A Novel Distributed Denial-of-Service Detection Algorithm

A Thesis

Presented to

the Faculty of

California Polytechnic State University, San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Brett Tsudama

June 2004

AUTHORIZATION FOR REPRODUCTION OF MASTER'S THESIS

I grant permission for the reproduction of this thesis in its entirety or any of its parts, without further authorization from me.

Signature

6/10/2004

Date

APPROVAL PAGE

TITLE: A Novel Distributed Denial-of-Service Detection Algorithm

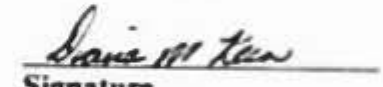
AUTHOR: Brett Tsudama

DATE SUBMITTED: June 10, 2004

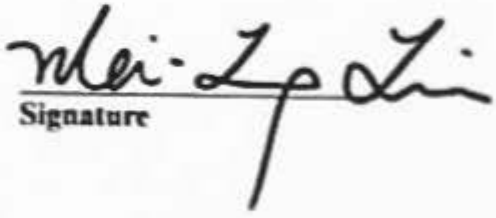
Dr. Hugh Smith
Advisor


Signature

Dr. Diana Keen
Committee Member


Signature

Dr. Mei-Ling Liu
Committee Member


Signature

ABSTRACT

A Novel Distributed Denial-of-Service Detection Algorithm

Brett Tsudama

Distributed Denial-of-Service (DDoS) attacks are becoming increasingly prevalent in the Internet today. A DDoS attack attempts to prevent a victim from providing a certain level of service to users, and the effects of a successful attack can range from minor inconveniences to major financial consequences. This thesis presents a novel approach for the detection of a DDoS attack called Ratio-based SYN Flood Detection (RSD). RSD is an adaptive threshold algorithm and will dynamically adjust its behavior to achieve optimal performance in the face of varying traffic conditions. A methodology is presented which optimizes RSD to make it more generally suited for widespread deployment, and its efficacy is validated through trace-driven simulations.

Network processors have been developed for routers in order to provide hardware-level performance with the flexibility of a programmable architecture. Network processors are located on the edge routers of a stub network, which is also where RSD is deployed. The applicability of implementing RSD on a network processor is explored, and initial results from this investigation are presented.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Denial-of-Service (DoS) attacks and Distributed Denial-of-Service (DDoS) are becoming increasingly prevalent in the Internet today. A DoS attack is "characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that service" [26]. In other words, a DoS attack aims to disrupt the normal operation of an Internet service through malicious means. A Distributed DoS attack is simply a DoS attack in which multiple distributed computers are used to perform the attack. The impact of these attacks can range from minor inconveniences, such as slow response from a website, to major financial consequences, such as the highly publicized DoS attacks in February of 2000. The latter attacks targeted high-profile companies who rely on the internet for their revenue, including Amazon.com, E*Trade, eBay, and Yahoo. Most recently, Microsoft, The SCO Group, and the RIAA have been the target of DoS attacks. In the case of The SCO Group, the attacks even succeeded in crippling their website.

Denial of Service attacks attempt to consume enough of a victim's limited resources until the victim can no longer provide a certain level of service to legitimate users. These resources can be network bandwidth, computing cycles, or operating system data structures [24]. In order to consume these resources, attackers use two methods: software exploits and flooding attacks. Software exploits take advantage of bugs in the victim's operating system, and flooding attacks send a large amount of traffic to the victim.

While most detected DoS attacks have been shown to come from a single source [29], a growing trend is to utilize multiple sources to send attack traffic. This is called a Distributed DoS attack, and this thesis will concentrate solely on DDoS flooding-based attacks.

This thesis presents a new approach for the detection of DDoS SYN-flood attacks called Ratio-based SYN Flood Detection (RSD). RSD is a type of adaptive threshold algorithm and will dynamically adjust its behavior to achieve optimal performance in the face of varying traffic conditions. A methodology is presented which optimizes RSD to make it more generally suited for widespread deployment, and its efficacy is validated through trace-driven simulations. This thesis will also investigate the applicability of implementing RSD on a network processor and will show, through simulation, that a minimal amount of processing overhead is required to implement RSD.

This thesis is organized as follows: Chapter 2 provides background information on DDoS attacks and network processors as well as current research trends in both areas. Chapter 3 presents the RSD algorithm in detail, along with a methodology that can be used to optimize its performance. Chapter 4 analyzes the results obtained when an implementation of RSD is validated against a number of real-world traffic traces. Chapter 5 investigates the applicability of implementing RSD on a network processor, and Chapter 6 presents conclusions along with suggestions for future work.

CHAPTER 2: BACKGROUND AND RELATED WORKS

This chapter will provide an understanding of different types of DDoS attacks and the mechanisms behind their effectiveness. Existing solutions that attempt to prevent and detect DDoS attacks are presented, along with a background on network processors and current research in the area.

2.1 Types of DoS Attacks

Denial of Service attacks attempt to consume enough of a victim's limited resources until the victim can no longer provide a certain level of service to legitimate users. These resources can be network bandwidth, computing cycles, or operating system data structures [24]. In order to consume these resources, attackers use two methods: software exploits and flooding attacks.

Attacks that fall in the software exploit category typically take advantage of bugs in the victim's operating system. By taking advantage of these bugs, attackers can succeed in crashing or disabling the victim with a small number of packets. An example of this type of attack is the "ping-of-death" attack [28] which sends an extremely large ICMP echo packet to the victim and causes some operating systems to crash, reboot, or freeze. Another example is the "Land" attack [17] which sends a TCP SYN packet containing the victim's IP as the source and destination address, thereby causing an infinite loop in the protocol stack [24]. These types of attacks can be mitigated through the use of patches and OS updates, and their prevalence has declined as users have become better about updating and protecting their systems.

A far more widespread and dangerous type of attack is a flooding-based attack [11]. Flooding attacks send a huge amount of traffic to the victim, resulting in bandwidth saturation or resource saturation as the victim attempts to deal with the traffic. The packets are typically generated to take advantage of the way a victim deals with certain protocols, as is the case with SYN flooding [16] which exploits TCP's three-way handshake mechanism. Other examples include attacks that use TCP packets with no flags set (TCP NULL attack), TCP packets with all flags set (Xmas attack), and packets with a non-existent IP protocol number [24]. In all of these cases, the victim allocates resources to handle each packet, and the sheer number of packets causes the victim to crash.

2.2 DDoS Flooding Attacks

As mentioned before, a DDoS attack utilizes multiple hosts on the internet to send a large volume of traffic to a victim, in the hopes of crippling it or making it unavailable. In general, there are two types of flooding attacks: direct attacks and reflector attacks [11].

2.2.1 Direct Attacks

A direct attack is when an attacker arranges to have a large number of packets sent directly to a victim. These packets can be generated solely by the attacker or, more commonly, through a DDoS attack network of compromised hosts. Such a network is composed of one or more attacking hosts which control some number of masters, or controllers. These masters in turn control a large number of zombies, or daemons (figure 1) which are the entities that actually send the attack traffic to the victim. Attacking hosts originate the attack sequence by notifying the masters who then instruct the zombies to

begin sending attack packets. Attackers will typically compromise an attacking host first, which is then used to scan for other vulnerable hosts in order to install specific DDoS master and zombie programs.

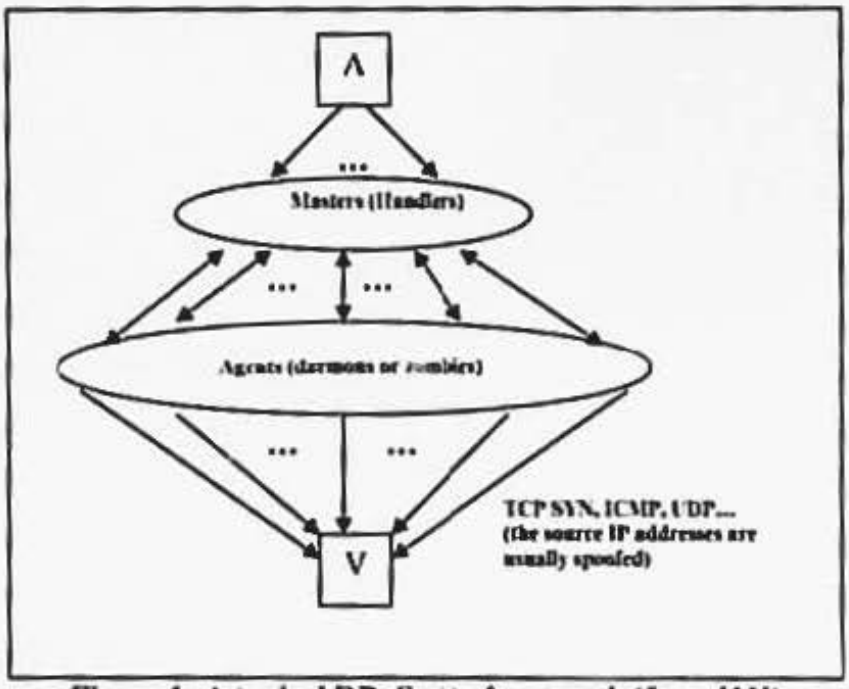


Figure 1: A typical DDoS attack network (from [11])

Setting up a DDoS attack network gives the attacker the advantage of anonymity (since packets are being generated by other legitimate, albeit compromised, hosts) and increased firepower due to the possibility of exponential growth using the master-zombie model. A number of tools are freely available on the Internet to aid in constructing an attack network and launching a DDoS attack, including Trinoo, Tribe Flood Network 2000, and Stacheldraht [11]. For a detailed description of some of these tools and how to setup an attack network, see [29].

Most direct attacks use common Internet protocol packets, such as TCP, UDP, and ICMP. In the case of TCP, a popular attack is a SYN flooding attack [11][16] where a large number of TCP SYN packets are sent to a victim using a spoofed source IP address in the packet. If the victim is listening for connection requests, it will respond with a number of SYN-ACK packets and will then wait for a response. However, since the original TCP SYN packets have spoofed source IP addresses, these SYN-ACK packets will be sent elsewhere in the Internet and the victim will be left waiting (figure 2a). A large number of TCP SYN packets will create a large number of half-open connections on the victim, quickly consuming resources and preventing the victim from accepting legitimate connections or, in the worst case, crashing the victim. UDP and ICMP packets can also be used in a similar fashion to overwhelm the victim by causing the victim to send a large number of UDP or ICMP echo reply response packets to fake IP addresses.

2.2.2 Reflector Attacks

A reflector attack is one in which reflector nodes such as routers or legitimate servers are used to launch an attack. Reflector attacks are indirectly launched by an attacker and are good at hiding the identity of the attacker or increasing the distribution of an attack [30]. In a reflector attack, an attacker will send packets to the reflectors that require a response, such as TCP SYN packets (SYN-ACK response) or ICMP echo request packets (ICMP echo reply response). However, these packets have the victim's IP address as the source IP address, causing the reflectors to send the response packets to the victim (figure 2a). In this way, a large amount of traffic is unknowingly generated by the reflectors and is targeted at the victim, flooding the victim's link.

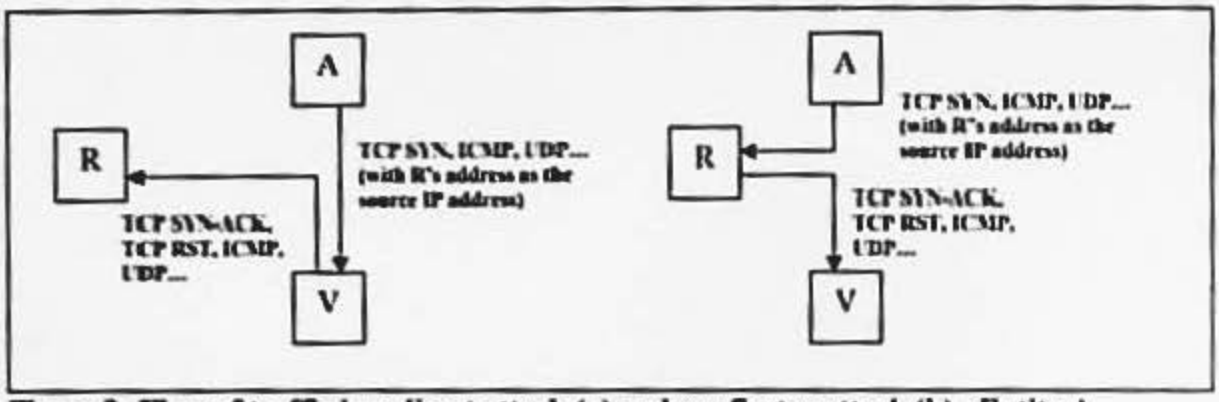


Figure 2: Flow of traffic in a direct attack (a) and a reflector attack (b). Entity A represents the attacker, V is the victim, R is the reflector (from [11])

Reflector attacks require the use of preselected reflectors and specific attack packets. Almost any protocol which employs an automatic response mechanism can be exploited in a reflector attack. In addition to TCP SYN packets and ICMP echo packets as mentioned above, ICMP error messages and DNS queries can be used. For example, attack packets with invalid destination ports would cause a reflector to send ICMP port unreachable messages to a victim. Also, DNS can be exploited to take advantage of bandwidth amplification where a small number of attack packets (DNS recursive queries) can trigger larger packets in response (DNS replies) [31]. Table 1 summarizes some common reflector attacks.

Because these reflected packets are generated by legitimate entities on the Internet, reflector attacks are more difficult to identify. Filtering based on spoofed IP addresses or route-based information becomes much less effective. In addition, reflector attacks can be more distributed than direct attacks since attackers do not need to compromise hosts. Instead, they can utilize the many legitimate reflectors across the Internet.

Type of Attack	Packets sent by an attacker to a reflector (with a victim's address as the source address)	Packets sent by the reflector to the victim in response
Smurf	ICMP echo queries to a subnet-directed broadcast address	ICMP echo replies
SYN flooding	TCP SYN packets to public TCP servers (e.g., web servers)	TCP SYN-ACK packets
RST flooding	TCP packets to nonlistening TCP ports	TCP RST packets
ICMP flooding	• ICMP queries (usually echo queries) • UDP packets to nonlistening UDP ports • IP packets with low TTL values	• ICMP replies (usually echo replies) • ICMP port unreachable messages • ICMP time exceeded messages
DNS reply flooding	DNS (recursive) queries to DNS servers	DNS replies (usually much larger than DNS queries)

Table 1: Examples of common DDoS reflector attacks (from [11])

2.3 Solutions to the DDoS Problem

As DDoS attacks become more prevalent and more costly, the need for solutions becomes vital. Generally speaking, there are three approaches towards the mitigation of DDoS attacks: attack prevention and preemption (prior to the attack), attack detection and filtering (during the attack), and attack source traceback and identification (after the attack) [11].

2.3.1 Attack Prevention and Preemption

Attack prevention and preemption is probably the most difficult of the three mitigation techniques. While this represents the first line of defense against DDoS attacks, the cost-benefit ratio is extremely high for users who implement these techniques. Attack prevention requires the cooperation of individual hosts to protect their systems from contributing to a DDoS attack. These individual hosts can use monitoring tools and attack signatures to scan for attack traffic or known attack messages sent between masters and zombies. They can also take the necessary steps to protect their systems from

becoming compromised and infected with a DDoS zombie process. However, most large ISPs and enterprise networks simply do not have the incentives to employ these difficult countermeasures [11]. Additionally, since the attack traffic is highly distributed amongst the sources, the amount of attack traffic is extremely low in relation to legitimate traffic. All of this combines to make attack prevention and preemption at the sources extremely tricky.

2.3.2 Attack Detection and Filtering

A second approach towards DDoS attack mitigation is attack detection and filtering. This approach attempts to identify an ongoing DDoS attack and then employs filters to block the attack traffic before it can reach the outside network.

A key consideration when dealing with attack detection is the placement of the detection and filtering mechanism in the network. Since DDoS attacks are highly distributed, the amount of attack traffic starts out relatively low at the source networks and becomes increasingly aggregated as it gets closer to the victim's network. Therefore, the closer the detection mechanism is to the victim's network, the "easier" it is to successfully detect an attack. In contrast, the closer the detection mechanism is to the source networks, the harder it is to detect an attack. In the latter case, the ratio of attack traffic to legitimate traffic is very low, making detection difficult compared to detection at the victim's network where the ratio is very high.

Unfortunately, the effectiveness of detection and filtering is also dependent on the location of the mechanism and is directly related to the difficulty of detection. By the

time an attack has been detected at the victim's network, the incoming bandwidth has already been clogged by the attack packets and the attackers have succeeded in preventing legitimate users from accessing the victim. The only real benefit to detection at the victim network is the prevention of a victim crash due to resource starvation (since attack packets are dropped before reaching the victim). Therefore, it would be much more effective to detect an attack near the source network and drop the attack packets before they enter the Internet, thereby saving Internet resources and preventing bandwidth starvation at the victim's network. However, as stated above, detection is more difficult as we move closer to the source networks. Therein lies the problem--as we move away from the victim's network, the effectiveness of detection and filtering increases, as does the difficulty. Figure 3 summarizes this relationship between detection effectiveness and difficulty.

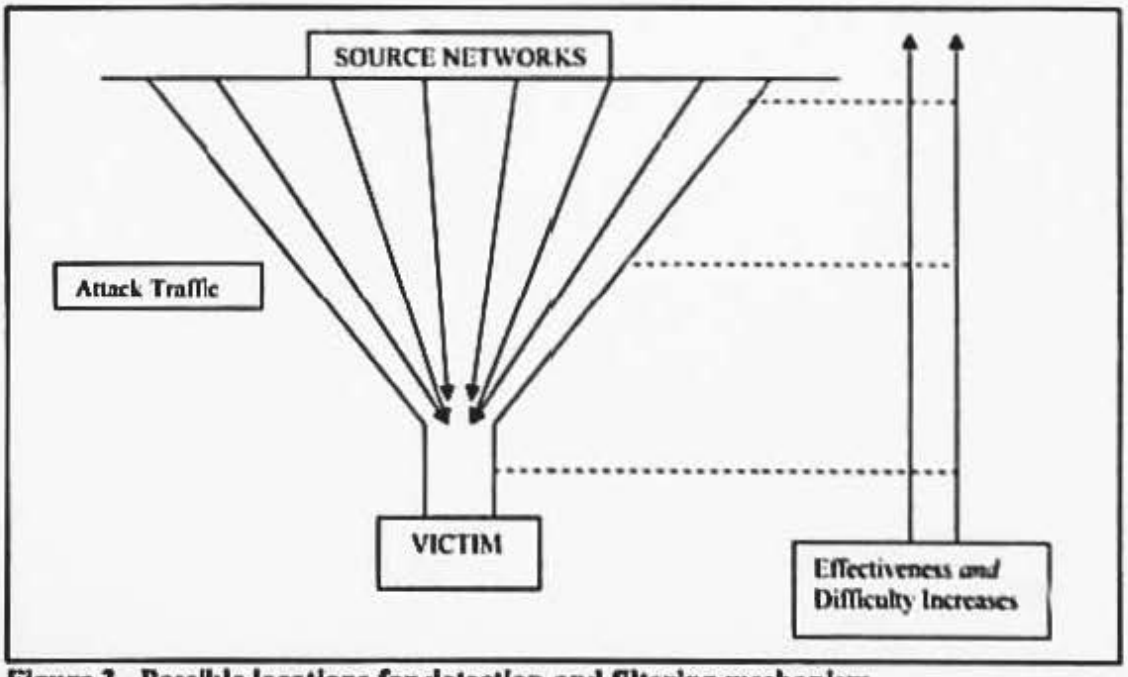


Figure 3. Possible locations for detection and filtering mechanism

As pointed out above, it becomes easier to detect DDoS attacks when the detection and filtering mechanism is located closer to the victim's network. As a result, a number of commercial products have been released which act as intrusion detection devices and are located at the firewall of the victim's server [33][34][35][36]. However, the methods which these devices employ are proprietary, and little information is publicly available. Also, they still do not address the problem of link bandwidth saturation. Therefore, most of the current research is focused on the difficult problem of detection and filtering closer to the source networks.

Most of the solutions fall into two broad categories. The first group contains the solutions that are based on the fact that many DDoS attacks utilize spoofed IP addresses in the attack packets, and a variety of methods are proposed to identify these malicious packets. The other group attempts to identify attack flows by monitoring traffic characteristics at various points in the network.

2.3.2.1 Spoofed IP Address Detection

One of the simplest methods to try and eliminate spoofed traffic is to perform network ingress filtering [17] at the source network's ingress router or at the ISP. This simply looks at the source IP address of all packets entering the Internet and only allows packets with source IP addresses that fall within the range of the source network to pass. This would effectively block most packets with randomly spoofed IP addresses. Of course, an attacker could always spoof IP addresses that fall within the range of the source network, but most unsophisticated attacks would be blocked. In addition, attacks that originate from compromised machines with valid IP addresses would pass unnoticed. While this

seems like a relatively simple solution to the DDoS problem, actual implementation is difficult. In many large-scale networks, the overhead of inspecting every outgoing packet for source address validity is infeasible. Widespread deployment would also require the cooperation from all ISPs and network administrators, who see little benefit in implementing such an expensive feature in their edge routers.

Other solutions that are attempt to identify spoofed traffic include iSAVE [22][19] and Source IP Address Monitoring (SIM) [12][13]. iSAVE attempts to detect spoofed IP addresses on routers by building tables of incoming interfaces and valid source IP addresses. The information contained in these tables are then shared between routers using iSAVE protocol messages and are used to filter out spoofed traffic. An implementation of the iSAVE protocol has been demonstrated on the Intel IXP and has been shown to minimally impact routing performance. SIM, on the other hand, uses off-line training to build IP Address Databases (IADs) which contain valid IP addresses seen at a network device. In the detection and learning phase, the amount of new IP addresses observed is modeled using a change detection algorithm called Cumulative Sum algorithm [16][37]. An attack is detected when the change-point variable crosses some threshold. SIM has been shown to effectively identify DDoS attacks at the source network using this scheme [13].

Park et. al. [15] take a different approach and attempt to eliminate spoofed IP addresses through the use of route-based distributed packet filtering (DPF). DPF uses routing information at network devices to determine if a packet arriving at a router is valid with

respect to its source and destination IP address fields. Park claims that if only 18% of autonomous systems implement DPF on their border routers, a "synergistic filtering effect is achieved whose collective filtering action proactively prevents spoofed IP flows from reaching other autonomous systems in the first place" [15].

Jin et. al. [23] also take a unique approach by filtering packets based on the TTL field (hop-count) in IP packets. The algorithm maintains a hash table containing address prefixes mapped to hop-counts and compares incoming IP packets against this table. Hop Count Filtering's (HCF) effectiveness relies on the assumption that attackers frequently choose some arbitrary value to place in the TTL field when generating attack packets, and these packets with "invalid" TTL fields can be easily identified. The authors have implemented the scheme in the Linux kernel and have shown a 90% success rate with "little collateral damage" [23].

2.3.2.2 Monitoring Traffic Characteristics

The second group of solutions uses traffic monitoring and traffic characteristics to identify DDoS attacks [38][39]. D-WARD [18][21] is detection mechanism that is placed at the source network and identifies attack flows by monitoring two-way traffic flows between the network and the rest of the Internet. It works by looking for signs of communication difficulties and by comparing observed traffic statistics with predefined models of normal traffic. When a possible DDoS attack is identified, D-WARD responds by rate-limiting the flow which can then confirm or refute the existence of an attack.

Gil et. al. [14] proposes a scheme called MULTOPS which utilizes a unique tree-like data structure to characterize traffic characteristics on a network device. It looks for inconsistencies between traffic going to and from specific IP addresses. Specifically, if a certain packet rate traveling to a host is higher than the returning packet rate, it is assumed that the destination host may be under attack and the outgoing traffic is then rate-limited. However, many flows on the Internet are asymmetric, which would increase the false positive rate of this scheme [12].

Wang et. al. [42] have proposed a mechanism to detect SYN floods at the source network called SYN-dog. The SYN-dog mechanism is located on a leaf router and monitors the difference between the number of SYN and SYN-ACK packets. Its effectiveness is based on the strong correlation between SYN and SYN-ACK packets in a normal TCP session. Under normal conditions, a SYN packet will be accompanied by a responding SYN-ACK during a TCP connection setup. However, during a SYN flooding attack, large numbers of SYN packets with spoofed IP addresses are sent to a victim. This prevents SYN-ACK packets from returning to the source network, and a discrepancy between SYN and SYN-ACK packets can be observed. SYN-dog views this difference between SYN and SYN-ACK packets as a "stationary, ergodic random process" [42], and models it as a Sequential Change Detection [46] problem. SYN-dog normalizes the difference using an estimated number of SYN-ACKs and then uses the non-parametric Cumulative Sum (CUSUM) method [47] to indicate possible attack-like traffic. For a detailed description of the SYN-dog algorithm, see [42].

Detecting DDoS attacks is a difficult task, especially when the goal is to detect attack traffic as close to the source as possible. Solutions must be easy to implement so that ISPs and network administrators will embrace enforcement while maintaining effective detection and low false-positive rates [40]. Research into detection and filtering is still being done, and novel solutions are still being proposed.

2.3.3 Attack Source Traceback and Identification

Attack source traceback, or IP traceback, is an after-the-fact solution to the DDoS problem. It attempts to identify, or trace, the origin of attack packets in order to take future action against the attacker (or compromised machine). Unfortunately, it is infeasible to use IP traceback to stop an ongoing DDoS attack [11]. IP traceback solutions cannot always trace packets beyond a firewall or a NAT proxy, and it is useless against DDoS reflector attacks since the packets are coming from legitimate network devices on the Internet. Nevertheless, IP traceback could be useful in some cases to identify certain attackers, allowing law enforcement agencies to take action. For a detailed survey on the state of IP traceback, see [41].

2.4 Network Processors

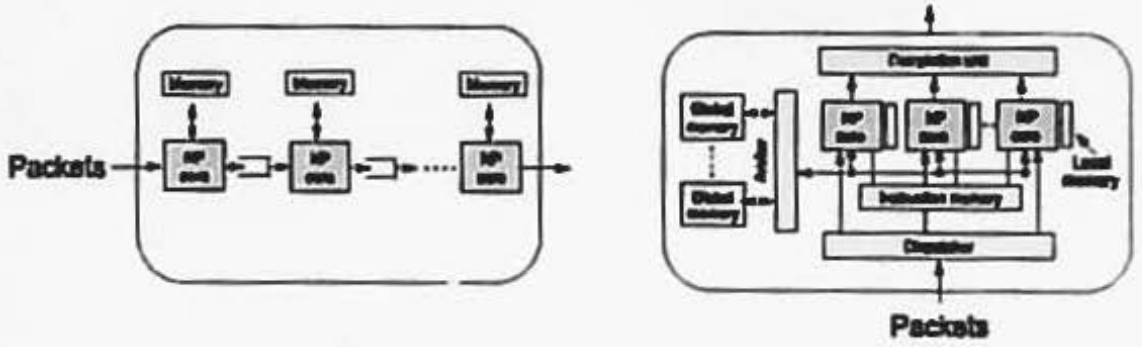
In any network infrastructure, routers play a key role by providing line-speed packet processing while supporting certain features including IP forwarding, quality of service (QoS), and virtual private networks. Many of these features go beyond a simple routing table lookup and require deep, stateful packet processing. The current trend in next-generation networks is to migrate these features towards the edges of service provider networks and away from the core routers, allowing the core to efficiently handle the high-

speed switching of large traffic aggregates. As a result, there is a need for flexibility at the edge routers as well as high performance to handle packets at line-speed. Network processors (NPs) have been developed to address this issue.

Traditionally, hardware-based design has focused on the use of Application-Specific Integrated Circuits (ASICs) to perform most of the processing load. However, the long hardware development cycle and the lack of reprogrammability of these ASICs soon became an issue, especially with the need for flexibility in the rapidly evolving network equipment market [1]. Offloading the processing load to a general-purpose processor (GPP) gave designers plenty of flexibility, but the performance of these GPP's was inadequate. Accordingly, network processors were developed to provide the flexibility of a programmable architecture with performance approaching that of a hardware-based solution.

Typically, network processors consist of a set of programmable processors that are optimized for packet processing. They usually have specific instruction sets which allow for efficient packet handling, along with specialized coprocessors to offload common processing tasks such as table lookups or checksum computations. Because network processors must operate under strict performance requirements, the entire design of the network processor (execution environment, memory, hardware accelerators, bus architecture, etc.) has been optimized for high-speed packet processing.

Network processor architectures fall into one of two models: serial or parallel. In the serial model (figure 4a), each core executes a small segment of the data-path code on every packet before sending it along to the next core for further processing. In the parallel model (figure 4b), a number of network processor cores are present within the NP, and each core executes the entire data-path code on a single packet. The Intel IXP architecture [49] is based on the serial (pipelined) architecture and the IBM PowerNP [6] is based on the parallel model.



(a) Serial model (b) Parallel model

Figure 4: Network processor architectures (from [11])

2.5 Current Research on Network Processors

Most of the network processor research today is being done using the Intel IXP architecture, specifically the Intel IXP 1200 [10]. Most of this research explores the performance and feasibility of porting existing software-based router functionality (QoS, DiffServ, encryption/decryption, VPN, etc.) onto a network processor. For example, Lin et. al. [7] has implemented DiffServ on the IXP 1200 with the aim of identifying bottlenecks within the IXP architecture and the DiffServ algorithm. The authors found that their implementation could support Per-Hop Forwarding Behavior (PHB) marking in

Diffserv at an aggregated throughput of 568kbps, with the main bottleneck being the shared memory between the microengines (DRAM).

Burns et. al. [8] implements address assurance on the IXP 1200 using the iSAVE protocol [22] and shows that the router suffers little to no performance degradation due to the added functionality. Spalink et. al. [3][5] provides an in-depth analysis of basic IP forwarding functionality on the IXP 1200 and concludes that it can easily sustain line speeds for 8 x 100 Mbps Ethernet ports, leaving ample processing cycles for additional packet processing. Similar to Lin et. al. [7], the performance bottleneck was in the DRAM, and the authors estimate that a 26% improvement in line-speed processing is possible with faster memory.

Haas et. al. [1] discusses the suitability of network processors for "quality-of-service (active queue management and traffic engineering), header processing (GPRS tunneling protocol), intelligent forwarding (load-balancing without flow disruption), payload processing (active networks code interpretation and just-in-time compilation), and protocol stack termination (SCTP)" [1]. Still other areas of research include QoS [9], active networks [10], packet classification [10], and various layer-4 applications such as web caching, load-balancing, and media streaming [10].

CHAPTER 3: RATIO-BASED SYN FLOOD DETECTION

This section will provide an explanation of SYN flooding, a common DDoS attack, and will also detail a proposed solution, dubbed ratio-based SYN flood detection (RSD). It will also introduce a methodology for optimizing RSD based on traffic patterns of specific networks.

3.1 SYN Flooding Attacks

A SYN flooding attack is so named due to its usage of TCP SYN packets as attack traffic. SYN flooding attacks fall into the flooding-based attack category and have become a popular weapon for DDoS attackers. In fact, it has been shown that more than 90% of DDoS attacks use TCP, and of these, SYN floods are the most prevalent type of attack [44].

SYN flooding attacks take advantage of TCP's three-way handshake mechanism in order to exhaust a victim's resources. In a normal TCP session, a server will listen for incoming SYN requests on a well-known port. Once a connection request comes in from a client, the server will respond with a SYN-ACK packet and will establish a half-open connection. Once the server receives the ACK from the client, the three-way handshake is complete and the session can proceed (Figure 5).

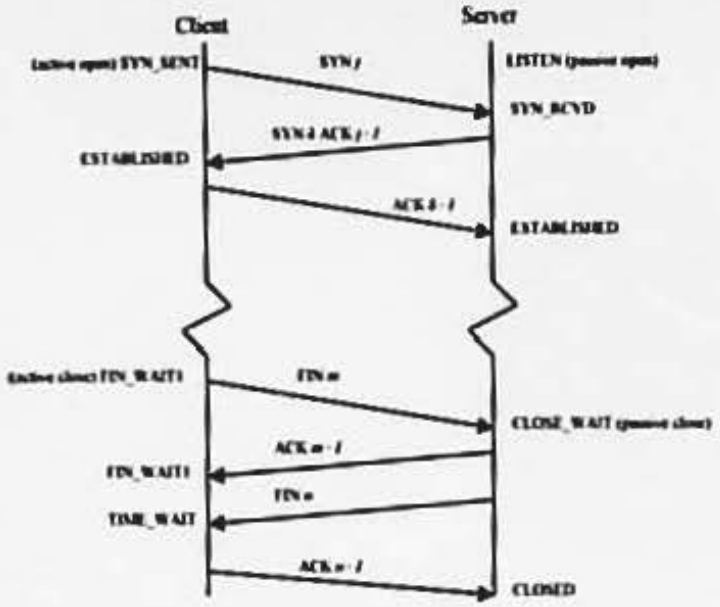


Figure 5: TCP states corresponding to normal connection establishment and teardown (from [45])

In a SYN flooding attack, a large number of spoofed SYN packets with false source IP addresses are sent to the victim's TCP listening port. For each SYN packet, the victim will respond with SYN-ACK packet and will establish a half-open connection. However, since the source IP address was spoofed, these SYN-ACK packets will be sent to the spoofed IP address and will (presumably) be lost somewhere in the Internet. The victim, meanwhile, will continue to keep the half-open connection active for a period up to the TCP connection timeout and will not close the connection until two more SYN-ACK retransmissions have failed. This period of time typically lasts 75 seconds [42]. Since the victim only has a limited amount of memory to store these half-open connections, a large number of spoofed SYN packets will quickly fill up the queue and will prevent the victim from servicing legitimate connection requests.

3.2 Ratio-based SYN Flood Detection (RSD)

Similar to the SYN-dog solution proposed by Wang et. al. [42], ratio-based SYN flood detection (RSD) aims to detect SYN floods at the source networks by monitoring the discrepancy between SYN and SYN-ACK packets at the edge router connecting a stub network to the Internet. However, RSD uses an adaptive threshold based algorithm to detect an attack instead of a cumulative sum algorithm, as in [42]. The success of RSD is dependent on two key assumptions regarding a typical SYN flooding attack:

- The SYN packets have randomly spoofed source IP addresses.
- The randomly spoofed source IP addresses do not fall within the source network subnet.

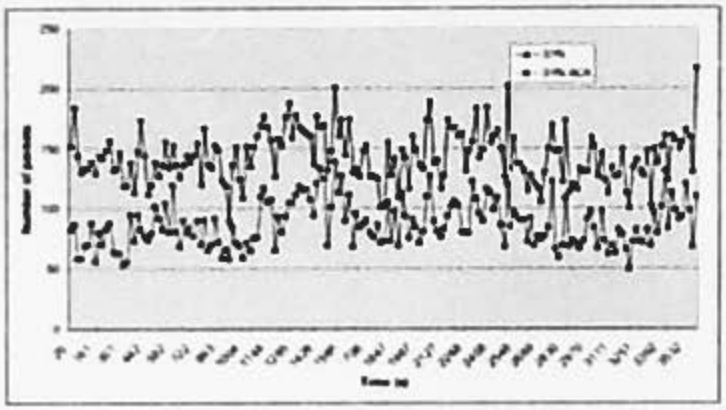
Under normal traffic conditions, outgoing SYN packets will be accompanied by a responding SYN-ACK packet within one round-trip time (Figure 5). However, if the SYN packets have spoofed source IP addresses, the victim server will respond to the spoofed address, and the RSD mechanism will never see these packets. Therefore, an inconsistency between the number of SYN packets and the number of SYN-ACK packets would indicate a possible attack. Unfortunately, under normal traffic conditions there is no perfect one-to-one correspondence between these packets. The most common reasons for this discrepancy are as follows [42].

- The TCP server receiving the SYN request is overloaded and is unable to reply with a SYN-ACK packet.

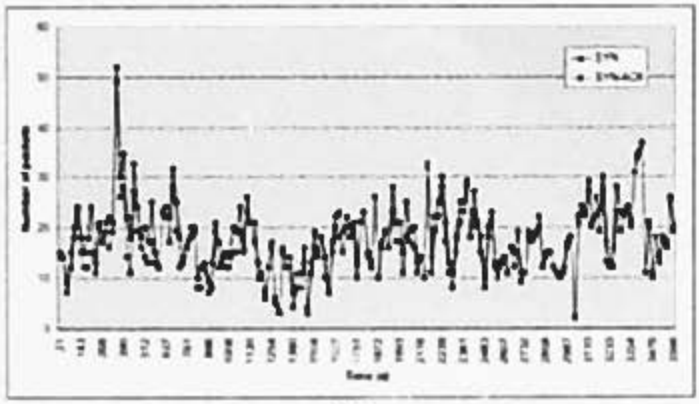
- The link between the client and server is congested, causing the SYN request to be dropped. Since the TCP server never receives the SYN request, no SYN-ACK packet is generated.

As a result, RSD cannot detect a SYN flooding attack simply when a difference between the number of SYN and SYN-ACK packets is observed. Instead, RSD monitors the *ratio* of SYN to SYN-ACK packets and signals an attack when an abrupt change of this variable is observed, using an adaptive threshold method based on previous ratio measurements. Similarly, SYN-dog [42] monitors the normalized difference between SYN and SYN-ACK packets and uses a non-parametric Cumulative Sum (CUSUM) method [47] to indicate an attack.

Under typical conditions, the number of SYN and SYN-ACK packets is strongly correlated, independent of sample time, site, and time-of-day. Traffic traces gathered at a stub network's connection to the Internet were obtained and analyzed, with the results shown in figure 6. Figure 6a represents a one hour trace of all traffic between the Digital Equipment Corporation and the Internet on March 8, 1995. Figure 6b represents a one hour trace of all traffic between the Lawrence Berkeley Laboratory and the Internet on January 21, 1994. Note that there is a strong correlation between SYN and SYN-ACK packets, but there is no one-to-one relationship.



(a)



(b)

Figure 6: SYN and SYN-ACK packet counts at DEC(a) and LBL(b)

During an attack, spoofed SYN packets will be observed at the leaf router but the number of SYN-ACK packets will remain relatively unchanged, since the victim will be sending the responding SYN-ACK packets to the spoofed IP address. As a result, the difference between SYN and SYN-ACK packets will increase, indicating a possible attack. However, we cannot infer an attack by simply using the difference as our measurement. Internet traffic will vary depending on a number of factors, including time of day and type of users, and a normal increase in traffic load may also lead to an increase in the difference between SYN and SYN-ACK packets. For example, assume one out of every three TCP connections requires a retransmission of the beginning SYN packet due to one

of the two factors mentioned earlier. If an edge router sees 30 new TCP connections every minute, then the router will see 10 more SYN packets than SYN-ACK packets every minute. However, if an edge router experiences a burst of high traffic and there are 60 new TCP connections every minute, then the difference between SYN and SYN-ACK packets will be 20. Obviously, the increase in the difference does not indicate an attack, but is a direct result of varying traffic patterns.

In order to make attack detection less sensitive to varying traffic patters, RSD monitors the *ratio* between SYN and SYN-ACK packets at the edge router. The algorithm for detecting an attack is relatively straightforward and is a type of adaptive threshold algorithm. It tests whether a given traffic measurement, the ratio of SYN to SYN-ACK packets in our case, exceeds a set threshold in a certain time interval. The threshold value is calculated using recent traffic measurements to make the algorithm applicable to differing traffic conditions.

3.3 RSD Details

Let

n = some time interval Δt

and let

R_n = *#SYN-ACK packets in interval n / #SYN packets in interval n.*

Then $R_n < 1$ and should be close to 1 due to the correlation of SYN and SYN-ACK packets. When an attack occurs, the number of SYN packets will increase while the number of SYN-ACK packets will stay the same, resulting in a decrease of R_n. During

the detection phase, RSD calculates R_n for every time interval Δt and signals an attack when

$$R_n \leq (a * O_{n-1})$$

Where a is some value between 0 and 1 and is our threshold variable and O_{n-1} is an approximate mean value of the past $n-1$ measurements of R. O_n is calculated using an exponentially weighted moving average function

$$O_n = \beta(O_{n-1}) + (1-\beta)R_n$$

where $0 < \beta < 1$ is the exponentially weighted moving average factor.

The tunable variables in this algorithm include a, β, and Δt. The following sections discuss the effects of altering these variables.

3.3.1 Varying a

a represents the threshold level for indicating a DDoS attack. By increasing a, RSD becomes more sensitive to small fluctuations in R_n and results in a higher probability of attack detection along with a higher number of false positives. By decreasing a, RSD becomes more immune to false positives but only signals attacks if the SYN flood rate is very high, thereby decreasing its effectiveness. Through initial trial-and-error, a is set to 0.7 in RSD in an attempt to balance a low false positive rate with a high degree of effectiveness.

3.3.2 Varying β

β represents the exponentially weighted moving average factor and determines how much weight is given to past measurements compared to the current measurement when calculating the moving average. A high value of β smoothes out the moving average O_n and makes it less susceptible to abnormal measurements of R_n. A low value of β will put more weight on current measurements and will adjust O_n quicker in response to large changes in R_n. Since internet traffic is typically very bursty on a short timescale and slowly varying on a long timescale, β is initially set to 0.9 in RSD.

3.3.3 Varying Δt

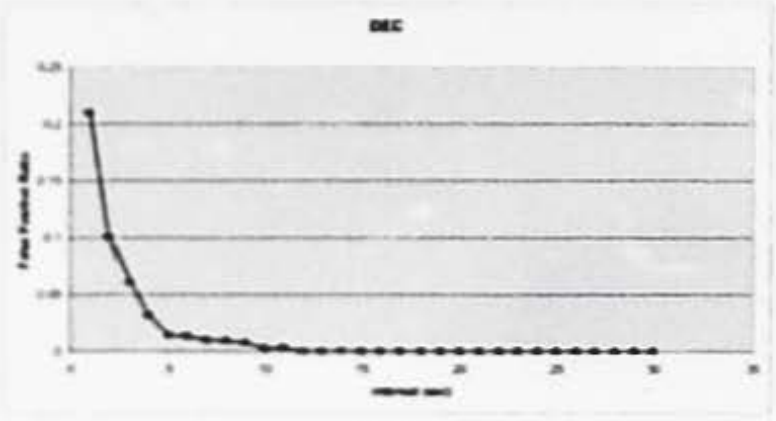
Δt represents the time interval (in seconds) that RSD will use when detecting SYN flooding attacks. It also represents the minimal attack detection time, as RSD can only flag an attack after each time interval is complete. As such, determining an optimal value for Δt is very important for good performance of RSD. If Δt is too small, RSD will not see a significant number of SYN and SYN-ACK packets during each interval, resulting in a high false positive ratio (FPR, or *#false positives in n intervals / n*) and poor performance. As Δt increases, the FPR decreases at the expense of attack detection time. Ideally, we would like for RSD to be able to detect SYN flooding attacks in the shortest time possible with a FPR of 0, but clearly a tradeoff exists between these two goals. The next section will explore this tradeoff in greater detail.

3.3.4 RSD Training Module

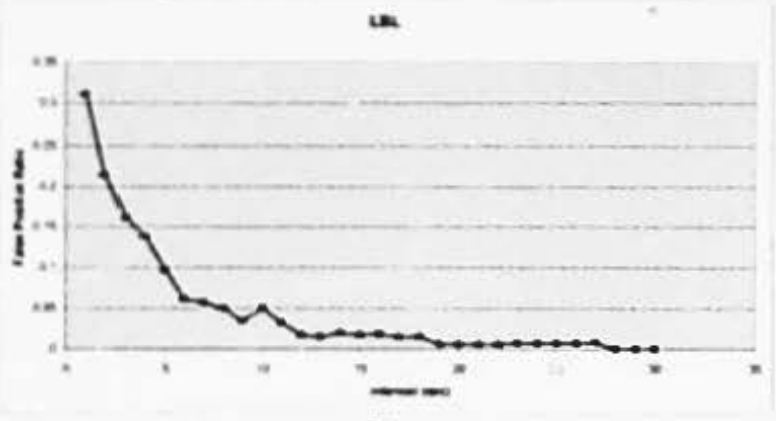
An optimal value of Δt will result in quick attack detection time with the lowest false positive rate (FPR) possible. Determining a good value for Δt is also very dependent on

the traffic patterns, even more so than α and β. For instance, a good Δt for a large stub network may result in extremely poor performance in a smaller stub network, as R_n can vary wildly when Δt is too small to accumulate an adequate number of SYN and SYN-ACK packets in the time interval. This same problem can occur on the same stub network when traffic patterns vary due to time of day fluctuations. As a result, a training module is needed to customize Δt for successful application in a specific network environment. This training phase is done offline and will result in an optimal Δt.

Offline training is done with a typical traffic trace obtained at the edge router where RSD will be deployed. It is assumed that this trace does not contain any SYN flood attacks. This is important, as it implies that any attacks caught by RSD can be considered false positives. Through simulation, different values of Δt can be set in RSD and the algorithm can be applied multiple times to the traffic trace, each time calculating the false positive ratio (FPR). Based on these simulations, an optimal value of Δt is then chosen. The results of applying RSD on the DEC and LBL traces with Δt varying between 1 and 30 seconds is shown in figure 7 below.



(a)



(b)

Figure 7: Effects of varying Δt at DEC(a) and LBL(b)

Clearly, there is an exponential relationship between the interval (Δt) and the FPR. Since we are interested in a low FPR with minimal detection time, we can see that a good Δt value is 10 seconds for the DEC network and 20 for the LBL network. The discrepancy between these two values can be confirmed by observing figure 6 and noting the difference in SYN and SYN-ACK traffic. The DEC trace has much more SYN and SYN-ACK traffic, and will therefore show less variation of R_n as smaller intervals are used, resulting in a lower FPR for the same value of Δt as in the LBL trace.

3.3.5 Dynamically Adjusting Δt

A key feature of RSD is the ability to adapt to changing traffic conditions. This is achieved by varying Δt dynamically, in order to achieve our goal of fast attack detection (low Δt) with a low FPR. As mentioned earlier, traffic conditions may fluctuate due to factors such as time of day or type of user, and as a result Δt may become less than optimal. To achieve adaptability, RSD first calculates a new value, $\#SA$, for every different Δt used in the training phase. $\#SA$ is the exponentially weighted moving average of the number of SYN-ACK packets in each Δt. Intuitively, it represents the typical amount of SYN-ACK traffic in each Δt. Figure 8 shows $\#SA$ values when Δt is varied from 1 to 30 seconds on the DEC trace. As expected, when we increase Δt, more SYN-ACK packets are observed per time interval.

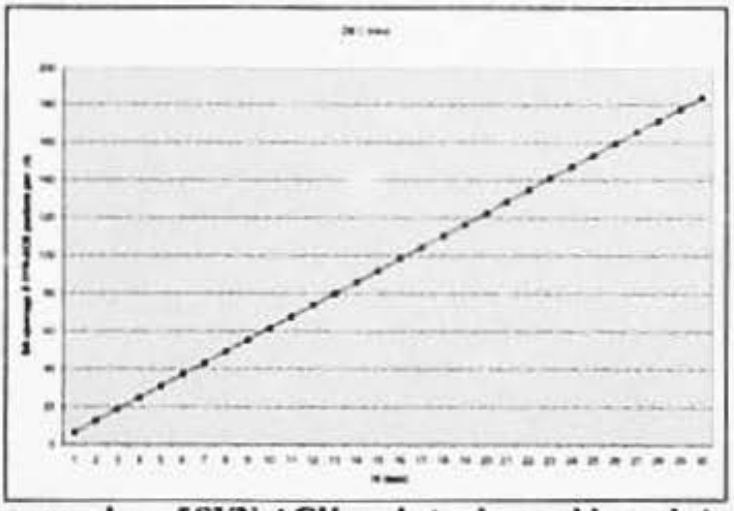


Figure 8: Average number of SYN-ACK packets observed in each Δt for DEC trace

After an optimal Δt is chosen (10 seconds in the case of the DEC trace), RSD saves the corresponding $\#SA$ value and stores it into a new variable, SA_{syn}. For the DEC trace, SA_{syn} = 61 SYN-ACK packets. It is important to remember that SA_{syn} is set after the training phase is completed and is not changed during normal operation of RSD.

During the detection phase, RSD compares SA_{opt} to the observed number of SYN-ACK packets (#SA) in each Δt. The following pseudo code illustrates the behavior of RSD during each time interval:

If $\#SA < SA_{opt}$ then

    Increase Δt by $2x$ seconds

Else if $\#SA > SA_{opt}$ then

    Decrease Δt by x seconds

Where x can be varied depending on how quickly we would like RSD to adapt to changing traffic conditions. x is set to 1 second in this implementation in order to keep Δt stable over the long run.

Using this simple test at each time interval allows RSD to adjust Δt dynamically in order to keep it close to optimal. For example, if a reduction in traffic is observed by RSD (indicated by a reduction in the number of SYN-ACK packets), then RSD will increase Δt in order to increase the number of SYN-ACK packets measured per time interval. This will keep the FPR from becoming too large. Concurrently, if an increase in traffic is observed then RSD will decrease Δt in order to keep our detection time as small as possible.

To show the effect of varying traffic conditions on the FPR if we were to use a static value of Δt, another trace collected at the DEC site was run through the training phase. This trace, labeled DEC-2, was gathered four hours prior to the original DEC trace (labeled DEC-1). Figure 9 shows the effects of varying Δt on both the DEC-1 and DEC-2 traces.

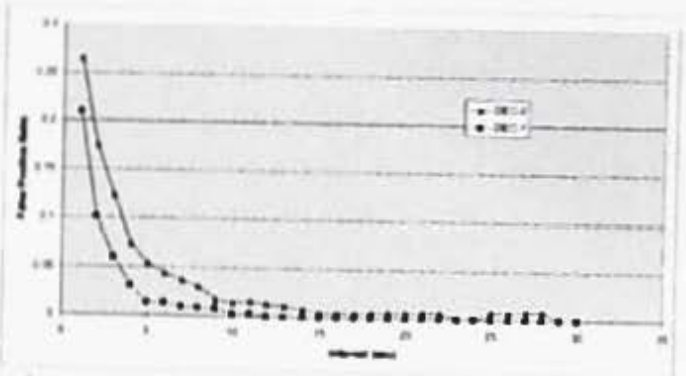


Figure 9: Effects of varying Δt on DEC-1 and DEC-2

From figure 9, we can see that the optimal Δt for DEC-2 will be longer than the optimal Δt for DEC-1. This is due to the lower amount of SYN-ACK traffic in the DEC-2 trace, as shown in figure 10.

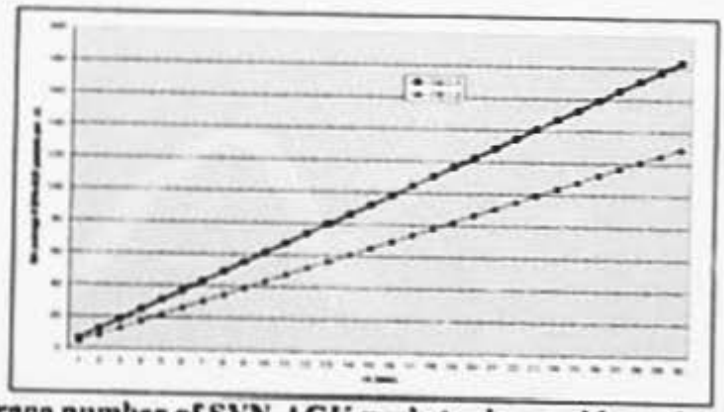


Figure 10: Average number of SYN-ACK packets observed in each Δt for DEC-1 and DEC-2

value of Δt, another trace collected at the DEC site was run through the training phase. This trace, labeled DEC-2, was gathered four hours prior to the original DEC trace (labeled DEC-1). Figure 9 shows the effects of varying Δt on both the DEC-1 and DEC-2 traces.

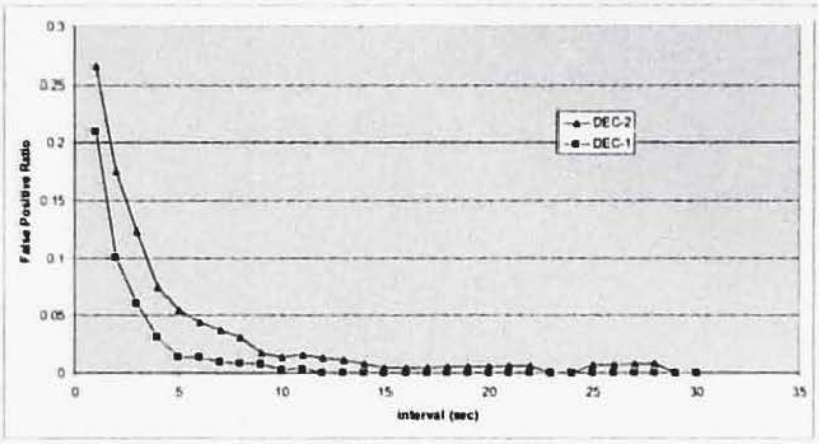


Figure 9: Effects of varying Δt on DEC-1 and DEC-2

From figure 9, we can see that the optimal Δt for DEC-2 will be longer than the optimal Δt for DEC-1. This is due to the lower amount of SYN-ACK traffic in the DEC-2 trace, as shown in figure 10.

From figure 9, we can see that the optimal Δt for DEC-2 will be longer than the optimal Δt for DEC-1. This is due to the lower amount of SYN-ACK traffic in the DEC-2 trace, as shown in figure 10.

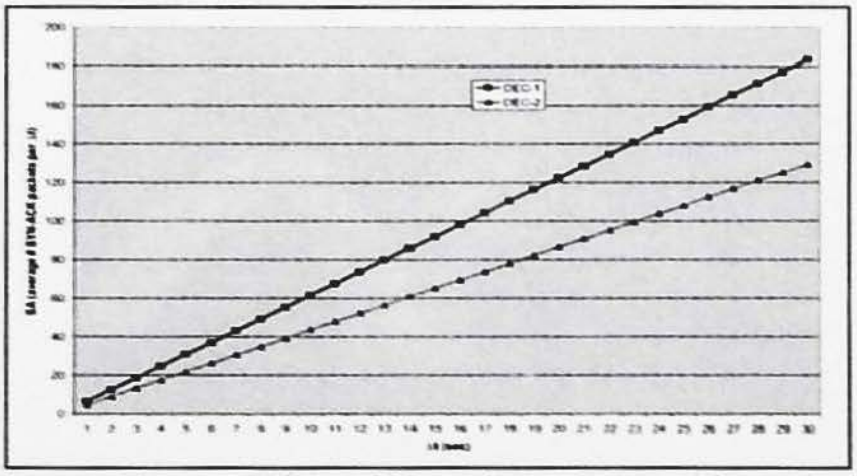


Figure 10: Average number of SYN-ACK packets observed in each Δt for DEC-1 and DEC-2

Clearly, if we have trained RSD on the DEC-1 trace, then a static value of Δt (10 seconds) will result in a higher FPR when traffic conditions such as in DEC-2 are present. Therefore, in the case of decreasing traffic load (i.e. DEC-1 to DEC-2), RSD will increase the time interval in order to ensure a low FPR. In contrast, when the traffic load is increasing (i.e. DEC-2 to DEC-1), RSD will decrease the time interval, which will result in a lower attack detection time. Since maintaining a low FPR is more important than ensuring the lowest possible detection time, RSD will increase Δt by $2x$ instead of x. This will keep Δt slightly larger than necessary in the long run.

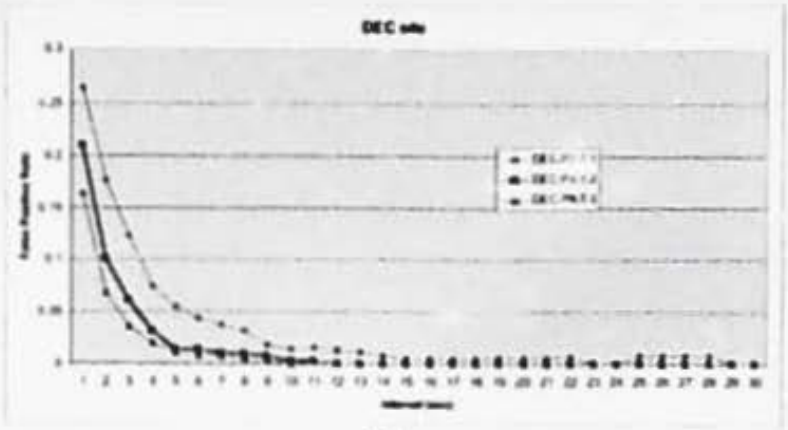
CHAPTER 4: RSD IMPLEMENTATION AND RESULTS

In order to investigate the efficacy of RSD, a number of trace-driven simulations were performed. Traffic traces were obtained from the link connecting two sites to the Internet: the Digital Equipment Corporation and the Lawrence Berkeley Laboratory [48]. Table 2 summarizes the pertinent information regarding the collected traces.

TRACE NAME	LOCATION	DURATION (min)	DATE COLLECTED	START TIME
DEC-PKT-1	DEC	60	March 8, 1995	22:00
DEC-PKT-2	DEC	60	March 9, 1995	02:00
DEC-PKT-3	DEC	60	March 9, 1995	10:00
LBL-PKT-4	LBL	60	January 21, 1994	14:00
LBL-PKT-5	LBL	60	January 28, 1994	14:00

Table 2: Summary of traffic traces used in simulations (from [48])

Following the methodology presented in the previous section, the RSD training phase was performed for each of the individual traces to determine optimal Δt and $S\Delta t_{opt}$ values. Figure 11 shows the results of this training phase, and table 3 shows the resulting optimal values chosen for each respective trace.



(a)

(b)

Figure 11: Results from RSD training phase on DEC (a) and LBL (b) traces

TRACE NAME	OPTIMAL Δt (sec)	$S\text{-}t_{opt}$ (packets per Δt)
DEC-PKT-1	23	99
DEC-PKT-2	15	91
DEC-PKT-3	10	102
LBL-PKT-4	30	25
LBL-PKT-5	45	28

Table 3: Optimal values for RSD for different traces

Table 3 illustrates the importance of the training phase for optimal performance of RSD. Initial values for Δt can range from very short intervals, as in the DEC-PKT-3 trace (10 sec), to long intervals, as in the LBL-PKT-5 (45 sec). Clearly, a static value for Δt will not provide optimal performance for varying traffic conditions and may result in a high FPR or reduced attack detection time, depending on conditions.

It is also important to note the correlation between optimal values of SA_{opt} within each site. Recall that RSD will dynamically vary Δt based on the number of SYN-ACK packets observed per time interval (#*SA*). In effect, RSD is adjusting Δt in order to keep #*SA* approximately close to SA_{opt}. The correlation between SA_{opt} values ({99,91,102} for the DEC site; {25,28} for the LBL site) shows that RSD will provide optimal detection even in the face of varying traffic conditions. For example, assume that RSD is trained on DEC-PKT-1 and begins the detection phase with Δt = 23 and SA_{opt} = 99. If traffic conditions change to those similar in DEC-PKT-2, suddenly Δt becomes less than optimal. In this case, RSD will see that #*SA* > SA_{opt}, and will decrease Δt until #*SA* approaches SA_{opt}, resulting in an optimal Δt (which is 15 in this case).

To verify this observation, the RSD training phase was run on DEC-PKT-1 and the variables were used when running simulations on DEC-PKT-2 (Δt = 23 and SA_{opt} = 99). As expected, RSD dynamically adjusted Δt by reducing it since a Δt value of 23 resulted in ##*SA* being greater than SA_{opt}. Δt was observed to approach its optimal value of 15 seconds in approximately 10 intervals. The correlation between SA_{opt} values within the DEC site ensured that the dynamic behavior of RSD kept Δt optimal.

Once the training phase was complete, simulated attacks were inserted into the traces and the RSD algorithm's attack detection capability was explored. All attacks began 10 minutes into the trace and lasted for 10 minutes. According to [44], this is the typical

attack duration observed in the Internet. The SYN flooding rate was varied from 1 to 20 SYN attack packets per second and these traces were run through the RSD algorithm.

4.1 DEC site results

Figure 12 below shows the results of varying the SYN flooding rate in the DEC traces and the ability of the RSD algorithm to correctly identify an attack condition. The x-axis represents differing SYN attack flooding rates (1-20 SYN packets per second). The y-axis represents the ratio of detected attack intervals over the total number of attack intervals. Recall that all simulated attacks last for 10 minutes, thus in the case of DEC-PKT-1 where Δt is initially set to 23 seconds, there will be approximately 26 attack intervals. It is clear that an increase in the SYN flooding rate results in RSD correctly identifying a higher percentage of the intervals which contain an attack. For example, in the case of DEC-PKT-1, RSD was able to detect 100% of the attack intervals (26 out of 26 intervals) when the SYN flood rate reached 6 SYN packets per second. In contrast, RSD was unable to detect an attack (0 out of 60 intervals) when a rate of less than 10 SYN packets per second was inserted into DEC-PKT-3.

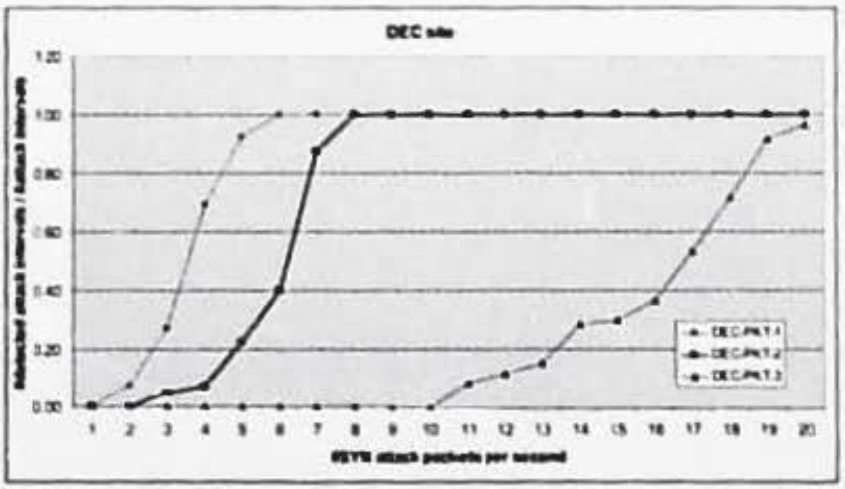


Figure 12: Results from RSD simulations (DEC site)

the case of DEC-PKT-1, RSD was able to detect the attack (1 out of 26 intervals) when the SYN flood rate reached 6 SYN packets per second. In contrast, RSD was unable to detect an attack (0 out of 60 intervals) when a rate of less than 10 SYN packets per second was inserted into DEC-PKT-3.

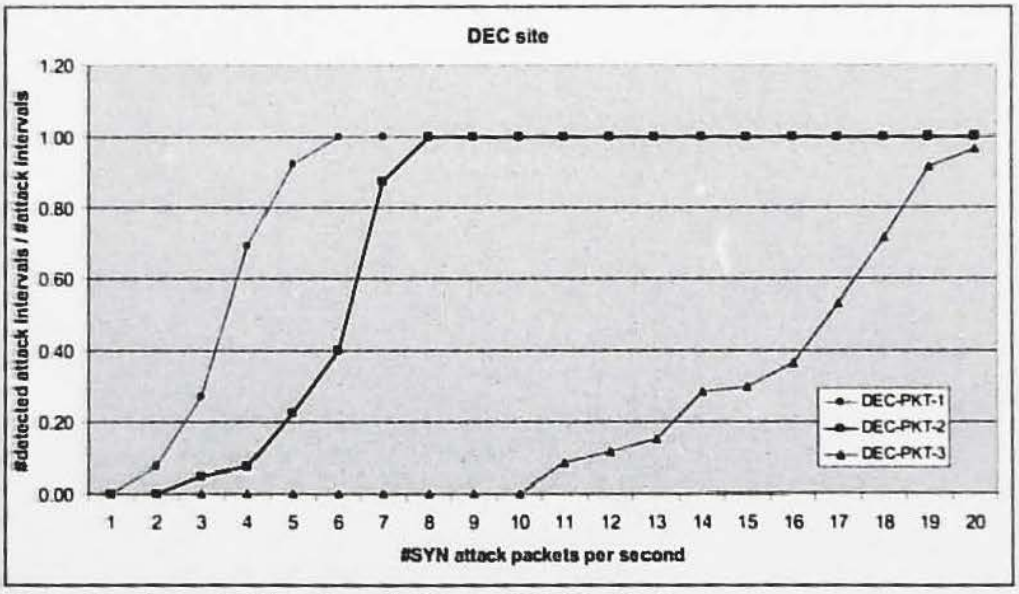
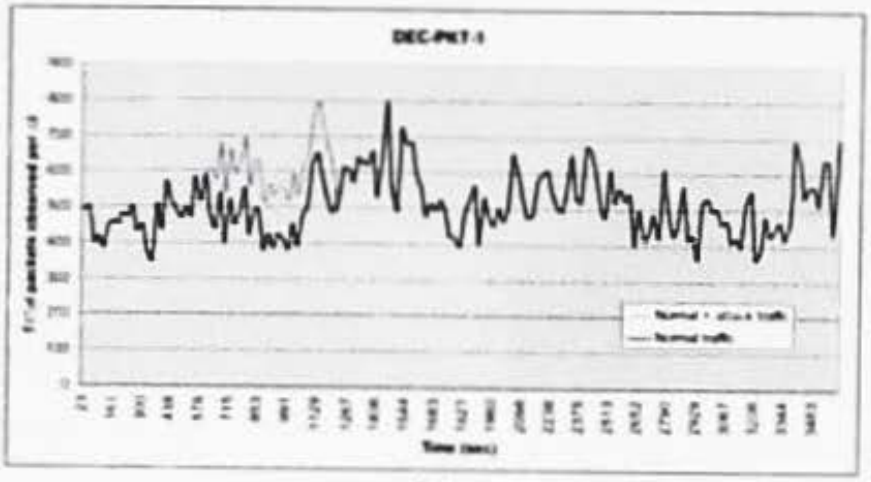
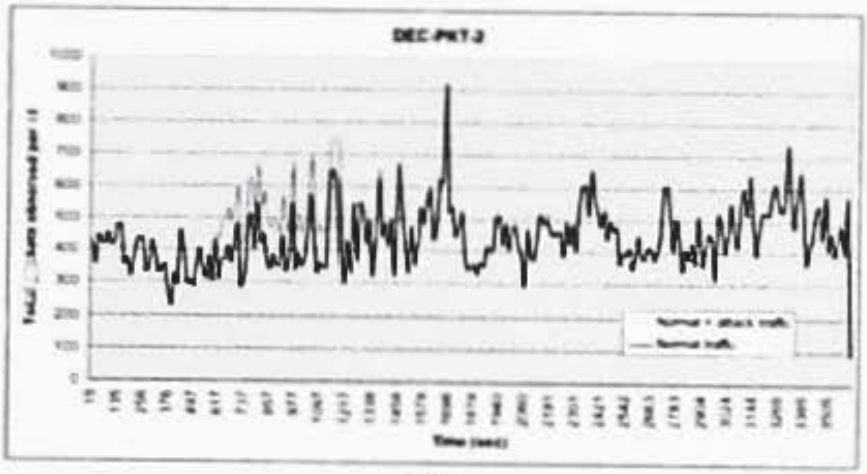


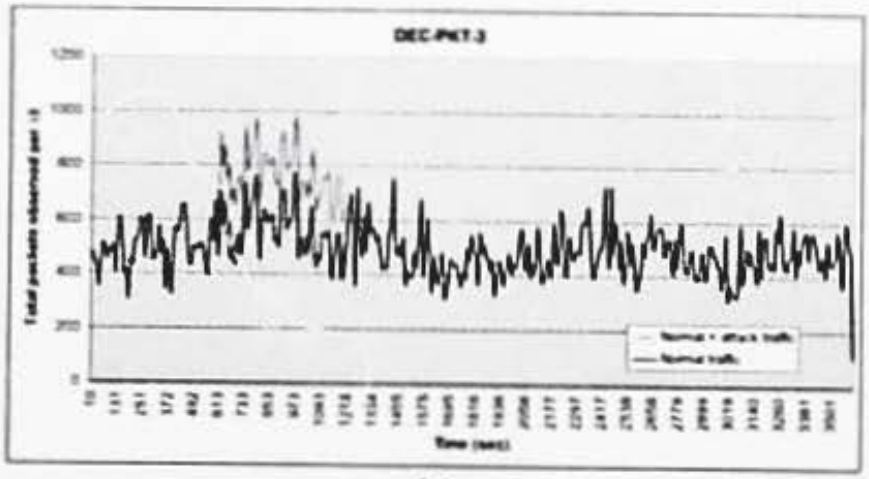
Figure 12: Results from RSD simulations (DEC site)



(a)



(b)



(c)

Figure 13: Traffic traces with and without attack traffic for DEC-PKT-1 (a), DEC-PKT-2 (b), and DEC-PKT-3 (c)

Figure 13 above shows the difference between the normal traffic traces and the traces with attack traffic added. Note that the attack begins 10 minutes into the trace and lasts for 10 minutes.

Interestingly, in the case of DEC-PKT-3, RSD detects 100% of a SYN flood when the rate is greater than 20 SYN packets per second. This is much higher compared to DEC-PKT-1, where RSD detected 100% of a SYN flood when the rate was 6 SYN packets per second. This is due to the differing traffic conditions present in the two traces. A SYN flood is harder to detect when there is a greater amount of background traffic since it is easier for the attack packets to blend in with normal traffic. In this case, successful detection of a SYN flood requires a higher rate of attack packets compared to a low traffic scenario. Figure 14 confirms this observation by presenting a comparison between the traffic rates at the DEC site. It shows that the DEC-PKT-3 trace has much more traffic than the DEC-PKT-1 trace, which explains why RSD can pick up attacks with lower SYN flood rates in DEC-PKT-1.

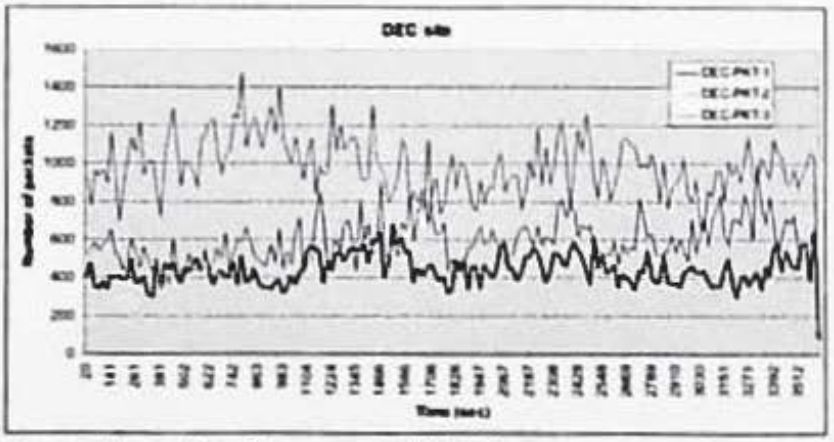


Figure 14: Comparison of traffic rates at DEC site

4.2 LBL site results

Results from simulated runs of RSD on the LBL traces are shown in Figure 15. From this figure, we can see that RSD can detect SYN floods with attack rates as low as 1 SYN packet per second. As explained in the previous section, this is a result of a low traffic rate at the edge router. In this case, the LBL traces contain very little traffic, which results in easy detection of very low-rate SYN floods. Figure 16 shows a comparison of traffic rates at the LBL and DEC sites, which verifies the correlation between low traffic rates and low minimum SYN flood detection.

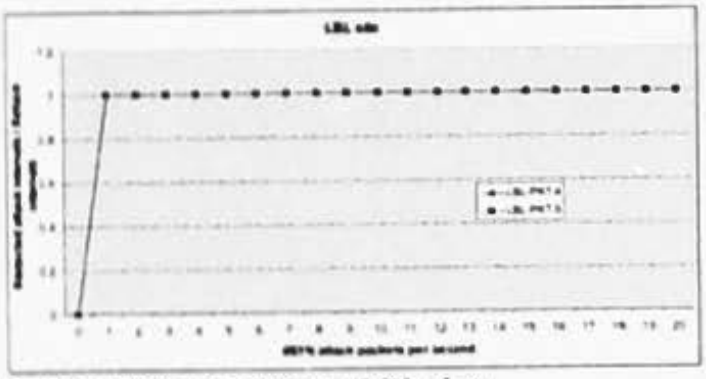


Figure 15: Results from RSD simulations (LBL site)

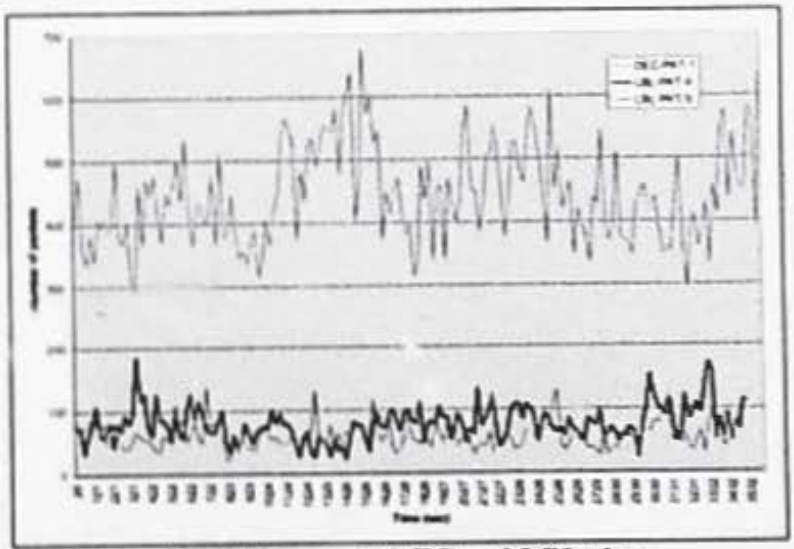


Figure 16: Comparison of traffic rates at DEC and LBL sites

4.3 Discussion

The previous sections have shown how a typical implementation of RSD will perform using trace-driven simulations. Data gathered from these simulations has shown the importance of optimizing Δt through the use of an offline training phase. This enables RSD to quickly detect an attack without incurring a high false positive rate.

The presence of varying traffic rates was also shown to have one of two effects on the performance of RSD: an increase in the FPR or a suboptimal (too large) minimal detection time. These effects are mitigated in RSD by dynamically adapting Δt based on current traffic conditions. This ensures optimal performance of the detection engine in the presence of long-term variations in traffic patterns.

Finally, the effectiveness of RSD was shown through the use of simulated attack traces, and performance data was gathered, based on the minimum SYN flood rate which results in a 100% attack detection rate. It was found that traffic rate, again, plays a key role in this metric. Lower background traffic rates make it easier to detect small rate SYN floods since the attack traffic is more anomalous. Concurrently, when larger background traffic rates are present, higher SYN flood rates are required to successfully detect an attack.

4.3.1 SYN-dog Comparison

Wang et. al. [42] have presented a similar solution to RSD called SYN-dog. Both take advantage of the correlation between SYN and SYN-ACK packets to detect DDoS

attacks, and both aim to detect attacks at the source networks. However, SYN-dog [42] uses a Cumulative Sum (CUSUM) method to detect SYN flood attacks rather than an adaptive threshold method, as in RSD. This results in a slightly higher degree of accuracy (less false positives) in SYN-dog, at the expense of a longer detection time. RSD, on the other hand, is more sensitive to anomalous traffic bursts and will throw more false positives compared to SYN-dog. This is an acceptable tradeoff for achieving our goal of fast detection time, which SYN-dog achieves.

SYN-dog does not feature a training phase to optimize Δt as in RSD. The authors of [42] simply set Δt to 20 seconds, thereby limiting the minimum detection time to this value. RSD, on the other hand, will determine an optimum Δt during the training phase and will dynamically adjust the time interval during execution. This allows RSD to keep the minimal attack detection time as small as possible, even smaller than SYN-dog in some cases. For example, in the case of the DEC-PKT-3 traffic mix, RSD features a minimum detection time of 10 seconds due to the training phase.

The ability of RSD to dynamically adjust Δt makes it more generally applicable to some network environments compared to SYN-dog, especially in low traffic situations. In these cases, SYN-dog's static interval of 20 seconds may not be enough time to obtain an adequate measurement of SYN-ACK packets, which can result in a higher false positive rate. For example, if only 10 SYN-ACK packets were seen in a single interval, then a single missing SYN packet would seem particularly anomalous, possibly resulting in an attack condition. Most likely, this single missing packet would not indicate an attack.

Contrast this to the case where 100 SYN-ACK packets are observed and the same SYN packet is missing. In this case, an anomalous condition would not be observed. RSD aims for the latter case by dynamically adjusting Δt so the number of SYN-ACK packets observed per time interval is close to optimal. While this may result in a longer minimum detection time, it provides a greater degree of accuracy and makes RSD more effective over a wide range of network environments.

Similar to RSD, the efficacy of SYN-dog is examined in [42] using trace-driven simulations. The results are consistent with those found from simulations using RSD—larger flooding rates lead to faster and easier detection of attacks, and detection is much more sensitive in the presence of a lower volume of traffic.

CHAPTER 5: APPLICABILITY OF RSD TO NETWORK PROCESSORS

As covered in section 2, network processors (NPs) have been developed to provide high-speed packet processing with the flexibility of a programmable architecture. A key feature of RSD is its ability to dynamically adapt to changing traffic conditions. This requires on-the-fly calculations which will result in changes in the behavior of RSD. Additionally, RSD requires offline training to optimize its variables, which then need to be programmed into the device. Clearly, network processors may provide RSD with the necessary platform for a successful and efficient implementation. NPs can give RSD the flexibility to modify its behavior while still maintaining line-speed packet processing. This section will explore the applicability of RSD on a network processor through the use of a NP simulator.

5.1 Simulator

The simulator used for performance testing of RSD is based on the SimpleScalar toolkit [50] and is designed to represent the architecture and performance of the Intel IXP 1200 network processor as shown in figure 17. The simulator gives a user the ability to simulate a generic out-of-order superscalar, taking the role of the StrongARM processor and the 6 microengines, each of which contains 4 context-switchable threads for packet processing. Support for C++ objects has also been added to the simulator to increase its flexibility and ease of use.

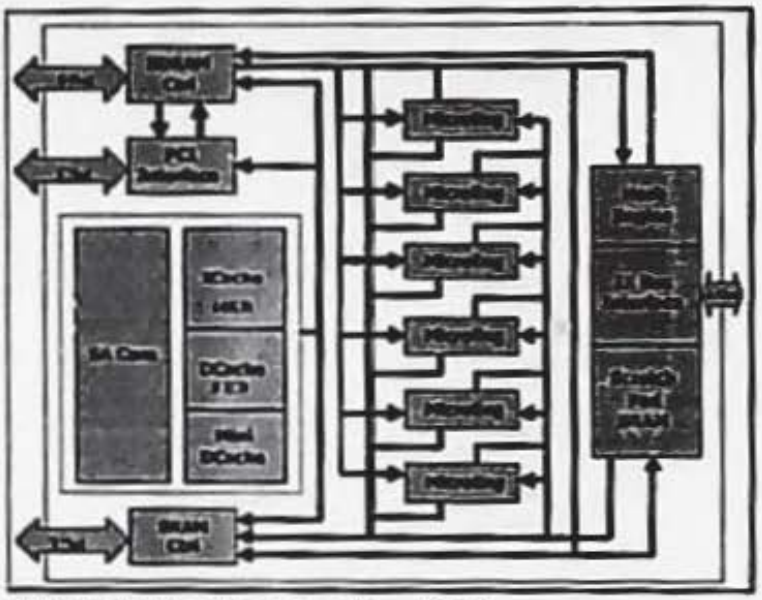


Figure 17: Intel IXP 1200 Architecture (from [49])

5.2 Implementation Results

The RSD algorithm was implemented in the simulator and simple performance measurements were collected to investigate the basic feasibility of running RSD on a network processor.

For comparison purposes, a baseline implementation of simple router functionality was completed and cycle counts were collected using the DEC-PKT-1 trace. The baseline implementation simply parses the destination IP address field of each packet and stores it into a temporary internal variable. In a fully functioning router, a routing table lookup would be performed using this destination IP address and outgoing interface information would be obtained, but for the purposes of this investigation a reference cycle count is all that is needed. Results from the baseline implementation are shown in figure 18 (1).

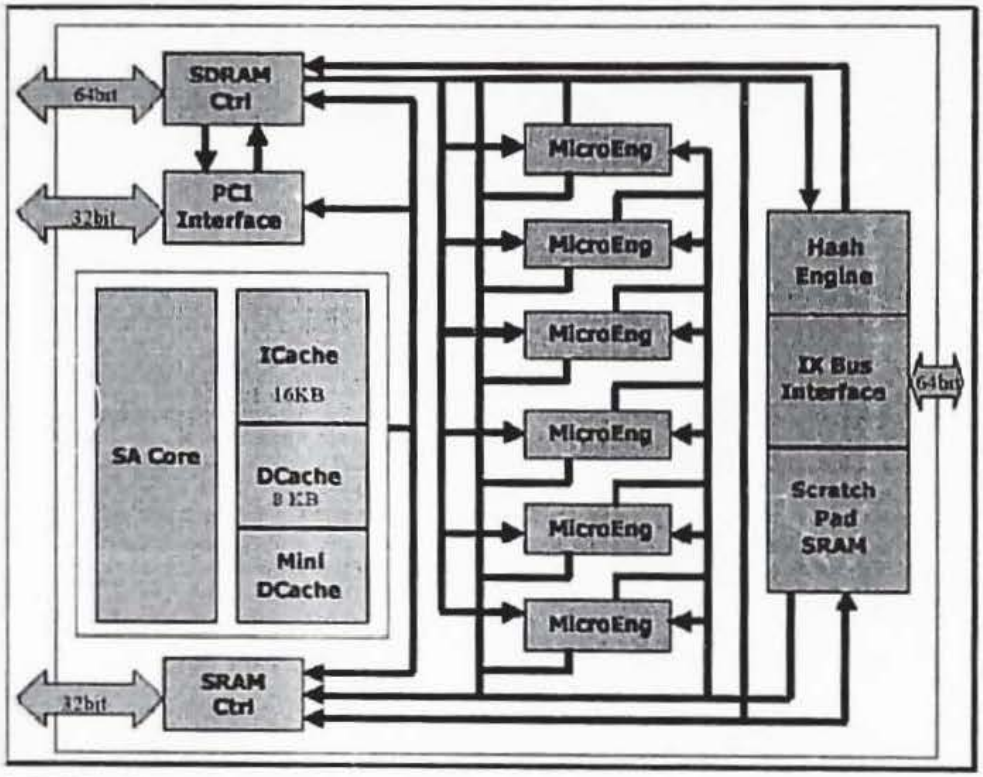


Figure 17: Intel IXP 1200 Architecture (from [49])

5.2 Implementation Results

The RSD algorithm was implemented in the simulator and simple performance

The RSD algorithm was then implemented and cycle count data was collected during runs using the DEC-PKT-1 trace. In the case of this trace, offline training was performed and Δt was set to 10 seconds.

The RSD algorithm requires that each packet be inspected to determine if it is a TCP SYN or TCP SYN-ACK packet, which requires inspection of the TCP flags field in the TCP header. This functionality was added to the baseline implementation and the difference in cycle count was calculated to determine the cost of the new functionality. Results showed that the cost of inspecting the TCP flags field and incrementing the appropriate counter ranged from 11 to 15 cycles per packet, depending on the type of packet and the value in the TCP flag field. Results are shown in figure 18 (2). This represents an approximately 1 percent increase in total cycle count.

Next, the cost of calculating the ratio R_n, updating $\bar{O}_n$, and checking for an attack condition was evaluated. These tasks are done every Δt seconds, not for every packet. The total number of SYN-ACK packets ($\#SA$) is also compared to SA_{cpt} at this point, and Δt is adjusted as necessary to keep RSD running optimally. The cost for these calculations and updates ranged from 51 to 72 cycles, depending on the behavior of RSD. Results are shown in figure 18 (3). This represents an additional 3-5 percent increase in total cycle count.

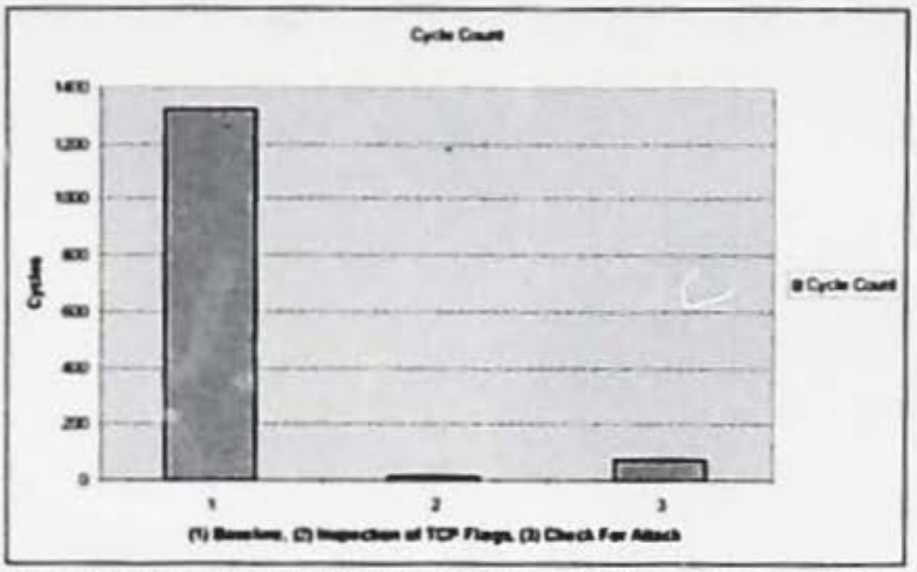


Figure 18: Results From Simulator. (1) represents total cycle count for the baseline implementation. (2) represents the additional cycles required for the inspection of the TCP flags field. (3) represents the additional cycles required when checking for an attack condition.

These results show that RSD has a minimal impact on the performance of a simulated network processor when performance is based solely on cycle counts. The addition of 11 to 15 cycles per packet (a 1 percent increase) should not present a problem for a network processor, and the additional 51 to 72 cycles (a 3-5 percent increase) required when checking for an attack condition will only happen every Δt seconds, not for every packet. In fact, Spalink et. al. [5] have shown that an additional 400 cycles are available in the IXP 1200 microengines for additional packet processing beyond simple IP forwarding. These results are encouraging and suggest that the processing requirements of RSD can easily be met by using a network processor. Additionally, these numbers were obtained using minimum-sized packets arriving at maximum speed on 8 100 Mbps ports. When a more realistic traffic mix is introduced, such as the DEC-PKT-1 trace, performance can be expected to increase, resulting in more available cycles for additional packet processing.

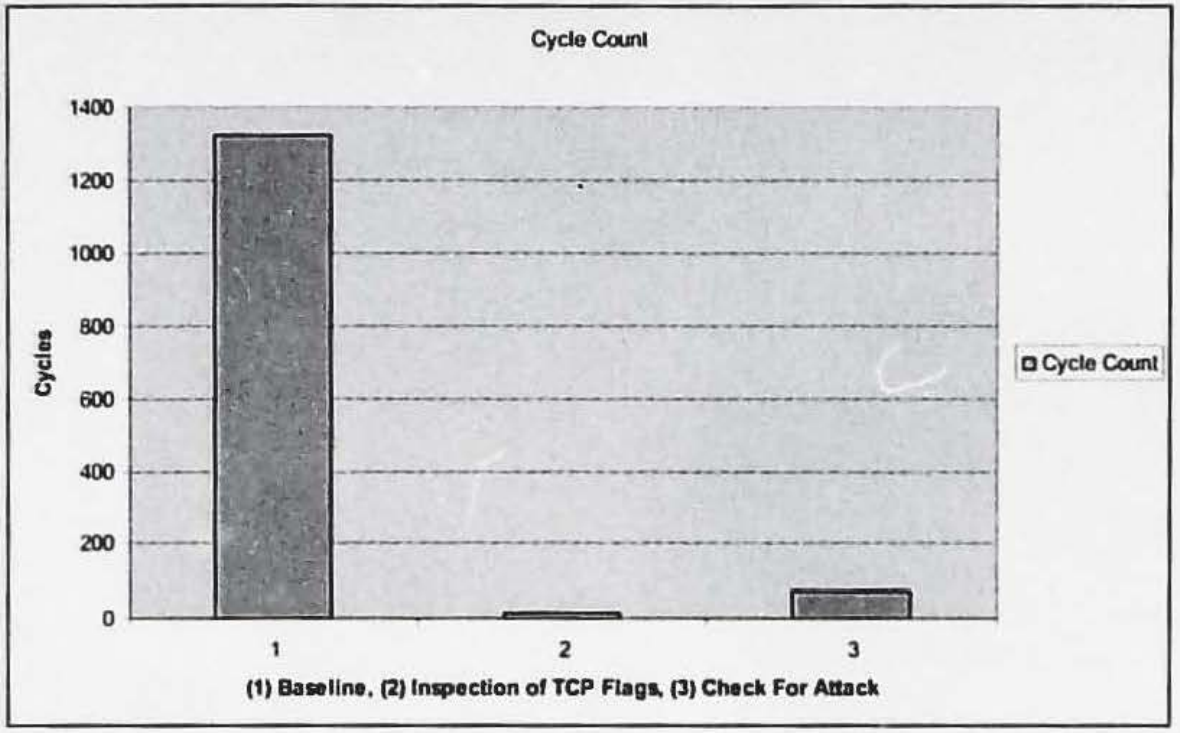


Figure 18: Results From Simulator. (1) represents total cycle count for the baseline implementation. (2) represents the additional cycles required for the inspection of the TCP flags field. (3) represents the additional cycles required when checking for an attack condition.

These results show that RSD has a minimal impact on the performance of a simulated network processor when performance is based solely on cycle counts. The addition of 11 to 15 cycles per packet (a 1 percent increase) should not

CHAPTER 6: CONCLUSION

This thesis presents a novel approach to detecting a certain type of distributed denial-of-service attack called a SYN flood. The detection algorithm, named Ratio-based SYN Flood Detection (RSD), is a type of adaptive threshold algorithm and features dynamically adjusting behavior to optimize its performance depending on the networking environment. RSD is designed to detect SYN flood attacks close to their sources, and is therefore located on the link that connects a stub network to the Internet. This allows attack traffic to be detected and filtered before it can enter the Internet and consume valuable bandwidth.

An in-depth analysis of RSD is presented, focusing on the tradeoffs between different values for parameters in the RSD algorithm. A methodology is discussed which can optimize these parameters to suit the particular traffic characteristics of the stub network.

The efficacy of this methodology is proven through the use of trace-driven simulations, and results are analyzed to determine the minimal rate of attack traffic which is needed for RSD to successfully detect 100% of the attack. For example, in the case of simulations using traces gathered from the Digital Equipment Corporation, RSD was able to detect an attack utilizing 8 SYN packets per second in as little as 15 seconds. Traffic rate is shown to have a significant effect on this performance metric as well as the false positive ratio, demonstrating the need for the adaptive behavior present in RSD.

Network processors may represent the ideal implementation environment for RSD, due to their flexibility and performance characteristics. RSD is implemented on a simple network processor simulator and preliminary data is collected which shows minimal degradation of performance. This indicates that RSD can realistically be deployed on a network processor platform.

Detection of DDoS attacks will always be a difficult problem, especially in the face of determined attackers. The unpredictable and constantly varying behavior of network traffic also makes it virtually impossible to detect attacks with 100% accuracy. However, RSD is applicable to a wide range of deployments due to its optimizing methodology and adaptive behavior.

6.1 Future Work

Future work should further validate the performance of RSD on a wide range of network deployments, from large institutional networks to small in-home networks. This may lead to further enhancements of the algorithm. Also, while the traces used in this thesis consisted of a short time period, validation of the performance of RSD over long time periods should be explored. For example, long-term variations in the traffic characteristics of a network may necessitate the need for another training phase at some point. Further study would need to be done to identify this point.

This thesis assumed that attacks were of a constant intensity and duration, which may or may not represent the current trend in attack signatures. Attackers are always coming up with novel ways to defeat current detection schemes, so different styles of SYN flood

attacks should be simulated in order to identify possible weaknesses in RSD. Improvements to RSD may be possible which could allow it to detect a wider range of attacks, including bursty and slow-growth attacks.

RSD does not attempt to identify the specific attack packet flows, it only detects when an attack is present. Successful mitigation of a DDoS attack will require filtering capabilities to identify and drop the malicious packets. The design of a filtering algorithm is beyond the scope of this thesis, but the observation of the correlation between SYN and SYN-ACK packets may help in the design of such an algorithm. Statistics gathered by RSD could be of use to the filtering engine, resulting in a close cooperation between detection and filtering.

RSD was shown to be theoretically appropriate for implementation on a network processor, and a simple implementation was presented along with some basic results. Further work will need to be done to explore the tradeoffs and performance limitations associated with implementing RSD on a network processor. This will require further development of the network processor simulator. Possible areas to explore include memory usage, latency, and throughput. An in-depth study into the tradeoffs between the two different network processor architectures (serial and parallel) would also be extremely valuable.

REFERENCES

[1] R. Haas, C. Jeffries, L. Kencl, A. Kind, B. Metzler, R. Pletka, M. Waldvogel, L. Frelechoux, P. Droz, "Creating Advanced Functions on Network Processors: Experience and Perspectives", *IEEE Network*, Vol. 17, No. 4, July 2003.

[2] P. Paulin, F. Karim, P. Bromley, "Network Processors: A Perspective on Market Requirements, Processor Architectures and Embedded S/W Tools", *Proc. of DATE (Design, Automation, and Test in Europe) 2001*, March 2001.

[3] T. Spalink, S. Karlin, L. Peterson, Y. Gottlieb, "Building a Robust Software-based Router Using Network Processors", *ACM SIGOPS Operating Systems Review, Proc. of the 18th ACM Symposium on operating systems principles*, October 2001.

[4] P. Crowley, J. Baer, "A Modeling Framework for Network Processor Systems", *IPCA-8 Workshop on Network Processors*, Cambridge, MA, Februrary 2002.

[5] T. Spalink, S. Karlin, L. Peterson, "Evaluating Network Processors in IP Forwarding", *Princeton University Technical Report TR-626-00*, January 2001.

[6] J. Allen, B. Bass, C. Basso, R. Boivie, J. Calvignac, G. Davis, L. Frelechoux, M. Heddes, A. Herkersdorf, A. Kind, J. Logan, M. Peyravian, M. Rinaldi, R. Sabhiki, M. Siegel, M. Waldvogel, "PowerNP Network Processor: Hardware, Software, and Applications", *IBM Journal of Research and Development*, Vol. 47, No. 2/3, March/May 2003.

[7] Y. Lin, Y. Lin, S. Yang, Y. Lin, "Diffserv over Network Processors: Implementation and Evaluation", *Proc. of IEEE 10th Symposium on Hot Interconnects*, Stanford University, Palo Alto, August 2002.

[8] M. Burns, G. Prier, J. Mirkovic, P. Reiher, "Implementing Address Assurance on the Intel IXP Router", *Proc. of NPC 2002*, 2002.

[9] T. Aoki, "Implementation Methods of Differentiated Services by Using CBQ with Dynamic Bandwidth Decision Method for Network Processors", *Master's Thesis*, University of Tsukuba, 2002.

[10] Intel IXA University Program Roster, http://www.ixaedu.com/roster/, 2003.

[11] R. Chang, "Defending against Flooding-Based Distributed Denial-of-Service Attacks: A Tutorial", *IEEE Communications*, Vol. 40, No. 10, October 2002.

[12] T. Peng, C. Leckie, K. Ramamohanarao, "Detecting Distributed Denial of Service Attacks Using Source IP Address Monitoring", *University of Melbourne Technical Report*, November 2002.

[13] T. Peng, C. Leckie, K. Ramamohanarao, "Protection from Distributed Denial of Service Attack Using History-based IP Filtering", *IEEE Intl. Conf. on Communications (ICC) 2003*, Anchorage, Alaska, May 2003.

[14] T. M. Gil, M. Poletter, "MULTOPS: A data-structure for bandwidth attack detection", In *Proceedings of USENIX Security Symposium 2001*, Washington, D.C., August 2001.

[15] K. Park, H. Lee, "On the effectiveness of route-based packet filtering for Distributed DoS attack prevention in power-law internets", In *Proceedings of ACM SIGCOMM 2001*, San Diego, CA, August 2001.

[16] H. Wang, D. Zhang, K.G. Shin, "Detecting SYN Flooding Attacks", In *Proceedings of IEEE INFOCOM 2002*, New York City, NY, June 2002.

[17] P. Ferguson, D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing", *RFC 2827*, May 2000.

[18] J. Mirkovic, G. Prier, P. Reiher, "Attacking DDoS at the Source", In *Proceedings of the ICNP*, Paris, France, November 2002.

[19] J. Li, J. Mirkovic, M. Wang, P Reiher, L. Zhang, "SAVE: Source Address Validity Enforcement Protocol", In *Proceedings of INFOCOM 2002*, New York City, NY, June 2002.

[20] J. Mirkovic, J. Martin, P. Reiher, "A Taxonomy of DDoS Attacks and Defense Mechanisms", *UCLA CSD Technical Report CS-TR-020018*.

[21] J. Mirkovic, G. Prier, P. Reiher, "Source-End DDoS Defense", In *Proceedings of Network Computing and Applications (NCA) 2003*, Cambridge, MA, April 2003.

[22] J. Mirkovic, Z. Xu, J. Li, M. Schnaider, P. Reiher, L. Zhang, "iSAVE: Incrementally Deployable Source Address Validation", *UCLA CSD Technical Report CS-TR-020030*.

[23] C. Jin, H. Wang, K. Shin, "Hop-Count Filtering: An Effective Defense Against Spoofed Traffic", In *Proceedings of 10th ACM Computer and Communication Security*, Washington, DC, 2003.

[24] A. Hussain, J. Heidemann, C. Papadopoulos, "A Framework for Classifying Denial of Service Attacks", In *Proceedings of 2003 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communicatons*, Karlsruhe, Germany, 2003.

[25] J. Xu, "Sustaining Availability of Web Services under Distributed Denial of Service Attacks", *IEEE Transactions on Computers*, Vol. 52, No. 2, February 2003.

[26] CERT Coordination Center, "Denial of Service Attacks", *http://www.cert.org/tech_tips/denial_of_service.html*.

[27] CERT Advisory CA-1997-28, "IP Denial-of-Service Attacks", http://www.cert.org/advisories/CA-1997-28.html.

[28] CERT Advisory CA-1996-26, "Denial-of-Service Attack via ping", http://www.cert.org/advisories/CA-1996-26.html.

[29] Distributed Denial of Service (DDoS) Attacks/tools, http://staff.washington.edu/dittrich/misc/ddos/, January 2004.

[30] V. Paxon, "An Analysis of Using Reflectors for Distributed Denial-Of-Service Attacks", *ACM Computer Communications Review (CCR)*, Vol. 31, No. 3, July 2001.

[31] CERT Incident Note IN-2000-04, "Denial of Service Attacks Using Nameservers", http://www.cert.org/incident_notes/IN-2000-04.html, April 2000.

[32] S. Gibson, "Distributed Reflection Denial of Service: Description and analysis of a potent, increasingly prevalent, and worrisome Internet attack", http://grc.com/dos/drdos.htm, February 2002.

[33] Mazu Networks, http://www.mazunetworks.com.

[34] SonicWall, http://www.sonicwall.com

[35] FortiNet, http://www.fortinet.com

[36] Cisco Systems, http://www.cisco.com

[37] B.E. Brodsky, B.S. Darkhovsky, Nonparametric Methods in Change-point Problems, Kluwer Academic Publishers, 1993.

[38] R.B. Blazek, H. Kim, B. Rozovskii, A. Tartakovsky, "A Novel Approach to Detection of Denial-of-Service Attacks via Adaptive Sequential and Batch-sequential Change-point Detection Methods", *In Proceedings of IEEE Systems, Man and Cybernetics Information Assurance Workshop*, June 2001.

[39] D. Yau, J. Lui, F. Liang, "Defending Against Distributed Denial-of-service Attacks with Max-min Fair Server-centric Router Throttles", *In Proceedings of*

IEEE International Workshop on Quality of Service (IWQoS), Miami Beach, FL, May 2002.

[40] J. Jung, B. Krishnamurthy, M. Rabinovich, "Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites", *In Proceedings of 11th World Wide Web Conference*, Honolulu, HI, May 2002.

[41] A. Belenky, N. Ansari, "On IP Traceback", *IEEE Communications Magazine*, Vol. 41, No. 7, July 2003.

[42] H. Wang, D. Zhang, K. Shin, "SYN-dog: Sniffing SYN Flooding Sources", *In Proceedings of IEEE ICDCS 2002*, Vienna, Austria, 2002.

[43] G. Malan et. al., "Observations and Experiences Tracking Denial-Of-Service Attacks Across a Large Regional ISP", *Technical Report*, Arbor Networks, 2001.

[44] D. Moore, G. Voelker, S. Savage, "Inferring Internet Denial of Service Activity", *In Proceedings of USENIX Security Symposium 2001*, August 2001.

[45] W. Stevens, TCP/IP Illustrated, Volume 1, Addision-Wesley Publishing Company, 1994.

[46] M. Basseville, I. Nikiforov, Detection of Abrupt Changes : Theory and Application, Prentice Hall, 1993.

[47] B. Brodsky, B. Darkhovsky, Non-parametric Methods in Change-point Problems, Kluwer Academic Publishers, 1993.

[48] "The Internet Traffic Archive", http://ita.ee.lbl.gov, Lawrence Berkeley National Laboratory.

[49] E. Johnson, A. Kunze, IXP 1200 Programming, Intel Press 2002.

[50] D. Burger, T. Austin, "The SimpleScalar Tool Set, Version 2.0", http://www.simplescalar.com, 2001.