CERIAS Tech Report 2002-15

Detecting Service Violations and DoS Attacks

by Ahsan Habib, Mohamed M. Hefeeda,
and Bharat K. Bhargava

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907

Detecting Service Violations and DoS Attacks

Ahsan Habib, Mohamed M. Hefeeda, and Bharat K. Bhargava
CERIAS and Department of Computer Sciences
Purdue University, West Lafayette, IN 47907
{habib, mhefeeda, bb}@cs.purdue.edu

Abstract

Denial of Service (DoS) attacks are a serious threat for the Internet. DoS attacks can consume memory, CPU, and network resources and damage or shut down the operation of the resource under attack (victim). The quality of service (QoS) enabled networks, which offer different levels of service, are vulnerable to QoS attacks as well as DoS attacks. The aim of a QoS attack is to steal network resources, e.g., bandwidth, or to degrade the service perceived by users. We present a classification and a brief explanation of the approaches used to deal with the DoS and QoS attacks. Furthermore, we propose network monitoring techniques to detect service violations and to infer DoS attacks. Finally, a quantitative comparison among all schemes is conducted, in which, we highlight the merits of each scheme and estimate the overhead (both processing and communication) introduced by it. The comparison provides guidelines for selecting the appropriate scheme, or a combination of schemes, based on the requirements and how much overhead can be tolerated.

1. Introduction

The San Diego Supercomputer Center reported 12,805 denial of service (DoS) attacks over a three-week period in February 2001 [16]. This is just one of the numerous incidents in which DoS attacks are causing serious security threats to many systems connected to the Internet. The DoS attacks can be severe if they last for a prolonged period of time preventing legitimate users from accessing some or all of computing resources. Imagine an executive of a financial institution deprived of access to the stock market updates for several hours or even several minutes. In [16], the authors showed that whereas 50% of the attacks lasted less than ten minutes, unfortunately, 2% of them lasted greater than five hours and 1% lasted more than ten hours. There were dozens of attacks that spanned multiple days. Wide spectrum of motivation behind these DoS attacks exists. They range from political conflicts and economical benefits for competitors to just curiosity of some computer geeks. Furthermore, cyber terrorism may not be excluded in the future.

In addition to DoS attacks, the quality of service (QoS) enabled networks are vulnerable to another type of attacks, namely, the QoS attacks. A QoS-enabled network, such as a differentiated services network [3], offers different classes of service for different costs. Differences in the charging rates may entice some users to steal bandwidth or other network resources. We define an attacker in this environment as a user who tries to get more resources, i.e., a better service class, than what he has signed (paid) for. QoS attacks are classified into two kinds: attacking the *network provisioning process* and attacking the *data forwarding process*. Network provisioning involves configuration of routers in a QoS network. This process can be attacked by injecting bogus configuration messages, modifying the content of real configuration messages, or delaying such messages. Networks can be secured against such attacks by encrypting the configuration messages of the signaling protocols. Attacks on the data forwarding process are of a more serious nature. These attacks inject traffic into the network with the intent to steal bandwidth or to cause QoS degradation for other flows. Since the differentiated services framework is based on aggregation of flows into service classes, legitimate customer traffic may experience degraded QoS as a result of the illegally injected traffic. Taken to an extreme, that excess traffic may result in a denial of service attack. This creates a need for developing an effective defense mechanism that automates the detection and reaction to attacks on the QoS-enabled networks.

In this paper, we first elaborate on the denial of service attacks and their potential threat on the system. We then classify the solutions proposed in the literature into two main categories: *detection* and *prevention* approaches. We briefly describe several mechanisms in each approach, focusing mainly on the salient features and highlighting the potential as well as the shortcomings of each mechanism. In addition, we propose network monitoring techniques

to detect service violations and to infer DoS attacks. We believe that network monitoring has the potential to detect DoS attacks in early stages before they severely harm the victim. Our conjecture is that a DoS attack injects a huge amount of traffic into the network, which may alter the internal characteristics (e.g., delay and loss ratio) of the network. Monitoring watches for these changes and identifies the congested links, which helps in locating the attacker and alerting the victim. Finally, we conduct a comparative evaluation study among the approaches presented. The aim of the study is to compare the behavior of the approaches under different situations of the underlying network. We draw insightful comments from the comparison that guide the selection of one or more defending approaches suitable for a given environment.

The rest of the paper is organized as follows. Section 2 discuses the DoS attacks and presents the classification of the approaches used to deal with them. In Section 3, we show how network monitoring can be used to detect service violations and to infer DoS attacks. The comparative study is presented in Section 4 and Section 5 concludes the paper.

2. DoS Attacks: Detection and Prevention

In the literature, there are several approaches to deal with denial of service (DoS) attacks. In this section, we provide an approximate taxonomy of these approaches. In addition, we briefly describe the main features of each approach and highlight the strengths and weaknesses of it.

We divide the approaches for dealing with DoS attacks into two main categories: *detection* and *prevention* approaches. The detection approaches capitalize on the fact that appropriately punishing wrong doers (attackers) will deter them from re-attacking again, and will scare others to do similar acts. The detection process has two phases: detecting the attack and identifying the attacker. To identify an attacker, several *traceback* methods can be used, as explained later in this section. The obvious way to detect an attack is just waiting till the system performance decreases sharply or even the whole system collapses. We propose a more effective method for detecting attacks before they severely harm the system. We propose to use monitoring for *early* detection of DoS attacks. The details are given in Section 3. The prevention approaches, on the other hand, try to thwart attacks before they harm the system. Filtering is the main strategy used in the prevention approaches.

To clarify the presentation, we use the hypothetical network topology shown in Figure 1 to demonstrate several scenarios for DoS attacks and how the different approaches react to them. The figure shows several hosts (denoted by Hs) connected to four domains[1] $D1, D2, D3$, and $D4$, which are interconnected through the Internet cloud. In the figure, A_i represents an attacker i while V represents a victim.

2.1. DoS Attacks

The aim of a DoS attack is to consume the resources of a victim or the resources on the way to communicate with a victim. By wasting the victim's resources, the attacker disallows it from serving legitimate customers. A victim can be a host, server, router, or any computing entity connected to the network. Inevitable human errors during software development, configuration, and installation open several unseen doors for these type of attacks.

Several DoS attacks are known and documented in the literature [14, 16, 21, 24]. Flooding a victim with an overwhelming amount of traffic is the most common. This unusual traffic clogs the communication links and thwarts all connections among the legitimate users, which may result in shutting down an entire site or a branch of the network. This happened in February of 2000 for the popular web sites Yahoo, E*trade, Ebay, and CNN for several hours [14].

TCP SYN flooding is an instance of the flooding attacks [22]. Under this attack, the victim is a host and usually runs a Web server. A regular client opens a connection with the server by sending a TCP SYN segment. The server allocates buffer for the expected connection and replies with a TCP ACK segment. The connection remains half-open (backlogged) till the client acknowledges the ACK of the server and moves the connection to the established state. If the client does not send the ACK, the buffer will be deallocated after an expiration of a timer. The server can only have a specific number of half-open connections after which all requests will be refused. The attacker sends a TCP SYN segment pretending a desire to establish a connection and making the server reserves buffer for it. The attacker does not complete the connection. Instead, it issues more TCP SYNs, which lead the server to waste its memory and reach its limit for the backlogged connections. Sending such SYN requests with a high rate keeps the server unable to satisfy connection requests from legitimate users. Schuba *et al.* [22] developed a tool to alleviate the SYN flooding attack. The tool watches for SYN segments coming from spoofed IP addresses and sends TCP RST segments to the server. The RST segments terminate the half-open connections and free their associated buffers.

Other types of flooding attacks include TCP ACK and RST flooding, ICMP and UDP echo-request flooding, and

[1] Throughout the paper, we use "domain" to refer to an Autonomous Systems (AS) domain, which is a network administered by a single entity.

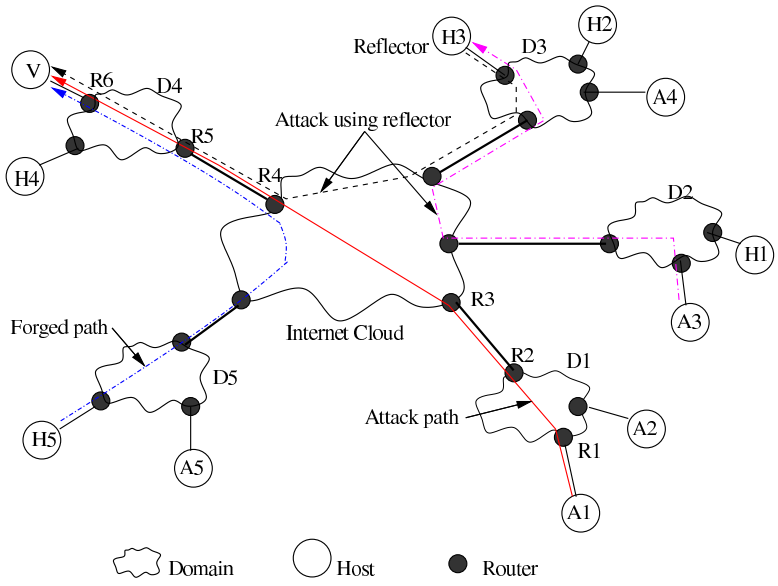


Figure 1. Different scenarios for DoS attacks. Attacker $A1$ launches an attack on the victim V. $A1$ spoofs IP address of host $H5$ from domain $D5$. Another attacker $A3$ uses host $H3$ as a reflector to attack V.

DNS request flooding [16, 24]. This list is by no means exhaustive.

A DoS attack can be more severe when an attacker uses multiple hosts over the Internet to storm a victim. To achieve this, the attacker compromises many hosts and deploys attacking agents on them. The attacker signals all agents to simultaneously launch an attack on a victim. Barros [1] shows that DDoS attack can reach a high level of sophistication by using *reflectors*. A reflector is like a mirror that reflects light. In the Internet, many hosts such as Web servers, DNS servers, and routers can be used as reflectors because they always reply to (or reflect) specific type of packets. Web servers reply to SYN requests, DNS servers reply to queries, and routers send ICMP packets (time exceeded or host unreachable) in response to particular IP packets. The attackers can abuse these reflectors to launch DDoS attacks. For example, an attacking agent sends a SYN request to a reflector specifying the victim's IP address as the source address of the agent. The reflector will send a SYN ACK to the victim. There are millions of reflectors in the Internet and the attacker can use these reflectors to flood the victim's network by sending a large amount of packets. Paxson [20] analyzes several Internet protocols and applications and concludes that DNS servers, Gnutella servers, and TCP-based servers are potential reflectors.

2.2. Detection Approaches

The detection approaches rely on finding the malicious party who launched a DoS attack and consequently hold him liable for the damage he has caused. However, pinning the real attacker down is not a straightforward task. One reason is that the attacker spoofs the source IP address of the attacking packets. Another reason is that the Internet is *stateless*, which means, whenever a packet passes through a router, the router does not store any information (or traces) about that packet. Therefore, mechanisms such as ICMP traceback and packet marking are devised to figure out the real attacker. In this subsection, we describe several techniques to identify the attacker after the attack took place. We defer the issue of early detection of an attack till Section 3.

2.2.1. ICMP Traceback

Bellovin [2] proposes the idea of ICMP traceback messages, where every router samples the forwarded packets with a very low probability (e.g., 1 out of 20,000) and sends an ICMP Traceback message to the destination. An ICMP Traceback message contains the previous and next hop addresses of the router, timestamp, portion of the traced packet, and authentication information. In Figure 1, while packets are traversing the network path from the attacker $A1$ to the victim V, the intermediate routers ($R1, R2, R3, R4, R5$, and $R6$) sample some of these packets and send ICMP Traceback messages to the destination V. With enough messages, the victim can trace the network path $A1 \rightarrow V$. The pitfall of this approach is that the attacker can send many false ICMP Traceback messages to confuse the victim.

To address Distributed DoS (DDoS) attacks by reflec-

tors, Barros [1] proposes a modification to the ICMP Traceback messages. In his refinement, routers sometimes send ICMP Traceback messages to the *source*. In Figure 1, $A3$ launches a DDoS attack by sending TCP SYN segments to the reflector $H3$ specifying V as the source address. $H3$, in turn, sends SYN ACK segments to the victim V. According to the modification, routers on the path $A3 \rightarrow H3$ will send ICMP messages to the source, i.e., to V. This *reverse trace* enables the victim to identify the attacking agent from these trace packets. The *reverse trace* mechanism depends only on the number of attacking agents, and not on the number of reflectors [20]. This achieves scalability because the number of available reflectors is much higher than the number of attacking agents on the Internet.

Snoeren *et al.* [23] propose an attractive hashed-based system that can trace the origin of a single IP packet delivered by a network in the recent past. The system is called source path isolation engine (SPIE). The SPIE uses an efficient method to store information about packets traversing through a particular router. The method uses n bits of the hashed value of the packet to set an index of a 2^n-bit *digest table*. When a victim detects an attack, a query is sent to SPIE, which queries routers for packet digests of the relevant time periods. Topology information is then used to construct the attack graph from which the source of the attack is determined.

2.2.2. Packet Marking

Instead of having routers send separate messages for the sampled packets, Burch and Cheswick [5] propose to inscribe some path information into the header of the packets themselves. This marking can be deterministic or probabilistic. In the deterministic marking, every router marks all packets. The obvious drawback of the deterministic packet marking is that the packet header grows as the number of hops increases on the path. Moreover, significant overhead will be imposed on routers to mark every packet.

The probabilistic packet marking (PPM) encodes the path information into a small fraction of the packets. The assumption is that during a flooding attack, a huge amount of traffic travels towards the victim. Therefore, there is a great chance that many of these packets will be marked at routers throughout their journey from the source to the victim. It is likely that the marked packets will give enough information to trace the network path from the victim to the source of the attack.

Savage *et al.* [21] describe efficient mechanisms to encode the path information into packets. This information contains the XOR (exclusive OR) of two IP addresses and a distance metric. The two IP addresses are for the start and the end routers of the link. The distance metric represents the number of hops between the attacker and the victim. To illustrate the idea, consider the attacker $A1$ and the victim V in Figure 1. Assume there is only one hop between routers $R3$ and $R4$. If Router $R1$ marks a packet, it will encode the XOR of $R1$ and $R2$ addresses into the packet and sets the distance metric to zero, that is, it will encode the tuple $< R1 \oplus R2, 0 >$. Other routers on the path just increase the distance metric of this packet, if they don't decide to mark it again. When this packet reaches the victim, it provides the tuple $< R1 \oplus R2, 5 >$. Similarly, some packets may get marked at routers $R2, R3, R4, R5$, and $R6$ and they will provide the tuples $< R2 \oplus R3, 4 >$, $< R3 \oplus R4, 3 >$, $< R4 \oplus R5, 2 >$, $< R5 \oplus R6, 1 >$, $< R6, 0 >$, respectively, when they reach the victim. The victim can retrieve all routers on the path by XORing the collected messages sorted by distance. (Recall that $Rx \oplus Ry \oplus Rx = Ry$.) This approach can reconstruct most network paths with 95% certainty if there are about 2,000 marked packets available and even the longest path can be resolved with 4,000 packets [21]. For DoS attacks, this amount of packets is clearly obtainable because the attacker needs to flood the network to cause a DoS attack. (Moore *et al.* [16] report that some severe DoS attacks had a rate of thousands of packets per second.) The authors describe ways to reduce the required space and suggest to use the identification field (currently used for IP fragmentation) of IP header to store the encoding of the path information. They also propose solutions to handle the co-existence of marking and fragmentation of IP packets [21].

The main limitation of the PPM approaches stems from the fact that, nothing prevents the attacker from marking packets. If a packet marked by the attacker does not get re-marked by any intermediate router, it will confuse the victim and make it harder to trace the real attacker. Park and Lee [17] show that for single-source DoS attacks, PPM can identify a small set of sources as potential candidates for a DoS attack. For DDoS attacks, however, the attacker can increase the uncertainty in localizing the attacker. Therefore, PPM is vulnerable to distributed DoS attacks [17].

2.3. Prevention Approaches

Preventive approaches try to stop a DoS attack by identifying the attack packets and discarding them before reaching the victim. We summarize several packet filtering techniques that achieve this goal.

2.3.1. Ingress Filtering

Incoming packets to a network domain can be filtered by ingress routers. These filters verify the identity of pack-

ets entering into the domain, like an immigration security system at the airport. Ingress filtering, proposed by Farguson and Senie [10], is a restrictive mechanism that drops traffic with IP address that does not match a domain prefix connected to the ingress router. As an example, in Figure 1, the attacker $A1$ resides in domain $D1$ with the network prefix a.b.c.0/24. The attacker wants to launch a DoS attack to the victim V that is connected to domain $D4$. If the attacker spoofs the IP address of host $H5$ in domain $D5$, which has the network prefix x.y.z.0/24, an input traffic filter on the ingress link of $R1$ will thwart this spoofing. $R1$ only allows traffic originating from source addresses within the a.b.c.0/24 prefix. Thus, the filter prohibits an attacker from using *spoofed* source addresses from outside of the prefix range. Similarly, filtering foils DDoS attacks that employ reflectors. In Figure 1, ingress filter of $D2$ will discard packets destined to the reflector $H3$ and specifying $V's$ address in the source address field. Thus, these packets will not be able to reach the reflector.

Ingress filtering can drastically reduce the DoS attack by IP spoofing if *all* domains use it. It is hard, though, to deploy ingress filters in *all* Internet domains. If there are some unchecked points, it is possible to launch DoS attacks from that points. Unlike ingress filters, egress filters [13] reside at the exit points of a network domain and checks whether the source address of exiting packets belong to this domain. Aside from the placement issue, both ingress and egress filters have similar behavior.

2.3.2. Route-based Filtering

Park and Lee [18] propose route-based distributed packet filtering, which rely on route information to filter out spoofed IP packets. For instance, suppose that $A1$ belongs to domain $D1$ and is attempting a DoS attack on V that belongs to domain $D4$. If $A1$ uses the spoofed address $H5$ that belongs to domain $D5$, the filter at domain $D1$ would recognize that a packet originated from domain $D5$ and destined to V should not travel through domain $D1$. Then, the filter at $D1$ will discard the packet. Route-based filters do not use/store individual host addresses for filtering, rather, they use the topology information of Autonomous Systems (ASes). The authors of [18] show that with partial deployment of route-based filters, about 20% in the Internet AS topologies, it is possible to achieve a good filtering effect that prevents spoofed IP flows reaching other ASes. These filters need to build route information by consulting BGP routers of different ASes. Since routes on the Internet change with time [19], it is a challenge for route-based filters to be updated in real time.

Finally, all filters proposed in the literature so far fall short to detect IP address spoofing from the domain in which the attacker resides. For example, in Figure 1, if $A1$ uses some unused IP addresses of domain $D1$, the filters will not be able to stop such forged packets to reach the victim V.

3. Monitoring to Detect Service Violations and DoS Attacks

In this section, we show how network monitoring techniques can be used to detect service violations and to infer DoS attacks. We believe that network monitoring has the potential to detect DoS attacks in early stages before they severely harm the victim. Our conjecture is that a DoS attack injects a huge amount of traffic into the network, which may alter the internal characteristics (e.g., delay and loss ratio) of the network. Monitoring watches for these changes and our proposed techniques can identify the congested links and the points that are feeding them. We describe the monitoring schemes in the context of a QoS-enabled network, which provides different classes of service for different costs. The schemes are also applicable to best effort (BE) networks to infer DoS attacks, but not to detect service violations because there is no notion of service differentiation in BE networks.

To monitor a domain, we measure three parameters: delay, packet loss ratio, and throughput. We refer to these parameters collectively as the service level agreement (SLA) parameters, since they indicate whether a user is achieving the QoS requirements contracted with the network provider. In our discussion, delay is the end-to-end latency; packet loss ratio is defined as the ratio of number of dropped packets from a flow[2] to the total number of packets of the same flow entered the domain; and throughput is the total bandwidth consumed by a flow inside the domain. Delay and loss ratio are good indicators for the current status of the domain. This is because, if the domain is properly provisioned and no user is misbehaving, the flows traversing through the domain should not experience high delay or loss ratio inside that domain. It is worth mentioning that delay jitter, i.e., delay variation, is another important SLA parameter. However, it is flow-specific and therefore, is not suitable to use in network monitoring.

The SLA parameters can be estimated with the involvement of internal (core) routers in a network domain or can be inferred without their help. We describe both *core-assisted* monitoring and *edge-based* (without involvement of core routers) monitoring in the following subsections.

[2] A flow can be a micro flow with five tuples (addresses, ports, and protocol) or an aggregate one that is a combination of several micro flows.

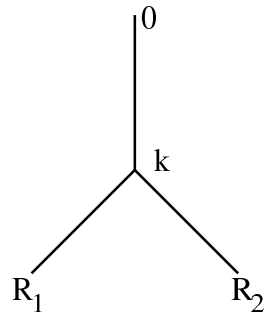


Figure 2. Inferring loss ratio from the source 0 to receivers R_1 and R_2.

3.1. Core-based Monitoring

A core-based monitoring scheme for QoS-enabled network is studied in [11]. In this scheme, the delay is measured by having the ingress routers randomly copy the header of some of the incoming packets. The copying depends on a pre-configured probability parameter. The ingress router forms a probe packet with the same header as the data traffic, which means that the probe packet will likely follow the same path as the data packet. The egress router recognizes these probe packets and computes the delay.

This monitoring scheme measures the loss ratio by collecting packet drop counts from core routers. It then contacts the ingress routers to get the total number of packets for each flow. The loss ratio is computed from these two numbers. To measure the throughput, the scheme polls the egress routers. The egress routers can provide this information because they already maintain this information for each flow. This scheme imposes excessive overhead on the core routers, therefore, it is not scalable. Other monitoring schemes that involve both core and edge routers are proposed in the literature, see for example [4, 7, 8].

3.2. Edge-based Monitoring

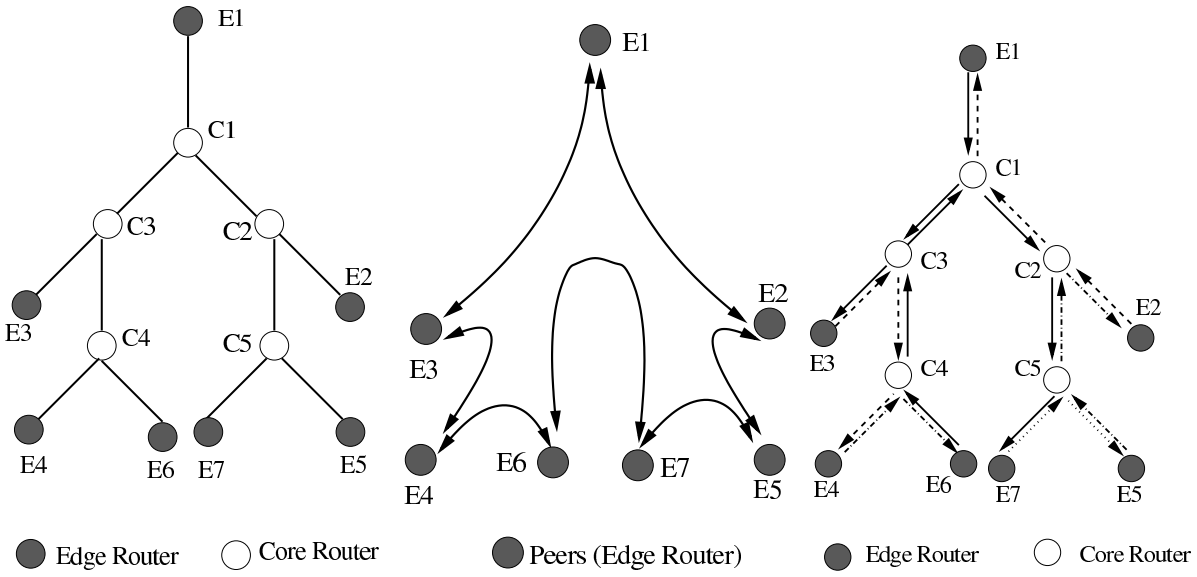
We describe two edge-based monitoring schemes: *stripe-based* and *distributed*. Both schemes measure delay and throughput using the same techniques as the previous core-based scheme. They differ, however, in measuring the packet loss ratio.

Stripe-based Monitoring. The stripe-based scheme infers loss ratio inside a domain without relying on core routers. We show how to infer loss ratios for unicast traffic as explained in [9] and refer the reader to [6] for the multicast traffic case. The scheme sends a series of probe packets, called a stripe, with no delay between them. Usually, a stripe consists of three packets. To simplify the discussion, consider a two-leaf binary tree spanning nodes 0, k, R_1, R_2, as shown in Figure 2. The loss ratio of the link $k \rightarrow R_1$, for instance, can be estimated by sending stripes from the root 0 to the leaves R_1 and R_2. The first packet of a 3-packet stripe is sent to R_1, while the last two are sent to R_2. If a packet reaches to any receiver, we can infer that the packet must have reached the branching point k. Further, if R_2 gets the last two packets of a stripe, it is likely that R_1 receives the first packet of that stripe. The packet loss probability is calculated based on whether all packets sent to R_1 and R_2 reach their destination. Similarly, the loss ratio of the link $k \rightarrow R_2$ is inferred using a complementary stripe, in which the first packet is sent to R_2 and the last two are sent to R_1. The loss ratio of the common path from $0 \rightarrow k$ can be estimated by combining the results of the previous two steps. For general trees, this inference technique sends stripes from the root to all ordered pairs of the leaves of the tree. Finally, this technique is extended in [11] for routers with active queue management in a QoS domain.

Distributed Monitoring. The distributed monitoring approach is proposed in [12] to further reduce the monitoring overhead. In this mechanism, the edge routers of a domain form an overlay network on top of the physical network. Figure 3(a) shows the spanning tree of the domain's topology. The edge routers form an overlay network among themselves, as shown in Figure 3(b). This overlay is used to build tunnel for probe packets on specified paths. The internal links for each end-to-end path in the overlay network are shown in Figure 3(c). In the distributed monitoring approach, an SLA monitor sits at any edge router. The monitor probes the network regularly for unusual delay patterns. The delay and throughput measurements are the same as described in *stripe-based* scheme. The two schemes differ in measuring loss. Since service violation can be detected without exact loss values, we need only to determine whether a link has higher loss than the specified threshold or not. The link with high loss is referred to as a congested link. The goal of the distributed monitoring is to detect all congested links.

When delay goes high, the SLA monitor triggers agents at different edge routers to probe for loss. Each edge router probes its neighbors. Let X_ρ be a boolean random variable that represents the output of probe ρ. X_ρ takes on value 1 if the measured loss exceeds the threshold in any link throughout the probe path, and takes on 0 otherwise. For example, if the outcome of $E1 \rightarrow E3$ probing path is 1, it means either $E1 \rightarrow C1$, $C1 \rightarrow C3$, $C3 \rightarrow E3$, or a combination of them is congested. If the outcome is 0, then definitely all internal links are *not* congested. In this way, we write equations to express all internal links in terms of the probe outcomes. Solving these equations and identifying the congested links are detailed in [12].

The distributed monitoring scheme requires less number of total probes, $O(n)$, compared to the stripe-based



(a) The spanning tree of the domain's topology.

(b) An overlay constructed over all edge routers.

(c) Direction of probing the internal links.

Figure 3. Network monitoring using the distributed mechanism.

scheme, which requires $O(n^2)$, where n is the number of edge routers in the domain. The distributed scheme is able to detect violation in both directions of any link in the domain, whereas the stripe-based can detect a violation only if the flow direction of the misbehaving traffic is the same as the probing direction from the root. To achieve same ability like distributed one, the stripe-based needs to probe the whole tree from several points, which increases the monitoring overhead substantially.

3.3. Violation and DoS Detection

In both the *stripe-based* and *distributed-based* monitoring schemes, when delay, loss, and bandwidth consumption exceed the pre-defined thresholds, the monitor decides on possible SLA violation. The monitor knows the existing traffic classes and the acceptable SLA parameters per class. High delay is an indication of abnormal behavior inside the domain. If there is any loss for the guaranteed traffic class and if the loss ratios of other traffic classes exceed certain levels, an SLA violation is flagged. This loss can be caused by some flows consuming bandwidth beyond their SLA. Bandwidth theft is checked by comparing the total bandwidth achieved by a user against the user's SLA for bandwidth. The misbehaving flows are controlled at the ingress routers.

To detect DoS attacks, set of links L with high loss are identified. For each congested link, $l(v_i, v_j) \in L$, the tree is divided into two subtrees: one formed by leaves descendant from v_i and the other from the leaves descendant from v_j. The first subtree has egress routers as leaves through which high aggregate bandwidth flows are leaving. If many exiting flows have the same destination IP prefix, we can infer that either this is a DoS attack or the traffic is a going to a popular site [15]. Decision can be taken with consulting the destination entity. If it is an attack, we can stop it by triggering filters at the ingress routers that are leaves of the *other* subtree.

We illustrate a scenario of detecting and controlling DoS attack using Figure 3. Suppose, the victim's domain $\mathcal{D}$ is connected to the edge router $E6$. The monitor observes that links $C3 \rightarrow C4$ and link $C4 \rightarrow E6$ are congested for a time duration Δt sec. From both congested links, we obtain the egress router $E6$ through which most of these flows are leaving. The destination IP prefix matching at $E6$ reveals that an excess amount of traffic is heading towards the domain $\mathcal{D}$ connected to $E6$. To control the attack, the monitor needs to figure out through which ingress routers the suspected flows are entering into the domain. The algorithm to identify these ingress routers is discussed in [12]. The monitor activates filters at these ingress routers to regulate the flows that are destined to $\mathcal{D}$.

The advantage of the monitoring-based attack detection is that the neighbor domains of the victim can detect the attack early by observing the violation of SLA parameters. By consulting with the potential victim, these domains can

Symbol	Description	Values used in comparison
$\mathcal{P}_{sch}$	Processing overhead for scheme sch	–
$\mathcal{C}_{sch}$	Communication overhead for scheme sch	–
M	Number of edge routers	[10 – 20]
N	Number of core routers	12
F	Number of flows entering through each edge router	100,000
P	Number of packets per flow	10
p	Probability to mark a packet	[0 – 0.20]
θ	Percentage of misbehaving flows	[0 – 20%]
h	Path length inside a domain or hop count	4, 6
s	Length of a stripe	3
f_s	Frequency of stripe per unit time in stripe-based monitoring	20
f_d	Frequency of probes per unit time in distributed monitoring	30
α_1	Processing overhead for filtering	–
α_2	Processing overhead for marking	–
α_3	Processing overhead for monitoring	–

Table 1. Symbols used in the comparison and their values.

regulate the intensity of the attack and even an early detection can thwart the attack. For each violation, the monitor takes actions such as throttling a particular user's traffic using a flow control mechanism.

4. Comparative Evaluation

In this section, we conduct a quantitative analysis of the overhead imposed by different schemes to detect and prevent DoS attacks. The objective of this comparison is to show the characteristics of each scheme and how they behave when different configuration parameters of a domain are changed. We do not emphasize on numeric overhead value of any specific scheme, rather, we draw a relative comparison among them. The comparison provides guidelines for selecting the appropriate scheme, or a combination of schemes, based on the requirements and how much overhead can be tolerated. The schemes we compare here are: Ingress Filtering (*Ingf*), route-based packet filtering (*Route*), traceback with probabilistic packet marking (*PPM*), core-based network monitoring (*Core*), stripe-based monitoring (*Stripe*), and distributed monitoring (*Distributed*).

4.1. Setup

For each scheme, we calculate two different overheads: processing and communication. The processing overhead is due to extra processing required at all routers of a domain per unit time. The communication overhead is due to extra packets injected into a domain. The communication overhead is computed as the number of extra *bytes* (not packets) injected per unit time. For processing overhead, the extra processing at routers may contain: more address lookups, changing some header fields, checksum re-computation, and any CPU processing needed by the scheme. For example, filters need to check the source IP address to verify whether a packet is coming from a valid source. This requires one extra address lookup (to check the source IP address) for each packet. The monitoring schemes inject probe packets into the network. Each router inside a domain requires processing such as address lookup, TTL field decrement, checksum computation for each probe packet. For simplicity, we charge the filtering scheme α_1 processing units, the marking scheme α_2 processing units, and the monitoring schemes α_3 processing units for each packet processed. We express the processing overhead in terms of α_1, α_2, and α_3 (processing units), and the communication overhead in terms of the total kilobytes (KB) injected in the domain.

We consider a domain $\mathcal{D}$ with M edge routers and N core routers. We assume there are F flows traversing through each edge router and each flow has P packets on average. We define θ as the percentage of misbehaving flows that may cause DoS attacks. We denote $\mathcal{C}_{sch}$ as the communication overhead and $\mathcal{P}_{sch}$ as the processing overhead respectively for scheme sch. Table 1 lists the variables used in the comparison and their values.

4.2. Overhead Calculation

Filtering and marking techniques do not incur any communication overhead. The monitoring schemes have both processing and communication overhead.

Ingress filtering. The processing overhead of ingress filtering depends on the number of packets entering a domain. It requires one processing unit to check the source IP address of every packet. For our domain $\mathcal{D}$, the total entering packets is $M \times F \times P$. Thus, the total

processing overhead of ingress filtering is given by:

$$\mathcal{P}_{Ingf} = M \times F \times P \times \alpha_1. \quad (1)$$

Route-based filtering. We need to deploy ingress filters in every domain in the Internet to effectively stop all possible attacks. The route-based filtering scheme, on the other hand, does not require every single domain to have a filter. Park *et al.* show that placing this filter at approximately 20% of all autonomous systems can prevent DoS to a great extent [18]. For a domain that deploys a router-based filter, the overhead is the same as the ingress filter. Globally speaking, the overhead of route-based filtering is one fifth of the overhead of ingress filtering on the average. In our comparison, we use

$$\mathcal{P}_{Route} = 0.2 \times \mathcal{P}_{Ingf}. \quad (2)$$

Probabilistic packet marking (PPM). *PPM* does not incur any communication overhead but adds extra α_2 processing units for every packet that gets marked at an intermediary router. *PPM* might need sophisticated operation such as taking hash of certain IP fields. The traceback with *PPM* marks packets with a probability p at each router on the path to the victim. If a packet passes through h hops, on the average, in the network domain $\mathcal{D}$, the processing overhead is computed as:

$$\mathcal{P}_{PPM} = M \times F \times P \times p \times h \times \alpha_2 \quad (3)$$

Core-based monitoring. The monitoring schemes inject probe traffic into the network and add processing overheads as well. The total number of injected probes and the size of each probe packet are used to calculate the communication overheads in terms of bytes. The *Core* scheme depends on the number of packets that core routers send to the monitor to report drop history. The drop history at each core router depends on the flows traversing the network domain and the percentage of these flows that are violating their SLAs at a particular time. For the domain $\mathcal{D}$, if d bytes are required to record the drop information of each flow, then each core needs to send $C = max(1, \frac{F \times \theta \times d}{packet_size})$ control packets to the monitor. The $packet_size$ is the size of a control packet, which depends on the MTU of the network. To obtain *loss ratio,* the monitor queries all edges for packet count information of the misbehaving flows. Every edge replies to this query. The total number of packets exchanged among all edge routers and the monitor is $(2M + N) \times C$ packets. Therefore, the communication overhead is given by:

$$\mathcal{C}_{Core} = (2M+N) \times max(1, \frac{F \times \theta \times d}{packet_size}) \times packet_size, \quad (4)$$

and the processing overhead is given by:

$$\mathcal{P}_{Core} = (2M + N) \times max(1, \frac{F \times \theta \times d}{packet_size}) \times h \times \alpha_3, \quad (5)$$

where $packet_size$ is a configurable parameter.

Stripe-based monitoring. In the stripe-based monitoring scheme, a stripe of s packets is sent from the monitor to every egress router pairs. For the network domain $\mathcal{D}$, the total number of probing packets is $s \times (M-1) \times (M-2) \times f_s$, where f_s is the frequency by which we need to send stripes per unit time. The communication overhead and the processing overhead are shown in equation (6) and equation (7) respectively.

$$\mathcal{C}_{Stripe} = s \times (M-1) \times (M-2) \times f_s \times packet_size, \quad (6)$$

$$\mathcal{P}_{Stripe} = s \times (M-1) \times (M-2) \times f_s \times h \times \alpha_3. \quad (7)$$

Distributed monitoring. For the distributed monitoring, each edge router probes its left and right neighbors. If it requires f_d probes per unit time, the communication overhead is:

$$\mathcal{C}_{Distributed} = 2 \times M \times f_d \times packet_size. \quad (8)$$

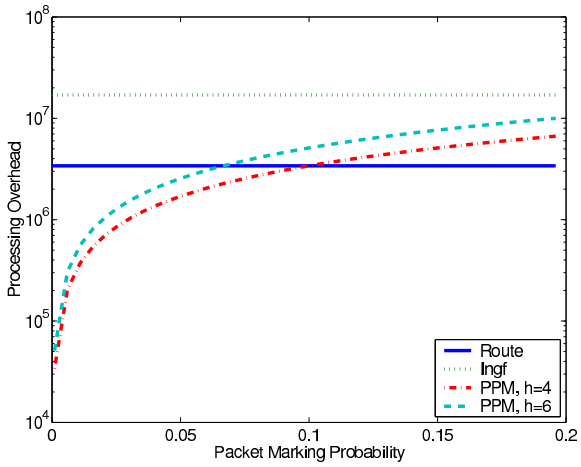
On the average, each probe packet traverses h hops and thus the processing overhead can be calculated as:

$$\mathcal{P}_{Distributed} = 2 \times M \times f_d \times h \times \alpha_2 \quad (9)$$

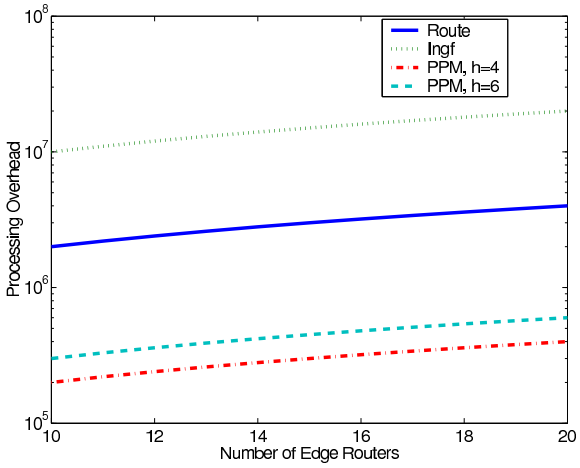
4.3. Results and Analysis

To visualize the differences among all schemes, we plot the processing and communication overhead for one of the domain shown in Figure 1. Usually, DoS attacks are directed towards a particular host or a set of hosts connected to a relatively small size domain. In the example, Figure 1, the DoS attack is directed towards domain $D4$ and the attack traffic is coming from various other domains. For our comparison, we use the parameters' values shown in Table 1 for domain D. We use *sec* as unit time in all comparisons.

Figure 4 (a) shows the processing overhead in terms of α_1 for ingress filtering, route-based filtering, and PPM when packet marking probability is varied along the X-axis. The route-based filtering requires less processing than marking scheme for $p \geq 0.07$ because this filtering scheme does not need to be deployed at all routers of all domains. Savage *et al.* use marking probability $p = 0.04$

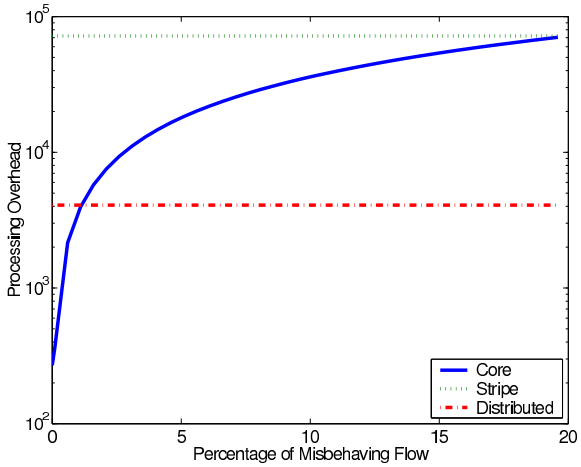


(a) Effect of varying the marking probability on the processing overhead.

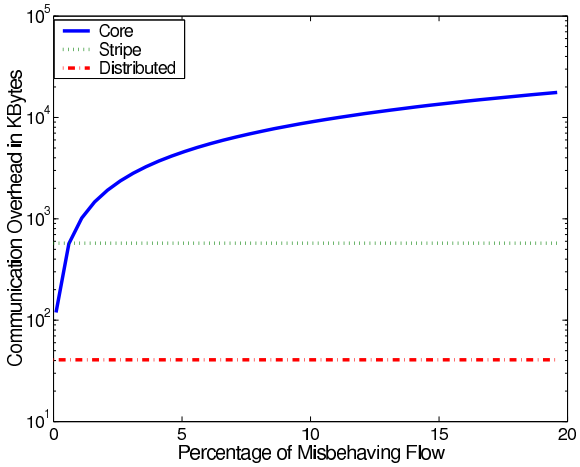


(b) Effect of varying the number of edge routers on the processing overhead.

Figure 4. The processing overhead per unit time for filters and probabilistic packet marking (PPM) schemes. Marking scheme has less processing overhead than filtering scheme if the marking probability is not too high (e.g., $p \leq 0.07$).

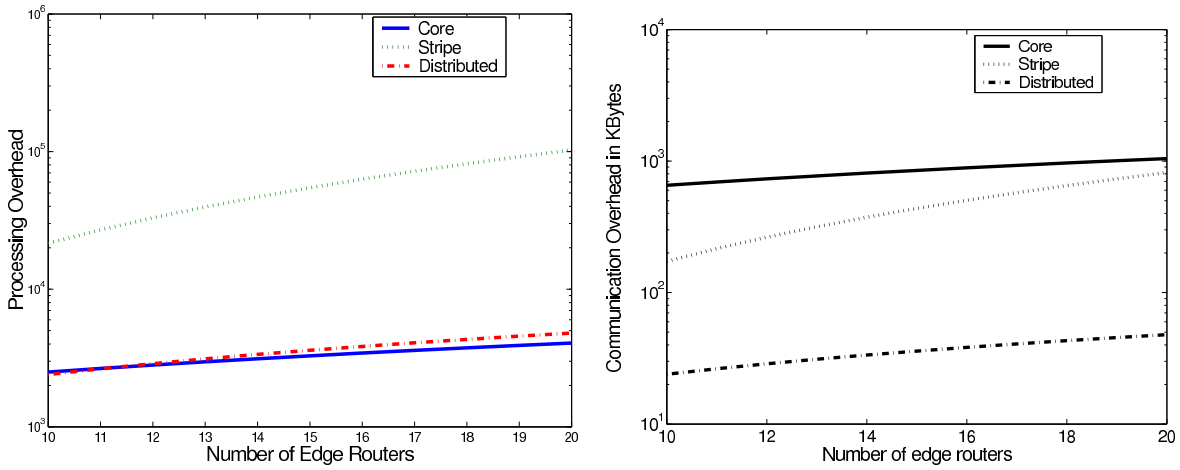


(a) Processing overhead.



(b) Communication overhead.

Figure 5. The processing and communication overhead for the monitoring schemes when the percentage of misbehaving flows is increased. The *Core* scheme has less communication overhead than *Stripe* scheme for $\theta < 20\%$. Both *Stripe* and *Distributed* schemes have less communication overhead than *Core* unless θ is very low.



(a) Processing overhead.

(b) Communication overhead.

Figure 6. The processing and communication overhead for the monitoring schemes when the number of edge routers in a domain is increased. The *Core* scheme has less processing overhead than both edge-based schemes when the number of edge routers in the domain is increased. Edge-based schemes always impose less communication overhead than the *Core* scheme. The *Core* might perform better than *Stripe* for a large domain (e.g., $M > 20$) depending on the value of θ.

in their traceback analysis [21]. Using this probability, the marking mechanism has less overhead than others. We use two different path lengths in the plot; one is $h = 4$ and another is $h = 6$. The path length does not increase the overhead substantially because the path length does not go up very high for a small-size domain. Figure 4(b) shows that when number of edge routers are increased in a domain the processing over is increased for all schemes.

Figure 5 shows both processing and communication overhead for different monitoring schemes. The processing overhead is low for *Core* scheme than the *Stripe* scheme for $\theta \leq 20\%$. This is because the control packet size of *Core* can be set equal to the maximum transmission unit of the network to minimize total number of packets sent, whereas the probe packet size of the *Stripe* is 20 bytes with 20 bytes of IP header. However, if the attack intensity is high, i.e., the value of θ is high, the overhead of *Core* exceeds the overhead of both edge-based schemes. In this example, probes injected by *Stripe* scheme consumes 600K bytes of bandwidth per sec, which is distributed over all links of the domain. If all links are OC3 type, on average each link experiences probe traffic less than 0.015% of the link capacity. The *Distributed* scheme consumes ten times less than the *Stripe* one in this setup. In Figure 6, we vary the domain size changing the number of edge routers while keeping the number of core routers fixed to $N = 12$. The percentage of misbehaving traffic θ is fixed and equals 1%. Figure 6 (a) shows that *Core* can result in less computation overhead than edge-based schemes when the number of edge routers increases. Even though the overhead of *Core* scheme depends on both core and edge routers, this scheme reduces processing overhead by aggregating flows when it reports to the monitor. When number of edge routers increases, overhead for both *Core* and *Distributed* schemes increase linearly. The overhead for *Stripe* increases substantially with the increase of edge routers. Depending on θ, Figure 6 (b) shows that the communication overhead for *Stripe* may exceed the communication overhead of *Core* when $M > 20$.

4.4. Summary

We summarize the important features of all schemes in Table 2. Ingress filtering and core-assisted monitoring schemes have high implementation overhead because the former needs to deploy filters at all ingress routers in the Internet and the latter needs support from all edge and core routers in a domain. But filtering and monitoring can provide better safety compared to the traceback which only can identify an attacker after the attack has occurred. All monitoring schemes need clock synchronization to measure SLA parameters, which is an extra overhead. But, they can detect service violations and DoS attacks as well. Filters are proactive in nature and all other schemes are reactive. Filters can detect attacks by spoofed packets whereas the rest of the schemes can detect an attack even if the attacker does not use spoofed IP addresses from other domains.

Property	*PPM*	*Ingress Filtering*	*Route-based Filtering*	*Core-assisted Monitoring*	*Stripe-based Monitoring*	*Distributed Monitoring*
Overhead depends on	attack volume	number of incoming packets	number of incoming packets	number of flows violating SLAs	routers, topology, attack traffic	routers, topology, attack traffic
Implementation overhead	all routers	all ingress edge routers	all routers of selective domains	all edge and core routers	all edge routers	all edge routers
Clock synchronization	—	—	—	at edge and core routers	at edge routers	at edge routers
Response	reactive	proactive	proactive	reactive	reactive	reactive
SLA violation detection	no	no	no	yes	yes	yes
Detect attacks initiated using	any IP	spoofed IP from other domains	spoofed IP from other domains	any IP	any IP	any IP

Table 2. Comparison among different schemes to detect/prevent service violations and DoS attacks.

5. Conclusions

We investigated several methods to detect service level agreement violations and DoS attacks. We showed that there is no single method that fits all possible scenarios. Specifically, in ICMP traceback and probabilistic packet marking mechanisms, the attacker may be able to confuse the victim by sending false ICMP traceback packets and by randomly marking attacking packets. Ingress filters need global deployment to be effective, whereas route-based filters strive against the dynamic change of the routing information.

We showed that network monitoring techniques can be used to detect service violations by measuring the SLA parameters and comparing them against the contracted values between the user and the network provider. We also argued that monitoring techniques have the potential to detect DoS attacks in early stages before they severely harm the victim. Our argument is based on the fact that a DoS attack injects a huge amount of traffic into the network, which may alter the internal characteristics (e.g., delay and loss ratio) of the network. The monitoring techniques watch for these changes and identify the congested links, which helps in locating the attacker and alerting the victim.

The presented comparative study showed several issues. First, it showed that while marking imposes less overhead than filtering, it is only a forensic method. Filtering, on the other hand, is a preventive method, which tries to stop attacks before they harm the system. Second, the core-based monitoring scheme has a high deployment cost because it needs to update all edge as well as core routers. However, the core-based scheme has less processing overhead than the stripe-based scheme because it aggregates flow information when it reports to the monitor. Third, The stripe-based monitoring scheme has lower communication overhead than the core-based scheme for relatively small size domains. For large domains, however, core-based may impose less communication overhead depending on the attack intensity. Fourth, The distributed scheme outperforms the other monitoring schemes in terms of deployment cost and overhead in many of the cases.

Acknowledgments

This research is supported in part by the National Science Foundation grants CCR-991712 and CCR-001788, CERIAS, and IBM.

References

[1] C. Barros. A proposal for ICMP traceback messages. Internet Draft http://www.research.att.com/lists/ietf-itrace/2000/09/msg00044.html, Sept. 18, 2000.

[2] S. M. Bellovin. ICMP traceback messages. Internet draft: draft-bellovin-itrace-00.txt, Mar. 2000.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for Differentiated Services. RFC 2475, Dec. 1998.

[4] Y. Breitbart, C. Y. Chan, M. Garofalakis, R. Rastogi, and A. Silberschatz. Effi ciently monitoring bandwidth and latency in IP networks. In *Proc. IEEE INFOCOM*, Anchorage, AK, Apr. 2001.

[5] H. Burch and H. Cheswick. Tracing anonymous packets to their approximate source. In *Proc. USENIX LISA*, pages 319–327, New Orleans, LA, Dec. 2000.

[6] R. Cáceres, N. G. Duffi eld, J. Horowitz, and D. Towsley. Multicast-based inference of network-internal loss characteristics. *IEEE Transactions on Information Theory*, Nov. 1999.

[7] M. C. Chan, Y.-J. Lin, and X. Wang. A scalable monitoring approach for service level agreements validation. In *Proc.*

International Conference on Network Protocols (ICNP), pages 37–48, Osaka, Japan, Nov. 2000.

[8] M. Dilman and D. Raz. Efficient reactive monitoring. In *Proc. IEEE INFOCOM*, Anchorage, AK, Apr. 2001.

[9] N. G. Duffield, F. L. Presti, V. Paxson, and D. Towsley. Inferring link loss using striped unicast probes. In *Proc. IEEE INFOCOM*, Anchorage, AK, Apr. 2001.

[10] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing agreements performance monitoring. RFC 2827, May 2000.

[11] A. Habib, S. Fahmy, S. R. Avasarala, V. Prabhakar, and B. Bhargava. On detecting service violations and bandwidth theft in QoS network domains. *Journal of Computer Communicatons (to appear)*, 2003.

[12] A. Habib, M. Khan, and B. Bhargava. Edge-to-edge measurement-based distributed network monitoring. Technical report, CSD-TR-02-019, Purdue University, Sept. 2002.

[13] S. Institute. Egress filtering v 0.2. http://www.sans.org/y2k/egress.htm, Feb. 2000.

[14] L. Garber. Denial of Service attacks rip the Internet. *IEEE Computer* , 33,4:12–17, Apr. 2000.

[15] M. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *ACM Computer Communication Review*, 32(3):62–73, July 2002.

[16] D. Moore, G. M. Voelker, and S. Savage. Inferring Internet denial-of-service activity. In *Proc. USENIX Security Symposium*, Washington D.C, Aug. 2001.

[17] K. Park and H. Lee. On the effectiveness of probabilistic packet marking for IP traceback under Denial of Service attack. In *Proc. IEEE INFOCOM*, Anchorage, AK, Apr. 2001.

[18] K. Park and H. Lee. A proactive approach to distributed DoS attack prevention using route-based packet filtering. In *Proc. ACM SIGCOMM*, San Diego, CA, Aug. 2001.

[19] V. Paxson. End-to-end internet packet dynamics. In *Proc. SIGCOMM '97*, Cannes, France, 1997.

[20] V. Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *ACM Computer Communication Review*, 31 (3), July 2001.

[21] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Network support for IP traceback. *IEEE/ACM Transaction on Networking*, 9:(3):226–237, June 2001.

[22] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on tcp. In *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, May 1997.

[23] A. Snoeren, C. Partridge, L. Sanchez, W. Strayer, C. Jones, and F. Tchakountio. Hashed-based IP traceback. In *Proc. ACM SIGCOMM*, San Diego, CA, Aug. 2001.

[24] G. Spafford and S. Garfinkel. *Practical Unix and Internet Security*. O'Reilly & Associates, Inc, second edition, 1996.