ABSTRACT

Title of dissertation: DETECTING DDoS ATTACKS IN STUB DOMAINS

Christopher Kommareddy,
Doctor of Philosophy, 2006

Dissertation directed by: Prof. Samrat Bhattacharjee
Department of Computer Science

Dr. Richard La
Department of Electrical & Computer Engineering

DoS attacks have least impact when detected and mitigated close to the attacks' source. This is more important for Distributed DoS (DDoS) attacks since they are difficult to mitigate at the victim without affecting service to legitimate flows. This is a challenging task since DDoS attack traffic may have relatively low flow rates and attack packets are indistinguishable from legitimate packets. Current source-end detection schemes such as MULTOPS and D-WARD are centralized and hence, are not easily deployable in multi-gateway stub networks with asymmetric traffic. Moreover, these systems require modifications to current routers for successful deployment.

We present a scalable, distributed DDoS detection system that can be deployed in single- as well as multi-homed stub networks to detect DDoS attacks using TCP packets. The detection system can detect attacks with very low flow rates and in multi-gateway networks, even with significant asymmetric TCP flows. We evaluate the performance of our detection system using extensive packet level simulations under different attack

scenarios. Our results show that with relatively less node state and processing, in networks with symmetric flows, our system can accurately detect attack flows that are one-third the intensity of an average flow in the network. In the case of multi-gateway networks, the detection system can detect all attacks for all rates of asymmetry when the attack rate is at least five times the average flow rate in the network.

We extend the system to detect attacks aimed at multiple hosts in a subnet instead of a single host. Subnet attacks seem more diffused for detection schemes designed to detect host attacks. Hence, it is harder for these schemes to detect these attacks. Our subnet attack detection scheme can detect attacks that target hosts in large subnets (/21) and in the presence of non-attack traffic to other hosts in the subnet. Our packet level simulations show that, in single gateway networks, our scheme can detect attacks with an aggregate flow intensity equal to an average flow in the network in less than a minute. Using these simulations, we also show that our scheme detects attacks in networks with up to four gateways and when up to 50% of the flows are asymmetric.

DETECTING DDoS ATTACKS IN STUB DOMAINS

by

Christopher R. Kommareddy

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

Advisory Commmittee:

Professor Samrat Bhattacharjee, Chair/Advisor
Professor Richard H. La, Co-Advisor
Professor Mark A. Shayman, Co-Advisor
Professor Ashok Agrawala
Professor Udaya A. Shankar

Dedicated

To World Peace

ACKNOWLEDGMENTS

endarO mahAnubhAvulu; andariki vandanamulu

there are several luminaries; my salutes to them all

— Thyagaraja (19th century composer and poet)

I owe my gratitude to all those who have made this thesis possible and my life and experience at University of Maryland memorable. I apologize if I do not adequately credit your contribution here.

I am extremely grateful to my adviser, Professor Bobby Bhattacharjee. He has been instrumental in my decision to pursue a Ph.D. after my Masters degree. I am greatly indebted for all he has done for me in his role as my adviser — for giving me the opportunity to work with him, for constantly supporting me financially, for letting me to work on interesting problems, for occasionally nudging me to do my best, for ensuring my work is thorough and complete, and for painstakingly revising my manuscripts countless times so that they read better. It has been a pleasure to work with and learn from such a dynamic and knowledgeable person.

I have been very fortunate to have both Professors Mark Shayman and Richard Hyongla as my co-advisers. Their suggestions during our weekly meetings were essential for me in completing this dissertation. They reviewed several of my manuscripts from cover-to-cover and suggested invaluable corrections and improvements.I also thank Professors

Ashok Agrawala and Uday Shankar for agreeing to serve on my dissertation committee and sparing their precious time to review this manuscript.

I offer my sincere gratitude also to Suman Banerjee, with whom I had the opportunity to work early on. My interactions with him helped me tremendously in developing the necessary flair for academic research. I also grateful to my other collaborators whose contributions to my dissertation are invaluable: Dave, Mehdi, Vahid and Tuna.

My colleagues at the PCHASM lab deserve a special mention. My interactions with Vijay Gopalakrishnan, Rob Sherwood, Dave Levin, Arunchander Vasan, Ruggero Morselli, Seungjoon Lee and Cristian Lumezanu have been very fruitful and have enriched my graduate life in many ways. In particular, I thank Vijay and Arun for helping me improve my presentations. I also thank Ray for teaching me how to play volleyball and thank Bujor for playing chess with me.

During my 6+ years at University of Maryland, I had several house mates and made several friends. I am thankful for their tremendous company. Their friendship and support have been crucial for me to have a wonderful time at college park and successfully complete my dissertation. In this regard, the following friends deserve a special mention: Vaibhav Kumar, DP Ayyadevara, Ashish Mundada, Kaveri Pant, Soumya Krishnamurthy, Nidhi Thareja, Lavan Gangisetty and Vijay Gopalakrishnan.

I am also grateful to all my friends from India. Their constant encourage, inducements, reminders and discouragements have been an important factor towards my successful completion of this dissertation. Siddhartha Nadella, Ramana Athota, Prakash Kaligotla, Sirisha Gandham, Goutam Nadella, Sunil Katepalli and Gunaranjan Vasireddy deserve a special mention. Due to these people, I was able to make frequent pleasure

trips. These trips tremendously helped me unwind from all the different pressures associated with living a graduate life.

Last but not least, words cannot sufficiently express my sense of appreciation and gratitude for my family. They have always stood by me and guided me throughout my career. My parents are a source of my inspiration and motivation. My dad, grandmother and sister have been instrumental in the development of my problem solving skills early in life. My brothers helped me in making right decisions at key stages in my academic life. The positive influence of my parents, brothers and sisters, the joy brought by my adorable niece and the playful memories of my cheerful dog helped me pull through difficult moments during different phases of my life. I also acknowledge the hand of the invisible figure in the successful completion of this dissertation and all I have achieved so far.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1

Introduction

Internet routers route datagrams without keeping per flow state. This architecture has served Internet well in terms of simplicity, scalability and heterogeneity. A drawback of this stateless approach is that routers cannot distinguish between legitimate packets versus malicious packets. Thus, identifying malicious hosts or preventing malicious traffic within the network is very difficult. Denial of service (DoS) attacks are a class of malicious traffic that aim to deny service to legitimate users. DoS attacks can be classified into two different types: protocol weakness attacks and resource exhaustion attacks. In protocol weakness attacks, attacks succeed by exploiting the weaknesses in protocol/application design or implementation. In resource exhaustion attacks, denial of service is achieved by overwhelming the resources required to service legitimate clients. We explain the two types of attacks using the examples of “Ping of death” and TCP “SYN” attacks.

IP packets are allowed to have a maximum size of $2^{16} - 1$ bytes. However, due to packet fragmentation and reassembly in the network layer, it is possible for a server to

receive an IP packet larger than the maximum allowed size. If the server does not check the size of such a fragmented packet while reassembling it, then the server can experience a buffer overflow during the reassembly. The server can crash due to this resulting in downtime for the server. These attacks were typically carried out by malicious hosts using ping (ICMP echo) packets. Hence, this attack is called “Ping of death”. Ping of death attack is successful when the systems do not check the size of a packet when reassembling it. Hence, this attack is an implementation weakness attack.

TCP SYN attack, on the other hand, is successful only when an attacker is able to send a large number of SYN requests. For a legitimate TCP connection, a client sends a SYN packet to the server and the server responds with a SYN-ACK packet. At the same time, it also allocates required data structures for the TCP connection. The client replies with an ACK packet and completes the connection setup phase. If a server does not receive the ACK packet, it resends the SYN-ACK packet a few more times, each time increasing the wait time between retransmissions. Eventually, after a total wait time of three minutes, if it does not receive the ACK packet from the client, it frees the allocated data structures and resets the connection. Due to the limited memory available for the TCP data structures, a server can at most service a fixed number of simultaneous connections. If an attacker sends several spoofed SYN packets, then it can exhaust the memory available at the server for TCP connections. As a result, the server will not be able to accept any new requests for connections from legitimate clients. Thus, an attacker can successfully deny service to legitimate clients. This is a resource exhaustion attack since the attacker exhausted the memory at the server for TCP connections. A SYN attack may also be considered an implementation weakness attack because SYN at-

tack succeeds when a server allocated memory for the “new” connection before the TCP three-way handshake is completed.

1.1 Bandwidth Attacks

Bandwidth attacks or flooding attacks are executed by sending a large number of packets to a victim. The links incident at the victim become heavily congested and drop a large fraction of packets sent to the victim. Since routers are stateless, they cannot distinguish between legitimate and attack packets and drop both types of packets uniformly. This results in deficient service to the legitimate users trying to access the victim. Thus, bandwidth attacks are resource exhaustion type DoS attacks. These attacks may be performed using packets that seem like TCP packets, UDP packets or some other type of IP packets. In this work, we develop two mechanisms to detect flooding attacks which use TCP like packets.

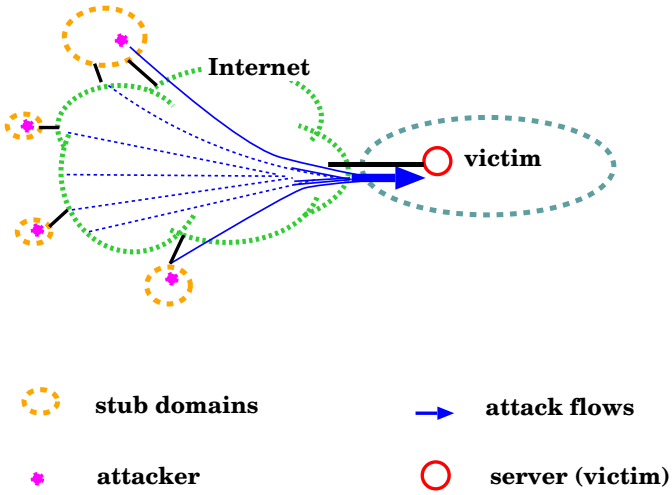

Figure 1.1: Typical Bandwidth Attack

Bandwidth attacks are a unique and important class of DoS attacks for the following

reasons. In most of these attacks, as shown in Figure 1.1, the attacking hosts are end hosts (often unwilling participants) and victims are server farms or enterprise sites. Typically, these victims have more resources, such as bandwidth and processing, than the combined resources of a few individual end hosts. Hence, several attackers need to participate in a bandwidth attack for the attack to be successful which results in a Distributed DoS (DDoS) attack. Next, because bandwidth attacks require large number of attack packets to be successful, the attacks consume significant network resources on the attack path. Their impact is felt more widely in the Internet compared to other DoS attacks. A case in point is the instability caused in global Internet routing during Nimda and the Code Red II worm propagation phase [1]. Inspite of not being bandwidth attacks by themselves, Internet instability during their propagation suggests that large DDoS attacks can also result in Internet instabilities. Finally, the large number of packets imply that taking defensive or corrective actions solely at the victim is very difficult.

The distributed nature of the problem and the inability to distinguish between attack packets and legitimate packets makes the problem interesting and challenging. Due to ease with which they can be mounted and the extent of damage they cause, preventing flooding attacks is important for stable functioning of the Internet.

1.2 Detecting TCP-based Bandwidth Attacks

If an attack is successful, only a fraction of TCP packets addressed to the victim reach their destination. The victim may respond to even fewer TCP packets because its resources such as processing and memory are overwhelmed. A legitimate host commu-

nicating with the victim during the attack will perceive the network to be congested and will multiplicatively decrease its sending rate. Attackers on the other hand will continue to send large number of packets to achieve denial of service at the victim. Thus, the number of packets sent to the victim by the attackers greatly outnumber the number of response packets received by the attackers from the victim during an attack. If the source addresses in the attack packets are spoofed, the respective response packets will not reach the attacking hosts and the ratio of attack packets to their response packets will be even larger at the attackers. We use this characteristic of flooding attacks to detect the attacks.

An important aspect of a detection scheme as described above is that it must observe a flow's packets in both directions to correctly determine if the flow is legitimate. Otherwise, the system will erroneously determine a legitimate flow as an attack. Routing in the Internet can be asymmetric, i.e., path for the packets of a flow in one direction is different than the path for the reverse packets. Hence, a detection system using packet ratios to detect attacks may be only deployed in stub networks, either at the source/attacker side of a flow (source-domain detection) or the destination/victim of a flow (victim-domain detection).

1.3 Detecting Attacks in Source, Transit and Victim Domains

Victim-domain detection of flooding attacks is easier because of the severe network congestion close to victim and presence of a large number of TCP flows unresponsive

to congestion control mechanisms. However, detecting and filtering attack traffic at the source is superior for the following reasons:

- Fewer network resources are wasted if attacks are filtered closer to the source of the attack.
- Excessive DoS traffic can cause severe network congestion even in the transit networks that can result in BGP instabilities [1]. This can affect all traffic in the Internet and not just victim's legitimate traffic.
- It is more difficult to distinguish between legitimate and attack traffic at the victim and would require more resources. If the resources are not available, legitimate traffic will also be filtered along with attack traffic.
- Network usage profiles of hosts may be easily captured at the source's host network and these profiles may be used to detect if the host's traffic is malicious.
- Source-domain detection reduces the need to deploy traceback mechanisms in transit networks.
- Finally, detecting malicious traffic in source networks helps network administrators identify compromised hosts or bots [2] inside their networks.

Victim-domain detection has the following advantage over source-domain detection. It can easily perceive an ongoing bandwidth attack and hence, need not constantly monitor the network traffic for the presence of DDoS flows. Source-domain detection will require constant monitoring of network traffic since the bandwidth attack may not have any affect in the source domains. In this dissertation, we focus on source-domain

detection. There has been extensive work on victim and transit domain detection that we discuss in Chapter 2.

1.4 Source-Domain Monitoring and Detection

Our system consists of a set of overlay nodes distributed at the routers within a stub network. These nodes form an overlay network over which they may communicate. These nodes sample traffic at the routers in the network and execute a pair of independent algorithms. The algorithms use the packet samples to detect ongoing DDoS attacks that originate within the network. One algorithm detects attacks to individual hosts while the other detects attacks to multiple hosts in a subnet. The difference between these two attacks is the following. Subnet attacks are diffused over more victim addresses and hence are more difficult to detect than the host attacks.

We envision that these overlay networks are deployed within a single AS due to administrative limitations. We also require that the network infrastructue and the data path in the network may not be altered. Changes to network infrastructure or the data path in the network may make the solution economically expensive. Moreover, if overlay nodes are deployed on the data path, they may delay the packets in the network.

The host attack detection scheme has two components. The first component processes sampled packets to quickly detect anomalous traffic flows at routers within the source AS. This is done in such a way that all attack flows, including weak ones, are found (i.e., there are no *false negatives*) but many legitimate flows may be suspected as attacks as well (i.e., there may be many *false positives*). The second component takes

as input the list of all suspected attack flows and removes the false positives, in part by allowing monitors to communicate with one another and vote on suspected attacks.

The subnet attack detection scheme, compared to the previous scheme, requires more processing and time to detect attacks. However, it can detect attacks that target hosts in large subnets (/12) and in the presence of non-attack traffic to other hosts in the subnet.

Our system is the first source-based detection system intended for deployment in multi-homed stub networks where TCP flows may enter and depart the domain using different gateway routers. Additionally, our system is passive: it does not induce any packet drops to detect attacks in the network. In fact, our system does not interfere with routers' fast path processing at all. It is also distributed: unlike [3] and [4], whose deployment consists of only a single border router, we distribute traffic monitors to routers throughout the AS. We are subsequently able to handle arbitrary stub AS topologies, and our system can easily be scaled to handle very fast access links. Finally, the system detects many different types of flooding attacks —such as direct attacks, pulse attacks, and reflector attacks —efficiently and with minimal false positives.

1.5 Contributions

In this dissertation, we propose to demonstrate that it is feasible to develop source-end DDoS detection systems with the following properties: *scales with network traffic, deployable in multi-gateway networks, flexible to allow tradeoffs between detection sensitivity and resources required, resilient to attacks against the system and does not require*

expensive changes to current deployed infrastructure. To support this thesis, we describe a distributed source DDoS detection system and, develop schemes that the system can execute to detect bandwidth attacks using TCP packets. We evaluate these schemes using traces collected from high bandwidth routers in the Internet. Our contributions in this dissertation are as follows:

- We describe an in-network overlay network that can be deployed in stub domains. We present in detail the problems such a system will encounter to monitor traffic at line speeds and our solutions to handle these problems.

- We next describe a host victim detection scheme that, with very little flow state, can detect bandwidth attacks using TCP packets against individual hosts. We will also present the extensions to the protocol so that attacks can be detected even in the presence of asymmetric traffic in the network. Ours is the first system to study this issue in detail. We investigate the effect of various tunable parameters on system performance. Network administrators can set these parameters so tune the detection system according to their individual preferences and requirements. We perform extensive simulations to evaluate the scheme under various host attack scenarios. We also investigate the resilience of the system when it itself is under an attack.

- We finally present our scheme that can effectively detect subnets attacks in source domains. To our knowledge, ours is the only system that explicitly detects subnet attacks. We discuss the extensions required to deploy the scheme in multi-gateway networks. We again investigate the effect of different detection scheme parameters

and how these parameters influence the detection sensitivity of the subnet attack detection scheme. We also perform detailed evaluations for different attack scenarios and present the results here.

- We compare our work with other source domain detection schemes. More specifically, we do a detailed comparision with DWARD [4].

1.6 Outline

The rest of the dissertation is structured as follows. We will discuss the related work in Chapter 2 and describe the system architecture in Chapter 3. We describe the host attack detection protocol for single- and multi-gateway ASes in Chapter 4. We present the evaluations of our host attack detection protocol in Chapter 5. In Chapter 6, we describe the subnet attack detection scheme for single- and multi-gateway networks. We present the evaluation results of this scheme in Chapter 7. We conclude in Chapter 8.

Chapter 2

Related Work

Different DoS detection and prevention approaches have been proposed in the literature. These approaches may be classified along many dimensions such as deployment location, detection heuristic, type and level of infrastructure changes, and so on. In this chapter, we will describe various detection mechanisms and compare it against our detection system. We will first describe D-WARD and MULTOPS, two stub domain detection systems proposed before ours. We then review other DDoS detection and prevention mechanisms including traceback schemes, proactive measures, detection methodologies, etc.

2.1 Source Domain Detection Schemes

MULTOPS and D-WARD are two DDoS detection schemes that use packet ratios of flows to detect attacks in stub domains. We have a detection strategy similar to these two schemes. In this section, we describe these two detection mechanisms and compare

our system against these in detail.

2.1.1 MULTOPS

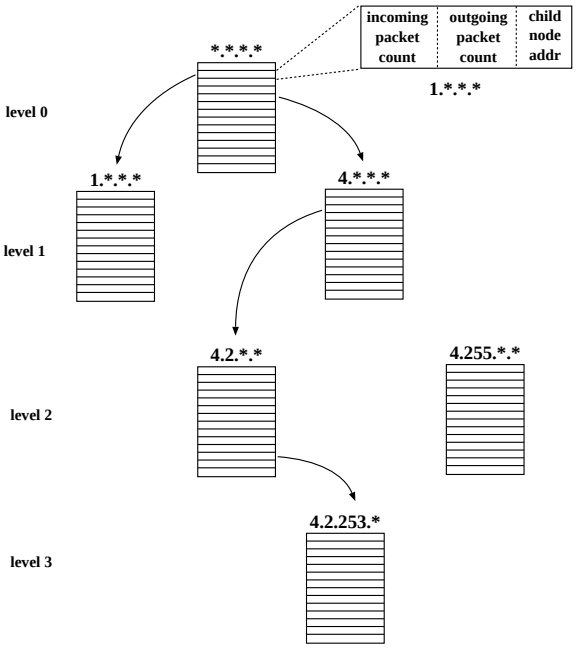

Figure 2.1: MULTOPS Data Structure

MULTOPS [3] was the first stub domain DDoS detection system proposed in the literature. A MULTOPS capable router maintains packet rate statistics for its traffic using a 4-level 256-ary tree data structure. The data structure is shown in Figure 2.1. Each node in the data structure corresponds to an IP prefix as shown in the figure. The position of the node determines the prefix it represents. For example, as shown in the figure, the 3rd child of node 4.*.*.* represents the prefix 4.2.*.*. Each node has 256 entries, each corresponding to one of the 256 children of the node. Each entry consists of 3 fields: outgoing and incoming packet count for the child's prefix and, if required, a pointer to the child node. Whenever a packet that maps to the entry, i.e., the packet's address prefix is

the same as that of the entry, the entry's counter corresponding to the packet's direction is updated. When a packet rate for a prefix reaches a certain threshold, the child node for the corresponding prefix is initialized (i.e., the entry is expanded). When the packet rate falls below a threshold, the node is contracted by deleting the node's children. In this manner, MULTOPS data structure can be adapted to changing traffic characteristics of the stub domain as well as the resources available at the router implementing the MULTOPS scheme.

MULTOPS detects attacks using the ratio of outgoing and incoming packet rates for each prefix for which it maintains packet rate statistics. Whenever a node's outgoing-to-incoming packet rate ratio falls outside a predetermined range, the corresponding prefix is flagged and the packets from (or to) the prefix are dropped.

Our system too uses flow ratio to distinguish between legitimate and attack flows. Besides this basic detection heuristic, our system is different from MULTOPS in several significant ways. The first important difference is that our detection system does not require any changes to the routers. MULTOPS is implemented in routers and thus, requires changes to the routing hardware.

Second, our distributed deployment provides several benefits over single point MULTOPS deployment. Due to distributed deployment and detection, our system does not have a single point of failure. As will be clear from protocol description, even if one monitor fails, the remaining monitors can still function to detect ongoing attacks. Also, due to distributed deployment, elements of our detection system can be deployed closer to attack hosts. This provides for traceback of the attack path. Since our detection system elements are deployed closer to end hosts in the AS, these nodes can sample packets at

a higher rate and hence, can detect attacks better. These elements also communicate between themselves. As a result, our detection system can detect attacks in high-speed and multi-gateway networks. It can also be adapted to detect attacks that originate and end within the network.

Next, our system maps flows to aggregates uniformly at random. This ensures that an attacker cannot use the knowledge of other active flows in the network to circumvent detection. In other words, the attacker cannot hide an attack using an active legitimate flow to another host close to the victim. For example, an attacker cannot use an active flow to destination $a.b.c.d$ to mask its attack to destination $a.b.c.e$ in our system. In MULTOPS, if the flow rate to $a.b.c.*$ as well as the attack rate to $a.b.c.e$ are low, the attack will be missed. More importantly, because of random mapping and distributed deployment, it is harder for an attacker to force our system to flag a legitimate flow as attack. Consider the scenario in Figure 2.2. In the figure, the MULTOPS capable router at the gateway (node G in the figure) cannot distinguish between flows a and c since they will always get mapped to the same entries in its data structure. It also cannot distinguish between flows a and b too if they are mapped to the same entries (which is dependent on the threshold set to expand the MULTOPS nodes). Our system can distinguish all three flows. Our system and MULTOPS router will fail to distinguish between flow a and d, however. Now, if flow a is legitimate but flows b, c and d are malicious, our system will flag a as attack only if d is present. On the other hand, MULTOPS will determine a as attack if any of the other three flows are present.

Next, our scheme detects attacks using only a fraction of packets. This provides two benefits to our system. It can scale better with increasing traffic rates. Also, our system

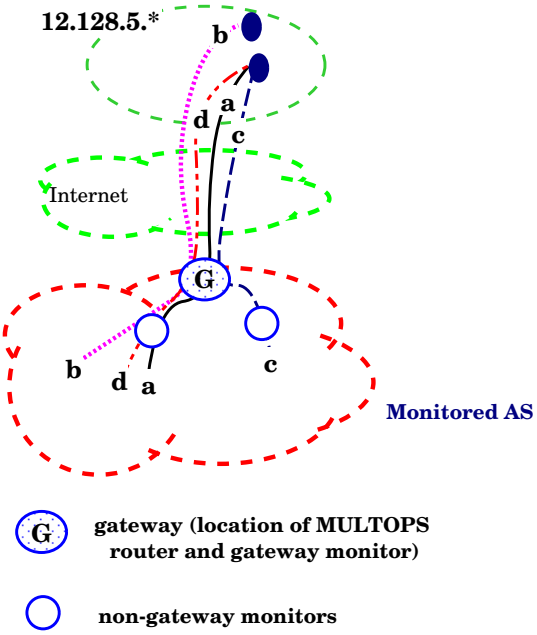

Figure 2.2: MULTOPS False Positives

is more protected from attacks that aim to overload the detection system. For instance, attackers can generate several random packets and send them to increase the processing overhead of the detection systems. Our system would react to such increase in processing by lowering the sampling rate. As a consequence, as our results show, detection time may be longer but the system is otherwise unaffected. Even though MULTOPS does not use packet sampling, it should be straight forward for it to employ the same and decrease packet processing overhead.

Finally, MULTOPS will miss reflector attacks if the attacker employs a large number of reflectors. In a reflector attack, the attacker sends packets to public servers (reflectors) in the Internet. These packets have the victim's address as the source address (i.e., the source addresses of these packets are spoofed with the victim's address) and are typically request packets (such as ICMP echo request or TCP connection request packets) that

generate a response from the servers. Since the source addresses for the request packets have the victim's address, all the response packets are sent to the victim. If the number of responses are high, it will congest the victim's access link. If the number of reflectors employed in the attack is high, the attack can be successful even if the number of request packets to each reflector is low. In that case, MULTOPS will miss all the individual flows to the reflectors and will be unable to detect the presence of an attack. While MULTOPS can detect bandwidth attacks, as shown in Figure 6.3, if a legitimate flow to a host in the victim subnet is stronger than the attack rate, MULTOPS will miss the attack.

2.1.2 D-WARD

D-WARD [4] is another source domain DDoS detection system. D-WARD maintains packet count at flow level (i.e., for each external destination) and at connection level (i.e., for each TCP connection). It uses different models to evaluate flows belonging to different protocols. For a TCP flow, it uses the packet ratio of the flow to determine if the flow is an attack. Whenever a flow's packet ratio is greater than a threshold, D-WARD drops packets of the flow. Due to the connection level packet counts it maintains, D-WARD is able to selectively drop packets of the flow, and thus, penalizes only errant connections of the flow. D-WARD uses packet drops to also determine if the flow becomes compliant with TCP specification [5]. If not, D-WARD rate limits the flow more severely. D-WARD uses similar models for non-TCP protocols such as ICMP and DNS traffic. For other UDP traffic it builds models for different applications that use UDP and applies those models on UDP flows to evaluate if the flows are legitimate. For other UDP

flows which do not have a built-in application model or the application model cannot be determined, it applies rate limits to limit the rate of such flows.

D-WARD uses many more models than MULTOPS to classify flows and the models are more complex. Hence, the complexity of D-WARD implementation can seriously impair the fast functioning of high speed routers. For example, D-WARD version 2.0 linux implementation could process around 10000 packets every second. The packet processing rate of D-WARD version 3.0 was not given in [6]. From the description in [6], it appears that D-WARD is designed for deployment in enterprise level networks.

D-WARD also processes all packets. Hence, like MULTOPS, it is also more prone to attacks against the system that aim to overload the processing capacity of the detection system. And, it is also less likely to scale to detection in networks with higher traffic rates. D-WARD implementation puts *traffic policing* component in the kernel space and other components (namely *observation* and *rate-limiting* components) in user space. As a result, this system, unlike MULTOPS, can adapt to changes in protocol models and introduction of new protocols.

Again, our distributed architecture can be deployed in complex networks and there is no single point of failure. D-WARD proposes two mechanisms to handle detection in multi-gatway networks. First, D-WARD capable routers are deployed at all the gateways in the network and these routers exchange information periodically before they classify flows and connections. The second mechanism is that D-WARD capable routers are deployed within the source network at connection points between stub subnetworks and the rest of the source network. In effect, each stub subnetwork is treated as an independent stub domain. This deployment approaches our distributed deployment. These two

mechanisms are shown in Figure 2.3.

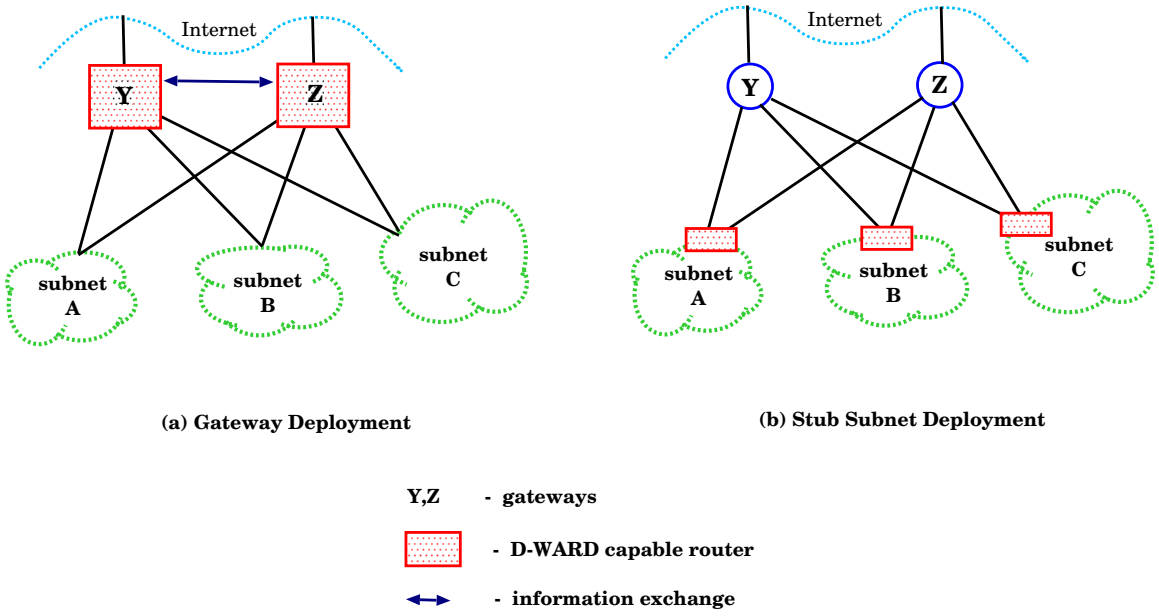

Figure 2.3: D-WARD Multi-gateway Deployment

D-WARD stores packet rate on a per-destination and a per-connection granularity. Hence, detection rate of D-WARD is better than our sytem's. Also, if attack packets are not spoofed, it can distinguish between legitimate versus attack traffic to a destination and selectively drop only attack packets. However, if attack packets are spoofed using other addresses in the domain, D-WARD may not correctly distinguish between attack and legitimate traffic. D-WARD maintains the flow and connection information in fixed size hash tables. This protects the D-WARD system from memory overflow and having to initialize memory. However, D-WARD still needs to check all entries in the table periodically to clear stale information such as the state of inactive flows from the table. Also, since D-WARD maintains per-flow and per-connection information, an attacker can exhaust D-WARD's memory resources by generating flows to arbitrary destinations and

connections to arbitrary ports. D-WARD handles the situation by deleting records of flows which sent few packets and bytes when the data structures are 90% full. An attacker can use this mechanism to sneak its attack packets past the D-WARD system. Another issue is the use of hashing to map flows with entries in the hash table. Hashing can result in collisions. To improve performance, flow information is not inserted in the hash table if the flow leads to collision even after three rehash attempts. Again, a smart attacker can utilize this to conduct an attack without detection.

Finally, unlike our system, D-WARD cannot detect attacks against subnets. If several attackers participate in an attack, each attacker can send attack packets at a low rate and yet the attack could be successful. Now, if an attacker has multiple addresses to use in an attack, it can decrease the attack rate to individual addresses accordingly. Since D-WARD does not combine the analysis for different addresses in a subnet and since it uses hash tables of limited sizes, it can miss one or more flows belonging to the subnet attack and unable to prevent the attack from succeeding.

2.2 General Detection Schemes

2.2.1 Traceback Techniques

Internet routing relies on the destination field in the packets. Hence, if the source address in the packets is spoofed, it is hard to detect the packet source. Spoofing packets to execute attacks has been discovered several years before [7]. Traceback is a mechanism which identifies the path, and potentially the source(s), of a packet or a group of

packets. In the past few years, several traceback mechanisms have been suggested that determine the source of the spoofed packets [8, 9, 10, 11, 12, 13, 14, 15, 16]. Traceback mechanisms come in several flavors. Some mechanisms generate additional ICMP packets while others use inline packet marking to relay path information to the destination. Other mechanisms maintain state locally at routers in the network and iteratively poll the upstream routers to discover the attack path. Yet others use input debugging and link testing to determine the path. We will describe these different approaches next and towards the end of this section, differentiate our approach with different traceback mechanisms.

ICMP Traceback ICMP message traceback [8], due to Bellovin, was among the first proposed traceback schemes. When forwarding packets, a router, with a low probability, generates a traceback message that is sent along to the destination. With sufficient messages from routers on a packet's path, the destination can determine the packet source and path. An important drawback of this approach is, it increases the traffic load in the Internet. Even if routers generate messages with low probability, if the path is long, each packet can generate several new packets. Next, when a victim is under attack, new ICMP packets will further contribute to congestion at the victim. Finally, ICMP packets are generated even when original packets are not spoofed. The destination has to handle additional traffic that is of no use to it.

Packet Marking Schemes IP marking mechanisms take a different approach by using IP header fields to encode packet path information instead of generating new packets. The first of the several schemes we discuss here is due to Savage et. al. [12, 13]. In

this, routers probabilistically mark packets they route with partial path information. A DDoS victim can combine a modest number of attack packets and construct the attack path/graph. In [16], authors show that this scheme has a high computation cost to compute the attack path and generate a large number of false positives when several attackers participate in the attack. In the mechanism proposed in [16], a router probabilistically encodes the following information in each packet it routes — the edge the packet traverses at the router and the edge's distance from the victim. The victim uses encoded information from sufficient packets to determine the attack graph while generating few false positives. The traceback mechanisms so far assume that routers in the network are not compromised. In the same work, authors propose also propose an authenticated mechanism to mark the packets. Another IP marking scheme is PI (path identification) by Yaar, Perrig and Song [15]. The full packet route is embedded in all packet which makes this scheme deterministic. A victim needs to classify one packet as malicious and inform upstream routers. All subsequent packets with the same path marking can be filtered by upstream routers and thus, these routers mitigate the attack.

Single Packet Traceback The traceback mechanisms we described so far are called "forward-looking", because they proactively send information to the destination. If the destination is under attack, it can determine the attack graph locally with the forwarded path information. Other traceback mechanisms rely on packet logging and input debugging (or link testing) to determine the attack path. In single-packet IP traceback [9], the authors rely on packet logging. They use packet hashes to generate audit trails for packets within the network. A router, for each of its packets, computes a digest over a portion of

the packet. It stores this digest using a Bloom filter. A router can determine if it routed a packet by verifying if the fields of the Bloom filter corresponding to the packet's digest are set. If so, it can contact its upstream routers to similarly determine if they routed the same packet. Thus, recursively, a packet's path can be determined. Unlike other previous schemes, this scheme can determine the path of even a single packet. Also, the scheme can be implemented using additional hardware, without requiring any changes to the deployed routers.

Input Debugging In input debugging, a victim develops an attack signature and reports it to the upstream router. The upstream router uses this signature to determine the incoming port for the attack which determines the next upstream hop in the attack path. This procedure is repeated recursively to determine the full attack graph. CenterTrack [17], deployed in ISP networks, is an overlay based prevention system that uses input debugging. IP tunnels are formed between edge routers and tracking routers to redirect the attack traffic onto the overlay network. All mechanisms described so far require global deployment and most require enhancements to router functionality.

Line Testing In [18], Burch and Cheswick propose an input debugging variant of traceback using only end hosts. Their mechanism, called *link testing*, determines the attack path of a DoS flow by selectively flooding links and observing its affect on attack traffic. Link testing has several drawbacks. First, a victim requires a good topology map of the Internet. Without a comprehensive topology map, it is difficult for the victim to realize which links have to be flooded. Second, the flooding itself can lead to DoS. Finally, link

testing is noisy and hence, it can be difficult to determine the attack graph of a distributed DoS.

Traceback mechanisms require global deployment to identify the remote attacker. Except for single packet traceback [9] and line testing [18] schemes, all traceback mechanisms require router modification. Most mechanisms require a flow's destination to receive a large number of packets for the destination to determine the path (and the source) of the flow. For these reasons, it is difficult to universally implement any of the traceback schemes. Finally, deployment of ingress filtering (described in the next section) can make traceback deployment in transit networks unnecessary.

Our system, unlike traceback, is deployed within a single domain. Hence, it does not require global coordination. Traceback cannot determine an ongoing attack. And when a host is under attack, the victim may not have sufficient communication or processing resources to initiate and successfully execute a traceback. Hence, unlike our detection scheme, traceback is not independently useful to mitigate the effects of a bandwidth attack. Our mechanism, however, detects only bandwidth attacks while traceback mechanisms can be used to determine the source of any malicious flow. Traceback is a general purpose scheme that can determine the origin of a spoofed packet. As such, if deployed universally, it is applicable in a wide variety of situations such as determining the source of a scan activity or worm activity and so on. Malicious traffic other than bandwidth attacks are outside the purview of our detection scheme.

2.2.2 Filtering

Traceback determines the path taken by a set of packets in the network and are necessary when attack packets are spoofed. However, traceback does not mitigate the attack. Filtering [19, 20, 21, 22, 23, 24, 25, 26] reduces the impact of a DoS attack at the victim by selectively dropping packets at routers. Packets may be determined malicious and dropped using several techniquess. In ingress filtering, edge routers determine if a packet's source address belongs to the peering stub network. In path filtering, routers use source and destination addresses of a packet and BGP information to determine if the packet's path is valid. SIFF routers [23] insert path specific information in packets. The destination can determine which packets are malicious and accordingly, request upstream routers on the path to filter the packets with matching path information. Pushback mechanisms determine misbehaving aggregates of flows and drop packets belonging to these aggregates. They also relay this information to the upstream router so that the packets are dropped as close to the source as possible and minimize the impact on the network. Congestion control mechanisms are similar to pushback in the sense that they determine the flows that cause congestion and drop packets belonging to those flows. With Pushback and congestion control, legitimate flows may suffer collateral damage. In this section, we will describe these different filtering mechanisms in detail and at the end, describe how our detection system differs from the filtering mechanisms described here.

Packet Filtering

Ingress Filtering Ferguson and Seine proposed ingress filtering to defeat DoS attacks that employ IP source address spoofing [19]. An edge router of a transit network can determine if the source address of an incoming packet corresponds to the ISP peering at the edge router. If the address does not, the router presumes that the address is spoofed and drops the packet instead of routing it. Egress filtering is similar to ingress filtering except that, in egress filtering, the gateway of a stub domain determines if the source address is valid instead of the edge router of the transit network. A stub domain can know which of its IP addresses are in use. On the other hand, unless an explicit mechanism is setup, transit domain only know the block of IP addresses of their peering stub domains. Hence, a transit domain will be unable to filter packets if the spoofed source address belongs to the IP block of the peering stub domain, but of an IP address not in use. This work was extended to multi-domain networks in [27].

Source address validity enforcement protocol (SAVE) [28] is developed to provide routers with information needed to validate the source address of a packet. For SAVE, each router is assumed to be associated with a source address space. In SAVE, each router has a table, called an incoming table, which specifies the correct incoming interface(s) for a given source prefix. A router builds its incoming table using SAVE updates it receives from its upstream routers. A router generates a SAVE update whenever an entry in its forwarding table changes. The update contains the destination prefix associated with the changed entry and the router's source address and is sent towards the destination prefix associated with the entry. When a packet arrives at a router, the router verifies if a packet

is spoofed using the packet's source address and incoming interface, and the incoming table at the router. If the packet's source address cannot be verified, the packet is filtered.

Path Filtering Path filtering [20] is a distributed packet filtering mechanism that drops spoofed packets at routers in the transit networks by using the BGP routing information. Unlike SAVE, it does not have the overhead of another routing protocol. The authors demonstrated in the paper that the mechanism can be effective when deployed at certain core routers. All the above filtering mechanisms cannot prevent DoS traffic from an attack host that does not spoof its attack packets[1] or that spoofs the packets using an address local to its AS. These mechanisms are still effective to prevent certain classes of DoS attacks.

SIFF SIFF [23] is a stateless mechanism that allows end hosts to selectively stop individual flows from reaching its network. Packets are classified as priviledged or otherwise and when contending for bandwidth, priviledged packets are given priority. Each client obtains capabilities for its flow that allow its packets to be marked as priviledged. Using the marking in the priviledged packets, an attack victim can inform upstream SIFF routers to receive packets of selective privileged flows. SIFF requires routers in the Internet to be modified. However, it can be deployed incrementally.

Congestion Control

Several congestion control studies [29, 30, 31, 32, 33, 34] aim to identify and selectively filter non-responsive TCP flows. They are different than packet filtering mechanisms because they drop packets only when links are congested. Congestion control

[1] This is likely to only be done if the attack is being launched from a compromised machine, or *zombie*.

mechanisms try to penalize or rate-limit non-responsive flows and provide fairness to the responsive flows in the network. As such, congestion control mechanisms act locally and do not require global deployment.

In [29], the authors discuss the need to control best effort traffic to increase fairness, goodput and avoid congestion collapse. Congestion collapse occurs when TCP connections unnecessarily resend packets that are either in transmission or have been received at the receiver. The objective of the study was to identify unresponsive flows and "not TCP-friendly" flows. Unresponsive flows at a router are those that do not reduce their sending rate due to increased packet drop rate at the router. Not TCP-friendly flows are one which have a steady state throughput is greater than that of conformant TCP flow in the same circumstances. It penalizes nonconforming flows for congestion control purposes. LRU-RED [34] is an active queue management technique. Such techniques use queue lengths at a router to indicate the severity of congestion. In LRU-RED, a router tries to drop less packets from responsive, high bandwidth flows at the time of congestion.

Congestion control is a general purpose mechanism. It can be used to alleviate congestion during an attack, at routers implementing congestion control. As congestion control is local to a router, it does not require global coordination. It also alleviates the effects of a DDoS attack. However, as rate limiting filters are applied to aggregates of flow, these filtering methods will result in collateral damage to all flows in the aggregates containing attack flows.

Congestion control algorithms were not proposed to handle congestion due to attack flows but congestion due to increase in legitimate best effort traffic in the network. However, network operators determined it is simpler and cost effective to over-provision the

networks than implement complex congestion control algorithms at the routers. Hence, congestion control methods will not be widely deployed and will be ineffective in mitigating the effects of a bandwidth attack.

Pushback Mechanisms

Pushback [25, 26, 35, 36] is also a type of filtering mechanisms. It detects aggregates of malicious flows in the network and selectively drops packets belonging to these aggregates. In pushback [26, 35], the authors use the heuristic that persistent congestion is due to a *differentiable* set of flows rather than a single flow or due to a general overall increase in traffic. A router implementing pushback executes two related mechanisms. The first, *local aggregate congestion control*, identifies misbehaving aggregates in the router's traffic and applies rate limiting congestion control to these aggregates. The router discovers upstream router(s) that send these packets using input debugging. In the pushback phase, the router requests these upstream routers to rate limit the misbehaving aggregates. Pushback can prevent upstream bandwidth from being wasted on packets that are only going to be dropped downstream. For determining upstream routers sending malicious traffic, input debugging is useless if routers are joined as VLANs or as a frame relay circuit. The reason is, input debugging can only identify physical upstream routers whereas in VLANs and frame relay networks, routers are connected on a virtual network. For these scenarios, the router receives estimated packet rates of a malicious aggregate from logical upstream routers and accordingly, sends the pushback message. In selective pushback [36], the authors overcome this problem by relying on the probabilistic marking traceback schemes. A router uses the probabilistic markings in packets to determine

the previous hops of malicious packets and send pushback messages to these upstream routers.

Pushback is a union of congestion control and traceback techniques. As with most traceback mechanisms, pushback mechanisms also require modification to all routers in the Internet. Like congestion control schemes, pushback mechanisms can filter packets of legitimate responsive flows due to aggregation.

2.2.3 Signal Analysis

Several detection mechanisms have been proposed using signal analysis and packet counting tools [37, 38, 39, 40, 3, 4]. Signal analysis mechanisms rely on anomaly (or change) detection to detect malicious flows in the network. Like our scheme, these are also detection tools. These mechanisms compute a baseline for normal traffic behavior and search for traffic patterns that deviate from this baseline. The degree of anomality of specific traffic patterns from the baseline determines the severity of the misbehaving traffic.

Wavelet Transformations In [37], Kim, et al. propose a wavelet analysis based stub domain detection system. They rely on the heuristic that outbound traffic from a stub domain is likely to have a strong correlation with itself over several timescales. In their three step approach, they first generate a signal for analysis. They use fields in packet headers such as destination address and port numbers to derive the signal. Next, using discrete wavelet transforms, they transform the signal for statistical analysis. Finally, they use thresholds developed for previous time periods to detect anomalous patterns in

the signal. Since traffic has strong correlation over time and different timescales, the belief is that thresholds determined for this time period should hold for periods in the near future. The authors consider only top 100 flows and hence, are likely to miss weak DoS attacks.

In [38], the author classifies network traffic into short-term and long-term flows using a least recently used (LRU) cache. While long-term flows are stored in cache for a long time, short-term flows are removed from the cache. These short-term flows (miss traffic) are used to determine DoS flows in traffic. The wavelet transform for short-term flows does not show any correlation over long-term time scales. However, when a DoS attack is executed with many flows, some of the long-term flows will also be removed from the LRU cache and get classified as short-term flows. As a result, wavelet transformations on *miss traffic* during this period begin to show correlation over time and indicate the presence of DoS flows. This work assumes that a DoS attack from the network consists of several flows, each with different source addresses. It will miss an attack if the attacker(s) use only one source address for the attack packets. Also, if the attackers frequently change the source address, the attack flows will seem as short flows. Even in this scenario, the system may miss attacks.

In [39], the authors detect traffic anomalies also using wavelet transformations. They use IP flow information and SNMP information to perform traffic analysis. As with other works in this section, the traffic showed daily and weekly cycles. They use this traffic behavior to identify and isolate anomalous components in the traffic. They classify the anomalies as either long-lived events or short-lived events, based on the observed duration of anomality. Flash crowds are classified as long-lived and attacks are classified

as short-lived. Short-lived events are similar to normal bursty network behavior. Hence, they use mid- and high-frequency components in the wavelet transformations to determine these anomalies. The authors demonstrated four types of events in this work, namely network outages, flash crowds, attacks and measurement failures.

All the above mechanisms can detect DoS attacks like our system. However, these anomaly detection schemes work on traffic aggregates and require significant variations in the traffic to detect the anomalies. Hence, if attacks in the stub domain network have a low flow rate, the same will not be detected by wavelet transformation and analysis methods. Secondly, these schemes do not identify the target of an attack. They only identify the presence of an attack. Hence, additional mechanisms are required to identify the victim. Our system, on the other hand, detects an ongoing attack as well as identifies the victim of the attack. However, unlike the work in [39], our system cannot detect attacks other than bandwidth attacks or other network events.

Sketch Based Detection In [40], the authors argue that the above change detection techniques can typically only handle a relatively small number of timescales. Hence, these schemes cannot be applied in ISPs with high traffic volume and where anomalies may not be apparent if highly aggregated traffic data (e.g., SNMP data for five minute intervals) is used for analysis. The authors assume that keeping per-flow state is too expensive in these networks. They use a modified sketch, $k-ary$ sketch, as a data structure to store the network data. The sketch stores interesting data in state and time efficient manner. Thus, they are able to analyze large amounts of network data at a very low level of aggregation. Then they apply different change detection (or time forecast) models to detect significant

anomalies. This scheme require significant pertubations in the traffic flows to be able to detect the presence of anomalous traffic. Hence, while this scheme may detect a DDoS attack in transit networks close to the attack victim, it will miss attacks in source domains where the individual DDoS flows are likely to have a low flow rate. Also, like the wavelet transformation schemes, it cannot determine the identity of an attack victim.

2.2.4 Intrusion Detection Systems

Attackers use compromised hosts (stepping stones) to avoid detection. The process of compromising an end host is refered to as Intrusion. Networks and hosts can be broken into for several reasons such as data theft, sending spam and so on. As we describe in Section 3.1.1, these compromised hosts are also used as attack hosts during a DDoS attack [41]. Hence, mechanisms that detect and prevent intrusion attempts are also useful to prevent DDoS attacks. These systems are called Intrusion Detection Systems. In [42], the authors use packet sizes and timing of packets to determine the presence of stepping stones in the network. Stepping stones is an after fact of a successful intrusion. However, if they are identified, they can be isolated so that they do not participate in nefarious activities. Stepping Stones work is part of a larger project called Bro [43] which is a real-time intrusion detection system. Snort [44] and Dshield [45] are well-known intrusion detection systems. The size and scope of DDoS attacks will be severely limited if these systems are effective and widely deployed in the Internet. However, intrusion detection is a hard problem and typically deployed only in sensitive enterprise networks. Hence, a good DDoS detection system such as our is still invaluable.

[46] is a very comprehensive site for information about recent popular DDoS attacks. It provides the history of DoS attacks and information about different DDoS attack tools. It also links to news articles related to DDoS attacks in popular media. The site provides rich information about different DDoS detection and mitigation mechanisms.

Chapter 3

Detection System

In this chapter, we first discuss the characteristics of a bandwidth attack and how it is executed using compromised hosts. We discuss the TCP protocol and how bandwdith attack flows differ from TCP flows. We then present the heuristic to detect these attacks when the attacks are executed using TCP packets. We then describe our detection system. We start by defining a few terms which we use throughout this dissertation and then give an overview of how the system functions. We next describe the issues with deploying such a system in the network. and how we resolve these issues. We give the functional architecture of a *monitor* which is the basic component in our system. We discuss the significance of fast and slow memory at the monitors. We conclude this chapter by describing types of bandwidth attacks our system detects.

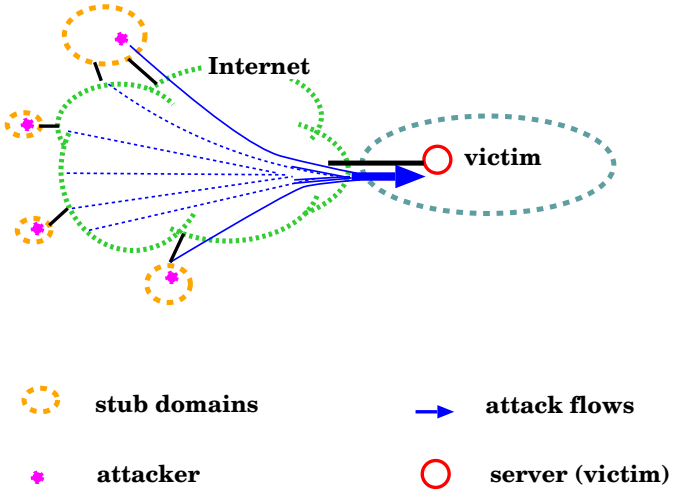

Figure 3.1: Bandwidth Attacks

3.1 Bandwidth Attacks

Bandwidth attacks flood a victim's access link with a large number of packets that are of no use to the victim. Since the victim's access links are congested, legitimate packets to the victim are dropped. Hence, other hosts in the Internet cannot communicate with the victim resulting in a denial of service for the victim as well as those other hosts interested in communicating with the victim. For most attacks [46], the victim of a bandwidth attack is a well accessed server and the attackers are compromised residential hosts acting as "bots". Hence, several bots need to participate in a bandwidth attack for the attack to be successful. Moreover, a bot may send attack packets at a rate much lower than it is capable of in order to prevent its discovery either to the legitimate user at the bot or to the bot's ISP operator. In such a case, even more bots need to be involved for the attack to be succeed. A typical bandwidth attack is shown in Figure 3.1. Since the attackers are distributed all over the Internet, this is distributed DoS (DDoS) attack.

3.1.1 Precursor to the Attack

Executing a DDoS attack requires access to several end hosts. The attacker uses compromised end hosts called bots or slaves to serve as attacking machines. A successful bandwidth attack requires several attackers. Hence, the process of compiling the set of bots is elaborate and typically drawn out. The attacker has a small set of compromised hosts called masters. The attacker uses the master hosts to discover the slaves and to orchestrate the attack. Master hosts use automatic tools that scan the Internet for end hosts with security holes. Vulnerable machines are then compromised using the discovered security holes. These machines are used as slaves. The attacker communicates with the slaves through the masters. The communucations are typically carried using IRC channels to avoid detection of the elements involved in the process [41]. The communication arrangement between the elements of an attack are shown in Figure 3.2. The attacker relays to the master nodes the target of a bandwidth attack and the time at which the attack should start. In the figure, the target is a.com and the attack time is X. Master nodes in turn communicate this information to the slaves. They spread the message propogation in time to avoid suspicion. At the determined time, the slaves carry out the bandwidth attack by sending a large number of attack packets. Bots are also used for other malicious activity such as for mailing spam and intruding networks.

3.2 Bandwidth Attacks Using TCP Packets

In order to understand our heuristic to detect bandwidth attacks using TCP packets, it is essential to understand how TCP functions. Here, we briefly cover TCP functioning.

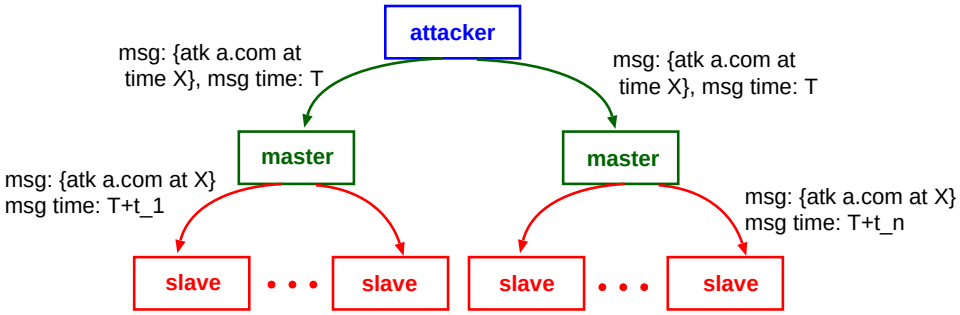

Figure 3.2: Communication between the Executioner and the Slaves

The primary role of TCP is to ensure reliable communication using ordered packets between applications on end hosts in a fair and efficient manner. A TCP session between two end hosts has three phases: connection establishment, data transfer and connection terminiation.

The TCP connection establishment phase is shown in Figure 3.3. A client initiates the connection by sending a SYN packet to a server and the server responds with an ACK packet. Server also tags its own SYN to the ACK packet. The client replies to server's SYN with its ACK and the TCP connection is established. If a host misses the ACK it expects, it resends its SYN as shown in Figure 3.4.

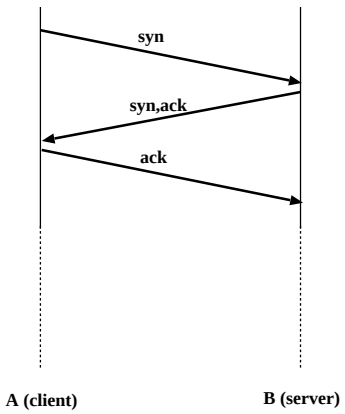

Figure 3.3: TCP Connection Establishment

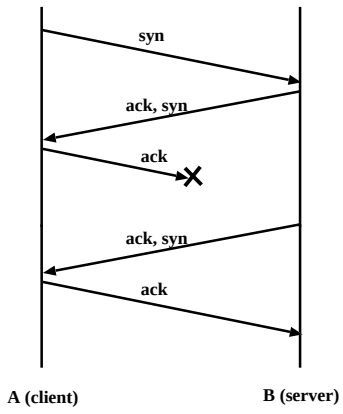

Figure 3.4: TCP Connection Establishment - Packet Loss

Throughout the connection, the client informs its "window size" to the server using the packets it sends to the server. The client's window size is the number of bytes it is willing to accept from the server at the time the client sent the packet to the server. The window size is called the client's advertised window. Similarly, the server advertises its window to the client using the packets it sends to the client.

The simplified data transfer phase of the TCP connection for bulk transfers is shown in Figure 3.5. After a connection is established, the client (receiver) sends a request and the server (sender) ACKs the request. The server at this point is said to be in "slow start" phase. A server in slow start should determine the packet transmission rate the network can sustain without packet losses for its packets. To determine this, the server uses a variable called *congestion window*, also represented as *cwnd*. Congestion window is the number of unacknowledged bytes the server can have for the client in the network. When the connection is established, the congestion window is set to one. The server increments cwnd by one packet for every ACK packet it receives from the client. The sender, at a time, can transmit up to the minimum of congestion window and the receiver's advertised window. Let the minimum of sender's congestion window and receiver's advertised window be called *sending window*.

The client, on the other hand, uses "delayed acknowledgements". On receiving a data packet, before sending the ack packet, it waits for up to 200 ms to see if it can piggyback the ack on some other data packet it may send within the next 200 ms. This saves it from sending the ack in a separate packet. However, if it receives another data packet from the server, it should ack the two outstanding packets immediately without delaying the ack anymore [5]. Thus, the client sends an acknowledgement packet for

at least every other packet. This is shown in the figure. From the figure, it is clear that a legitimate TCP connection, without packet losses, has an outgoing-to-incoming packet ratio between 0.5 and 1.0 at the client or the receiver and between 1.0 and 2.0 at the server or the sender. In general, a TCP connection without packet loss will have a ratio between 0.5 and 2.0 at each of its end hosts.

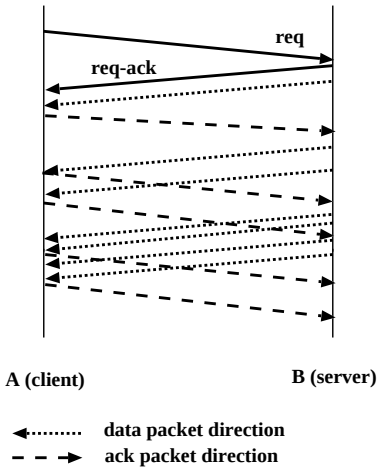

Figure 3.5: TCP Connection Data Exchange

Now consider the case when the TCP connection experiences packet losses. The sender can learn of an undelivered packet in one of two ways.

- the sender received three or more duplicate acknowledgements for a packet it sent.
- the sender sent a packet but did not receive an acknowledgement for the packet within a precalculated time since the packet was sent.

In either case, when the sender discovers that a packet is dropped, it sets a variable called *ssthresh* to half of the size of the current *sending window*. Recall that sending window is the minimum of sender's congestion window and receiver's advertised window. Further, if the sender discovers the packet loss due to non-receipt of an acknowledgement,

it sets its congestion window (and hence, the sending window) to 1. Now, if congestion window is less than or equal to *ssthresh*, the connection will function as if it is performing slow start. During this phase, the sender increments its congestion window by one packet for each acknowledgement it receives. Slow start continues until the congestion window becomes greater than *ssthresh*. Whenever the congestion window is greater than *ssthresh*, the connection will be in congestion avoidance state. Note that the connection will be in congestion avoidance state even when the packet loss is discovered due to duplicate acknowledgements since even then congestion window will be greater than *ssthresh*. During congestion avoidance, the sender increments the congestion window (*cwnd*) by at most one packet for every packet round trip time. Thus, for each acknowledgement the sender receives when the connection is in congestion avoidance, the sender increments the *cwnd* by $\frac{1}{cwnd}$. Since the sender can have at most *cwnd* outstanding acknowledgements, it can increase *cwnd* by at most one packet for each round trip period when in congestion avoidance state.

During the period of occasional packet losses, the sender will be in congestion avoidance state and reacts to packet loss by increasing the sending rate at a much slower rate than during the beginning of the connection. If the packet loss is more frequent, the connection could know about the packet loss through duplicate acknowledgements but will eventually realize about it due to non-receipt of acknowledgements. Then, the connection will restart in slow start phase and transits to congestion avoidance state when the congestion window size is half the size when packet losses occured. In congestion avoidance state, it again slowly increases the sending rate so that it does not send more than the network's forwarding rate for its packets. If the packet loss is severe, the sender

will always realize about the packet loss due to non-receipt of acknowledgements. The connection, in that situation, will lower *ssthresh* and eventually discover the rate at which it can transmit packets without packet losses. If the packet losses are extreme (which is usually the case during a bandwidth attack), the sender will receive very few acknowledgements for the packets it sends. The effective packet rate of the connection will be very low. If the sender does not receive an acknowledgement even for a duplicate packet it sent, it increases the time between the packet retransmissions from 1 second to 3 seconds to 6 seconds and so on. In effect, the sending rate at the sender will become the minimum possible.

Detecting Bandwidth Attacks Now compare the behavior of an attack flow with that of a legitimate TCP flow. A bandwidth attack is successful if the attack packets severely congest the victim's access link so that a majority of packets to the victim are dropped. Depending on the rate of packet losses, a legitimate TCP connection may go into congestion avoidance or slow start. If the packet loss is severe, the connection will be in slow start but the packet sending rate will the minimum possible, a packet is sent once every several seconds. On the other hand, the attack will last only if the attacker(s) continue to send the attack packets at high rate. The attack packets too undergo congestion at the victim and many of the attack packets will be dropped. Only a fraction (say κ) of attack packets reach the victim and even if the victim responds to all attack packets it receives, the outgoing-to-incoming packet ratio at the attacker for the victim's flow will be at least $\frac{1}{\kappa}$. Typically, the congestion during a bandwidth attack will be severe and hence, κ will be much less than 0.5 and hence, the ratio will be much greater than 2.0. Recall from earlier

discussion that legitimate flows have an outgoing-to-incoming packet ratio of at most 2.0. If the victim does not respond to the attack packets, the ratio will be even larger. The attacker(s) may spoof the source addresses of the attack packets either to hide their identity or because the attack is a reflector attack. If the attack packets are spoofed, even if the victim responds to these packets, they do not reach the attacker(s). Again the ratio will be much larger than $\frac{1}{\kappa}$ at the attacker for victim's flows. Thus, a detection system can use packet ratios for TCP connections to detect bandwidth attacks using TCP-like packets.

3.3 Detection Philosophy

Any DDoS attack detection system should have the following important characteristics for it be practical and deployable in real networks.

- The most important requirement of a detection system is that it should be **feasible** to deploy such a system. In other words, a practical detection system should be implementable — the input for the detection algorithm should be easy to collect and maintain, the algorithm should have low computation costs and the output of the algorithm should be useful to detect as many attacks as possible with few false positives, if any. Typically, the algorithm input should be computable from local sources using simple mechanisms. Also, the cost of deploying a detection system should not be high. In most circumstances, this would mean that the system should be deployable with minimal changes to the existing infrastructure. Otherwise, cost of the system will be a deterrant to deploy these systems.

- The detection system should **flexible**. The system should be able to tradeoff de-

tection accuracy with resources available for the system. With more resources and more detection time, for example, the system should be able to detect most attacks without any false positives. With some false positives, the system should detect attacks more quickly. If the system has few resources, it must detect at least strong attacks. The system should be simple and intuitive so that system parameters can be tuned such that the system performs in a predictable manner.

- The detection scheme should be **scalable**. The detection scheme should able to handle high rates of network traffic. In effect, the system should not become obsolete if the traffic rate in the network suddenly increases. The system should be able to fail gracefully with increasing network traffic. The system should be flexible such that with additional resources, even with increased traffic, it should be able to detect attacks better.

- The system should be **useful**. The system should be able to detect ongoing attacks and identify the victim(s) and the source(s) of an attack. The system should be able to distinguish between legitimate and attack traffic to the victim and filter attack traffic such that the effect of the attack is minimized. The system should not have a single point of failure. More importantly, adversaries should not be able to manipulate the system such that the system flags legitimate packets as attack traffic and filter such traffic. I.e., the detection system should not be a source of DoS attack. Equally important, an adversary should not be able to hide an actual attack using other legitimate or attack flows. Lastly, the system should not be prone to DoS attacks against itself. A system such as this will provide few incentives to be attacked

by a malicious user.

- Finally, the system should be **adaptable**. Most systems are designed to handle current attacks. A good system, on the other hand, detects current attack types and with few changes, also detect new attack types. An ideal system should detect all types of attacks under an attack class for which it is designed.

Our system detects bandwidth attacks that originate in the stub domains where the system is deployed and have the following properties. The attacks use TCP packets as attack packets and the attack flows have a much larger outgoing packet rate than their incoming packet rate. Our system consists of a set of *in-network overlay nodes*, overlay nodes deployed at routers in the network. These nodes, with the help of their associated routers, sample packets to monitor the network. They process the sampled packets to build relevant state, in a scalable manner, about flows in the network. These nodes coordinate among themselves to detect ongoing attacks using the state maintained at the nodes.

Our system, with its overlay architecture, requires minimal changes to the existing infrastructure. In our system, the process of collecting and maintaining the state of network flows is distributed. The cost for collecting and maintaining this state as well as the cost for processing this state to detect attacks is low in our system. It is flexible, such that with more resources, it can detect attacks better. We will show in later chapters, using simulation results, that our detection system generates few false positives. Because of our system's distributed architecture to collect state and detect attacks, it has no single point of failure. For the same reason, it can detect attacks in high-speed networks as well as in multi-gateway networks. It can identify the victim of an ongoing attack. The

system's scale of deployment determines the system's ability to identify the source(s) of the attack. The deployment scale, along with routers' capabilities to selectively forward flows to their overlay nodes, also determines the system's ability to distinguish between legitimate and attack flows to the victim and filter attack packets. Finally, the detection algorithms are executed at overlay nodes. Thus, it is easy to enhance the detection algorithms to reflect the changes in traffic characteristics and to implement newer algorithms to get newer attacks. Thus, our system to detect bandwidth attacks using TCP packets is feasible, flexible, scalable, useful and adaptable.

3.4 Nomenclature For Our Detection System

Before we further discuss our system, we define some terms that we use throughout this dissertation. We assume attackers may spoof their packets' source IP addresses. The detection schemes therefore identify flows by the IP address external to the AS. Multiple traffic streams to the same external IP address are considered the same *flow*. Due to this, irrespective of whether an attacker spoofs source addresses or multiple attackers are present in the domain, all packets of an attack at a monitor will be mapped to the same flow. We use the term "attacker" to mean a host within the monitored stub domain that participates in the attack by sending attack packets to the victim during the attack. Thus, when we refer to an attacker, we mean the compromised end hosts (i.e., slaves or bots) that are present in the stub domain and are participating in the attack.

The monitoring nodes of our detection system use the ratio of a flow's "outgoing" to "incoming" packets to determine if the flow is an attack flow. Here, incoming and

outgoing are defined with respect to monitored source domain (i.e., packets coming into the monitored domain are defined to be incoming and are called outgoing otherwise). We define the ratio of a flow's outgoing to incoming packets as the flow's ratio. A flow is considered *legitimate* if its flow rate is not asymmetric (i.e., the outgoing packet rate is not very large compared to incoming packet rate or vice versa). In other words, a flow is legitimate if, not accounting for packet losses, its ratio is within the range specified by the TCP protocol. As we described previously, this ratio range is between 0.5 and 2.0. A suspect flow is a flow that a monitor suspects has a flow ratio more than 2.0 or less than 0.5. As we will demonstrate later, suspect flows may be legitimate but all attacks will be suspect flows.

A flow is considered asymmetric if the path of its outgoing packets is different from the path of its incoming packets. However, we are interested only of the flow paths within the AS. Thus, we call a flow *symmetric* if the incoming and outgoing packets have the same path within the AS. Otherwise, the flow is called *asymmetric*. Note that an asymmetric flow is still legitimate if the flow has symmetric outgoing and incoming packet rates. Finally, our detection scheme maps a flow to one or more sets of flows. We call this mapping *aggregation* and refer to a set of flows as a *flow aggregate* or an *aggregate*.

3.5 Detection System Overview

Our detection system detects bandwidth attacks that use TCP packets if the attack flows have high packet ratios due to reasons described above. Our system consists of a set

of overlay nodes distributed at routers within a stub network. Each node is associated with a router and monitors the traffic at its router using packets randomly sampled at the router. The node maps each flow at the router to one or more set of flows, called *aggregates*. Using the sampled packets, it counts outgoing and incoming packets belonging to each aggregate. It uses the packet counts to estimate the flow ratios of different aggregates. As described previously, attack flows will have a flow ratio much larger than 2.0. As a result, an aggregate containing an attack flow is also likely to have a flow ratio greater than 2.0. Each node, using the estimated flow ratios for sets of flows, locally determine the set of suspicious aggregates, i.e., the sets that are likely to contain attack flows. The flows belonging to a node's suspicious aggregates are regarded as *suspect flows* by the node. Each node collects information regarding its set of suspect flows. It communicates with other overlay nodes in the network to determine which of its suspect flows are attacks. A suspect flow is considered an attack if it is suspect at all overlay nodes on its path. Hence, the overlay node can determine a suspect flow is an attack by verifying if it is suspected at all other overlay nodes on the flow's path. Thus, attack flows are determined. Since flows are identified by the destination of the outgoing packets, the victim of the attack is readily identified by the detection system.

Our system is able to process packets at line-speeds and scales well with increasing network traffic rates because we employ packet sampling to collect flow state and maintain flow state in an aggregated manner. We use overlay nodes to process the sampled packets and maintain state. Hence, routers in the network do not require any changes. Nodes are deployed throughout the stub domain and communicate with each other about suspect flows in the network. Thus, they can be deployed to detect attacks in all types

of stub domains including multi-gateway stub domains. In the remainder of this chapter, we discuss in detail the difficulties with deploying overlay nodes within the network and how our system overcomes these problems. In later chapters, we describe the exact mechanisms to detect different bandwidth attacks using TCP packets in single- and multi-homed domain networks.

3.6 Deploying Our System

Our system consists of a set of overlay nodes distributed at routers within a stub network. These nodes monitor network traffic using randomly sampled packets and detect DoS attacks. We refer to overlay nodes deployed within a network as *in-network overlay nodes*. In the context of our detection system, we refer to these in-network overlay nodes as *monitors*. Sampling helps a monitor to obtain an accurate picture of monitored traffic using only a fraction of packets. This results in low packet processing overhead at the monitors and enables the monitors to have simple hardware and yet process packets at line speeds.

Each monitor is associated with a router in the AS [1]. It constructs the state of flows at its router using packets that transit the router. A monitor can easily access these packets if it is deployed on the data path at the router. However, this can result in two problems. First, it will increase the processing time for the packets at the router. Second, if the monitor misbehaves or is compromised, it could affect the correct functioning of the router.

[1] Unless noted otherwise, a router is always discussed in the context of its associated monitor.

3.6.1 Monitor-Router Interaction

Port-Mirroring versus Using Line Taps A monitor can access packets at the router without being deployed on the data path using two mechanisms. One mechanism is for the monitor to use line taps at the router's interface cards [9, 47, 48]. A line tap is a specialized hardware device that is deployed at a router's interfaces (or ports) and provides access to the traffic at those ports. A simpler mechanism that does not require additional hardware is for the router to port-mirror its traffic. Also known as port spanning, it requires the router to have port-mirroring capabilities. Such a router will duplicate the traffic that occurs at one or more of its ports and directs the duplicate traffic onto a mirror port. The monitor will listen to the mirror port of the router to determine the packets transiting the router. The second approach is more economical as it does not require any additional hardware. In this work, we assume that routers can port-mirror their traffic and use this scheme for the monitors to access packets at their associated routers.

Port-Mirrored Links The monitors are only concerned with detecting attacks where the victims are outside the network. As such, they only require access to "external" or "non-local" packets[2]; they do not need access to local packets. Hence, a router only has to port-mirror links that lie on the path between the router and the AS's gateways. For example, consider the network in Figure 3.6. Assume that hop count is used as the distance metric for routing in this network. Then, at router R_1, links e_3 and e_4 lie on the path between itself and the network gateway G. Hence, R_1 only has to port-mirror traffic on these two links for R_1's monitor to access external packets at R_1. Similarly, R_3 only

[2]Here, "local" packets are those that are exchanged by two end hosts that are both within the stub AS. External packets are those that are destined to or received from external hosts.

port-mirrors link e_7 and G port-mirrors its external links, e_1 and e_2. Each router can use gateway addresses and its routing table to determine which links lie between itself and the gateways and thus identify the ports that it has to port-mirror.

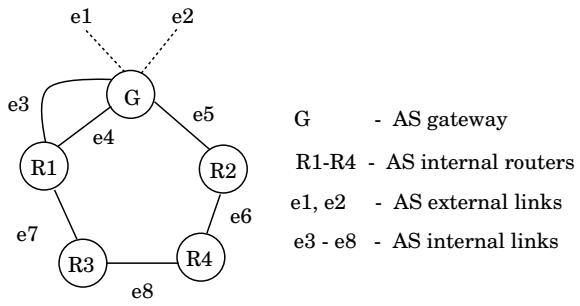

Figure 3.6: Port Mirroring at a Monitor

Address Classification As mentioned earlier, a monitor requires to access only to external packets. Choosing port-mirror links as described previously eliminates those links that may carry significant local traffic. However, port-mirror links at non-gateway routers may transmit local as well as external traffic. Thus, monitors can decrease their processing overhead if they can easily identify local packets among port-mirrored packets. We describe a mechanism here for monitors to achieve the same. We assume that a monitor knows its router's routing table and the AS topology. If the AS has a contiguous block of IP addresses for its hosts, it can classify a packet as local or external by comparing the source and destination IP addresses against the block of IP addresses. If both the IP addresses in a packet are within the AS's block of IP addresses, the packet is considered local and external otherwise. If only the destination (source) IP address is within the address block, the packet is considered incoming (outgoing). The interesting case is when both the IP addresses are outside the address block. In such a case, the source address

may have been spoofed and either of the two addresses of the packet may be the target of an attack. (If the spoofed source address in the packet is the target, such an attack is called a reflector attack). If the AS does not have a contiguous block of IP addresses, the monitor uses the routing table and the network topology to classify packets. It can consider a packet as outgoing if it will be routed to one of the AS's gateways and incoming otherwise. It will have to use the addresses of the AS topology to particularly distinguish between external and local packets. If this step is computationally prohibitive, the monitor can skip this step at the cost of additional packet processing.

A monitor receives port-mirrored traffic from its corresponding router and performs simple tests to identify (seemingly) anomalous traffic flows at line speeds. In addition to performing this local determination, monitors deployed within the AS compare their suspects with one another and collectively determine which suspects are indeed attacks. To effectively collaborate, monitors in the system form a logical network, called *overlay network*.

3.6.2 Sampling

Monitor nodes deployed "near" routers detect possible attacks. This arrangement can lead to a number of problems.

- The monitor's packet processing rate could be far less than the traffic rate at the router,
- The router's bandwidth to the monitor could be less than the traffic rate, or
- The router may not be able to port-mirror all of its traffic.

We employ uniform sampling to address all of these issues. The sampling rate must be low enough for the monitor to process it but high enough for the monitor to quickly capture the traffic profile at the router.

Sampling rate may be set as follows. Assume that for any of the reasons mentioned above, the monitor is able to process only m packets every second. For example, if the monitor processing is the bottle-neck and the monitor takes c processing cycles to process each packet, then m will be $\frac{1}{c}$. Now, if the instantaneous aggregate packet arrival rate of the port-mirrored links at a monitor is r packets per second, then the monitor's sampling rate would be $\frac{m}{r}$. Since the instantaneous packet arrival rate may vary with time and we do not know the value beforehand, we compute the aggregate packet rate frequently over short intervals and use it to recompute the sampling rate accordingly. Thus, each monitor will have different sampling rates determined by the packet arrival rate at the monitor. In Section 5.2, we explore the tradeoffs between sampling rate and the speed of detection. Note that it is crucial that the sampling be performed uniformly at random so as to avoid biasing of flows or, worse, an attacker determining how to make its traffic invisible to the monitor.

3.6.3 Monitoring Approach

Sampling, by itself, is insufficient for monitors to have simple hardware and yet process packets at line speeds and detect attacks. First, monitors require to maintain state for the flows at the router to accurately distinguish between legitimate flows [3] and attacks. Routers may have millions of active simultaneous flows. Thus, maintaining per-flow

[3] flows that are not bandwidth attacks

state is not feasible. Our monitors maintain state in an aggregated manner to account for possible large number of flows at its router.

Second, packet sampling reduces processing overhead at the monitors. However, sampling rate at the monitors must be sufficient for monitors to detect anomalous flows quickly and accurately. Thus, monitors can further reduce packet processing overhead if their per-packet processing is very low. At a monitor, the per-packet processing consists of few simple functions such as mapping the packet to an aggregate, updating counters associated with the aggregate and performing light-weight tests on these counters. At regular intervals, the monitors perform more complex functions to detect anomalous flows and detect attacks. We next describe various generic per-packet and periodic functions of monitors in more detail.

3.7 Functional Architecture

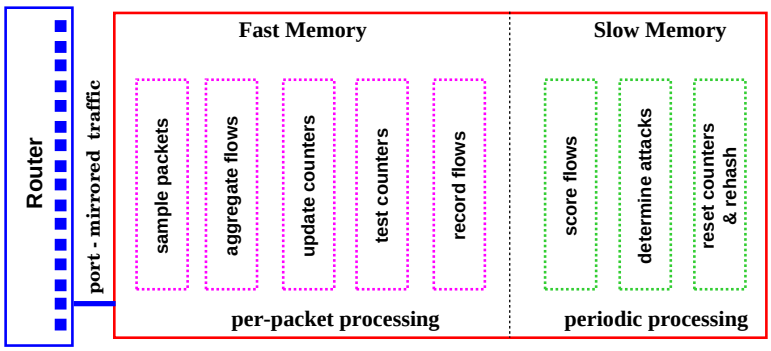

Figure 3.7: Monitor Architecture.

In Figure 3.7, we present a schematic functional architecture of a monitor. The monitor is composed of two functional components: a per-packet processing component

and a periodic processing component. The per-packet processing component performs lightweight processing on each sampled packet. It maps the packet to one or more sets of flows using a random hash function and updates counters associated with the set of flows to the which the packet is mapped. The updates are typically simple computations such as addition and are determined by the specific detection scheme and packet direction (i.e., whether the packet is "incoming" or "outgoing"). It performs simple tests on these counters and if these counters exhibit suspicious behavior, the flow corresponding to the packet is assumed to be suspect and is recorded. Because a monitor records only suspect flows, the per-flow state due to suspect flows will be low if the tests are accurate. The periodic processing component is responsible in identifying attacks, if any, in the network. The monitor periodically computes a score for suspect flows which is a measure of anomality of these flows. This scoring helps the periodic processing component to identify legitimate flows among suspect flows. Further, the periodic components of the monitors exchange information among themselves to determine attacks quickly and accurately.

3.8 Fast Memory vs Slow Memory

Each monitor builds and updates its flow state by processing packets sampled at the router. Since the monitor accesses this state on a per-packet basis, its access latency to the state has to be very low. Otherwise, the monitor will not be able to process packets at line speeds. Hence, a monitor has to maintain its flow state in its fast memory. As the size of the fast memory is limited, the monitor cannot maintain state on a per-flow basis.

Instead, it aggregates flows at its router and maintains state on a per-flow-aggregate basis. All monitor functions that require access to the state on a per-packet basis are performed using state in the fast memory.

The monitor transfers its state from fast memory to slow memory intermittently. This transfer may occur either periodically or when the fast memory is low on available memory. Monitor's functions that do not access flow state for every processed packet may use flow state in the slow memory. Since these functions are not performed frequently, these functions may be more complex than per-packet functions of the monitor. These functions include flow scoring, rehashing and information exchange between the monitors. These functions aid in accurate determination of attack flows among suspicious flows.

3.9 Types of Detected Attacks

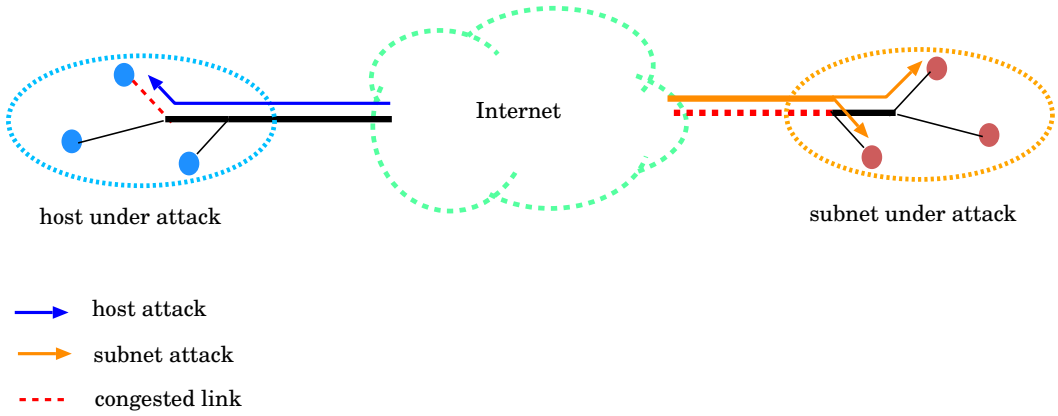

Figure 3.8: Types of Bandwidth Attack

Bandwidth attacks may be classified as either *host attacks* or *subnet attacks*. In host attacks, all attackers participating in a bandwidth attack target a single host. In subnet

attacks, the attackers target multiple hosts in a subnet. These two attacks are shown in Figure 3.8. As shown in the figure, the link targetted in a host attack is the victim host's access link. In the subnet attack, this link is the subnet's access link. The difference between the two attacks is that in host attack, all attack packets have the same destination address, that of the victim host. In a subnet attack, the packets can use any of the multiple addresses in the victim subnet as the destination address. In other words, a subnet attack is composed of several victim attacks. Thus, a subnet attack can be more diffuse and harder to detect in source domains than a host attack if the detection system identifies attacks by victim host's address. If the size of the victim subnet increases, the attack will be more diffuse and much harder to detect. We will show in Chapter 6 that detecting attacks exclusively using subnet prefixes has its own problems; the size of subnet prefix to monitor is unknown and monitoring subnet prefixes can result in host attacks being missed.

Our monitors execute two independent algorithms; one to detect host attacks and the other to detect subnet attacks. The host attack detection scheme is described in Chapter 4 and the subnet attack detection scheme is described in Chapter 6. Both the schemes can be deployed in single- as well as multi-homed networks and in networks with only symmetric flows as well as in networks where some of the flows are asymmetric [4]. The host attack detection scheme has lower per-packet processing and memory overheads and identifies attacks, including weak attacks, quickly. The subnet attack detection scheme, requires relatively more processing and state, compared to host attack detection scheme.

[4] A flow is asymmetric if its incoming and outgoing packets have different paths within the source domain. Refer to Chapter 4.2 for further explanation.

Moreover, the total attack rate originating from within the source domain also has to be stronger to be detected by this scheme compared to host attack detection scheme. However, it can detect subnet attacks irrespective of the size of the victim subnet, including host attacks [5]. In the case of a host attack, the attack rate has to be higher for the subnet attack detection scheme to detect the attack compared to host attack detection scheme.

In Chapter 4, describe the host attack detection scheme in single- and multi-gateway networks. In Chapter 6, we describe the subnet attack detection scheme in single- and multi-gateway networks.

[5] A host attack is also a subnet attack where the size of the subnet is one host.

Chapter 4

Host Attack Detection

Bandwith attacks are of two types — host attacks and subnet attacks. The difference is, a host attack has a single host as the attack's victim while a subnet attack has multiple hosts in a subnet as victims of the attack. In host attacks, attackers send a large number of attack packets to a single victim and congest the victim's access link. To detect these type of bandwidth attacks, each monitor, with the help of packet copies obtained through port-mirroring or line taps, counts incoming and outgoing packets belonging to flows at its associated router. A flow is identified by the destination address of the outgoing packets (and thus, the source address of the incoming packets). After processing each packet, the monitor performs simple tests on the counts it maintains to identify (seemingly) anomalous traffic. Periodically, the monitor compares its list of anomalous flows with other monitors to determine suspects which are indeed attacks.

In Figure 4.1, we present the monitors' schematic for host attack detection. Per-packet processing component of the monitor is used to quickly identify the set of anomalous flows. This set may also contain legitimate flows, i.e., flows that are not attacks. The

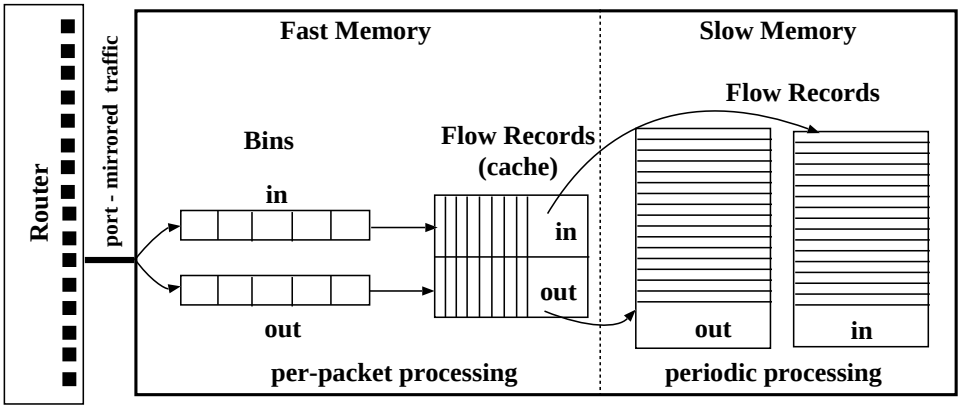

Figure 4.1: Host Attack Detection - Monitor Architecture

periodic processing component operates on this set of flows to accurately identify attacks. In the rest of this chapter, we elaborate on these two processing components of the monitors in single gateway networks. In the next section, we assume that all flows in the domain are symmetric within the domain, i.e., the paths of the incoming and the outgoing packets of a flow inside the domain are the same. In the following section, we describe the protocol enhancements required for deployment in multi-gateway stub networks. The enhanced detection scheme assumes that any number of flows within the domain may be asymmetric, i.e., the outgoing and incoming packets of flows within the domain may have different paths. In Section 5.3, we evaluate the scheme and the chapter with a discussion in Section 4.3.

4.1 Single Gateway Networks

4.1.1 Per-Packet Processing

A monitor's per-packet processing component creates a set of *suspects* by passively observing the packets at its associated router. For each flow, it counts the outgoing and incoming packets and compares the outgoing-to-incoming packet ratio against a threshold called the *attack threshold* to determine if the flow is anomalous. Since monitors may encounter large traffic volumes, they perform their per-packet processing using only fast memory (i.e., memory with lowest data access latencies). Size limitations of fast memory may make it infeasible for a monitor to have counters for all flows. Instead, the monitor randomly groups flows into *flow aggregates* and counts incoming and outgoing packets on a per-aggregate basis. The monitor tests the outgoing-to-incoming packet ratio of a flow aggregate against the attack threshold and if the test fails, records all flows that map to the flow aggregate. An aggregate fails the test only if it has an anomalous flow. These aggregates are called suspicious and the flows that map to such aggregates are marked "suspicious". Suspicious flows are reported to the periodic processing component of the monitor which performs further analysis and determines attacks from amongst the list of suspect flows.

Aggregating Flows

Each monitor's per-packet processing component aggregates flows and stores state on a per-flow aggregate basis. A flow aggregate's state is represented in fast memory as

a *bin*[1] which consists of two counters: the number of the flow aggregate's incoming and outgoing packets (see Figure 4.1). (Here, "incoming" and "outgoing" are with respect to the stub AS.) Each monitor maintains a fixed number of bins. Each bin corresponding to one or more flows. Number of active flows at the router and the number of bins at the monitor determine the flow aggregation rate at the monitor.

Each monitor is preconfigured with a hash function which it applies on the external IP address (destination address for outgoing packets and source for incoming) of a packet to map the packet to its corresponding bin. A monitor uses only one hash function to minimize the number of memory accesses required for each processed packet. For each sampled packet, the monitor maps it to its appropriate bin and increments the corresponding counter (incoming or outgoing). The bin counters are reset periodically to reflect the current state of the flow aggregate. Otherwise, after a flow with very high packet rate, it may take a long time for the bin counters to reflect the ratios of active flows accurately (see the discussion below).

Detecting Suspect Flows

The bin counters described above represent the only network measurement our system requires. According to RFC 2581 [5], a legitimate TCP may have a data-to-ACK packet ratio of no more than 2. On the other hand, as described previously, flooding attacks often exhibit packet ratios that are well beyond 2. A monitor therefore uses the bin counters to detect attacks by observing the ratio of the bin's outgoing counter to its incoming counter, as in [3]. If this ratio exceeds a pre-defined *attack threshold*, $\mathcal{R}(\geq 2)$,

[1] We use the terms flow aggregates and bins interchangably

the monitor flags the bin. In practice, $\mathcal{R}$ is set slightly higher than two to account for dropped ACKs and variations due to random sampling of packets. If the ratio of the flow aggregate is greater than $\mathcal{R}$, the bin and each flow mapped to it is said to be *suspicious*.

Each time a monitor processes an outgoing packet,[2] it tests if the corresponding bin's ratio exceeds $\mathcal{R}$. If the bin is indeed suspicious, a *flow record* corresponding to the packet's flow is created in fast memory, if such a record does not already exist. The advantage of checking the bin ratio for every (outgoing) packet is that the monitor can dynamically collect information regarding suspicious flows rather than having to store information for all flows. If the monitor knows the identities of all flows at the router, then it may perform the ratio comparison periodically (instead of for every packet) and still be able to identify suspicious flows. Thus, the monitor trades off processing for lower fast memory requirements.

To avoid filling up fast memory, records are moved to slow memory either periodically or when there is not enough space in fast memory. Flow records are used to record suspect flows and some measure, or *score*, of their abnormality. The precise determination of scores is described in Section 4.1.2.

Flow Aggregation's Effect on Suspect Detection

Aggregating flows at monitors affects suspect flow detection in two ways:

- Attack flows will not be detected if the legitimate flows in their bin have enough incoming packets to compensate for the attacks' lack thereof.

[2] Since we are concerned with being the source of an attack, it is not necessary to perform this check for incoming packets.

- Attacks with rates significantly higher than rates of other flows in the bin will result in all flows in the bin being flagged as suspect flows.

Consider the example in Figure 4.2. The table in the figure lists the incoming and outgoing packet rates for the four flows. Flow 4 is an attack and its ratio is therefore considerably higher. The attack threshold, $\mathcal{R}$, is set to 3 for this example. If the flows are mapped as in (A), the bin with the attack flow will be flagged as suspicious and flows 3 and 4 will be added to suspect list. On the other hand, if flows are mapped as in case (B), none of the bins will be flagged and flow 4 will not be detected.

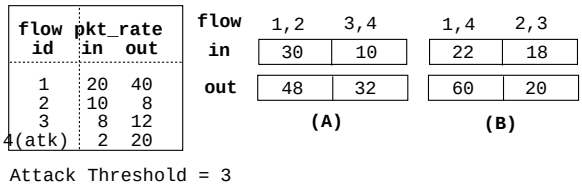

Figure 4.2: Aggregation Effect on Detection.

The effects of aggregation only increase as more flows are mapped to the same bins. Thus, there exists a tradeoff between the amount of fast memory required at a monitor and the monitor's detection sensitivity. For a given aggregation rate, detection sensitivity may be increased by periodic activies such as rehashing which are described next.

4.1.2 Periodic Processing

The monitor periodically takes as input the flow records from the per-packet processor and determines which (if any) of the suspect flows are indeed attacks. This is indicated in Figure 4.1, where it makes use of per-flow state in slow memory for this purpose. Periodically, it also communicates with other monitors to determine attacks earlier

and more accurately. At short intervals, the monitor also calculates a score for each flow record, allowing it to track anomalies at a per-flow as opposed to per-aggregate granularity. Over longer time periods, it generates new hash functions for the per-packet processor and collaborates with other monitors to vote on which suspects are attackers. Flow scores are calculated every second and hash functions are reset every five seconds. Flow scoring and rehashing steps are described in greater detail below.

Rehashing

At pre-defined *rehash intervals*, each monitor resets all of its bin counters and *rehashes*, i.e., it generates a new hash function which changes its packet-to-bin mapping. Rehashing aids in decoupling flows from their flow aggregates and helps in alienating anomalous flows of a flagged bin from other flows of the flagged bin. This is because, after rehashing, it is unlikely that the legitimate flows from that flagged bin are mapped to the same bin as the anomalous flows again. Setting the bin counters to zero at the beginning of each rehash interval ensures that the new mapping's state is not affected by the old.

The flow decoupling provided by rehashing has several benefits of note. As we observed in Section 4.1.1, an attack flow will not be detected if the other flows in its bin have enough incoming packets to compensate for the attack's lack of. Rehashing enables such attack flows to get mapped to other bins that may contain flows with lower incoming rates, increasing the probability of attack detection. Further, rehashing facilitates flow scoring (below), thereby reducing the number of suspect flows to be voted upon.

Flow Scoring

Flow records, when created, contain no information that indicates which of the included flows may be anomalous; each flow mapped to a flagged bin is an equally likely candidate. To help in classifying the flows with flow records as legitimate flows or attacks, a monitor scores suspicious flows periodically. The *flow score* measures how anomalous a given flow is by recording its corresponding bins' packet ratios across multiple rehash intervals. For a flow f, the score is initially set to zero. Every second, it is incremented by the difference between the ratio of the bin currently corresponding to f (say r_f) and the attack threshold $\mathcal{R}$. Since $(r_f - \mathcal{R})$ is greater than zero only if the bin is suspicious. f's score is incremented only if it is mapped to a suspicious bin in successive rehash intervals. As seen previously, this happens only if the flow is an anomalous flow. Hence, rehashing is important to flow scoring as described here.

The record is further processed, as described later, if the score crosses a *score threshold* and deleted when the score reaches zero. An interesting aspect of the flow score is that it captures the attack intensity. If the flow rate of an attack is high, r_f will also be high, the score is incremented more and will cross the score threshold earlier. Bin ratios are stored in fast memory and records are stored in slow memory. To reduce the number of fast and slow memory accesses, scores are computed periodically (and not for every processed packet). However, scores need to be computed at shorter intervals than rehash interval to capture the intermediate state of the bins accurately.

4.1.3 Detecting Attacks

A monitor may have a list of suspect flows that have exceeded the score threshold. Some of these suspects may be legitimate. Most of the suspect flows that are not attacks are eliminated from consideration at the monitor due to rehashing and scoring. The monitor narrows down the suspect list further using remote intersection and voting. Remote intersection works similar to rehashing. The difference is as follows: rehashing happens at a single node across different rehash intervals while remote intersection happens across different monitors during the same rehash interval. Thus, remote intersection is dependent on the fact that different monitors have different hash functions and hence, map flows differently. The remote intersection procedure is explained further below.

Remote Intersection

Every monitor forwards the records of suspect flows with scores above the score threshold to the next hop monitor on the flow's path. Similarly, it forwards any records from its previous hop monitors and forwards them to their respective next hop monitors.[3] The last monitor on a path performs an intersection of the suspect lists forwarded by monitors along the path. The flow identifier in the flow records is used to perform intersection of suspect lists. The monitor reports the result of these intersections as attacks to the detection system.

For a flow to be considered an attack, it needs to have been a suspect at each monitor on the path from the attacker to the network gateway. Since a given legitimate flow is

[3]For outgoing flows, for routing purposes, a single gateway network may be considered to be a tree rooted at the border router. Hence, all outgoing flows have the same next hop monitor.

unlikely to be mapped with an anomalous flow and flagged suspicious at each monitor, the resulting intersection will result in only the attack flows.

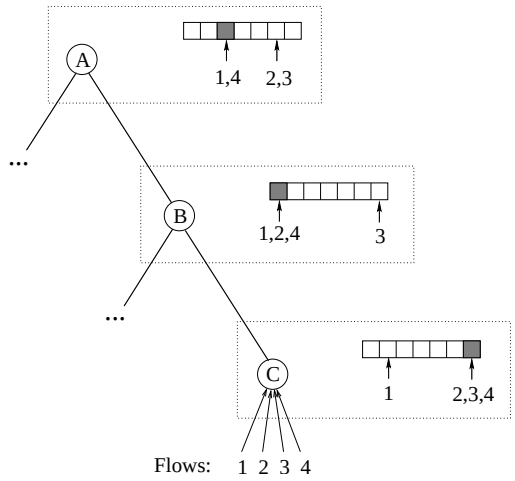

Figure 4.3: Anomaly Detection Example

Consider the example in Figure 4.3. Three monitors (A, B, and C) have detected anomalous activity (shaded boxes denote bins with ratios exceeding $\mathcal{R}$) and have their own list of suspects: $\{1,4\}$, $\{1,2,4\}$, and $\{2,3,4\}$ respectively. Since A is the topmost monitor on the path to this egress point, it collects these lists and performs the intersection, resulting in the final (correct) suspect list of $\{4\}$. A then reports the attacker.

Voting

A voting protocol is used to decrease the detection times and increase the detection rates. Due to sampling and different aggregate flow rates at each monitor, some nodes may take longer to detect anomalies than others. Moreover, monitors with low sampling rates may miss low packet rate attacks and hence, a simple intersection of suspects will fail to detect the attack. So, instead of strict intersection, the detection system uses a simple voting scheme. A flow is considered an attack if at least k out of a total n monitors

on the flow path report the flow as a suspect.

The detection scheme presented in this section is flexible, allowing for various configurations or extensions. Its distributed nature enables it to perform traceback. We describe the protocol extensions for detection system deployment in multi-gateway stub network in the next section.

4.2 Multiple Gateway Extensions

In ASes with multiple border routers, there are likely to be asymmetric traffic flows (i.e., their ingress and egress routers are different). Detection systems designed for single gateway networks will detect these asymmetric legitimate flows as attacks. Consider the network shown in Figure 4.4 with two border routers A and E and an asymmetric flow. Multiple deployments of a single-gateway solution, such as D-WARD, is insufficient because the data monitored at each gateway would not necessarily be independent. Any detection system designed for single gateway networks will flag the asymmetric flow in the figure as an attack. On the other hand, the monitors of our detection system are deployed throughout the AS. Hence, those monitors deployed on the flow's symmetric subpath in the network (e.g., monitors C and D in Figure 4.4) can sample the flow's packets in both directions. This helps the detection system to correctly distinguish between asymmetric flows and attacks, minimizing the number of false detection of attacks.

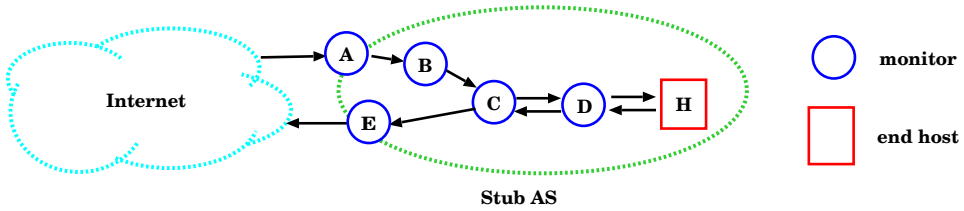

Figure 4.4: Multi-Gateway AS — Asymmetric Flows

4.2.1 Attacks and Asymmetric flows

In the extreme case, there may be no monitors deployed on the symmetric subpath of asymmetric flows. Then, the protocol described thus far will also flag asymmetric flows as attacks and hence, is insufficient. For example, in Figure 4.4, without monitors C and D, the asymmetric flow will be flagged as an attack by monitor E. However, monitors A and B would sample the incoming packets of the flow and can detect that the flow is anomalous in the incoming direction (i.e., disproportionately more incoming packets than outgoing packets). If E communicates with A and B, they can correctly determine that the flow is not anomalous. For the multiple gateway deployment discussion, we will consider only this extreme case where the common subpaths of asymmetric flows do not have any monitors. An example of this extreme case is shown in Figure 4.5. In this setup, monitors are deployed at routers A, B, C, D, W, X, Y, and Z. An asymmetric flow's outgoing packets exit the network at router A and its incoming packets enter the network at router W. Thus, monitors in the network will only sample outgoing or incoming packets of the asymmetric flows. In this multi-gateway scenario, to detect low rate attacks without generating many false positives, the detection scheme requires enhancements to per-packet and periodic processing components at the monitors. We describe these enhancements in

the next two sections.

4.2.2 Multi-Gateway Per-Packet Processing

Per-packet processing in a multi-gateway network is similar to that of a single gateway; monitors maintain aggregate counters for incoming and outgoing packets and create flow records when a bin's packet ratio seems suspicious. However, monitors perform two tasks differently: they apply a more elaborate test on bins' packet count ratios to detect the incoming components of asymmetric flows and record suspect flows differently. We describe these two modified per-packet functions next.

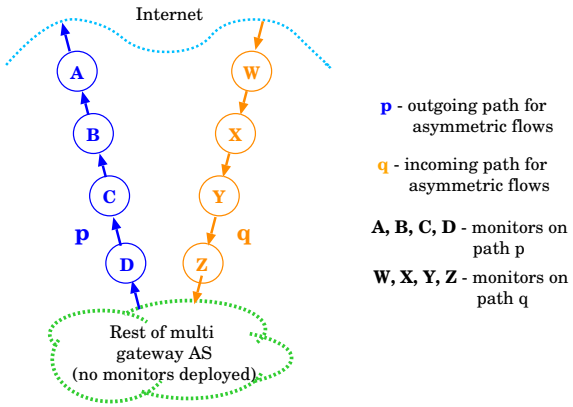

Figure 4.5: Multi-Gateway AS Detection System

Detecting Incoming Asymmetric Flows

Monitors must detect incoming components of asymmetric flows so that the detection system correctly distinguishes between attacks and legitimate asymmetric flows. From a monitor's perspective, the incoming and outgoing components of an asymmetric flow behave very much like an attack. The difference is that the incoming components

will have a large incoming-to-outgoing packet count ratio. Thus, a monitor may detect an incoming asymmetric flow by comparing the bin's outgoing-to-incoming packet ratio with the inverse of the attack ratio threshold. We accomodate this extension by requiring the ratio test to use two threshold ratios instead of one: $\mathcal{R}_{out}$, the overflow threshold, and $\mathcal{R}_{in}$, the underflow threshold. $\mathcal{R}_{out}$ is the same as $\mathcal{R}$ (≥ 2) from Section 4 and $\mathcal{R}_{in}$ is its inverse ($\leq 1/2$). Every time a monitor processes a sampled packet, it compares the corresponding bin's packet ratio against $\mathcal{R}_{out}$ and $\mathcal{R}_{in}$. If the ratio is greater than $\mathcal{R}_{out}$ or less than $\mathcal{R}_{in}$, the bin is flagged accordingly.

Flow Recording

A bin is flagged if its ratio is greater than $\mathcal{R}_{out}$ or less than $\mathcal{R}_{in}$. In single gateway networks, the flow corresponding to an outgoing packet is recorded if the packet's bin is flagged. In multi-gateway networks, the monitor creates a record for an incoming packet's flow if the bin corresponding to the incoming packet is flagged. Thus, in addition to outgoing suspect flows, records are created for incoming suspect flows also. Additionally, the monitor maintains a count of the sampled packets for outgoing flows. The flow scoring algorithm uses the count to score the outgoing syspects. The algorithm estimates the flow's outgoing packet rate using the counts and uses the estimate to minimize the number of false positives. The flow scoring algorithm is described in detail in Section 4.2.3.

4.2.3 Multi-Gateway Periodic Processing

The monitors' periodic activities in the multi-gateway case are also similiar to those of the single gateway. The differences lie in how messages are exchanged, flows are

scored, and voting is performed to classify suspects as attackers.

Consolidating Data with Rendezvous Points

In a single gateway network, the monitor closest to the border router consolidated the messages from other monitors. There may be multiple such monitors in a multiple gateway network. We therefore explicitly specify such a monitor, called the *rendezvous point*, for each flow in the network. A single rendezvous point may suffice for networks with few asymmetric flows while multiple rendezvous points may be used for networks that have significant number of asymmetric flows. A system-wide hash function can be used to map flows (using the external IP address) to one of the rendezvous points. For the purpose of this dissertation, we assume that the detection system has a single rendezvous point.

Each monitor delivers to the rendezvous point, its outgoing and incoming suspects gathered during the rehash interval. For outgoing suspects, the monitor also reports to the rendezvous point, the estimated packet rates of the outgoing suspects. Packet rates at a monitor are estimated by normalizing the count of outgoing packets of a suspect flow during the preceding rehash interval with the monitor's sampling rate and length of the rehash interval. We explain in the next section how we use flow rates of outgoing suspect flows to decrease the number of false positives.

In addition to performing the same data consolidation as the single gateway case, rendezvous points also have to differentiate between attacks and legitimate asymmetric flows. The rendezvous point matches outgoing and incoming suspects; if an outgoing suspect has a corresponding incoming suspect, it is deemed legitimate. Of course, this

will not capture all legitimate flows, so the rendezvous point performs further work, as explained in the rest of this section.

Flow Scoring

False positives, exclusively due to asymmetric flows, may occur for two reasons in multiple gateway networks. Consider Figure 4.6(A), which uses the network in Figure 4.5. Flow 1 is asymmetric with outgoing traffic on path p and incoming traffic on path q. Its traffic on p is flagged as an outgoing attack. However, since there are symmetric flows on q, they mask the incoming portion of flow 1's traffic; its bin in q has a ratio of 2/3 but $\mathcal{R}_{in}$ is 1/2 so it will not be reported to the rendezvous point as an incoming suspect. Thus, a rendezvous point using a simple matching of outgoing and incoming suspects would incorrectly classify this flow, yielding a false positive. In Figure 4.6(B), we see another source for false positives. In this case, flow 2 is a symmetric flow on path p, aggregated with asymmetric flows on p. This bin on p will overflow. Though there is a corresponding underflowing bin on q, flow 2 is not mapped to it. Thus, again, a simple matching of outgoing to incoming suspects will yield a false positive. In short, false positives arise because some flows *mask* portions of other flow's asymmetric traffic

Observe that, in Figure 4.6(A), if flow 1 has a higher packet rate than the sum of the rates of 5 and 6, then flow 1 could not have been masked. Similarly, in (B), if flow 2's rate is greater than the sum of the rates of 3 and 4, it could not have been flagged incorrectly. Thus, a reasonable heuristic to determine if a flow is being masked, and hence may be a false positive, is to check if its flow rate is less than the expected sum of the other flows' rates in its bin. We refer to this expected sum of other flows' rates in a bin as the *score*

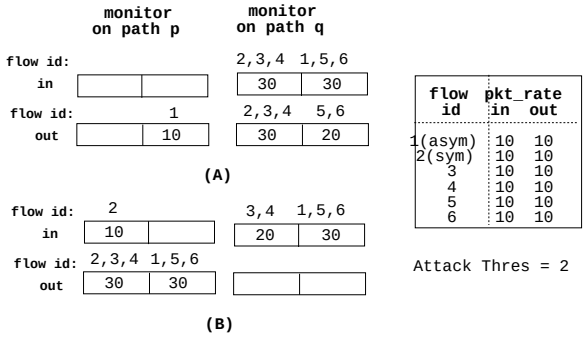

Figure 4.6: False Positive Scenarios in Multi-Gateway AS

threshold at the monitor.

If a flow is masked, that is, has a flow rate less than the score threshold, the detection system will not classify the flow as an attack. This will reduce the incidence of false positives reported by the detection system. The penalty paid for this approach is that attacks with rates less than the score threshold will not be reported and increase the false negative rate.

A monitor's score threshold is proportional to the average number of flows in its bin and the average flow rate of its flows. In practice, for simplicity, the detection system may use a single value for the score threshold for all flows and monitors. In such an event, the score threshold used will be the maximum of the score thresholds at individual monitors. This unified score threshold, σ, is defined as follows: [4]

$$\frac{avg.\ flow\ rate \cdot (\#\ flows\ per\ bin)}{2 \cdot \mathcal{R}_{in}}$$

If the asymmetric traffic rate in the network is known beforehand, σ may be set lower,

[4] A flow may be masked only by either the outgoing packets or incoming packets. Hence, only packet rates in the direction of interest should be included in the flow rates. We approximate this packet rate by evenly dividing the flow rates. Hence, the factor $\frac{1}{2}$ in the above equation.

reducing the possibility of false negatives.

Multi-Gateway Scoring Method

The rendezvous point updates the scores for outgoing suspects using the estimated packet rates for the flows reported by the monitors. For an outgoing suspect flow f, let M_f denote the list of monitors on its path, and $N_m[f]$ the estimated packet rate of f as reported by $m \in M_f$. Then the score is computed as follows:

$$\Delta score[f] \; = \; \frac{\sum_{m \in M_f} N_m[f]}{|M_f|}$$

$$score[f] \; += \begin{cases} \Delta score[f], & \Delta score[f] \geq \sigma \\ -1, & \text{otherwise} \end{cases}$$

Thus, the Δ $score$ of f is the average of the its estimated packet rates reported by the monitors on its path. Outgoing suspect f is deleted from the suspect list if *(i)* its score reaches 0, *(ii)* there is a corresponding incoming suspect, or *(iii)* its $\Delta score$ for a rehash interval is 0. The $\Delta score$ for f for a rehash interval will be 0 only if it is not suspected by any of the monitors during the rehash interval. In this case, short-lived or weak overflows will be deleted by the rendezvous point until again flagged by the monitors.

Observation Period

If the average flow rate in the network is high, the score threshold, σ, can be high, resulting in missed detection of weak (low rate) attacks. However, lowering the score threshold will result in more false positives. In this section, we demonstrate how increasing the minimum number of rehash intervals to flag an attack allows us to lower the score threshold without increasing the number of false positives.

False positives in this case arise when the outgoing component of an asymmetric flow is flagged but the corresponding incoming component is not. The key observation we employ is that flow rates in a network may vary by several orders of magnitude [49]. As the incoming component is subjected to more rehash intervals, it is likely to be mapped to an aggregate consisting of low-rate outgoing traffic. The legitimate, incoming component will therefore be flagged as an incoming suspect, removing the false positive. Using this approach, we reduce the score threshold value defined above by half, thereby trading off detection time for improved detection accuracy.

Outgoing suspects are observed by the rendezvous points for a minimum number of rehash intervals before they may be determined as attacks. During a rehash interval, the rendezvous point considers each message corresponding to outgoing suspects as a positive vote for the respective outgoing suspects. After the minimum number, say m, of rehash intervals, if an outgoing suspect receives at least $m \cdot k$ positive votes (out of a possible total of $m \cdot n$), its score is more than the score threshold σ, and it does not have a corresponding incoming suspect, then it is confirmed as an attack. Otherwise, it is removed from the suspect list. k and n above correspond to k and n in Section 4.1.3.

Our multigateway detection system is designed to perform well in networks with any amount of asymmetry and any traffic flow setup. Most practical multi-gateway networks will not experience these extreme amounts of asymmetry, in which case some of the detection rules can be relaxed so that the system can detect weaker attacks or detect attacks earlier. In the next chapter, we describe our packet level simulator, inputs for our simulator and present the results of simulations for different attack scenarios and under different network conditions.

4.3 Discussion

In this section, we discuss other aspects of our host attack detection scheme. We start with describing how attackers can avoid detection with the aid of other attackers deployed in other stub domains. We then explain how our distributed architecture can help in identifying legitimate packets to a victim if these packets do not share the full path with attack packets. In the latter sections we describe the built-in traceback mechanism of our detection system and conclude the section by presenting a sample hardware implementation for our monitors for host detection.

4.3.1 Escaping Detection

Attack flows avoid detection if their corresponding bins do not overflow. One method to do this is to flood the stub AS with a large number of packets with random source address from outside the AS. A more efficient mechanism, however, is for a host outside the AS to send *mask packets*. Mask packets are packets destined to the attacking

host in the AS and have the source address spoofed as the victim's address. In essence, the external attacker would be providing the ACKs that the victim is unable to deliver, making the attack flow look legitimate. If the rate of mask packets is the same of attack packets, the bin ratios corresponding to the attack will not overflow and the attack will not be detected. This loophole will be minimized if all stub ASes implement egress filtering. In that case, only hosts that are in the same AS as the victim can participate in masking. However, attackers are unlikely to do this, as it would easily reveal their identity.

4.3.2 Traceback

The distributed nature of our detection system helps in tracing back the traffic to the subtree that contains the attacking host. The smallest such subtree will be rooted at the monitor that *(i)* is closest to the attack host that also *(ii)* detects the attack by the attacker. This also means that the monitor lies on the attack path between the attacker and the network gateway. For example, in Figure 4.7, for an attack originating in subtree Z, if D detects the attack, then the subtree with its root at D will be identified as the subtree that contains the attacking host. Similarly, if B detects an attack originating in Y and C does not, then the subtree for the attacker will be rooted at B.

Our system is intended for deployment at source AS's, so the task of traceback is tractable. Since each machine involved in performing the traceback is in the same AS, they are presumably trusted and can be configured suitably. Further, to perform traceback, there must be a communication mechanism between the nodes performing the trace. Our monitors communicate periodically using an overlay, providing a basis for periodic partial

traceback of attacks.

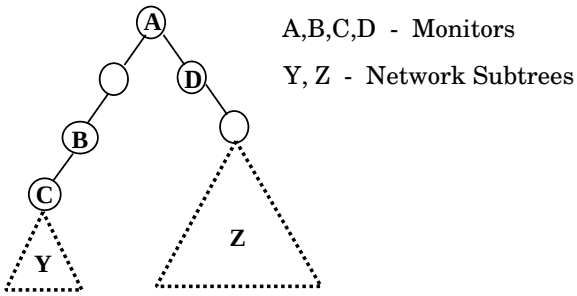

Figure 4.7: Traceback in Detection System

4.3.3 Hardware Implementation

The current fastest routers have multiple OC-192 and OC-768 interfaces [50], translating to link speeds of 10Gbps and 40Gbps respectively. At these speeds, inter-packet arrival time can be as low as a few nanoseconds. In Figure 4.8, we show a hardware design for our monitors that is suitable to process packets at these high speeds.

The monitor would first read a packet from the FIFO queue at the router interface. The Flow Identifier unit would extract the packet direction (entering or leaving the AS) and the flow identification from the packet header and forward the information to both the CAM's FIFO and the Hash Unit. Since the hash function is time consuming, the Hash Unit would have multiple hash modules and compute hashes in parallel using the flow identifier. The hash result is used to access the corresponding bin counters at the on-chip memory. The on-chip memory may be fabricated using the 1T-SRAM [51] or 3T-iRAM [52] to achieve larger amounts of memory, compared to traditional SRAM. The bin counter values and packet direction would be forwarded to the increment module. The Increment Module would increment the corresponding counter (incoming or outgoing)

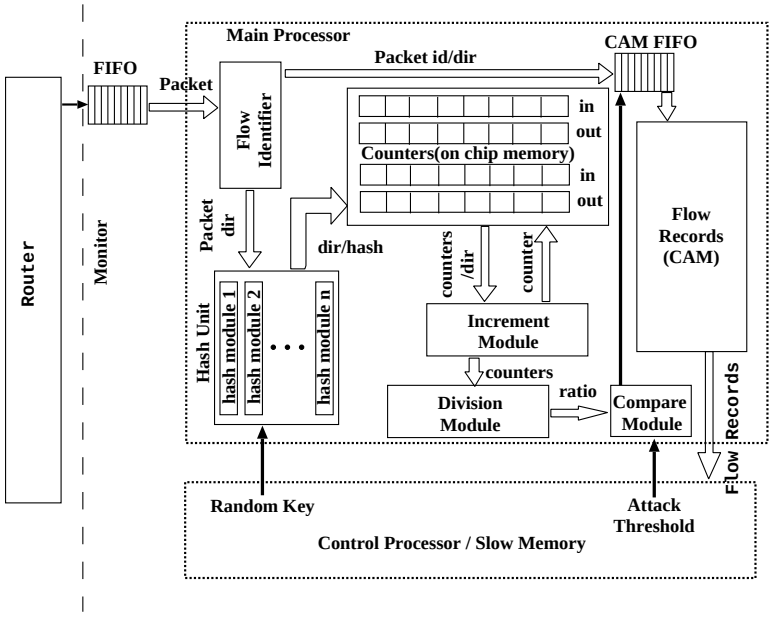

Figure 4.8: Hardware Design

based on the packet direction, write the result to the on-chip memory, and forward the counter values to the Division Module. The Division Module would compute the bin ratio and forward it to the Compare Module, which would determine if the aggregate is suspicious (i.e., if its ratio is outside the range $[\mathcal{R}_{in}, \mathcal{R}_{out}]$). The Compare Module would then forward the result of the ratio's comparison to the CAM FIFO. If the bin were suspicious, the CAM FIFO would forward the corresponding packet ID and the direction to the flow records memory. Otherwise, if the bin were not suspicious, the CAM FIFO would delete the corresponding entry from the queue. The flow records memory should be implemented as CAM so that multiple instances of the same flow record are not created. Periodically, the CAM would flush its contents to the main (slow) memory. At the end of every rehash interval, the control processor would write a new random key to the hash module and store the key. The control processor would also be able change the values used for attack thresholds.

Chapter 5

Host Attack Detection Evaluation

We ran extensive packet-level simulations to evaluate choices of system parameters and study our detection system under different attack scenarios. We used two Internet traces as background traffic. We mapped synthetic, unidirectional TCP flows onto the background traffic to simulate DoS attacks. In this chapter, we describe the metrics to evaluate our detection system and describe the important system parameters. We evaluate how different parameters affect system performance and present the results of our system performance simulations for different attack scenarios.

5.1 Simulation Setup

5.1.1 Packet Traces

We use two traces from the National Laboratory for Applied Network Research's website (NLANR) [53]. These traces differ by more than one order of magnitude in terms of their average packet rates. The smaller trace, which we call the Bell Lab trace,

was constructed from four smaller traces collected outside the Bell Labs gateway in May 2002 and has an average packet rate of 2736 packets per second. The IP addresses in the Bell Labs trace are anonymized. Hence, for each flow, we choose one address randomly as internal to the AS and assume the other to be external.

The larger trace, which we call the Abilene trace, was collected at the Cleveland interface of the Indianapolis Abilene router (part of Internet2) in Oct 2002. With an average packet rate of 100,000 packets per second, this trace is significantly larger than the Bell Labs trace. Packets traversing different directions in the Abilene trace are logged in different files. We randomly used one of the files for incoming traffic and the other for outgoing traffic. Some of the traffic flows in the Abilene trace appear to be asymmetric; half of the flow is not present. We removed these from the trace, as they would be flagged as attacks by our detection system.[1] None of the traces contain intra-AS traffic. Recall that, if there were, the monitors would have filtered it.

Trace characteristic	*Bell Labs*	*Abilene*
simulation duration	25 min	10 min
number of flows	65,000	235,000
avg # active flows per sec	200	3500
incoming pkt rate per sec	1194	55,583
outgoing pkt rate per sec	1586	45,867
number of internal addresses	1291	24,257
number of external addresses	3445	23,647

Table 5.1: Characteristics of the Packet Traces

We summarize the properties of the two traces in Table 5.1. The average number of active flows per second provides a lower bound on the amount of state required by

[1] In fact, some of these asymmetric flows may have been attacks.

systems maintaining per-flow state. As described in the following section, we use this to normalize the number of bins. Further details of the traces, including the trace collection sites and the measurement hardware, are available from [53].

5.1.2 AS Topology

The majority of our experiments concern the single gateway version of our protocol. Since we do not know the network architecture of Bell Labs or of the sites served by Internet2, we simulate a typical stub network as a tree. To represent the single gateway network, we use a full binary tree of depth 5 (15 internal and 16 leaf nodes) where each internal node represents a router and each leaf represents a subnet. In Section 5.4, we describe how we model the multi-gateway network.

5.1.3 Attack Traffic

The attack traffic is composed of synthetically generated TCP packets. Each simulation has eight attacks executed in parallel. All attacks last for eight minutes and each attack has a victim randomly chosen from the external addresses. A new attack is started every two minutes with the Bell Labs trace and every 15 seconds for the Abilene trace. One or more hosts may participate in an attack. All attackers are attached to a leaf node uniformly at random.

5.1.4 Measured Values

To investigate the detection sensitivity of our system, we measure the following:

- Detection Rate: the percentage of the attack flows that are detected.
- False Positives: the number of legitimate flows that are flagged as attacks.
- Detection Time: the time from the beginning of an attack to when the system reports it. It does not include any penalty for undetected attacks.

5.1.5 Experiments Outline

We performed three sets of experiments. The first investigates the effect of various system parameters on detection sensitivity. It is run on a single gateway topology using the Bell Labs trace. In the second set of experiments, we examined the performance of our system under different attack scenarios, again in a single gateway topology, using the Abilene trace. Finally, we used the Abilene trace again to study the system's performance when deployed in stub AS's with multiple gateways. In the following sections, we discuss these experiments in turn. All of the results are an average of eight runs of our simulator.

5.2 System Parameters

In this section, we investigate how the choice of system parameters affects the accuracy of our system's detection. The key parameters of our system are:

- Deployment Scope: defined to be the vector (k/n, *location*) in which
  - k is the minimum number of monitors required to flag a flow to conclude it as an attack,

  - n is the number of monitors on the path from the hosts to the border router, and
  - *location* denotes where the monitors are deployed in the AS. Monitors are placed at the n topmost levels (denoted *root*) or at the n bottommost levels (denoted *leaf*) of the tree representing the AS.

- Normalized number of bins: the ratio, $\mathcal{NB}$, of the number of flow aggregates maintained at each monitor ($\mathcal{B}$) to the average number of active flows per second in the trace. As this number decreases, the average number of flows per bin increases. Thus, we use this metric to demonstrate the trade-off between detection accuracy and the amount of required memory.

- Sampling rate: the maximum rate at which monitors can process the packets. This is measured in packets per second, thus it is the same at each monitor, regardless of the amount of traffic its router receives. [2] Intuitively, by sampling more packets, the monitors receive a more accurate picture of the traffic in the AS. Thus, sampling rate represents the trade-off between detection accuracy and the amount of required processing.

In these experiments, unless specified otherwise, we use a deployment scope of (3/4, root), an $\mathcal{NB}$ of 0.2, and a sampling rate of 300 packets per second. Each attack consists of a single attacker and has a rate of 20 packets per second (roughly 1.5 times the average flow rate in the trace). All experiments in this section were performed using the

[2] In terms of percentage of packets processed, the sampling rate of a monitor is dependent on its router's traffic rate. For example, assume a monitor processes 10 packets per second. Now, its sampling rate will be 10% if traffic rate at its router is 100 pps and its sampling rate is 20% if the traffic rate is 50 pps.

Bell Labs trace.

5.2.1 Deployment Scope

We demonstrate the effect of deployment scope, (k/n, root/leaf), on detection accuracy. Recall that root/leaf denotes the monitor deployment strategy (root-down or leaves-up) and that at least k of the n monitors on a path must vote for a flow for it to be confirmed as an attack. We tested our system without voting ($k = n$) as well as with voting ($1 < k < n$). We present the results of these tests in Tables 5.2 and 5.3 respectively. The detection rate in all these cases was 100% and hence, is not reported in these tables.

Deployment Scope	*Avg # False Pos.*	*Detect. Time (sec)*	*Deployment Scope*	*Avg # False Pos.*	*Detect. Time (sec)*
(1/1, root)	4.00	39.39	(1/1, leaf)	15.29	7.24
(2/2, root)	0.00	41.00	(2/2, leaf)	0.12	10.84
(3/3, root)	0.00	45.75	(3/3, leaf)	0.00	17.94
(4/4, root)	0.00	50.57	(4/4, leaf)	0.00	50.57

Table 5.2: Deployment Scope vs Detection Accuracy (Without Voting)

Deployment Scope	*Avg # False Pos.*	*Detect. Time (sec)*	*Deployment Scope*	*Avg # False Pos.*	*Detect. Time (sec)*
(1/3, leaf)	47.43	7.06	(1/4, root)	57.00	6.86
(1/3, root)	44.71	8.78	(2/4, root)	0.38	8.70
(2/3, leaf)	0.25	9.37	(3/4, root)	0.00	15.28
(2/3, root)	0.00	15.11			

Table 5.3: Deployment Scope vs Detection Accuracy (With Voting)

In Table 5.2, we observe that the false positives are high when there is only one monitor on a flow's path. In this scenario, the detection scheme cannot differentiate between an attack flow and a legitimate flow mapped to the same bin as the attack flow at the

monitor. Hence, the detection system flags the legitimate flow also as an attack. There are more false positives in the leaf setup for the following reason. Fewer flows are mapped on average to each bin in the leaf setup compared to the root setup. Hence, attack packets to legitimate packets ratio in a leaf setup is high in the leaf setup. Therefore, a bin with attack flows will get flagged faster and will have a bin ratio much larger than the attack threshold in leaf setup. Now, when a legitimate flow is mapped to this bin, its score will reach the score threshold faster in leaf setup compared to root setup. Hence, the probability that the legitimate flow's score reaches threshold before rehashing is higher in the leaf setup case compared to the root setup case. As a result, the leaf deployment generates more false positives than root deployment. As the number of monitors deployed on a flow path increases, a legitimate flow that is mapped in the same bin as an attack flow at one monitor will be mapped to a bin other than the attack flow's with a high probability. Thus, the detection system can identify attack flows accurately, generating almost no false positives.

In Table 5.2, we also see two sources for increased detection time: root-down monitor deployment and the number large n with no voting. Detection times are lower for leaf-up deployments than for root-down deployments. This is because, in the leaf configuration, monitors maintain state for fewer flows than in the root configuration. Thus, attack flows are not as easily masked by legitimate flows within flow aggregates and are therefore detected earlier (see the discussion on on flow aggregation and suspect detection in Section 4.1.1).

When suspect lists are strictly intersected ($k = n$), increasing the number of monitors, n, *increases* the detection time. This is because the system must wait for all of the

monitors on the attack path to detect the attack. Since monitors may use different sampling rates[3] e traffic at their associated routers, different monitors and since nodes closer to the root will likely sample less attack packets than those closer to the leaves, there can be a large difference in detection times at individual monitors. For instance, note that the (3/4, root) deployment scope in Table 5.3 takes less than a third the time of (4/4, root) to detect the attack.

When voting is allowed ($1 <= k < n$), the system need not wait for the slower monitors (again, due to different sampling) to report an attack. The results in Table 5.3 show that detection time decreases with k. Interestingly, except for $k = 1$ case, this happens without significant rise in the number of false positives. False positives for $k = 1$ arise for the same reason as false positives are generated when $n = 1$ in Table 5.2. For all other cases, on average, false positives are still less than one per experiment. Further, when keeping k constant, it is helpful to increase the number of monitors.

From these results, we can determine which deployment scopes are most effective. An effective deployment scope should have high detection rates, low detection times, and very few false positives. We choose (3/4, root) over the other configurations because, since the monitors vote, it detects attacks quickly. Further, as we show in Section 4.3, it is more resilient to attacks.

[3]Monitors have same sampling rate in terms of number of packets processed. Thus, the packet rate at a router determines the percentage of packets that the router's monitor processes. A monitor whose router has high traffic rate will have lower sampling rate in percentage compared to a monitor whose router has a low traffic rate.

5.2.2 Normalized Number of Bins

The normalized number of bins, $\mathcal{NB}$, provides a means of trading off detection accuracy and memory requirements. In Table 5.4, we present our system's accuracy with various values of $\mathcal{NB}$. Observe that, as we increase $\mathcal{NB}$, the system is able to detect attacks more accurately.

$\mathcal{NB}$	*Avg # false pos.*	*Detect. Rate (%)*	*Detect. Time (sec)*
0.05	0.00	89	97.95
0.10	0.00	100	27.25
0.20	0.00	100	15.28
0.40	0.00	100	12.47
0.60	0.00	100	11.00
0.80	0.00	100	10.50
1.00	0.12	100	10.79

Table 5.4: Normalized Number of Bins vs Detection Accuracy

For low values of $\mathcal{NB}$, the number of flows in a flow aggregate is very high. Under these circumstances, a weak attack (20 pps) will not be strong enough to cause the ratio of the flow aggregate to cross the attack ratio threshold, resulting in missed detections of some attacks and higher detection times for other attacks.

As $\mathcal{NB}$ is increased, fewer flows are aggregated to the same bin. Hence the relative strength of an attack in the flow aggregate increases. Subsequently, both detection rate and detection time are improved. Thus, at the cost of more memory, the system is able to improve both detection rate and detection time. The false positives generated when $\mathcal{NB}$ is set to 1.0 is due to noise in the system.

At a certain point, we experience diminishing returns from increasing the normal-

ized number of bins. As made evident in Table 5.4, $\mathcal{NB} = .20$ is a reasonable trade-off between increased state and detection accuracy. We therefore use this value for the remainder of our experiments.

5.2.3 Sampling Rate

Another trade-off present in our system is that between detection accuracy and the amount of required processing at the monitor. This is made evident with monitors' sampling rate. In Table 5.5, we evaluate the effect of varying sampling rate on detection accuracy.

Samp. Rate (%)	*Avg # False Pos.*	*Detect. Rate (%)*	*Detect. Time (sec)*
2.5	0.00	72	98.21
5	0.07	99	52.00
10	0.00	100	15.28
20	0.00	100	12.04
40	0.00	100	9.95
60	0.00	100	10.15
80	0.00	100	9.78
100	0.00	100	9.67

Table 5.5: Sampling Rate vs Detection Accuracy

From the Table 5.5, we see the results of this tradeoff; as the sampling rate increases, detection accuracy is improved. Observe that the sampling rate has no discernable effect on the number of false positives. As with the normalized number of bins, there are diminishing returns to increasing the sampling rate. We therefore use 10% sampling rate for the remainder of our experiments for the monitor at the router with the highest traffic rate. It is a reasonable choice, given Table 5.5.

5.3 Detecting Attacks — Symmetric Traffic

In this section, we present the performance of our system in the face of attacks. We use the Abilene trace and the system parameters chosen from Section 5.2: deployment scope (3/4, root), 0.2 normalized number of bins, and 10% sampling rate. As described above, we find that this choice of parameters gives a reasonable trade-off between detection accuracy, required memory, and the amount of processing at a monitor.

In addition to measuring detection accuracy, as before, we also measure network overhead. Recall that monitors communicate via an overlay to share flow records. Every time a monitor reports a flow record, it incurs at least 6 bytes of network overhead (excluding TCP or IP header sizes): 4 bytes for the destination address and 2 bytes for the flow score. In practice, the system could include more information such as flow ratio for use by administrators or higher-layer applications.

5.3.1 Single Attacker — Varying Attack Rates

We begin our attack analysis by considering a single attacker's attack rates in a single-gateway AS. As the attack strength grows, the corresponding flow aggregates' packet ratios reach the attack threshold, $\mathcal{R}$, more quickly, and is therefore easier to detect. To simulate a wide range of attacks, we tested our system with attack rates ranging from 0.3 times to 6 times the amount of traffic for the average flows from the traces. We present the results in Table 5.6.

The results in Table 5.6 show that as the attack rate increases, detection accuracy is improved. This is because, at low attack rates, attack flows are masked by other le-

Atk Rate (pps)	*Avg # False Pos.*	*Detect. Rate (%)*	*Detect. Time (sec)*	*Over-head (Bps)*
10	0.25	99	106.25	77.50
20	0.12	100	27.88	43.75
35	0.25	100	16.47	38.86
50	0.25	100	13.35	39.85
75	0.25	100	10.63	38.69
100	0.25	100	10.14	44.52

Table 5.6: Attack Rate vs Detection Accuracy

gitimate flows in the same bin. In this case, it takes longer for the bin's packet ratio to cross the attack ratio threshold, resulting in longer detection times and possibly missed detection. As the attack rate increases, fewer legitimate flows have comparable flow rates and the probability that the attack can be masked decreases, yielding faster, more accurate detection.

Attacks with low traffic rates induce greater network overhead between monitors. In such cases, monitors closer to the attacker may detect the attack significantly sooner than monitors close to the border router (by the same argument in Section 5.2.1). Until the minimum number, k, of monitors detect the attack, the monitors which have detected it will continue to periodically send flow records along the path.

With higher attack rates, all of the monitors on the attack path will detect the attack sooner, thereby decreasing the amount of time until the attack is reported by the system. Once a flow is reported as an attack, monitors no longer exchange information regarding it. Thus, a greater attack rate effectively reduces the amount of network overhead between monitors.

5.3.2 Multiple Attackers

DDoS attacks achieve large-scale Internet service disruption by using a large number of compromised hosts, often with each host sending packets at a low rate. The low attack rate and large number of attackers make the task of identifying the attack difficult; with low enough rates, users may not even notice that there is malicious traffic originating from their machine. In this section, we present our system's behavior when there are multiple, simultaneous attackers with varying attack rates. Since we are modelling a DDoS attack, each of the attackers targets the same victim.

We study our system's performance by investigating various numbers of attackers in two scenarios. In the first, each attacker sends attack traffic at the same rate. Attackers are distributed randomly among the end hosts and have attack rates of 20 packets per second. We present our results of this experiment in Table 5.7.

From Table 5.7, we see that our system detects attacks quicker if there are more attackers. This is because, since the attackers are targeting the same victim, the aggregate attack rate is increased. The reasoning is therefore similar to those in the attack rate experiment of Section 5.3.1.

The second multiple-attacker scenario we consider maintains a constant amount of attack traffic and varies the number of attackers. In this case, as the number of attackers increases, their individual attack rates decrease. We present the results from this experiment, with an aggregate attack rate of 100 packets per second, in Table 5.8.

In this scenario, detection accuracy worsens with more attackers, as shown in Table 5.8. Since the aggregate attack rate is held constant, monitors close to the root will

Aggr Atk Rate	*# of Atta-ckers*	*Avg # False Pos.*	*Detect. Rate (%)*	*Detect. Time (sec)*	*Over-head (Bps)*
20	1	0.12	100	27.88	43.75
60	3	0.12	100	16.96	43.00
100	5	0.25	100	12.38	45.38
160	8	0.25	100	11.27	65.65
200	10	0.25	100	10.21	73.84

Table 5.7: Multiple Attackers vs Detection Accuracy: Equal Individual Host Attack Rates.

Aggr Atk Rate	*# of Atta-ckers*	*Avg # False Pos.*	*Detect. Rate (%)*	*Detect. Time (sec)*	*Over-head (Bps)*
100	1	0.25	100	10.14	40.52
100	3	0.25	100	11.81	41.49
100	5	0.25	100	12.38	45.38
100	8	0.00	100	16.75	66.47
100	10	0.12	99	14.71	72.02

Table 5.8: Multiple Attackers vs Detection Accuracy: Constant Aggregate Attack Rate.

see similar traffic patterns irrespective of the number of attackers. Monitors closer to the clients, however, are effectively working with lower attack rates. As shown in Section 5.3.1, this results in increased detection times or potentially undetected attacks at those monitors. Consequently, the minimum number, k, of nodes to report an attack is never (or, at best, slowly) reached. Note that the detection time is calculated only for attacks that are detected. For the case when number of attackers is 10, since the missed attack is not used in calculating the detection time, the average detection time in this case is less than the case when there are 8 attackers.

In both scenarios, as the number of attackers increase, more bins will overflow, increasing the amount of communication overhead between monitors. Due to rehashing,

the number of false positives remains low.

5.3.3 Pulse Attacks

In this section, we examine how our system deals with *pulse attacks*. Pulse attacks are characterized by alternating on- and off-periods in which the attacker sends packets only during the on-period. The key difficulty in detecting pulse attacks is in being able to quickly react to the attack while it is in an on-period. Since our system's per-packet processing component detects anomalies quickly, it is effective in detecting pulse attacks.

We simulate pulse attacks with on-periods of 1 second and several off-periods (1, 3, and 5 seconds). We also vary the attack rate (during the on-period) between 20 and 100 packets per second. With a longer on-period, our system would more easily detect the attack, so we do not vary it in our simulations. The results of these experiments are presented in Table 5.9.[4]

Atk Rate (pps)	*Avg. # False Positives*			*Detection Rate (%)*			*Detection Time (sec)*			*Byte Overhead (Bps)*		
on/off	**1/1**	**1/3**	**1/5**	1/1	1/3	1/5	1/1	1/3	1/5	1/1	1/3	1/5
20	0.12	0.25	0.25	94	5	2	130.04	91.88	58.00	90.66	118.23	74.90
40	0.25	0.25	0.12	100	99	47	31.38	145.69	240.25	43.39	85.46	103.74
60	0.25	0.25	0.38	100	100	97	19.32	53.07	119.43	38.25	51.90	68.20
80	0.38	0.25	0.12	100	100	100	15.93	33.75	67.88	40.16	47.98	51.20
100	0.12	0.38	0.25	100	100	100	13.82	29.03	47.55	38.27	41.42	47.04

Table 5.9: Pulse Attacks vs Detection Accuracy

From the table, we see that the system's detection rate is determined by both the attack rate and the off-period duration. Since these two values together determine the

[4]Note that, since the detection times are not penalized for undetected attacks, the trend of the detection time in Table 5.9 is skewed.

average attack rate, this is expected. For a given attack rate, longer off-periods degrade detection accuracy. For instance, in the 20 pps attack flows, 94% of the attacks with off-period 1 are detected, as compared to 2% of the attacks with off-period 5. This is because, for each rehash interval that occurs during an off-period, the flow's score is decreased. It is therefore difficult for the flow's score to cross the score threshold for the minimum number of rehash intervals.

Let $(\mathcal{A}, \mathcal{O})$ denote a pulse attack where $\mathcal{A}$ is the attack rate and $\mathcal{O}$ is the length of the off-period. Note the detection rate for (40 pps, 5 sec). Under ideal conditions, because of the low average attack rate, the score at the root monitor for this attack will be always 0. At the monitor below the root (which processes about twice the number of attack packets since traffic rate at its router is about half of that of root's router), the amount that the score is increased during the on-period is precisely the amount that is decreased during the off period. In realistic scenarios, conditions in the network may result in this monitor to oversample and detect the attack. Since, with equal probability, a monitor may over- or under-sample, attacks are detected with 50% probability, as shown by the 47% detection rate of (40 pps, 5 sec).

Comparing the results of Tables 5.6 and 5.9, we see that our detection system detects pulse attacks comparably to direct attacks for the same average attack rates. For example, in Table 5.9, the (20 pps, 1 sec), (40 pps, 3 sec), and (60 pps, 5 sec) attacks each have an average attack rate of 10 pps. The 10 pps attack in Table 5.6 has similar detection accuracy as these three pulse attacks.

The communication overhead increases with longer off-periods. As length of off-periods increases, detection time increases. As noted above, when attack flows remain

suspicious but are not classified as an attack, they continue contributing to the communication overhead by sending flow records along the path. Finally, as expected, the number of false positives in these set of simulations is not very different from the previous set of simulations.

5.4 Detecting Attacks — Asymmetric Traffic

In this section, we demonstrate how our system performs in multi-gateway AS's using the protocol extensions detailed in Section 4.2. We present the results from experiments where we vary the asymmetry of the AS's traffic. All of our simulations make use of the worst-case topology for our system.

Instead of the binary tree used in the other experiments, we model our topology on the network shown in Figure 4.5. This network has two gateway routers, A and W, and monitors are deployed at routers A through D and W through Z. As shown in the figure, outgoing and incoming asymmetric flows use path p and q, respectively, and the two paths do not have any common monitors. As mentioned in Section 4.2.3, this monitor deployment can generate a large number of false positives at high asymmetric traffic rates. A monitor common to paths p and q would help the detection system to correctly distinguish between legitimate asymmetric flows and attacks. Thus, with no common monitors between p and q, the network structure in Figure 4.5 is the worst-case topology for our detection system.

In the multi-gateway experiments, we use two additional variables to characterize different asymmetric traffic scenarios. Based on the network of Figure 4.5, let A^p_{out} be the

percentage of all outgoing traffic that traverses the monitors on path p. Similarly, let A^p_{in} be the corresponding percentage for incoming traffic on p. In order to have a measure of the number of symmetric flows in the network, we set the percentage of symmetric flows on p to $min\{A^p_{out}, A^p_{in}\}$. Consequently, the percentage of flows in the network that are asymmetric and use path p is $|A^p_{out} - A^p_{in}|$. For instance, if $A^p_{out} = 50\%$ and $A^p_{in} = 20\%$, then 20% of the flows throughout the network are symmetric on p and 30% of the flows are asymmetric with p as their outgoing path. To study a wide array of traffic asymmetries, we vary A^p_{in} between 0.0 and 1.0 in steps of 0.2 and set A^p_{out} to 0.1 and 0.5.

Some combinations of A^p_{out} and A^p_{in} represent traffic profiles in which one of the paths has mostly asymmetric traffic while the other has a significant amount of the symmetric traffic. For instance, when $A^p_{out} = 10\%$ and $A^p_{in} = 0\%$, 100% of flows on path p is asymmetric and 90% of traffic on q is symmetric. Recall from Section 4.2.3 that, in situations like this, the monitors on the asymmetric path (p in this example) will report a large number of outgoing suspect flows to the rendezvous point. We mitigate this by choosing a large score threshold, described in Section 4.2.3.

For this experiment, our system parameters are as follows. We set the interval score threshold (σ) to 75 packets per second, the number of observation intervals at the rendezvous point (n) to 4, and the number of votes required to determine an attack (k) to 12 ($= (n-1) \cdot m$, where m is defined in Section 4.2.3). We use the Abilene trace for background traffic and generate attack traffic as in the single gateway experiments. Monitors sample up to 10% of network traffic and use a normalized bin size ($\mathcal{NB}$) of 0.2. We set the attack rate to 150 packets per second, the attack threshold to 2.0, and rehash interval to 5 seconds.

We present our results in Table 5.10. There are two important features to note. First, as discussed in Section 4.2.3, the number of false positives is increased in situations of extreme asymmetry due to flow masking. Second, there is a non-intuitive conservation in the number of false positives for various cases of asymmetric traffic.

To see this, first note that when $A^p_{out} = 50\%$, path p receives five times the amount of traffic as when $A^p_{out} = 10\%$. Intuitively, one would expect that the number of false positives would be five times as much also, but this is not the case. Since more traffic is diverted to path p, the incoming suspect flows on path q need not have as high a flow rate as the 10% case. Thus, although more outgoing suspects are detected on p, this is balanced by the increased number of incoming suspects detected on q.

A^p_{out}	A^p_{in}	*False Positives*	*Detection Time (sec)*	*Overhead (bytes/sec)*
10%	0%	1.12	53.60	7434.7
	20%	0.00	37.19	7829.8
	40%	0.00	29.61	10797.6
	60%	0.00	27.34	13575.2
	80%	0.00	28.36	16263.6
	100%	0.12	33.05	18536.0
50%	0%	1.38	56.57	12671.2
	20%	0.25	35.32	10586.1
	40%	0.00	27.81	8256.7
	60%	0.00	26.48	8301.4
	80%	0.25	28.98	10687.8
	100%	0.38	43.83	12676.1

Table 5.10: Flow Asymmetry and Detection Accuracy

As asymmetry increases, attacks on paths with increased incoming traffic will be increasingly masked, as the incoming traffic may appear to be acknowledgements to the attack traffic. Thus, the time to detect these attacks increases. This is demonstrated nicely in the $A^p_{out} = 50\%$ case in Table 5.10; the less symmetric the traffic (i.e., the greater the difference between A^p_{in} and A^p_{out}), the greater the detection time.

In the case for $A^p_{out} = 10\%$, as A^p_{in} increases from 0% to 60%, attacks on path p take longer to be detected whereas those on q are detected earlier. This follows from the same reasoning as above. In this case, the overall detection time decreases because 90% of the attack flows are on q. As A^p_{in} increases from 60% to 100%, the time to detect the attack flows on p is increased, ultimately overcoming the improved detection time of attack flows on q.

Our system detects all attacks so we do not report detection rate in the table. As expected, message overhead increases with increased in traffic asymmetry. When more flows are asymmetric in the network, monitors detect more incoming and outgoing suspect flows and hence report more information to the rendezvous point.

5.5 Attacking the System - Inducing False Positives

Our detection system can be potentially manipulated to generate a large number of false positives. Consider the situation, similar to the one mentioned in Section 4.2.3, when a large fraction of bins at monitors on a path are suspicious. If monitors are deployed according to the deployment scope (k/n, *), the probability that a legitimate flow maps into k suspicious bins at n monitors on its path increases. The flow will be considered an attack, thereby generating a false positive, when its score crosses the score threshold. In Table 5.11, we list the theoretical probabilities that a flow is a suspect at k of the n monitors on its path for different deployment scopes and fractions of suspicious bins at the monitors.

We ran simulations for the above scenario using the Abilene trace on the single

Deployment scope	*Fraction of Suspect Bins* 0.1	0.2	0.3	0.4	0.5
(2/2,*)	0.0100	0.0400	0.0900	0.1600	0.2500
(2/3,*)	0.0280	0.1040	0.2160	0.3520	0.5000
(3/3,*)	0.0010	0.0080	0.0270	0.0640	0.1250
(2/4,*)	0.0523	0.1801	0.3294	0.5248	0.6875
(3/4,*)	0.0037	0.0272	0.0648	0.1792	0.3125
(4/4,*)	0.0001	0.0016	0.0081	0.0256	0.0625

Table 5.11: Analytical probability that a legitimate flow is suspected at k monitors

gateway network. We varied the fraction of bins at a monitor that are subject to overflow. Thus, the number of attacks in the system is set to fraction of targetted monitors times the number of bins at a monitor. The rate of an attack flow, as we have seen, determines if the corresponding bin will be flagged suspicious. Hence, we also vary the attack rate and in the table, report the attack flow rate normalized by the average flow rate in the network. The fraction of attack traffic in the network can be computed by multiplying the normalized attack flow rate with fraction of bins targetted by the attack flows (i.e., the product of row and column headers in the table). The deployment scope, normalized number of bins and sampling are set to (3/4, root), 0.2, and 10,000 pps (i.e., 10% for lowest sampling rate monitor) respectively.

Normalized Attack Rate	*Fraction of Bins Targetted* 0.1	0.2	0.3	0.4	0.5
0.6	0.00	0.00	0.00	0.00	0.00
1.5	0.00	0.00	0.50	2.75	7.75
2.4	0.00	1.00	7.50	43.00	129.00
3.3	0.00	3.12	25.51	112.50	318.00

Table 5.12: False Positives Generated by Our Detection System Under Attack

From Table 5.12, we observe that the attack traffic in the network should be at least 10% to generate any false positives. For the extreme case in the table, when attack traffic

rate is 33% of the total network traffic, our detection system generates 318 false positives. With 230K flows in the network, this number is significantly less than the theoretical number of false positives predicted for this case (about 70K out of 230K flows in the trace). Thus, we can conclude that our detection system is robust against system attacks due to mechanisms such as local rehashing and flow scoring.

Chapter 6

Attacks Against Subnets

The scheme discussed thus far is used to detect bandwidth attacks against single hosts. Attacks may also target multiple addresses in a subnetwork and congest the access link to the subnet. We refer to these as subnet attacks. In this chapter we describe how our monitors can detect subnet attacks. First, we explain the behavior of our host attack detection scheme in the presence of subnet attacks.

6.1 Host Attack Detection Scheme and Subnet Attacks

Typically, access links to subnets have higher bandwidths than access links to single hosts. Hence, the aggregate attack rate of a subnet attack may be more significant than the attack rate of a host attack. But, a subnet may include several addresses and packets of a subnet attack may be destined to any of the addresses of the subnet. Hence, traffic to individul addresses in the subnet may be very diffuse. As a result, the host attack detection scheme would not detect the attack if the complete address is used as the flow identifier.

For example, consider the example in Figure 6.1. The network has four flows a, b, x and y. The table in the figure lists the packet rates of these four flows. a and b are legitimate flows and x and y are attack flows as shown in the figure. Attack flows are targetting hosts in the subnet $p.q.r.*$. The monitors in the host attack detection scheme will map attack flows to hosts a.b.c.1 and a.b.c.2 (namely, flows x and y) to different aggregates with high probability. As shown in the figure, since these two individual flows have very low flow rates compared to legitimate flows a and b, the system will miss the attack.

flow	in packet rate	out packet rate
a (legit)	12	9
b (legit)	12	9
x (atk)	0	9
y (atk)	0	9

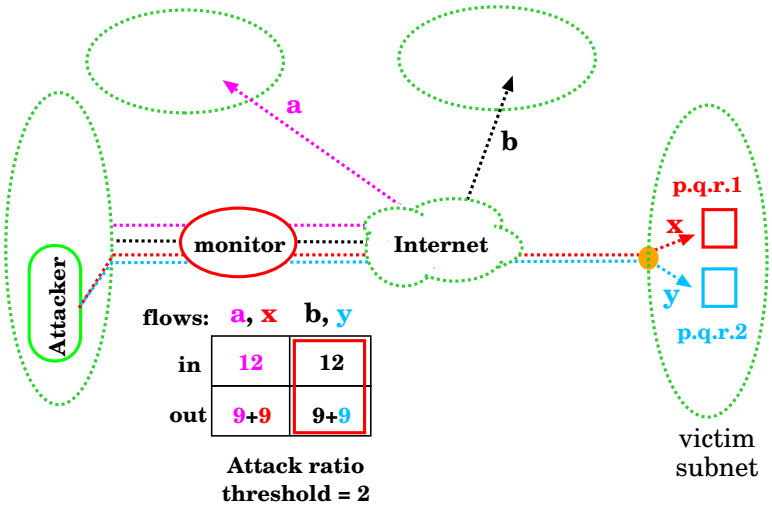

Figure 6.1: Missed Detection of Subnet Attacks

One solution is to use address prefixes to map flows to bins. (Recall that a flow is

the set of traffic streams to the same external address.) For this example, assume that the host attack detection scheme uses 24 bit prefixes to map flows to bins. In such a case, the system will map both flows x and y to the same bin and the attack is detected. This scenario is shown in Figure 6.2.

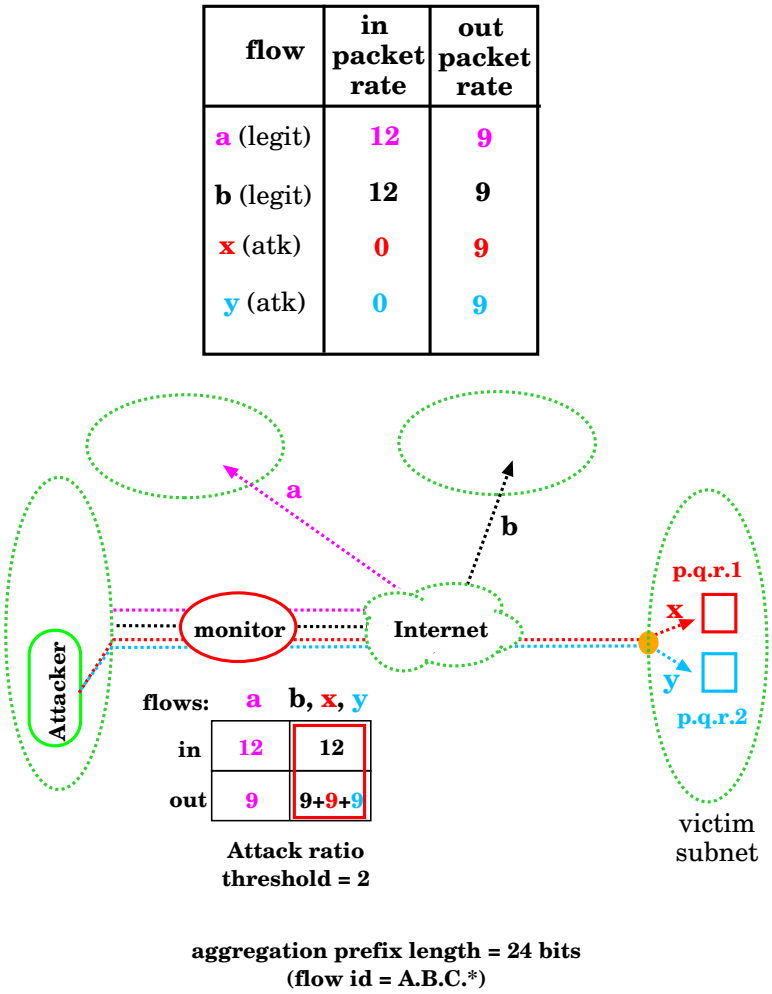

Figure 6.2: Subnet Attack Detection Using Address Prefixes

The problem with this solution is two fold. First, it is difficult to determine the prefix length and prefix range to use because subnets may be of different sizes and address range for subnets may not be contiguous. Second, a monitored prefix may have hosts which are not under attack. Hence, legitimate flow(s) with large flow rates to these hosts may mask

the attack flows to the victim hosts in the subnet and the attack will be missed. This scenario is shown in Figure 6.3. In this case, flow y is a legitimate flow and hence, attack flow x will be missed by the monitor due to flows b and y.

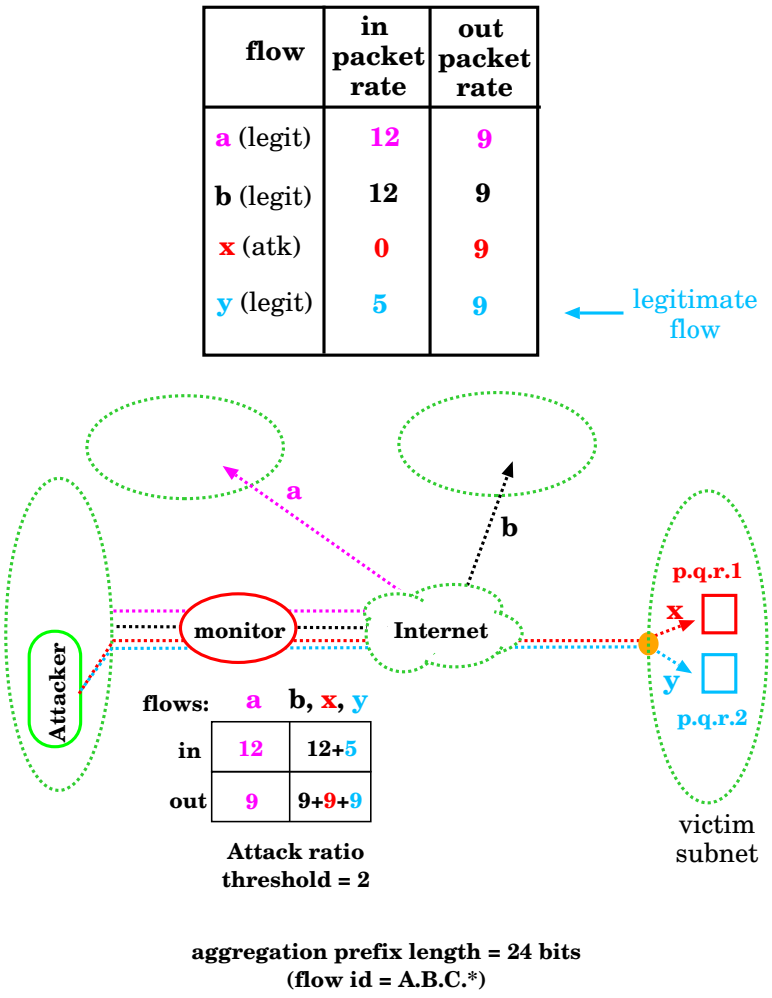

Figure 6.3: Missed Detection due to Prefix-based Aggregation

We next describe a scheme that is executed by an overlay node to detect subnet attacks. The scheme assumes that legitimate traffic at the node have symmetric paths and using packet samples, detects attacks that originates in the subtree network rooted at the overlay node. For simplicity, we assume that the subnet attack detection scheme is executed by the overlay node that is deployed at the AS gateway. We describe the extensions

for multi-gateway networks in the following section and present the evaluations of the scheme in the following chapter.

6.2 Subnet Attack Detection — Single-homed Domains

6.2.1 Detection Technique

We use the following technique to detect subnet attacks. Consider a counter to which a set of flows F are mapped. Let κ_i denote the outgoing-to-incoming packet ratio of a flow $i \in F$. For each sampled packet of flow i, let the counter be incremented by 1 if the packet is outgoing and decremented by κ_i if the packet is incoming. Then, at the end of the flow, the net increment to the counter due to flow i's outgoing and incoming packets will be zero. This is true for all flows in F. From the discussion in Chapter 3, we know that legitimate flows have the ratio κ_i at most 3 and bandwidth attacks have κ_i much larger than 3. Thus, consider the case when an incoming packet can decrement the counter by at most 3 (i.e., all κ_is are assumed to be upper bounded by 3). Assume that flow $j \in F$ is an attack flow (κ_j is greater than 3). After counter updates due to j's outgoing and incoming packets, the counter will have a net increment. Thus, increments to a counter's value over time indicates the presence of an attack flow among the set of flows mapped to it.

In Figure 6.4, we demonstrate with an example how per-packet updates to a counter can be used to detect an attack flow mapped to the counter. In the figure, flow 1 has a flow ratio of 2 and flow 2, an attack, has a flow ratio of 4. In the first case, κ is unbounded

flow ratio for flow 1 = 2

flow ratio for flow 2 = 4

sequence of sampled packets	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
flow id	**1**	**1**	**1**	**1**	**2**	**2**	**1**	**2**	**1**	**2**	**2**	**2**	**2**	**2**	**2**	**2**
packet direction	o	**i**	o	o	o	o	o	o	**i**	o	**i**	o	o	o	**i**	o
counter when kappa unbounded	1	-1	0	1	2	3	4	5	3	4	0	1	2	3	-1	**0**
counter when kappa is upper bounded by 3	**1**	**-1**	**0**	**1**	**2**	**3**	**4**	**5**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**

Figure 6.4: Subnet Attack Detection Technique

and hence, packet updates do not result in net increments to the counter. In the second case, κ is upper bounded by 3. Hence, increments due to flow 2's outgoing packets will be more than the decrements due to incoming packets. Hence, the counter will have net increments over time which indicates the presence of an attack.

Our detection scheme has to address two issues before we can use the technique to detect subnet attacks.

- It requires a data structure that aggregates flows by subnets to which they are destined. We need a limited size data structure that can fit in the fast memory of our monitors. Also, it is useful if the counters in the data structure not only indicate the presence of an attack but also easily identify the victim of the attack. We describe our data structure which has both the properties in Section 6.2.2.

- Estimating flow (outgoing-to-incoming packet) ratios accurately is difficult. Further, a flow ratio may vary with time because of variations in mix of applications

and traffic, and because of changes in network state. Hence, static estimation of flow ratios are useless. In our scheme, instead of estimating flow ratios, we use a single κ value, which is the maximum possible flow ratio for legitimate flows. We design our data structure and packet updates such that, decrements because of a flow's incoming packets at most equal the increments because of the flow's outgoing packets. We describe the precise packet update mechanism in Section 6.2.3.

Overview In subnet attack detection scheme, a flow is mapped to multiple flow aggregates and a counter is associated with each flow aggregate. The mapping of a flow to aggregates is determined by the IP address of the flow's destination (flow identifier) as well as a hash of the flow identifier. The set of counters can be represented as 2-D tables of counters. (See the next section for details). If a flow's sampled packet is outgoing, all counters corresponding to the flow are incremented by one. If the packet is incoming, all of the flow's counters are decremented by a positive value κ, as long as all the counters are greater than κ. The decrement policy is that net increments of counters due to packets of a legitimate flow are almost zero. An attack flow, because of its high outgoing-to-incoming packet ratio, will increment its counters more than it decrements the counters. Hence, attack flow's counters will have significantly larger values than other counters. Ongoing attacks can be detected by periodically processing the counters and identifying those counters that have values significantly larger than other counters. The flow to counter mapping makes identification of victim subnet straightforward. In the rest of this section, we describe the subnet attack detection scheme for single gateway networks. We describe the scheme for multi-gateway networks in the next section.

6.2.2 Mapping between Flows and Counters

In this algorithm, we use two equal sized 2-D tables of counters. A flow is mapped to a counter in each of the columns of both the tables. While the flow's external IP address (*flow identifier*) is used to index the counters in one table, a hash of the identifier is used to index counters in the other table. We refer to these two tables as IP indexed table and hash indexed table respectively.

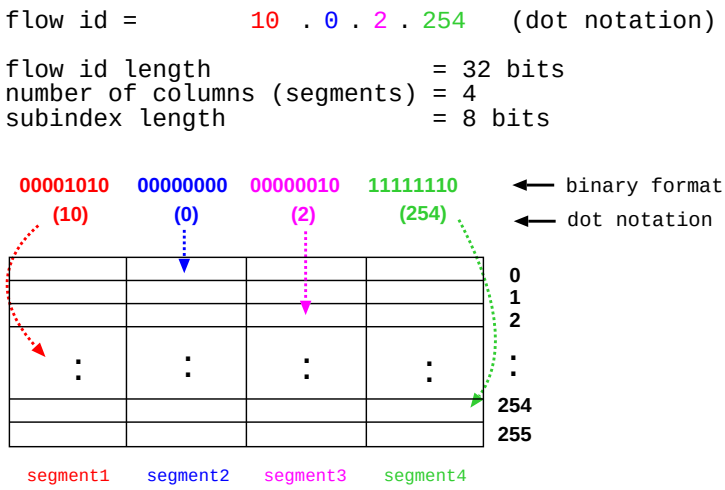

Figure 6.5: Flow Identifier to IP indexed Table Mapping

Mapping of a flow to the IP indexed table's counters, using the flow identifier, is shown in Figure 6.5. Flow's mapping to hash indexed table is the same as for IP indexed table. However, a hash of the flow identifier, computed using a hash function associated with the hash indexed table, is used instead of the flow identifier to map the flow to hash indexed table. Assume that flow identifiers are p bits long and the table has c columns. If p is a multiple of c, the identifier is split into c non-overlapping parts, all of equal length. The parts are refered to as *segments*. (By non-overlapping, we mean none of the bits of the identifier are mapped to more than one segment.) Each segment corresponds to

that column of the table as determined by its position in the flow identifier. Thus, the first segment corresponds to the first or leftmost column of the table and so on. Then, as shown in the figure, the value of a segment is used to index the counters in the corresponding column. If p is not a multiple of c, then $p \ mod \ c$ (p%c) least significant bits of the identifier are ignored and the rest of the index is split as described above. In our example in Figure 6.5, we have chosen to be byte aligned; this is not necessary in general. We discuss the impact of the number of table rows in the next section.

Hash Indexed Tables

A hash indexed table has a static hash function associated with it. The hash function takes the full flow identifier (32 bits for IPv4) of a flow as input and generates a hash value of equal size (i.e., 32 bits again for IPv4). If the hash function is perfectly random, the probability that two flow identifiers are mapped to the same hash value by the hash function will be $\frac{1}{|flow \ id \ space|}$ (= $\frac{1}{2^{32}}$ for IPv4). The flow is mapped to the hash indexed table in the same manner as the IP indexed table, except that the hash of the flow identifier is used for the mapping.

The flow mapping in IP indexed table causes flow identifiers with same prefixes to map to same counters in the left columns of the IP indexed table. For example, assume a four column IP indexed table. Flows to hosts 10.0.2.11 and 10.0.2.187 will get mapped to same counters in the first three columns of the IP index table. In contrast, use of hashes makes the flows with same prefixes to map to different counters in the hash indexed table with high probability. Thus, flows of a subnet attack are mapped to same counters in the left columns of the IP indexed table and with high probability, to different counters

in the hash index table. As we will observe later, mapping in the IP indexed table will help detect attacks and mapping in hash indexed table will reduce false positives and false negatives.

6.2.3 Per Packet Processing

When a packet is sampled, the packet's flow is mapped to corresponding counters in both the tables as explained above. For an outgoing sampled packet, all corresponding counters are incremented by one. If the packet is incoming, the corresponding counters are decremented by a positive value κ, if all of these counters are at least κ. κ should be set to the maximum acceptable ratio of outgoing-to-incoming packet ratio for legitimate flows in the network. We assume that a legitimate flow may have this ratio up to 3. Hence, we use a value of 3 for κ.

The idea behind this decrement scheme is the following. Attacks can be detected if net increments to counters the attacks map to are significantly larger than increments to other counters in the tables. Hence, net increments due to attack flows should be much larger than due to legitimate flows. Also, legitimate flows should not cause net decrements. Otherwise, legitimate flows aggregated with an attack can mask the attack's increments with their decrements. Thus, decrementing by κ for incoming packets as outlined above ensures that attacks can be detected. We demonstrate this with an example next (Figure 6.6):

Assume that flow ids are 4 bits long, counter table has two columns and flow 1011 is an attack flow. The arrival sequence of packets is shown on the left of the figure. κ

is set to 3. At time 10, the outgoing packets of (attack) flow 1011 will increase counters corresponding to 10 in the left column and 11 in the right column of the table by 5. At time 20, flow 1001's outgoing packet will increment corresponding counters by 1. At time 30, the incoming packet of flow 1001 cannot decrement the counters because counter 01 in the right column is less than three. At time 40, this counter is incremented to three and time 50, this counter as well as counter 10 in the left column are decremented by three. Hence, counters corresponding to attacks will have net increments, while other counters will not. By analyzing the counters, attacks can be detected.

Effects of Flow Aggregation Aggregation (mapping packets to fewer rows of a column) can lead to counter manipulations that results in false positives and and false negatives. In the example in Figure 6.7, there are three flows of which, flow *1011* is an attack flow. Again, packet sequence is shown on the left of the figure. With the given aggregation and order of packet updates, packets of flow 1001 will decrement its corresponding counters by 6 at time 40 even though it incremented the counters only by two. Hence, the attack signal in the counter corresponding to 10** is masked. Moreover, the counter corresponding to 00** has values significantly larger than other counters in the table. The result is, in extreme cases, these wrong updates can supress attack detection and generate false positives. The effect of aggregation is dependent on the size of the table. With more rows, flows are aggregated less and hence, incorrect decrements will be less and only attacks with very small flow rates may go undetected (false negatives). If aggregation increases, incorrect decrements may also increase and can result in more false positives and false negatives.

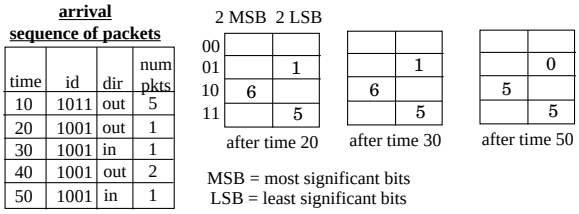

Figure 6.6: Decrementing Counters

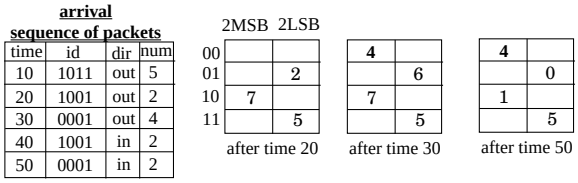

Figure 6.7: Aggregation Effect on Updates

Hash Indexed Table vs. Aggregation The hash indexed table reduces the impact of aggregation in two ways. First, it increases the number of counters a flow is mapped to, thereby reducing the possibility of wrong decrements. With more counters and different sets of flows mapped to these counters, probability of decrementing incorrectly by a flow decreases. Note that in general, the detection scheme can be extended to multiple hash tables. However, one such table is sufficient. Second, and equally importantly, it prevents an attacker from masking his attack flow with the help of a large legitimate flow to a host in victim's subnet. Thus, using hash table reduces the possibilities of false positives and false negatives due to flow aggregation. We next describe the procedure to detect counters that may correspond to attacks.

6.2.4 Periodic Processing

Flows to a subnet, as described, map to the same set of counters in the left columns of the IP indexed table. Hence, increments due to flows of a subnet attack are highest

procedure $DetectAttackCounters$

1: $S \leftarrow\ counters \in IPtable$
2: $incr[c,t] \leftarrow c[t]\ -\ c[t-1];\quad \forall\, c\ \in\ S$
3: compute $avg[t]\ and\ std.dev[t]\quad \forall\, incr[c,t]\ >\ 0$
4: **for all** $c \in S$ **do**
5: $\quad score[c]\ +=\ incr[c,t] - avg[t] - std.dev[t]$
6: $\quad score\ =\ max(0,\ score)$
7: $\quad score\ =\ min(score,\ detec\, thres)$
8: $\quad$ **if** $score[c] \geq detec\, thres$ **then**
9: $\quad\quad$ flag c
10: $\quad$ **end if**
11: **end for**

Figure 6.8: Periodic Processing Procedure

in counters in the left columns of the IP indexed table. We periodically analyze the IP indexed table to detect ongoing bandwidth attacks. The interval between periodic processing should be larger than the RTTs for most flows in the network. This enables the incoming packet updates to compensate the outgoing packet updates at counters. The procedure for periodic processing is given in Figure 6.8.

We compute a score over time for each of the counters to detect counters to which attack flows are mapped. Scores are computed using the net increments of counters over the period. Score computation for counter c at the end of period t is shown in the figure. We use a threshold called detection threshold to flag counters whose score reach the threshold as those counters to which an attack flow is mapped. The detection threshold is set proportional to the sampling rate. Also, the score is bounded between zero and detection threshold to minimize the effect of old state on detection.

The rationale behind the score computation is as follows. The net increment of a counter could be due to flow aggregation or because an attack flow mapped to the counter.

Assume that the net increment of a counter during a period is a random variable and that the number of attacks is very small compared to number of rows in the table. Since increments due to attack flows will be larger than those due to flow aggregation, we can treat increments that correspond to attack flows as outliers. To determine these outliers, we compute the average and standard deviation of the net increments and use their sum as the baseline for net increments due to aggregation. Increments of counters above the baseline are considered outliers.

According to Grubb's test [54], using only one standard deviation in the baseline computation can result in a high false positive rate. But, increasing the number of standard deviations in the baseline will lower the detection rate of attacks with low flow rates. We note that counters that attacks map to will have increments during the whole duration of the attack while other counters may have net increments for a few periods. The score of a counter is the net increments of the counter above the baseline. Hence, the score for a counter to which an attack is mapped will increase over time. Scores for other counters will be at zero most of the times and may peak at a low value briefly. Thus, we can use a detection threshold for scores to detect counters to which attack flows are mapped and supress false positives. We show the evolution of scores over time for three counters from actual simulations in Figure 6.9.

In the figure, one counter has an attack mapped to it. The attack started at time step 72 and ended at time step 96. A detection threshold of 100 is used to detect the attacks. As we described above, score for the attack counter increases over time and the system flags the counter when its score reaches the detection threshold. By bounding the scores, detection scheme stops flagging attack's counters as soon as the attack stops. The

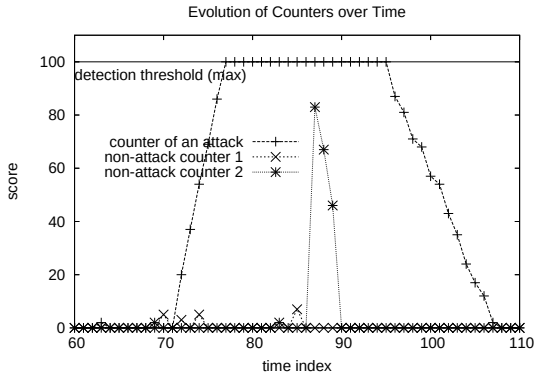

Figure 6.9: Counter Score Evolution

plot corresponding to the score of non-attack counter 1 is the typical behavior of most of the scores. They have a value close to zero most of the time. Some counters, may experience net increments in their values and hence temporary increases in their scores due to aggregation. As the plot for non-attack counter 2 shows, these increases in scores are only temporary. If the detection threshold is set low, the system will flag these types of counters as attacks.

Columns in the IP indexed table map to different parts of the flow identifier. If the sampled traffic has only one subnet attack, only one counter from different columns of the table will be flagged and the prefix of the subnet under attack can be constructed by combining the indices of the flagged counters. If the sampled traffic has more than one ongoing attack at a time, the combinations to determine subnet prefixes of victims will increase exponentially. For example, assume that the IP indexed table has two columns and there are three attacks simultaneously in the network. Then, three counters will be flagged in each of the two columns, which will give a maximum of nine possible subnet victims. Thus with n columns in the table and m attacks, there could be up to m^n pos-

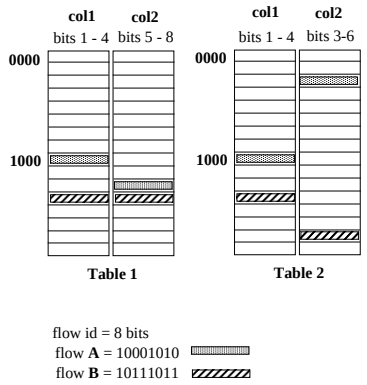

Figure 6.10: Mapping Enhancement

sible combinations for m victim subnet. This problem can be addressed using a slightly different mapping scheme described next.

6.2.5 Mapping Enhancement

The mapping scheme described thus far assumed that adjacent segments do not overlap. We can probabilistically reduce the combinations due to multiple attacks in the network if we let adjacent segments overlap. Consider the example shown in Figure 6.10. Assume that the IP addresses are 8 bits long and the IP indexed table has two columns. Also assume that flow A (1000 1010) and flow B (1011 1011) are two attack flows and the detection algorithm flags the counters to which these two flows are mapped. We show two cases in the figure: in Table 1, the segments do not overlap while in Table 2, the segments overlap in bit positions 3 and 4. The flagged counters are shown in all the columns in the tables.

Without overlap, the detection system cannot match counters in column 1 with counters in column 2 of table. Hence, there are four possible combinations for victim

subnets, which are (1000 1010), (1000 1011), (1011 1010) and (1011 1011). With the overlap in Table 2, flagged counter 1000 of column 1 will only map to counters 0000, 0001, 0010 and 0011 of column 2 in the table. Of these four counters, only 0010 is flagged. Hence, the only possible combination for the victim subnet prefix corresponding to A will be 100010. Similarly, for B, the flagged counter 1011 in column 1 will match only with flagged counter 1110 in column 2 in table 2 and the prefix that can be constructed will be 101110. An additional column in table 2 that corresponds to bits 5 through 8 would enable determining the victim prefixes completely without any ambiguity. In general, with n columns and overlap in y bits, a table can reduce the possible combinations for constructing prefixes by $\frac{1}{2^{y.(n-1)}}$.

6.3 Subnet Attack Detection — Multi-homed Domains

The subnet attack detection scheme described in the previous section is deployable only in networks in which all legitimate flows are symmetric. If the network has legitimate asymmetric flows, those flows will also be flagged as attacks by the system described so far. Consider the example in Figures 6.11 and 6.12. The network has two gateways m and n and the first of the two figures shows the three flows and their flow rates. Flow 00 is a symmetric flow, flow 01 is an attack flow and flow 10 is an asymmetric flow. The asymmetric flow's outgoing packets exit the network at gateway m and incoming packets enter at gateway n. The result of deploying the single gateway subnet detection scheme in this network is shown in the second figure. The scheme, as expected, will flag flow 01 as an attack. However, as seen in the figure, it will also flag flow 10 as an attack even

though the flow is legitimate

flow id	flow type	egress gateway	out rate (packets per sec)	ingress gateway	in rate (packets per sec)
00	legit, sym	m	10	m	20
01	attack, sym	m	40	m	2
11	legit, asym	m	15	n	10

Two gateways: **m** and **n**

Figure 6.11: Example Flows in Multi-Gateway ASe

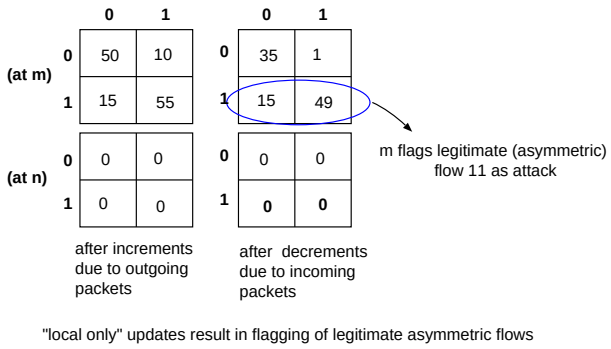

Figure 6.12: Subnet Attack Detection — False Positives in Multi-Gateway ASes

We will describe here the extensions to the system so that the scheme will detect attacks accurately even in the presence of legitimate asymmetric flows. The extension assumes that aggregate flow rates to subnets do not vary abruptly and that the duration of a bandwidth subnet attack is long (in the order of minutes). Under these assumptions, overlay nodes can perform packet updates delayed in time and space (location) and distinguish between malicious and legitimate asymmetric flows. More specifically, during an interval, each overlay node collects information that identifies all asymmetric flows in its traffic. During succeeding intervals, other overlay nodes use this information to identify any incoming asymmetric flows and together aid in determining attack flows.

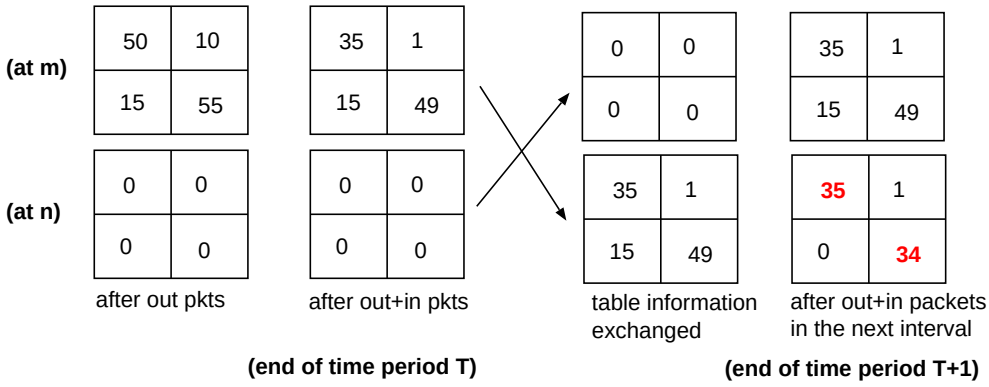

Figure 6.13: Detection in Multi-Gateway ASes — Monitors periodically exchange information about asymmetric flows. In subsequent intervals, monitors identify attacks from among all asymmetric flows

We demonstrate the idea in Figure 6.13. At the end of every period, each monitor receives information from other monitors about asymmetric flows at those monitors. If legitimate asymmetric flows are active over multiple intervals, the monitor can determine local legitimate incoming asymmetric flows. All monitors can pool this information and distinguish legitimate asymmetric flows from attacks and thus, detect attacks. We next present the extensions to detect subnet attacks in multi-homed networks in detail.

6.3.1 Data Structure

In the extended version, monitors are deployed at all the gateways in the network. Each monitor in the extended scheme has two such pair of IP and hash indexed tables. All tables are of equal dimensions. We refer to the first pair as the *symmetric* IP and hash indexed tables and the second pair as the *asymmetric* IP and hash indexed tables. The mapping between a flow and a table is the same as in the basic version (i.e., as shown in Figure 6.5). Like the hash indexed table in the basic scheme, symmetric and asymmetric

hash indexed tables use a hash function to map flows to entries in the tables. This mapping function is the same for both the tables. In fact, all the monitors in this scheme use the same mapping function for the hash indexed IP tables.

Each monitor uses its symmetric tables to detect asymmetric flows in its traffic. Symmetric tables are updated in the same manner as the IP and hash indexed tables in the basic scheme; they are updated at the monitor locally using packet samples. Thus, counters of symmetric tables corresponding to asymmetric flows behave in the same manner as the counters corresponding to attacks in the single gateway case. Periodically, each monitor computes increments in the values of these counters over the period. At the end of the period, these increments are added to corresponding counters in the asymmetric tables at all monitors. Thus, asymmetric tables at all monitors have the state of all outgoing asymmetric flows, legitimate and attacks, that are active in the previous interval. During the current interval, each monitor uses the asymmetric tables to identify legitimate asymmetric flows that enter the network at the monitor. It uses the packets of these flows to decrement the counters accordingly. Thus, using their asymmetric tables, monitors collectively identify legitimate asymmetric flows and detect attacks.

In the next two sections, we discuss in detail how the tables are updated. To keep the discussion simple, we describe the updates using only symmetric and asymmetric IP indexed table (or simply symmetric and asymmetric tables). However, any update to entries of an IP indexed table will result in a similar update to entries in the corresponding hash indexed table. (Only difference is what entries are updated. This is determined by the flow address for the IP indexed tables and the hash of the flow address for the hash indexed tables.) In Section 6.3.3, we explain, using asymmetric IP indexed tables, how

we detect attacks.

6.3.2 Per-packet Processing

A monitor's per-packet processing component uses sampled packets to update the symmetric table. When the monitor samples an outgoing packet, it increments fields of the symmetric table corresponding to the packet by one. If the packet is incoming, the same fields are decremented by the value κ, if all the fields are at least κ. If not, it decrements the corresponding fields in the asymmetric table by value κ, again only if all these fields are at least κ. Otherwise, the incoming packet is ignored. As mentioned in Section 6.2.3, κ should be set to the maximum acceptable ratio of outgoing-to-incoming packet ratio for legitimate TCP flows in the network. We use a value of 3 for κ.

Identifying Asymmetric Flows The per-packet updates to the symmetric tables are identical to the per-packet updates to the tables in the single gateway scheme. In Section 6.2.3, we described the effect of our update mechanism on entries of the tables. From that discussion, recall that entries in the tables corresponding to an asymmetric flow have net increments. In a single gateway network, the asymmetric flows at the gateway are attacks. In a multi-gateway network, asymmetric flows at a gateway may be attacks or legitimate. Thus, due to above described update mechanism, only entries of the symmetric table that correspond to either attacks or outgoing components of legitimate asymmetric flows will have net increments. If symmetric tables at all monitors are combined, then the entries of this aggregate symmetric table will identify all asymmetric flows in the network. The periodic processing function at the monitor use this aggregated symmetric

table to update its asymmetric table which in turn helps distinguish between legitimate asymmetric flows and attack flows. We describe the same in the next section where we also describe how per-packet updates to entries in asymmetric tables are used.

6.3.3 Periodic Processing

Let $S_m(t)$ and $A_m(t)$ represent respectively the symmetric and asymmetric tables at a monitor m at the end of interval t. At the beginning of interval $t+1$, the periodic processsing component of monitor m computes increment table, $\Delta\ S_m(t)$, as follows:

$$\Delta\ S_m(t)\ =\ S_m(t) - S_m(t-1)$$

Monitor m reports its increment table to a designated monitor, called the *rendezvous node*. The rendezvous node, on receiving the increment tables for period t from all monitors, computes the aggregate increment table, $\Delta\ S(t)$, by adding all the increment tables. The aggregate increment table values, as mentioned in the previous section, reflects the state of asymmetric flows in the network during period t. The rendezvous node sends the aggregate increment table to all the monitors. After m receives the aggregate increment table for period t, it adds the table to its asymmetric table $A_m(t)$, resulting in $A_m(t+\delta)$.

During interval $t+1$, the per-packet processing component of m updates the asymmetric table as described in the previous section. An incoming packet of flow f at m decrements corresponding entries in $S_m(t)$ if all these entries have values greater than κ. If the packet fails to decrement, ignoring the effects of aggregation, it means one of two things. First, decrements by f's previous incoming packets equaled the increments by

its outgoing packets thus far. This could occur in the single gateway version too. More interesting reason why the decrement fails is that f is an asymmetric flow. Flow f's outgoing packets exit the network at a gateway other than m and its incoming packets enter the network at m. Since its outgoing packets did not increment counters in $S_m(t)$, its incoming packets cannot decrement these counters in the table. However, the aggregate increment table and consequently $A_m(t+\delta)$, have counters that have been incremented by the outgoing packets of f. Hence, incoming packets of f will be able to decrement corresponding counters of $A_m(t+\delta)$. If f has excess incoming packets, these packets will be unable to decrement counters of $A_m(t+\delta)$ just as excess packets of a symmetric flow at m cannot decrement counters of $S_m(t)$.

At the end of interval $t+1$, incoming packets of f would have decremented the increments due to f's outgoing packets elsewhere, if f is legitimate. For each asymmetric flow in the network, its incoming packets during interval $t+1$ would have compensated at some monitor, the increments due its outgoing packets at some other monitor during interval t. The attack flows would not, however, have significant incoming packets. Hence, their corresponding counters in asymmetric tables will not be adequately decremented at any of the monitors. Consequently, the monitors coordinate and detect counters of asymmetric tables that exhibit net increments at all monitors during interval $t+1$. We use mechanims similar to detection in single gateway networks to detect attacks. We describe how we detect attacks next.

Monitors use only the asymmetric IP indexed table to detect ongoing attacks and identify victims. At the end of each interval t, monitor m, similar to the increment table, computes a decrement table, $\Delta\ A_m(t)$. It computes the decrement table using the

asymmetric IP indexed table as follows.

$$\Delta\ A_m(t)\ =\ A_m(t) - A_m(t-1+\delta)$$

m sends its decrement table to the rendezvous node. The rendezvous node computes the aggregate decrement table $\Delta\ A(t)$ by adding all the decrement tables for period t. [1] The rendezvous node sums up the aggregate increment table for period $t-1$ and the aggregate decrement table for period t to result in the anomaly table for period t, $\Gamma(t)$.

$$i.e., \Gamma(t) = \Delta\ S(t-1) + \Delta\ A(t)$$

Example

The steps of periodic processing are shown in the example in Figure 6.14. The active flows and their flow rates are as shown in Figure 6.11. The state of the tables at the end of intervals x is shown in the figure. Due to the outgoing and incoming packets of flows $00, 01$ and 11, the per-packet processing at m will result in the state after interval $x+1$ as shown in the figure. Since, there are no outgoing packets at monitor n, per-packet processing at n does not change the state of table $S_n(x)$. At the end of interval $x+1$, $\Delta\ S(x+1)$ is computed and added to tables $A_m(x+1)$ and $A_n(x+1)$. During the $x+2$ interval, the per-packet processing at n due to incoming packets of flow 11 will decrement the corresponding counters in table A_n resulting in the table $A_n(x+2)$. Table $A_m(x+2)$ is unchanged. As in the figure, table $\Delta\ A(x+2)$ is computed at the end of

[1] Note that entries of $\Delta\ A_i(t)$ and consequently, $\Delta\ A(t)$, will be less than or equal to zero.

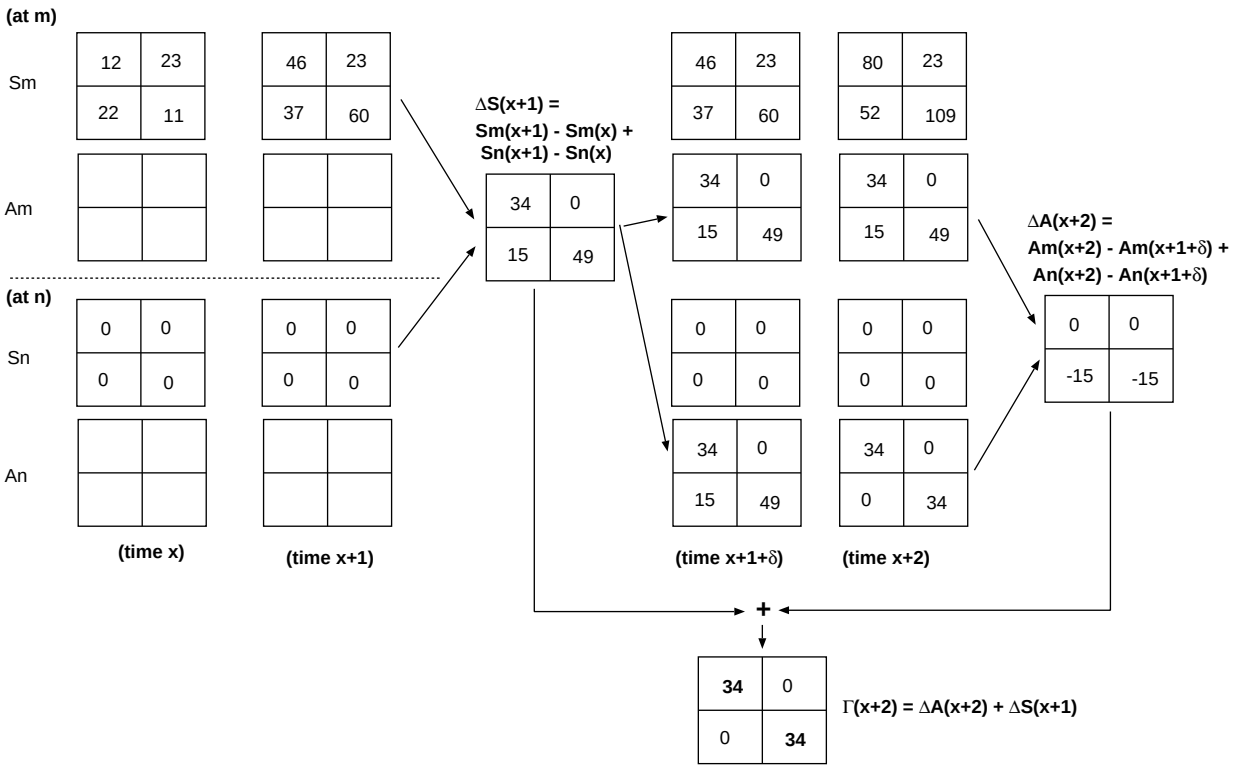

Figure 6.14: Detection in Multi-Gateway ASes — Example

interval $x + 2$, using the states of tables A_m and A_n at times $x + 1 + \delta$ and $x + 2$. Now, $\Gamma\ (x + 2)$ is computed by simply adding $\Delta\ A(x + 2)$ and $\Delta\ S(x + 1)$. $\Gamma\ (x + 2)$'s entries corresponding to flow 01 demonstrate that the flow is an attack. We next describe the precise attack detection mechanism.

Detecting Attacks

We described in previous sections how attacks cause increments in the counters of Γ. Flow aggregation, as discussed in previous section, will also result in some counters that do not correspond to attacks to show net increments for brief periods of time. We rely on flow scoring described for single gateway detection scheme to distinguish between counter increments corresponding to attacks and counter increments corresponding to

procedure $DetectAttackCounters$

1: compute $avg[t]$ *and* $std.dev[t]$ $\forall\, \gamma[t] > 0;\ \gamma \in \Gamma$
2: **for all** $\gamma \in \Gamma$ **do**
3: $score[\gamma] += \gamma[t] - avg[t] - std.dev[t]$
4: $score[\gamma] = max(0,\ min(score[\gamma],\ detec\, thres))$
5: **if** $score[\gamma] \geq detec\, thres$ **then**
6: $anomaly - count[\gamma] += 1$
7: **end if**
8: **if** $flagged[\gamma] \geq anomaly - count\ thres$ **then**
9: flag γ
10: **end if**
11: **end for**

Figure 6.15: Attack Detection Procedure

flow aggregation. We compute a score over time for each of the counters in the Γ table and use the scores to detect subnets under attack. This procedure is given in Figure 6.15.

As in the single gateway case, the average and standard deviation are computed over all entries in $\Gamma(t)$ that have a positive value. These values, as shown in the figure, are used as a base line and are used to calculate 'instantaneous scores' for counters in Γ for period t. Instantaneous score of a counter is the counter's score for the period for which it is computed and is added to the counter's actual score.

Once a counter's score is updated, it is compared against a *detection threshold*. The detection threshold is used to detect counters to which attacks are mapped. It is set proportional to sampling rate used for sampling the packets. If the counter's score is greater than or equal to the detection threshold, an anomaly count associated with the counter is incremented. A counter's anomaly count is the number of periods the counter's score was above detection threshold. Only if a counter's anomaly count reaches a threshold, called the *anomaly count threshold*, is the counter flagged.

An anomaly count for counters is the only addition to the attack detection phase in multi-gateway networks and is used for the following reason. Contrary to our assumption, asymmetric flows may be short. They may also have decreasing flow rates over time. Both can result in increments to corresponding counters that will not be sufficiently decremented during the next interval. As a result, the score for such a counter will increase during consecutive intervals and will cross the detection threshold. However, the score will fall below the score threshold as soon as the flow stops, which is typically one or two intervals after the flow starts. On the other hand, a bandwidth attack is active over several intervals and hence, the scores of corresponding counters will be above the threshold for several intervals. Thus, a threshold for number of intervals when a score is above the detection threshold, can differentiate between attack flows and short duration asymmetric flows. We call such a threshold, anomaly count threshold. When a counter's anomaly count crosses the anomaly count threshold, we flag the counter as an attack's counter. We study the effect of detection threshold and anomaly count threshold in Section 7.3. We next present the evaluations of our subnet attack detection schemes for single- and multi-gateway networks using simulations.

Chapter 7

Simulations

7.1 Simulation Setup

We next present the simulations for subnet attack detection scheme. We ran simulations to investigate the quality of the detection scheme for different table sizes and different attack thresholds. We used a sampling rate of 10% for the following simulations. As before, we measure the quality of the detection scheme using:

- # False Positives: the number of flagged table entries that do not correspond to attacks.
- Detection Rate: the percentage of the attacks that are detected.
- Detection Time: the time from the beginning of an attack to when the system reports it. It does not include any penalty for undetected attacks.

For a given table size and attack threshold, we vary the attack rate and measure detection accuracy. Finally, we simulate attack scenarios where prefixes of victim subnets are not

aligned with segment partitions of the table and measure detection accuracy. We use Abilene trace for these simulations and all results are an average of eight runs. The characteristics of the trace are presented in Table 5.1 in Section 5. The 95% confidence interval for detection times is less 2 seconds for all the attack scenarios.

Trace and anonymization The Abilene trace has IP addresses anonymized, mapped to a /12 prefix. The result is that flows map to one entry in the leftmost columns and few entries in the next column of the IP indexed table. To simulate practical scenarios, we randomize addresses in the trace and use them as flow identifiers resulting in a uniform assignment of flows to entries in the left columns.

Attack Flows The attack traffic is composed of synthetically generated TCP packets. Each simulation has five attacks, all attacks last for two minutes and none of the attacks overlap in time. Victim subnets are chosen randomly and have sixteen addresses (in this first step of experiments). The attack packets of a flow are randomly distributed among the addresses in the subnet. Unless noted, attack rate is set to 60 packets per second and detection threshold is set at 100. Recall that the average flow rate in the trace is 30 packets per second.

7.2 Detecting Attacks With Symmetric Traffic

7.2.1 Aggregation vs. Detection

We first consider the effect of aggregation on detection. We used two detection threshold values, 50 and 100. We varied the number of rows in the tables from 128 rows to 1024 rows to determine the effect of aggregation on detection. The flow identifier is partitioned into 4 parts when the table has 128 and 256 rows and is partitioned into 3 parts when the table has 512 and 1024 rows. Results are presented in Table 7.1. We observe from the table that as the number of rows is increased, false positives decrease and detection time decreases. This is because, as the number of rows increases in the table, flows are aggregated less. As we described previously, this decreases the probablity for incorrect decrements and hence, there are fewer false positives, reduced detection time and better detection rate.

Increasing the detection threshold, as can be expected, decreases number of false positives while increasing the detection time. This behavior can be seen in the results shown in the table. When all attacks are detected, the increase in detection time is approximately proportional to increase in the detection threshold. Thus, detection threshold can be used to vary detection sensitivity of the detection scheme.

7.2.2 Attack Rate vs. Detection

In this set of simulations, attack rate is varied from 10 packets per second to 100 packets per second. Number of rows in the table is set to 512 and detection threshold is

	Avg. # False Positives		*Detection Rate (%)*		*Detection Time (sec)*	
Table Size	*Detec. Thres*		*Detec. Thres*		*Detec. Thres*	
(# rows)	50	100	50	100	50	100
128	9.8	1.5	85	42.4	51.03	63.64
256	2	0	100	100	21.24	36.87
512	2	0	100	100	14.61	26.00
1024	2	0	100	100	12.74	22.12

Table 7.1: Table Size vs. Detection Accuracy

set to 100. Results are presented in Table 7.2. There are no false positives in these runs. From the table, we can observe that very low intensity attacks cannot be detected by the detection scheme. This is expected because the mean and standard deviation computed for net increments will mask the counter increments due very low rate attacks. As attack rate increases, detection time decreases and detection rate increases.

	Attack Rate			
	10	30	60	100
Detec. Rate	0	97.6	100	100
Detec. Time	0	42.77	26.00	15.00

Table 7.2: Attack Rate vs. Detection Accuracy

7.2.3 Non-Byte Aligned Subnet Prefixes

In our experiments so far, the attack targetted /28 subnets with prefixes of type a.b.c.*. Hence different flows of a subnet attack mapped to the same set of counters in all but the rightmost column of the IP indexed table. In these simulations, we vary the size of victim subnets from /21 through /24. We also byte align the columns in the IP and hash indexed table. I.e., a column of the IP (or hash table) starts with the first bit of corresponding byte in the flow identifiers. For tables with 512 and 1024 rows, we make

this possible by making segments overlap. Note that a /21 mask corresponds to a subnet of 2048 hosts and a /24 mask corresponds to a subnet of 256 hosts. Hence, IP addresses of the victim subnet map to multiple entries in the two rightmost columns of the IP indexed table (except when the IP table has 256 rows and the victim subnet has 256 hosts). We set the attack rate to 60 packets per second, detection threshold to 100 and sampling rate to 10%. We measure detection rate and time using the counters in columns 1 and 2. The detection rate for these results is 100%. The detection times are presented in Table 7.3. From the results in the table, we can conclude that detection times do not increase by varying the size of the victim subnets.

	Subnet Size (in # of hosts)			
# rows	/24	/23	/22	/21
256	35.49	35.12	35.24	34.37
512	25.87	26.62	26.62	26.49
1024	22.37	22.74	22.74	22.74

Table 7.3: Subnet Size vs. Detection Accuracy

7.3 Detecting Attacks With Asymmetric Traffic

In this section, we evaluate our subnet attack detection scheme in multi-gateway networks. We use the Abilene trace for the background traffic and, as in all previous evaluations, use synthetic traffic for attack traffic. The target subnets for attacks in the following evaluation simulations have 256 hosts. The aggregate attack traffic rate, unless otherwise mentioned, is 100 packets per second. We use tables with three columns and 512 counters in each column for our symmetric and asymmetric tables. Thus, each segment of the flow identifier has 9 bits and the segments do not overlap. The victim subnets

for the simulations in this section have 256 hosts. In the first set of simulations, we evaluate the effect of detection threshold and anomaly count threshold on detection accuracy. We next evaluate the performance of our system for different rates of traffic asymmetry and different attack rates. In all these scenarios, we assume that the multi-gateway network has two gateways. We conclude this chapter by studying the effectiveness of the detection scheme when the network has more than two gateways.

7.3.1 Multi-Gateway Detection Parameters

In this set of simulations, we study the effect of detection parameters, namely detection threshold and anomaly count threshold, on detection sensitivity. We set the attack rate to 100 packets per second. Flows to 50% of the subnets which have flows from the stub domain are asymmetric. Recall that, for single gateway scheme, we used a detection threshold of 100. Here, we vary detection threshold between 100 and 300. We also vary anomaly count threshold between 1 and 4. A value of 1 for anomaly threshold implies that the corresponding counter is flagged as soon as its score reaches the detection threshold. A value of 2 implies that counter's score is above the threshold during two intervals. The results of the simulation are presented in Table 7.4.

When the detection or the anomaly count threshold are increased, as expected, the detection time increases and the number of false positives decrease. It is interesting to compare the result when detection and anomaly count thresholds are 100 and 3 with the result when they are 200 and 1 respectively. The earlier result has no false positives and yet, the detection time is lower than the second result which has one false positive. This

Attack Rate (pps)	*Detection Threshold*	*Anomaly Count Threshold*	*Detection # Time (sec)*	*Detection Rate*	*# of False Positives*
100	100	1	21.99	1.0	4
100	100	2	27.99	1.0	1
100	100	3	32.99	1.0	0
100	100	4	38.99	1.0	0
100	200	1	37.99	1.0	1
100	200	2	41.99	1.0	0
100	200	3	46.99	1.0	0
100	200	4	51.99	1.0	0
100	300	1	50.99	1.0	1
100	300	2	55.99	1.0	0
100	300	3	60.99	1.0	0
100	300	4	65.99	1.0	0

Table 7.4: Detection Parameters vs Detection Accuracy

is because of the following reason. In the second result, the false positive is due to a large asymmetric flow of short duration. Thus, even though the detection threshold is double in this case, the scores of the flow's counters are reaching the threshold easily. On the other hand, the scores for counters corresponding to attacks take longer to reach the threshold and hence, the higher detection time. In the first result, the lower threshold implies that scores for attack's counters reach the threshold faster. The scores of legitimate asymmetric flow's counters too reach the threshold faster. However, the flow is of short duration and the wait time involved with anomaly count threshold prevents it from being flagged as a false positive.

		asymmetry=10%			asymmetry=50%		
Attack Rate (pps)	*Anomaly Count Threshold*	*Detection Time (sec)*	*Detection Rate*	*False Positives*	*Detection Time (sec)*	*Detection Rate*	*False Positives*
100	1	19.99	1.0	0	21.99	1.0	4
100	2	24.99	1.0	0	27.99	1.0	1
100	3	29.99	1.0	0	32.99	1.0	0
100	4	34.99	1.0	0	37.99	1.0	0

Table 7.5: Traffic Asymmetry vs Detection Accuracy

7.3.2 Detection Accuracy vs Traffic Asymmetry

In Table 7.5, we evaluate the behavior of our detection scheme as we vary the amount of asymmetric traffic in the network. Here, we set the percentage of subnets whose flows are asymmetric to 10% and 50%. The detection and anomaly count thresholds are set to 100 and 3 respectively. We can observe from the results in the table that, as asymmetry decreases, number of false positives as well as detection time decrease. This is because, as asymmetry in the network decreases, the net increments decrease for counters corresponding to asymmetric flows. Hence, the average and the standard deviation computed for the base line also decrease and scores for attack counters increase at a higher rate. Hence, attacks are detected earlier. False positives decrease because there are fewer legitimate asymmetric flows in the network which may cause undue increments to counters.

7.3.3 Detection Accuracy and Attack Rate

We now vary the attack rate and evaluate the performance of our system. We varied the attack rate between 50 and 150 packets per second. The detection threshold and

Attack Rate (pps)	*Detection Time (sec)*	*Detection Rate*	*False Positives*
50	70.99	1.0	0
100	32.99	1.0	0
150	25.99	1.0	0

Table 7.6: Attack Rate vs Detection Accuracy

anomaly count threshold are again 100 and 3 respectively. The results are presented in Table 7.6. As we expect, as the attack rate increases, detection time decreases. As attack rate increases, detection rate would also increase. However, our detection system detects all attacks even with an attack rate of 50 packets per second. Hence, there is no change in the detection rate as attack rate increases.

7.3.4 Detection Accuracy vs Border Gateways

	Anomaly Count Threshold = 3			*Anomaly Count Threshold = 4*		
# of Gateways	*Detec # Time (sec)*	*Detect. Rate*	*False Posit*	*Detec # Time*	*Detect. Rate*	*False Posit*
2	32.99	1.0	0	37.99	1.0	0
3	34.99	1.0	0	39.99	1.0	0
4	36.99	1.0	1	41.99	1.0	0

Table 7.7: Number of Gateways vs Detection Accuracy

The final result we consider is the performance of the system when the number of gateways is increased. We use up to 4 gateways for this result. We set detection threshold to 100 again but we use 3 and 4 for anomaly count threshold. Traffic to 50% of subnets is asymmetric. The results are presented in Table 7.7. From the table, we observe that our system performs well even when the number of gateways in the network

increases. As the number of gateways in the network increases, the detection time also increases. The reason is as follows. Consider a network with two gateways, say A and B. Then, on average, 50% of asymmetric flows are incoming at A. Hence, incoming packets at A will be able to decrement counters in its asymmetric table corresponding to only 50% of asymmetric flows. If the network has three gateways, then the incoming packets at a gateway, on average, can only decrement counters in its asymmetric table corresponding to only 33% of asymmetric flows. With four gateways, this figure will be 25%. Hence, as the number of gateways in the network increases, the number of uncompensated counters in asymmetric tables increases. As a result, incoming packets of asymmetric flows will decrement counters (including counters to which attacks are mapped) more than they should (i.e., more than the increments by their corresponding outgoing packets). The consequence is, it will take longer for attack's counters to be flagged which will increase the detection time. In short, as the number of gateways in the network increases, the effects of aggregation increases and it will take longer to detect attacks. For the same reason, number of false positives also increase when number of gateways increases. However, as shown in the table, false positives can be prevented by increasing the anomaly count threshold.

Thus, we can conclude that our scheme performs well in single- as well as multi-gateway networks under different network and attack scenarios.

Chapter 8

Conclusions and Future Work

8.1 Thesis and Contributions

In this dissertation, we demonstrated that it is feasible to develop source-end DDoS detection systems with the following properties: *scales with network traffic, deployable in multi-gateway networks, flexible to allow tradeoffs between detection sensitivity and resources required, resilient to attacks against the system and does not require expensive changes to current deployed infrastructure.* We described our heuristic to detect bandwidth attacks using TCP packets and the architecture of our distributed source DDoS detection system. We described schemes that the system can execute to detect host and subnet bandwidth attacks. We also demonstrated the performance of our schemes using extensive simulations using real traces, collected from high bandwidth routers in the Internet. Our contributions in this dissertation are as follows:

- We described in detail what a bandwidth attack is and how a large scale bandwidth

attack can directly and indirectly affect a large set of unrelated Internet hosts besides the intended victims. We also discussed various approaches to detect these attacks and why source domain detection is most useful among different approaches.

- We stated different issues with source domain detection and discussed in detail important problems such as line speed processing of packets in the network. We described an in-network overlay network with distributed architecture that can address these issues and detect different types of bandwidth attacks in stub domains.

- We next described our host attack detection scheme that, with very little flow state, can detect bandwidth attacks using TCP packets against individual hosts, even the attack rate is very low. We preesented the extensions to the protocol detect attacks in multi-gateway networks and in the presence of asymmetric traffic in such networks. Ours is the first system to study this issue in detail. We demonstrated how changing system parameters alters the detection sensitivity of the detection scheme. We performed extensive packet-level simulations to evaluate our scheme under various host attack scenarios and demonstrated that ours is a robust system that can detect different types of host attacks quickly and effectively. We also demonstrated that our system is resilient against attacks that target the detection system itself.

- We demonstrated that subnet attacks are more diffused than host attacks and hence as a result, that host attack detection schemes for source domain detection are insufficient to detect subnet attacks. We described our subnet attack detection scheme for source domain detection which can quickly, effectively and efficiently detect subnets attacks in source domains. To our knowledge, ours is the only system

that explicitly detects subnet attacks. We discussed the extensions for deploying the scheme in multi-gateway networks where flows can be asymmetric. We again investigated the effect of different scheme parameters and how these parameters influence the detection sensitivity of the scheme. We presented detailed evaluations of our detection scheme for different attack scenarios which demonstrated that our scheme detects subnet attacks when the attack rate is low as well as when the attack is very diffused (i.e., when the size of the victim subnet is very large).

8.2 Future Work

A natural extension for our source domain detection system is to detect other malicious traffic that is generated from within the domain. In the first part, we will discuss extentions to the current work to detect bandwidth attacks using IP and UDP packets. Next, we discuss the directions for our detection scheme to detect DDoS attacks that target victim's resources other than bandwidth. Hosts within a domain may participate in other malicious activities such as port scanning and network and host intrusion. Hosts may also participate to send spam and in virus propagation. We believe that stub domain detection is also useful to detect and prevent these types of malicious traffic in the network. We will conclude by briefly discussing some of the approaches our system may take to detect these different types of malicious traffic in stub domains.

8.2.1 Detecting Bandwidth Attacks using non-TCP Packets

Bandwidth attacks may be performed using protocols other than TCP, e.g. ICMP ping attacks and attacks using UDP segments. We will extend our system with mechanisms to detect these type of attacks. Some of these protocols such as DNS queries and ICMP ping messages have proportional bidirectional traffic; i.e. number of packets in forward direction is proportional to number of packets in the reverse direction. Our detection system can detect attacks using these protocols by employing the techniques and data structures described previously for TCP flows. This provides the benefit that attacks that use packets of different protocols to flood the victim network will still be detected. An important consideration is to identify all UDP and IP protocols that have proportional bidirectional flows and build flexibility into the system such that newer protocol models can be easily integrated.

DDoS attacks that use flow types without proportional bidirectional traffic are hard to detect because these flows lack mechanisms that suggest the legitimacy of these flows. In such a scenario, passive detection require heuristic solutions to determine the state of these flows in source domains. Such a heuristic would incorporate history and state profiles of a flow's end-hosts to determine the legitimacy of a flow. For example, several simultaneous UDP connections to a destination may be viewed as suspicious if that destination has not served UDP flows previously. In [55], authors use similar history profiles at edge routers in source domains to detect suspicious flows. In D-WARD [4], the authors use thresholds for flow rates and number of connections to limit traffic belonging to such flows. In particular, they restrict the number of non-TCP connections to an external ad-

dress and the maximum flow rate for each connection. A drawback of these approaches is that legitimate flows may get affected even when the destination of the flows is not under attack.

A detection system may also first detect if a destination is under attack and then use this detection to determine if a “uni-directional” flow is legitimate. The detection system may passively monitor the TCP traffic to an end-host or any anomalous traffic to/from the end-host to identify the state of an end-host. The detection system may also use active probing schemes to determine the state of the end-host. Once the detection system identifies a host under attack, it may maintain state for the uni-directional flows and apply protocol specific tests to correctly determine attack flows from among the unidirectional flows in the network. We will investigate both the methods mentioned and evaluate the pros and cons of the two schemes in detail.

8.2.2 Detecting Other DDoS Attacks

In the discussion so far, we considered only DDoS attacks that exhaust the bandwidth of the victim’s access link. DDoS attacks may also target a victim’s memory or processing capacities. The difference between bandwidth and other DDoS attacks is, in the latter type of attacks, the attack packets reach the victim. The victim may elect to respond to as many packets as it can, legitimate and otherwise. In such a case, it may respond to attack packets belonging to illegitimate TCP flows with reset packets. The consequence is, the ratio for these type of attack flows may not be higer than the threshold for legitimate TCP flows. Thus, we require different mechanisms to detect these types of

attacks. One possible mechanism is to track the number of reset messages received from an external host. The other mechanism is to track the variations of round trip times to a subnet and use the variations to determine if one or more hosts in the subnet are under attack. Similar approaches can be used to detect non-TCP flows with bidirectional traffic.

8.2.3 Detecting Malicious Traffic

End hosts in a domain can be used for various other malicious activity in the Internet. They are used as stepping stones [42] to perform port scanning at other end hosts and consequently, identify and compromise susceptible end hosts. Compromised end hosts are used to hack into sensitive networks and are also used for stealing identifies of users in compromised networks. Internet will be come more secure and reliable if all this malicious traffic is detected and prevented.

It will be very difficult to detect some types of malicious traffic in source domains. For example, some of the attack packets can be specifically constructed to exploit the weakness(es) at a particular host, say weakness due to a proprietary OS or application. In such cases, especially if the host is outside the source domain, it will be very hard for the detection system deployed within the system to identify the weakness and corresponding exploits.

A general approach to detect malicious traffic within the source domain can be as follows. Most of the traffic within a domain will be legitimate. Hence, the first step for the detection system will be to quickly identify traffic that is likely to legitimate with high confidence. The detection system can consider a packet as legitimate if it is sent by a

trusted local host or to a well-known secured destination. The detection system can also consider a packet as legitimate if it belongs to well-known applications that are considered secure. For the remaining packets, the detection system can apply perform packet, flow, protocol and source level analyses to determine if the packet may be malicious. It can consider factors such as sizes of packets, fields in the packets, packet rate of a flow belonging to a particular protocol and so on. It can use special packets such as TCP reset packets to determine if packets sent earlier are possibly suspicious. Thus, the detection system can use various approaches to help to classify if a packet is legitimate or not. If it cannot make such a determination for packets, then it can store these packets briefly to verify the status of the packet using future packets of the flow. Finally, the detection system can deploy honey pots within the domain to quickly identify the popular types of malicious attacks and derive signatures online for these attack flows. It can use the signatures to determine which of the flows that originate within the network are malicious. Thus, our detection system can be generalized to detect different malicious traffic in source domains.

Bibliography

[1] J. Cowie, A. Ogielski, B.J. Premore, and Y. Yuan. www.renesys.com/projects/bgp_instability, September 2001.

[2] `http://www.honeynet.org/papers/bots/`.

[3] Thomer M. Gil and Massimiliano Poletto. MULTOPS: A Data-Structure for bandwidth attack detection. In *In Proceedings of 10th Usenix Security Symposium*, pages 23–38, August 2001.

[4] J. Mirkovic, G. Prier, and P. Reiher. Attacking DDoS at the source. In *Proceedings of International Conference on Network Protocols, Paris, France, November 2002.*, 2002.

[5] M. Allman, V. Paxson, and W. Stevens. RFC 2581: TCP Congestion Control, April 1999.

[6] J. Mirkovic. *D-WARD: Source-End Defense Against Distributed Denial-of-Service Attacks*. PhD thesis, University of California, Los Angeles, August 2003.

[7] L. Heberlein and M. Bishop. Attack class: Address spoofing. In *Proceedings of the 19th National Information Systems Security Conference*, pages 371–377, 1996.

[8] S. Bellovin, M. Leech, and T. Taylor. The ICMP traceback messages. In *Internet draft, work in progress*, February 2003.

[9] A. C. Snoeren, C. Partridge, L. A. Sancheq, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based IP traceback. In *ACM SIGCOMM*, 2001.

[10] A. Belenky and N. Ansari. IP traceback with deterministic packet marking. *IEEE Communications Letters*, 7(4), April 2003.

[11] A. Mankin, D. Massey, C. Wu, S. Wu, and L. Zhang. On design and evaluation of intention-driven ICMP traceback. In *IEEE International Conference on Computer Communications and Networks*, October 2001.

[12] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. *ACM SIGCOMM*, 2000.

[13] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Network support for IP traceback. *IEEE/ACM Transactions on Networking*, 9(3), June 2001.

[14] D. Dean, M. Franklin, and A. Stubblefield. An algebraic approach to IP traceback. *ACM Transactions on Information and System Security*, 5(2):119–137, 2002.

[15] A. Yaar, A. Perrig, and D. Song. Pi: A path identification mechanism to defend against DDoS attacks. In *In IEEE Symposium on Security and Privacy*, May 2003.

[16] D. X. Song and A. Perrig. Advanced and authenticated marking schemes for IP traceback. In *Proceedings of IEEE Infocomm*, 2001.

[17] R. Stone. Centertrack: an IP overlay network for tracking DoS floods. In *Proceedings of the USENIX Security Symposium*, August 2000.

[18] H. Burch and B. Cheswick. Tracing anonymous packets to their approximate source. In *Proceedings of the USENIX Large Installation Systems Administration Conference*, pages 319–327, December 2000.

[19] P. Ferguson and D. Senie. Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing. *RFC 2827*, May 2000.

[20] K. Park and H. Lee. On the effectiveness of probabilistic packet marking for IP traceback under a denial of service attack. In *Proceedings of the IEEE Infocom*, April 2001.

[21] C. Jin, H. Wang, and K. Shin. Hop-count filtering: An effective defense against spoofed DoS traffic. In *Conference on Computer and Communications Security (CCS)*, pages 30–41, October 2003.

[22] M. Sung and J. Xu. IP traceback-based intelligent packet filtering: A novel technique for defending against internet DDoS attacks. *IEEE Transactions on Parallel and Distributed Systems*, 14(9):861–872, September 2003.

[23] A. Yaar, A. Perrig, and D. Song. SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks. In *Proceedings of the IEEE Security and Privacy Symposium*, May 2004.

[24] K. Lakshminarayanan, D. Adkins, A. Perrig, and I. Stoica. Taming IP packet flooding attacks. *SIGCOMM Comput. Commun. Rev.*, 34(1):45–50, 2004.

[25] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proceedings of Network and Distributed System Security Symposium*, February 2002.

[26] S. Floyd, S. Bellovin, J. Ioannidis, K. Kompella, R. Mahajan, and V. Paxson. Push-back messages for controlling aggregates in the network. In *Internet Draft, work in progress.*, 2001.

[27] F. Baker and P. Savola. Ingress filtering for multihomed networks. *RFC 3704*, 2004.

[28] J. Li, J. abd Mirkovic, M. Wang, P. Reiher, and L. Zhang. Save: Source address validity enforcement protocol. In *INFOCOM*, 2002.

[29] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4), 1999.

[30] W. Feng, K.G. Shin, D.D. Kandlur, and D. Saha. The Blue active queue management algorithms. *IEEE/ACM Transactions on Networking*, 10(4):513–528, 2002.

[31] J. Widmer, D. Denda, and M. Mauve. A survey on TCP-friendly congestion control. *IEEE Network*, 15(3):28–37, May/June 2002.

[32] S. Floyd, M. Handley, and J. Padhye. A comparison of equation-based and AIMD congestion control. *IEEE/ACM NOSSDAV*, June 2001.

[33] S. Floyd and K. Fall. Router mechanisms to support end-to-end congestion control. *LBL Tech. Report*, 1997.

[34] S. Reddy. LRU-RED: An active queue management scheme to contain high bandwidth flows at congested routers. In *Proceedings of Global Telecommunications Conference*, 2001.

[35] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *Technical report*, 2001.

[36] T. Peng, C. Leckie, and K. Ramamohanarao. Defending against distributed denial of service attack using selective pushback. In *Proceedings of IEEE International Conference on Telecommunications*, 2002.

[37] Seong Soo Kim and A. L. Narasimha Reddy. Detecting traffic anomalies at the source through aggregate analysis of packet header data. In *Proceedings of Networking*, 2004.

[38] A. Ramanathan. WADES: A tool for distributed denial of service attack detection. Master's thesis, Texas A&M University, College Station, 2002.

[39] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *Internet Measurement Workshop 2002.*, 2002.

[40] B. Krishnamurthy, S. Subhabrata, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Ineternet Measurement Conference*, 2003.

[41] `http://wind.lcs.mit.edu/~dga/ddos.txt`.

[42] Y. Zhang and V. Paxson. Detecting Stepping Stones. In *USENIX Security Symposium*, 2000.

[43] `http://www.bro-ids.org/`.

[44] `http://www.snort.org`.

[45] `http://www.snort.org`.

[46] `http://staff.washington.edu/dittrich/misc/ddos/`.

[47] `http://www.flashnetworks.com`.

[48] `http://www.finisar.com`.

[49] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. On the characteristics and origins of Internet flow rates. In *ACM SIGCOMM*, 2002.

[50] http://newsroom.cisco.com/dlls/prod_121003.html.

[51] http://www.mosysinc.com/products/1t_sram.html.

[52] http://www.tezzaron.com/products/technology/3t-iram.htm.

[53] NLANR PMA: Special traces archive *http://pma.nlanr.net/special/*.

[54] `http://www.graphpad.com/articles/outlier.htm`.

[55] T. Peng, C. Leckie, and K. Ramamohanarao. Protection from distributed denial of service attack using history-based IP filtering. In *Proceedings of IEEE International Conference on Communications (ICC)*, 2003.