproportional behavior. This is done by letting the tree expand and contract ("zoom in and zoom out") based on observed (disproportional) traffic patterns.

MULTOPS stores packet rate statistics for flows between hosts (or subnets) A and B using either A's IP address or B's IP address. As a consequence, MULTOPS can either establish the victim, or the source(s) of the attack. We distinguish between these two modes by defining them as *victim-oriented* mode and *attacker-oriented* mode, respectively. In victim-oriented mode, MULTOPS tries to identify the IP address of the victim of an attack. In attacker-oriented mode, MULTOPS tries to identify the IP address(es) of the attacker(s). The difference between these two modes becomes important when dropping packets: either packets going to the victim are dropped, or packets coming from the attacker are dropped. Note that in both cases the attack is stopped. In one case this is done based on the IP address of the victim, in the other case it is done based on the IP address(es) of the attacker(s). Throughout this paper we assume that MULTOPS runs in victim-oriented mode, unless specified otherwise.

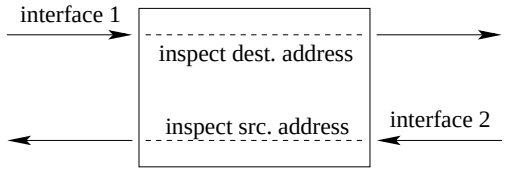

Figure 1: Schematic MULTOPS in victim-oriented mode

MULTOPS expects two streams of IP packets as input—each connected to a different network interface. Packets going in one direction ("forward packets") are inspected on their destination address; packets going in the opposite direction ("reverse packets") are inspected on their source address. Figure 1 illustrates this. Exchanging the network interfaces switches between attacker-oriented and victim-oriented mode.

MULTOPS presents a query interface that returns an approximation to $R(P)$. $R(P)$ is the ratio of forward packets with destination IP address prefix P to reverse packets with source IP address prefix P.

In victim-oriented mode, MULTOPS determines a victim's IP address by looking for prefixes for which $R(P)$ is greater than some threshold. Dropping packets with destination addresses matching such prefixes might defeat the attack, though it may also impose "collateral damage" by dropping legitimate packets. In attacker-oriented mode, MULTOPS determines the addresses of attackers by looking for prefixes for which $R(P)$ is less than some threshold. Dropping packets based on source addresses matching such prefixes might defeat the attack, though IP spoofing introduces complications that are discussed in Section 7.1. Note that a single MULTOPS cannot detect both attacker and victim addresses.

In our current design, we also assume that packets are being sent using IPv4. Our approach should easily extend to IPv6, although it will consume sig-

nificantly more resources.

4.2 MULTOPS heuristic

Packets are defined to be malicious (and, thus, may be dropped) if they are destined for a host or subnet from which too few packets are coming back. This heuristic is based on the assumptions that (1) most Internet traffic consists of packet flows, and (2) during normal operations, the rate of packets in a flow going from A to B is proportional to the packet rate going from B to A. Thus, during normal operations on the Internet, the packet rate of traffic going in one direction is proportional to the packet rate of traffic going in the opposite direction. If not, something must be wrong.

This heuristic appears to hold broadly. TCP, the protocol mainly used on the Internet, acknowledges every single—or every k—received packets by sending back a packet, and, therefore, has proportional packet flows.

The following example illustrates the heuristic. If machine A is sending legitimate TCP packets to machine B, but B is suffering under a bandwidth attack, then A's packets will not reach B. Even *if* some of A's packets reach B, then B's packets may not reach A because of the overloaded links and routers. In reaction to the absence of B's packets, A will automatically decrease the sending rate and, eventually, stop sending packets to B altogether. If, on the other hand, A is an attacker that blasts (any type of) packets at B, a MULTOPS-equipped router routing A's packets to B will detect the disproportional packet rates between them and could decide to drop packets going to B. Consequently, B will not have to cope with A's packets.

Let $R(P)$ be the ratio between the packet rate going to and coming from addresses with prefix P. Under normal circumstances, R is close to some constant k for all P, i.e., packet rates are proportional for all prefixes. If R drops below R_{min} or exceeds R_{max}, then a (host in) subnet with prefix P is either under attack or a subnet with prefix P harbors an attacker.

MULTOPS collects packet rates to and from address prefixes so that, given a certain P, $R(P)$ can be calculated. Packets may be dropped if they are destined for a host or subnet from which disproportionally fewer packets are coming back, i.e., if $R(P)$ is not between R_{min} and R_{max}. The sensitivity of MULTOPS can be tuned by changing the values of R_{min} and R_{max}.

4.3 Data structure

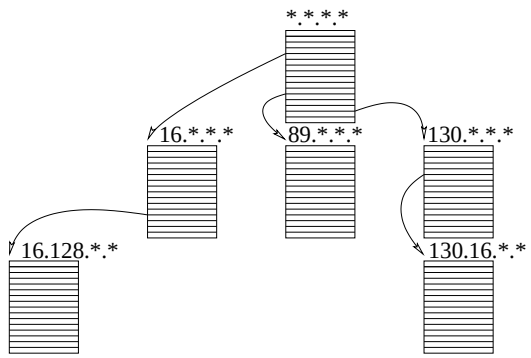

Figure 2: MULTOPS

MULTOPS is organized as a 4-level 256-ary tree to conveniently cover the entire IPv4 address space. Each node in the tree is a table consisting of 256 records, each of which consists of 3 fields: 2 rates—to-rate and from-rate—and 1 pointer potentially pointing to a node in the next level of the tree. A table stores all packet rates to and from IP addresses with a common 0-bit, 8-bit, 16-bit, or 24-bit prefix, depending on the level of the tree. Deeper levels of the tree contain packet rates for addresses with a longer prefix. Thus, the root node contains the aggregate packet rates to and from address 0.*.*.*, 1.*.*.*, 2.*.*.*, etc. The 90th record in the root node, for example, contains the packet rates to and from addresses with 8-bit prefix 89, and a pointer to a node that keeps tracks of the aggregate packet rates to and from addresses with that prefix, i.e., 89.0.*.*, 89.1.*.*, 89.2.*.*., etc. The sum of all 256 to-rates and the sum of all 256 from-rates in a node are equal to the to-rate and the from-rate in the parent record of that node. Figure 2 shows a sample MULTOPS.

When the packet rate to or from a subnet reaches a certain threshold, a new subnode is created on the fly to keep track of more fine-grained packet rates, potentially down to per-IP address packet rates. For example, if the aggregate packet rate to or from subnet 130.17.*.* exceeds R_{max}, a new node is created to keep track of packet rates to and from subnets 130.17.0.*, 130.17.1.*, etc. Creating new nodes is called *expansion*. The reverse, i.e., removing nodes

or entire subtrees, is called *contraction*. Contraction is done when the packet rate from and to a given IP address prefix drop below a certain threshold, or when memory is running out, possibly due to a memory exhaustion attack against MULTOPS itself.

Expansion and contraction enable MULTOPS to exploit the hierarchical structure of the IP address space and the fact that a bandwidth attack is usually directed at (or coming from) a limited set of IP addresses—with a common prefix—only. MULTOPS detects the attack on a high level in the tree (where prefixes are short) and expands toward the largest possible common prefix of the victim's IP address(es), potentially establishing single IP address(es) that are under attack.

4.4 Algorithm

Each packet (or every nth packet) that is routed causes packet rates in applicable nodes in the tree to be updated; starting in the root, and going down to the deepest available node. This works as follows. The first byte of the IP *destination* address of a *forward packet* is used as an index in the root node to find the record in which to update the *to-rate*. For *reverse packets* the first byte of the IP *source* address is used as an index in the root node to find the record in which to update the *from-rate*. If the record has a child, the process descends down to the child and continues. If no child exists, it is created if either the from-rate or the to-rate exceeds a certain threshold. In any case, the process may follow the pointer in the record to the child node. In this child node, the *second* byte of the IP address is used as an index to find the record and update the packet rates. This process may descend down to the deepest level in the tree where per-IP address packet rates are kept. The full algorithm is given in pseudo-code in Algorithm 4.1.

Method UPDATE() is called by method HANDLE_PACKET() described in Section 5.2. Parameter *addr* is the 4-byte IP source or destination address of packet *packet*, depending on whether MULTOPS is set up in victim-oriented or attacker-oriented mode. Parameter fwd tells UPDATE() whether this packet is a forward packet or a reverse packet. Statement 1 immediately after the **for** -loop annotates the packet with r's from-rate and to-rate. This annotation can later be used by a part of the system that implements the heuristic to determine whether or not this packet is part of a malicious flow and should, thus, be dropped.

Algorithm 4.1: UPDATE($addr, packet, fwd$)

```
TABLE t ← root
for i ← 1 to 4
  do { RECORD r ← t[addr[i]]
       if fwd
         then update r's to-rate
         else update r's from-rate
       if r has no child node
         then break
       t ← r's child node
annotate packet with r's from-rate and to-rate          (1)
if (r's from-rate > threshold
 or r's to-rate > threshold)
 and t is not a node in deepest level of tree
  then create child table t' under r
```

4.5 Expansion and contraction

If the to-rate or the from-rate for an address with an n-bit prefix P exceeds the *expand threshold*, MULTOPS creates a child node under the record for prefix P to keep track of packet rates for addresses with $(n+8)$-bit prefix P'. Lowering this expand threshold increases precision of MULTOPS, but also increases its memory use. Figure 3 shows how a new node is added to the tree to keep track of all packet rates to and from addresses with prefix 130.16.120.

The reverse of expansion is contraction. Contracting a record involves removing a subtree from under a record. A subtree is contracted when the aggregate packet rate for that subtree drops below R_{max}. Contraction is done to constrain memory use and to avoid (maliciously intended) memory exhaustion. Figure 3 shows how a node is contracted.

Traversing the entire tree in search of subtrees to contract is potentially expensive and its frequency should be chosen with care. Traversing the tree for every x routed packets is dangerous because a router should have its resources free for routing, not for contracting when packet rates go up. Traversing the tree every t ms is safer, but choos-

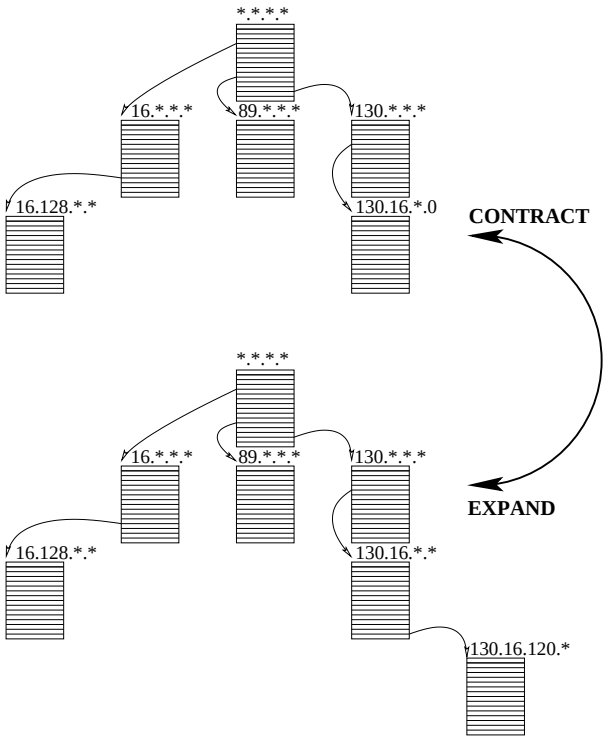

Figure 3: Expansion and contraction

ing t correctly is tricky: if t is too high, memory might run out before traversal starts. The strategy we chose is to never allocate more memory than a certain limit m—thereby making memory exhaustion impossible—and to traverse the tree every t ms in search of subtrees to contract. In the time period between reaching memory limit m and the next "cleanup," MULTOPS cannot create new nodes. It is, therefore, important to choose t low, but not so low as to trigger cleanups too often and, thus, waste the router's resources.

An attacker might try to launch a memory exhaustion attack against MULTOPS by causing it to branch profusely. The two opposing forces are the attacker causing nodes to be created versus contraction causing nodes to be destroyed. Since a subtree is contracted when the packet rates to and from addresses with a certain prefix are less than the expand threshold, the attacker will have to sustain a higher packet rate for as many different address prefixes as possible. Section 5.4 deals with this issue in a quantitative context.

5 MULTOPS implementation

MULTOPS is implemented using Click [KMC+00]. 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. Each element has 0 or more inputs and 0 or more outputs. Inputs are used to receive packets from other elements. Outputs are used to hand off packets to other elements. Configuration of a Click router is done by feeding it a file describing which elements to use and how the inputs and outputs of these elements interconnect.

MULTOPS is implemented as 2 separate elements: `IPRateMonitor` and `RatioBlocker`. Adding these elements to the configuration adds the MULTOPS detection mechanism and the related dropping policy to the router. `IPRateMonitor` tags each packet with from-rate and to-rate such that `RatioBlocker` may decide to drop the packet based on these tags and based on the defined thresholds (i.e., R_{min} and R_{max}). Thus, `IPRateMonitor` implements the tree, `RatioBlocker` implements a dropping policy based on the MULTOPS detection heuristic.

`IPRateMonitor` has 2 inputs and 2 outputs. Each input should be connected to a different physical network interface. `RatioBlocker` has 1 input and 1 output.

5.1 Data structure

`IPRateMonitor` is a C++ class that defines two private `structs`: `Record` and `Table`. Figure 5.1 contains the C++ code that defined these `structs`.

`from_rate` and `to_rate` in `Record` are used to store packet rates. `EWMA` implements an exponentially weighted moving average and is used to keep track of rates. `child` contains a pointer to a child or `NULL` if no child exists. Besides 256 pointers to `Record`, `Table` contains a pointer to the parent record (`parent`) and two pointers (`prev` and `next`) that are used to maintain a doubly-linked list of nodes—their use is explained in Section 5.3. `root` points to the root node.

```
struct Record {
  EWMA from_rate;
  EWMA to_rate;
  Table *child;
};

struct Table {
  Record *parent;
  Table *prev, *next;
  Record* record[256];
};

Table *root;          // root node
```

Figure 4: C++ code that defines `Record` and `Table`

5.2 Algorithm

`IPRateMonitor`'s method HANDLE_PACKET() (given in pseudo-code in Algorithm 5.1) implements the functionality represented by Figure 1. It is, together with method UPDATE(), responsible for implementing the algorithm described in Section 4.4.

Algorithm 5.1: HANDLE_PACKET(*port*, *packet*)

if port == 0
  then UPDATE(packet.dest_addr, packet, true)
  else UPDATE(packet.src_addr, packet, false)

`IPRateMonitor`'s 2 input ports should each be logically connected to one of the network interfaces. Port 0 connects to the interface for forward packets, port 1 connects to the interface for reverse packets. (This is achieved through Click configuration.) *port* is the input of the `IPRateMonitor` element that packet *packet* arrived on. This information is passed to UPDATE() using its fwd parameter.

5.3 Expansion and contraction

In addition to the tree itself, MULTOPS maintains a doubly-linked list of pointers to nodes in the tree using `prev` and `next` in `Table`. Each time a new node is created in the tree, i.e., expansion occurs, a pointer to that node is added at the end of the linked list. During a cleanup, the list is traversed. A node (and all its children) is deleted when the sum of all from-rates and the sum of all to-rates in that node are both lower than the expand threshold. (Both sums are, by definition, stored as from-rate and to-rate in the parent record of that node; hence the need for the `parent` pointer in `Table`.) The root node is never deleted. The list is either traversed backwards or forwards to avoid checking the same nodes every time thereby causing starvation-like phenomena.

To avoid heavy memory fluctuations and to avoid spending too much time on a single cleanup, contraction stops when a certain fraction f of all allocated memory has been freed. If none of the nodes can be deleted, but memory is at its imposed maximum, then some nodes *must* be deleted. In that case, the expand threshold is decreased by some factor and the cleanup starts again. This may have to be repeated multiple times until fraction f of all memory has been freed.

5.4 Memory exhaustion attacks

To defeat our mechanism, an attacker may try to exhaust a router's memory by making `IPRateMonitor` allocate many nodes. (Of course, memory exhaustion is only possible when `IPRateMonitor` has no imposed memory limit.) An attacker achieves this by sending packets with a wide variety of spoofed IP source addresses through that router. (This is a problem only when MULTOPS is in attacker-oriented mode.) Each stream of packets with a common IP source address needs to have a bandwidth higher than the expand threshold of MULTOPS—otherwise MULTOPS contracts the nodes, thereby defeating the attacker's goal to run it out of memory. If an attacker is not bound by any resource constraints, nor by ingress/egress filtering, he can create a worst-case scenario by sending spoofed IP packets such that the number of nodes in MULTOPS is maximized.

Given the structure of the MULTOPS tree, the size of a `Table` (1040 bytes), the size of a `Record` (28 bytes), a packet size of 34 bytes, and an expand threshold of 1000 packets per second, an attacker, launching such a worst-case scenario memory exhaustion attack, needs to generate traffic with 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 router. This number was derived by calculating the amount of allocated memory based on the number of different address prefixes stored in the tree. The expand threshold can be set to a value that ensures that memory will never run out. It is safe to conclude that, even without an imposed memory limit, it is impossible to run `IPRateMonitor` out of memory.

6 Measurements

To measure the performance of `IPRateMonitor`, a simple Click configuration was run in a Linux kernel 2.2.16 on an off-the-shelf PC (700 Mhz Pentium III, 256 KB cache, 256 MB memory) that sends packets through an `IPRateMonitor` element. Bogus UDP packets were generated by Click itself to avoid time consuming interaction with network interfaces. IP spoofing attackers were simulated by generating UDP packets with an IP source address picked from a fixed set of IP addresses in round-robin fashion. Measurements were done for different memory limits and for an expand threshold of 0, i.e., maximum expansion.

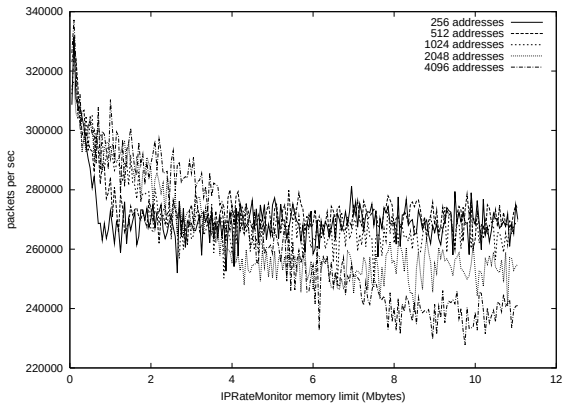

Figure 5: Packet rate as a function of memory limit

The graph in Figure 5 shows the number of packets that `IPRateMonitor` can handle as a function of its imposed memory limit. The graph shows this for 5 UDP flood attacks that differ only in the number of attackers (i.e., IP source addresses) involved. The IP source addresses used in the malicious UDP packets constitute a worst-case scenario (see Section 5.4).

The graph shows that `IPRateMonitor` performs better when it has little memory at its disposal. A small tree fits in cache entirely and is, therefore, fast. When more memory is available, the tree size increases up to the point where it is too big to fit in cache, and cache misses result. The performance of `IPRateMonitor` for 256, 512, and 1024 addresses is roughly the same (270,000 packets/sec), because in these cases the tree is small enough to fit in cache entirely. For 2048 and 4096 addresses, rates drop proportional to the total memory consumption of the tree, up to the point where the tree reaches its maximum size, after which memory consumption—and thus performance—fluctuates around the same point.

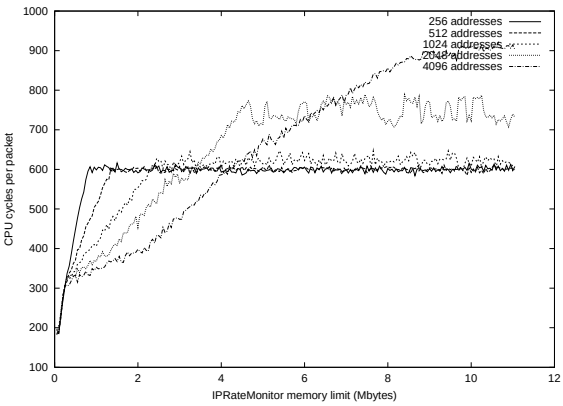

Figure 6: CPU cycles per packet as a function of memory limit

The graph in Figure 6 shows the number of CPU cycles that `IPRateMonitor` consumes per packet as a function of its imposed memory limit. `IPRateMonitor` consumes more CPU cycles when it has more memory at its disposal. These extra cycles are, most likely, spent on waiting for a memory fetch after a cache miss. Unsurprisingly, the graph in Figure 6 is essentially the reciprocal of the graph in Figure 5.

`IPRateMonitor` performs better when it has little memory at its disposal. Unfortunately, its ability to expand and, therefore, to precisely determine the source(s) and/or target(s) of the attack, is also more limited. Thus, the tradeoff is precision vs. performance.

7 Discussion

7.1 IP spoofing

MULTOPS in victim-oriented mode is not influenced by IP spoofing. However, MULTOPS may impose "collateral damage" by dropping legitimate packets going to the victim.

When attackers randomize IP source addresses—a common practice—then a problem arises for MULTOPS in attacker-oriented mode. There could be so many different (spoofed) IP source addresses that MULTOPS does not have enough available memory to establish all "malicious" IP source addresses. In that case, MULTOPS *can* establish a set of prefixes that malicious IP source addresses share. Better randomization implies shorter address prefixes. Shorter prefixes implies that MULTOPS drops more packets, which may include legitimate packets. In other words: collateral damage as a result of MULTOPS' dropping policy is greater when IP spoofing gets more randomized.

When attackers *perfectly* randomize IP source addresses, each malicious stream of packets with a common IP source address (or prefix) is either too insignificant to be seen as part of an attack, or *all* malicious streams are seen as part of an attack. In the former case, MULTOPS does not detect the attack at all. In the latter case, all packets are considered part of an attack, and, hence, dropped. Both cases constitute a successful denial-of-service attack.

7.2 Distribution

The IP spoofing problem described above closely relates to the problem of attacker distribution. As more (spoofing or non-spoofing) attackers participate in a bandwidth attack, it becomes harder (for MULTOPS in attacker-oriented mode) to identify a single attacker because its relative share in the total mass becomes smaller and, therefore, the disproportional quality of the traffic less conspicuous.

When a total number of T packets per second is required to crash the victim's infrastructure, and N attackers participate, then each attacker needs to generate an average of T/N packets per second. As N gets larger, T/N gets smaller.

Even though MULTOPS' sensitivity can be tuned, if N is too large and, consequently, T/N too small, one single attacker might go undetected by MULTOPS. If, though, attackers do not spread out geographically, their combined generated traffic might go through a single MULTOPS-equipped router that could decide to drop all the packets. Even *if* the attackers are perfectly distributed throughout the world, the malicious packets get funneled on their way to the victim by routers. The chance of being detected as a malicious stream by one of these routers gets larger as the stream gets more bundled (and, thus, packet rates become more disproportional).

7.3 Different protocols

MULTOPS relies on the assumption that, during normal operations, packet rates between two communicating parties are proportional. There are, however, different protocols, each with different implementations. With TCP, for example, implementations differ in their acknowledgment policy, although most TCP implementations acknowledge at least every other packet. Nonetheless, defining the MULTOPS detection heuristic quantitatively, i.e., choosing suitable values for R_{min} and R_{max}, is tricky. In the current implementation of `RatioBlocker`, $R_{min} = 0.66$, and $R_{max} = 2.5$. These values were experimentally determined. One can imagine implementing a `RatioBlocker` that adjusts these values based on observed traffic patterns during normal operations, making the heuristic more flexible.

Protocols such as UDP and ICMP do not require acknowledgments at all. However, several applications such as NFS and DNS display proportional behavior similar to TCP, which is advantageous for the MULTOPS detection heuristic. Since most services on the Internet are TCP-based, we suggest rate-limiting all non-TCP traffic during an attack. Even though this is a drastic measure, it will allow most Internet traffic to proceed normally.

7.4 Asymmetric routes

MULTOPS needs to see traffic in both directions to detect disproportional packet rates—this requires symmetric routes. However, Paxson demonstrated

that many routes on the Internet are asymmetric [Pax97]. To circumvent this problem, MULTOPS should be placed on the edges of the network—in a data center, for example. If such a site is multi-homed, then packet rate statistics from all on-site routers need to be combined. This requires (preferably out of band) communication between several MULTOPS-equipped routers. The details of such a setup are beyond the scope of this paper.

7.5 Granularity

When MULTOPS has more memory at its disposal, it can expand to deeper levels, thereby increasing its precision. Dropping packets based on disproportional packet rates in a record in the root node will affect many machines, i.e., all machines with a common first byte in their IP address. If, however, dropping packets is done based on disproportional packet rates from/to a single IP address—stored in the deepest level of the tree—then only the machine with that IP address will be affected. It is, therefore, important to not restrict MULTOPS' memory use too much.

8 Conclusion

This paper proposes MULTOPS. MULTOPS enables routers or network monitors to detect ongoing bandwidth attacks using a simple heuristic: a significant, disproportional difference between the packet rate going to and coming from a host or subnet. This is based on the assumption that, during normal operations on the Internet, the packet rate of traffic going in one direction is proportional to the packet rate of traffic going in the opposite direction.

MULTOPS is a tree of nodes that contains packet rate statistics for subnet prefixes at different aggregation levels. It dynamically adapts its shape to (1) reflect changes in packet rates, and (2) avoid (maliciously intended) memory exhaustion.

MULTOPS successfully detects bandwidth attacks that create disproportional packet flows between the sender(s) and the receiver. To our knowledge, no such detection mechanism has been proposed yet. Depending on the situation, MULTOPS can point out the source(s) of the attack.

MULTOPS is not a complete solution against bandwidth attacks. However, it enables network devices to maintain statistics to establish whether or not a bandwidth attack may be going on.

Measurements show that the performance of MULTOPS is primarily influenced by the size of the cache and the number of IP source addresses involved in the attack. It is exceedingly difficult to run a MULTOPS-equipped router out of memory.

9 Acknowledgements

We would like to thank Frans Kaashoek, Robert Morris, Andrew Tanenbaum, and the entire PDOS team for their help.

References

[AH99] Thamer Al-Herbish. Secure Unix Programming FAQ, 1999. Available at http://www.whitefang.com/sup/secure-faq.html.

[Bel00] Bellovin. ICMP Traceback Messages. Technical report, AT&T, 2000. Available at http://www.ietf.org/internet-drafts/draft-bellovin-itrace-00.txt.

[Bel01] Steven Bellovin. DDoS Attacks and Pushback, February 2001. NANOG 21, Atlanta, GA, USA.

[CC96] CERT Coordination Center. CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks, 1996. Available at http://www.cert.org/advisories/CA-1996-21.html.

[CC98] 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.

[Cisa] Cisco. Netflow Services and Applications. Available at http://www.cisco.com/warp/public/732/netflow/.

[Cisb] Cisco. RMON. Available at http://www.cisco.com/warp/public/614/4.html.

[CNN00a] CNN. Cyber-attacks batter Web heavyweights, February 2000. Available at http://www.cnn.com/2000/TECH/computing/02/09/cyber.attacks.01/index.htm%l.

[CNN00b] CNN. 'Immense' network assault takes down Yahoo, February 2000. Available at http://www.cnn.com/2000/TECH/computing/02/08/yahoo.assault.idg/index.ht%ml.

[DFS01] Drew Dean, Matt Franklin, and Adam Stubblefield. An Algebraic Approach to IP Traceback. In *Proceedings of the 2001 Network and Distributed Systems Security Symposium*, February 2001.

[Dit00] 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.

[ea81] J. Postel et al. RFC 792. Internet Control Message Protocol. Technical report, IETF, 1981. Available at http://sunsite.cnlab-switch.ch/ftp/doc/standard/rfc/7xx/792.

[ea00] P. Ferguson et al. RFC 2827. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. Technical report, IETF, February 2000. Available at http://sunsite.cnlab-switch.ch/ftp/doc/standard/rfc/28xx/2827.

[ec00] The electrohippies collective. Client-side Distributed Denial-of-Service, 2000. Available at http://www.gn.apc.org/pmhp/ehippies/files/op1.pdf.

[Ins00] SANS Institute. Egress filtering v 0.2, 2000. Available at http://www.sans.org/y2k/egress.htm.

[KMC+00] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[Net00] Netscape. Leading Web sites under attack, February 2000. Available at http://technews.netscape.com/news/0-1007-200-1545348.html.

[Pac00] Packetstorm. Packetstorm, 2000. Available at http://packetstorm.securify.com.

[Pax97] Vern Paxson. End-to-End Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking*, Vol.5, No.5:601–615, October 1997.

[Pax99] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23–24):2435–2463, December 1999.

[SP01] Dawn Xiaodong Song and Adrian Perrig. Advanced and Authenticated Marking Schemes for IP Traceback. In *Proceedings of the IEEE Infocom 2001*, April 2001.

[Spi00] Lance Spitzner. The Tools and Methodologies of the Script Kiddie. Know Your Enemy, 2000. Available at `http://www.enteract.com/~lspitz/enemy.html`.

[Sto99] Robert Stone. CenterTrack: An IP Overlay Network for Tracking DoS Floods, October 1999. NANOG 17, Montreal, Canada.

[SWKA00] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical Network Support for IP Traceback. In *Proceedings of the 2000 ACM SIGCOMM Conference*, pages 295–306, August 2000.