INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt 2004; **14**: 000–000 (DOI: 10.1002/nem.543)

Scalable multicast based filtering and tracing framework for defeating distributed DoS attacks

By Jangwon Lee,† and Gustavo de Veciana*

In this paper we present a distributed scalable framework to support on-demand filtering and tracing services for defeating distributed denial of service attacks. Our filtering mechanism is designed to quickly identify a set of boundary ***filter locations so that attack packets might be dropped as close as possible to their origin(s). We argue that precisely identifying the origins of an attack is not achievable when there is only a partial deployment of tracing nodes—as is likely to be the case in practice. Thus we present a tracing mechanism which can identify sets of candidate nodes containing attack origins. Both mechanisms leverage multicasting services to achieve scalable, responsive and robust operation, and operate with a*** partial ***and*** incremental ***deployment.***
Performance evaluations of proposed approaches on both real and synthetic topologies show that a small coverage of filtering and tracing components throughout a network can be effective at blocking and localizing attacks. Copyright © 2004 John Wiley & Sons, Ltd.

1. Introduction

Denial of Service (DoS) attacks are one of the greatest threats to today's Internet. They not only degrade performance but deprive legitimate users of basic access to network services. As seen in frequent news headlines attacks are becoming increasingly prevalent and evolving since the first spectacular attack on high-profile web sites in February 2000.[1] Despite their diverse character, such attacks share a common feature: they exploit defects or weaknesses of various network components ranging from applications, operating systems to protocols. Attacks using implementation defects or bugs in network components, can be prevented by frequent system updates and software patch work. However, it is much harder to thwart attacks which exploit intrinsic vulnerabilities and characteristics of the existing IP infrastructure.

For example, in IP, irrespective of a receiving side's intent, any host can basically send packets to any other host provided that those hosts are connected to the Internet. This simple feature of IP allows attackers to launch DoS attacks by simply inundating victims with large amounts of useless traffic. In turn, such traffic consumes network resources along the way to victims and eventually degrades performance for other users sharing these network resources. In a distributed DoS (DDoS) attack, i.e., one which is carried out by multiple compromised hosts, the damage can become exceedingly detrimental. Moreover, IP has no mechanism for checking or controlling the correctness of the sender's address. This facilitates *spoofing*, i.e., placing incorrect source IP addresses

Jangwon Lee works in the Computer Science Department, North Carolina State University.

Gustavo de Veciana works in the Electrical and Computer Engineering Department, University of Texas at Austin.

**Correspondence to: Jangwon Lee, Computer Science Department, North Carolina State University, USA.*
†E-mail: jangwlee@ece.utexas.edu

in packets to conceal the true origins of the packets. Furthermore, it is difficult to identify the physical locations of attacks in an IP network due to its stateless nature (even without spoofing), i.e., routers forward packets based on destination addresses alone and maintain no state information on traffic flows. The identification of the origins of attacks is even more difficult in the case of a DDoS attack where attackers may inject multiple, identical packets at multiple locations.

The critical innovation in DDoS attacks is its distributed nature. Even a small number of attack packets from each compromised host can eventually become a large traffic flow, inundating a target system. We argue that the solutions to thwart DDoS attacks should also be *distributed* to be effective. The *local* solutions on the victim computer or in its local network without outsider's cooperation, can neither identify where the packets are coming from nor effectively mitigate the possibly large volume of attack traffic.

We argue that the solutions to thwart DDos attacks should also be distributed to be effective.

One way to deal with such attacks is *proactive* filtering.[2,3] The key idea is to configure routers to drop spoofed packets whose source IP addresses are inconsistent with the network topology. Note that the strength of this approach is its proactiveness, i.e., attacks can be eliminated before they affect victims. However, if DoS attacks are infrequent, most resources allocated to proactive filtering are wasted except when spoofed packets are actually dropped. Furthermore, attackers may evade proactive filtering by forging IP addresses using hundreds or thousands of legitimate host addresses within a given domain.

By contrast, in this paper our focus is on two *reactive* approaches: *on-demand filtering* and *tracing*. Our approach is *reactive* in that actions are initiated after an attack reaches a victim. Both tracing and on-demand filtering mechanisms can make up for the above-mentioned deficiencies of IP networks. That is, a tracing mechanism can identify the true origins of an attack and a filtering mechanism can enable a host to request unwanted packets to be dropped early on, before they reach the victim.

The following are some desirable characteristics that on-demand filtering and tracing mechanisms should have.

- *Scalability.* Mechanisms should be scalable to benefit a co-operation across a number of different administrative domains.
- *Promptness.* By the nature of reactiveness, filtering and tracing should be performed quickly before the victim is seriously damaged or there no longer exists a trail of information.
- *Flexibility.* Mechanisms should allow heterogeneous equipment and proprietary operation across different administrative domains.
- *Distributed.* From the perspective of robustness, it is preferable that mechanisms be implemented in a distributed manner rather than relying on central points.
- *Partial and incremental deployment.*

Based on the above desirable characteristics, in this paper we propose a framework which provides filtering and tracing services to aid in thwarting DDoS attacks. We set two goals in the paper. The first one is to block attack packets as close as possible to attack origins by way of filtering components that are distributed over the Internet. Second, under a *partial* deployment environment where only a subset of routers are tracing-enabled, it becomes impossible to pinpoint the precise origins of attack packets. Thus, our objective for tracing is to identify sets of candidate nodes containing attack origins. The proposed solutions are based on multicasting service to achieve a number of desirable characteristics, e.g. scalability, distributedness, quick response and robustness. Furthermore, they rely on existing monitoring and filtering mechanisms, allowing heterogeneity from different network domains.

This paper is organized as follows. Section 2 introduces components and attack models used throughout the paper. In Sections 3 and 4, we discuss our objectives and our solutions for filtering and tracing mechanisms respectively. Section 5 includes a performance evaluation for the proposed framework and is followed by Section 6 wherein we include comments on implementation and various possible modes of operation. In Section 7, we discuss the advantages and short-

comings of previous approaches to defeating and mitigating attacks and contrast these with our work. Section 8 concludes the paper.

2. Models

—2.1. Attack Model—

Consider an attack whose target is a node, v, referred to as the *victim*. A victim can be an end host, a router or a network border device such as a firewall. In the sequel, we will refer to the set of *attack nodes* A as those participating in the attack. We will leave the initiation time of the attack unspecified. Thus, the tuple of the set of attack nodes and the victim, i.e., (A, v) represents an attack incidence. In the case of a distributed attack, $|A|$ is greater than 1 and the locations of attack nodes are potentially widespread. For each attack node, say, $a_i \in A$, the *attack path* for a_i is an ordered list of nodes traversed by attack packets from a_i to v, excluding a_i itself. An *attack graph* induced by (A, v), denoted by $AG_{(A,v)}$, consists of all links and nodes traversed by attack packets associated with nodes in A and the victim v. For example, in Figure 1, (r_3, r_4, r_5, r_6, v) is a_3's attack path and the dotted lines represent an example of the attack graph for attack incidence $(\{a_1, a_2, a_3\}, v)$. Attack paths and graphs may vary during the attack period due to routing instabilities and dynamic attack patterns. Throughout this paper, an *attack origin* is referred to as a local router to which the attack node is attached. For example, r_1 is the attack origin of a_1 in Figure 1. We will refer to an *attack signature* $AS_{(A,v)}$ as a common feature shared by attack packets generated from A. For example, for a smurf DoS attack case, an attack signature could be ICMP echo protocol and a range of source IP addresses.[4] For example, a smurf attacker sends a stream of ICMP echo requests to the broadcast address of the reflector subnet. Since the source addresses of these packets are falsified to be the address of the target, many hosts on the reflector subnet will respond, flooding the target. Thus, the source addresses of the echo reply packets are clustered in a few address prefixes. Note that extracting an attack signature is a crucial starting point for most reactive mechanisms and it is not easy to obtain a clear and well-matching attack signature with real attacks. However, in practice, common rough characteristics for attack packets can be generally identified and used for filtering or tracing purposes.[5] In the case where extracting the attack signature is impossible, the attack packet itself can be used for the signature, as with the approach in Reference 6. For mitigating DDoS attacks without attack signatures, refer to Reference 7.

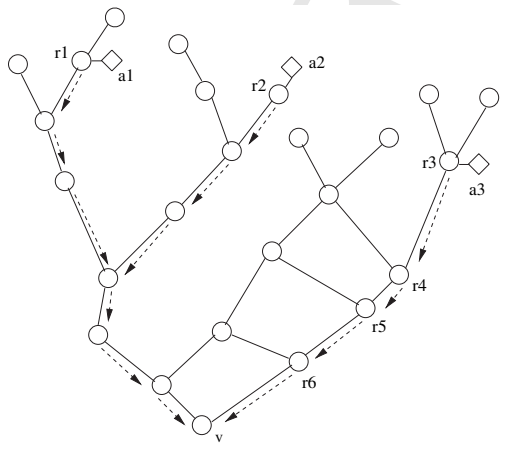

Figure 1. An example of an attack incidence

—2.2. Framework Model and Assumptions—

Our framework includes the following components.

- *Detecting component.* Reactive approaches to DoS attacks require a component which can detect incidences of attacks and generate an attack signature by extracting a feature shared by attack packets. This attack signature is further used in the tracing and filtering processes.
- *On-demand filtering component.* An on-demand filtering component drops packets conforming to a rule set derived from an attack signature.
- *Tracing component.* When a tracing component is queried for a given attack signature, it can check for the existence of packets with the attack signature in the past or current traffic traversing the component.

In our framework, we assume that the above components have multicasting-enabled functionality. In this paper, we assume that these components can communicate with each other in a

secured manner. How to secure multicasting channels is not in the scope of this paper. Refer to References 8–10 on this issue. Multicasting can be supported at either network-layer or application-layer,† and there are various available implementation methods for each component. We defer those discussions to Section 6. Note that it is not our intent to specify multicasting methods or propose a specific or new implementation method for the above components. The main goal of this paper is to propose a framework which provides flexible filtering and tracing services.

We refer to entities which provide detecting, filtering and tracing services to victims as *detector*, *filter* and *tracer* which are equipped with detecting, on-demand filtering and tracing components, respectively. Multiple functionalities can be co-located and perform multiple roles. Note that each entity is associated with a location of interest. That location may be an end host, a router or a link depending on the implementation. Without loss of generality, throughout the paper, we assume that filters and tracers are located at internal nodes (i.e., routers), and detectors are co-located at victim nodes. Usually detector nodes, D, will be located at highly-defended and secured strong points such as high-profile web servers or network entry points. Filter nodes, F, and tracer nodes, T, could be located at network entry points or distributed all over the Internet. In our framework, we assume a *partial* deployment of components over the Internet, i.e., only a subset of nodes are equipped with detecting, filtering and tracing functionalities—this is deemed a realistic environment.

Given a particular attack incidence (A, v), any set of nodes S can be partitioned into two subsets: *positive* and *negative* nodes, where positive nodes are on the attack graph of (A, v) and negative nodes are not. That is, positive nodes, denoted by $S_{(A,v)}$, are given by $S \cap AG_{(A,v)}$. For a given set of nodes S and $a_i \in A$, we define a *boundary* node, $b(a_i)$, as the node in S that is first encountered by attack packets making their way towards a victim v. We refer to a *boundary interface* as the interface where attack packets enter in a boundary node. We denote by $BS_{(A,v)}$ the collection of boundary nodes

†*As will be seen in the sequel paper, for network-layer multicasting, our framework does not require reliability due to its soft-state property.*

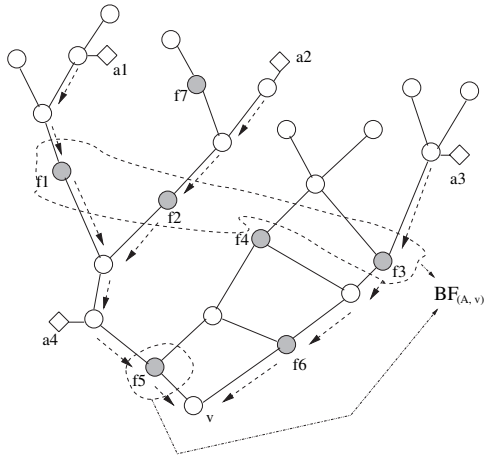

Figure 2. An example of filter deployed network

associated with an attack incidence (A, v). Letting S be a set of filter nodes, F, or a set of tracer nodes, T, we can obtain sets corresponding to positive, negative and boundary filter (tracer) nodes associated with a given attack incidence. For example, in Figure 2 the filter nodes (shadowed nodes) have been deployed in the network shown in Figure 1. For an attack incidence $(A, v) = (\{a_1, a_2, a_3, a_4\}, v)$, $F_{(A,v)} = \{f_1, f_2, f_3, f_5, f_6\}$, $b(a_1) = f_1$ and $BF_{(A,v)} = \{f_1, f_2, f_3, f_5\}$.

3. On-Demand Filtering

—3.1. Objective—

The objective for on-demand filtering considered in this paper is to block attack packets as close as possible to the attack nodes by using a set F of co-operative filter nodes distributed over the Internet. One might consider a centralized solution wherein each detector maintains and queries all the filters, F, over the Internet. However, this approach may incur high overheads and seems to scale poorly when there is a large number of filter or attack nodes. We observe that boundary filter nodes play a key role in achieving the objective, since they are the filtering nodes first met by attack packets from A to v. That is, performing filtering only at boundary filter interfaces is resource-efficient while blocking attack packets as early as possible given the available set of filters F. Thus, the filtering objective becomes a resource discov-

ery problem, i.e., finding boundary interfaces given a set of filter nodes, F, and a particular attack (A, v).

The objective for on-demand filtering is to block attack packets as close as possible to the attack nodes.

—3.2. Our Solution—

3.2.1. Filter multicast session—The key idea in our filtering solution is that all filter nodes subscribe to a *filter multicast session*. As with the '911' telephone number allotted to serve police, fire and emergency situations in the United States, in our approach, a filter multicast session is dedicated to a communication channel to support on-demand filtering service.

Once an attack, (A, v), is launched and detected by a victim v, it joins the filter multicast session and requests filtering service by multicasting a *filter request packet*. In the case where a detector and a victim are not co-located, the detector will act as an agent on behalf of the victim. For simple exposition, we assume that the victim detects attack and requests filtering operation before attack packets may force the victim to give in. The request packet contains an attack signature, $AS_{(A,v)}$, used to generate appropriate filtering rule sets, and a *filtering period* d_v, the desired duration over which filtering is to be performed. Once each filter receives an *initial* request packet, it associates its interfaces with either the ON or OFF state based on the following decision rule (Figure 3).

The state of each interface is basically determined according to whether it carries the attack packets over two equal-sized intervals at the beginning and the end of the filtering period: $[t, t + d_i]$ and $[t + d_v - d_i, t + d_v]$. Note that the range for α in Line 2 guarantees that there is no overlap between two intervals.

Given an attack incidence, (A, v), interfaces at filter nodes can be classified into three types: (1) negative: whose interface state should be OFF after the first interval (Line 6), (2) boundary: whose interface state should be ON after the second interval (Line 12), and (3) positive but not boundary:

```
1:  { For each interface, i, except the one receives the re-
    quest packet}
2:  { t : filtering initiation time, d_i = α d_v (0 < α ≤ 0.5) }
3:  { install a filtering rule set conforming to AS_(A,v) at t}
4:  if no attack packets dropped in [t, t + d_i] then
5:      stop filtering
6:      make i state OFF
7:  else
8:      keep filtering
9:      if no attack packets dropped in [t + d_v − d_i, t + d_v]
        then
10:         make i state OFF
11:     else
12:         make i state ON
13:     end if
14: end if
```

Figure 3. Interface state decision algorithm

whose interface state should be OFF after the second interval (Line 10). The third type may happen because filtering requests are handled in a distributed and asynchronous manner at each filter node. Note that in the above decision rule, even a single packet matching with the attack signature enables an interface to be put in the ON state. One may consider a threshold method, i.e., only when the number of packets matching the attack signature is larger than some specified threshold value, does the interface enter the ON state.

Each filter node which has at least one interface in the ON state (referred to as ON filter node) sends a *filtering report* packet to the victim containing some filtering statistics, e.g., the number of packets dropped on each of its ON interfaces. Based on report packets (whether the attack persists or not), the victim can renew its filtering request by multicasting another filtering request packet. As the state of each interface at filter nodes has been decided after the initial filtering request packet, OFF filters simply ignore the request, while ON filter nodes keep filtering at ON interfaces until either a filtering timeout (d_v) expires, or a renewal arrives, in which case the filtering period is restarted and again a filtering report packet is sent to the victim.

Note that state information should be preserved long enough that the next filtering request packet arrives, i.e., next renewal. However, it should be eventually eliminated if no renewals arrive. Thus,

once a node gets a filtering request packet, it restarts the *state elimination timer* (e.g., $3*d_v$). After the timer expires (i.e., no renewal packet prior to time-out), it eliminates state information associated with the attack signature, $AS_{(A,v)}$.

3.2.2. Adaptiveness—In the above protocol, filter states are determined upon receipt of an initial request packet, and remain fixed before they expire. However, even for the same attack incidence, the boundary filter nodes may change due to (1) dynamic attack patterns, e.g., an attacker may have a strategy that periodically turns on and off some attack nodes, and (2) routing instability, i.e., packets injected by the same host to the same destination may travel different paths. This requires a filtering mechanism to adapt to dynamic changes in the set of boundary filter nodes (interfaces).

To this end, we include a *reset flag* in the filtering request packet and add the following behavior at filter nodes: (1) if the received request packet's reset flag is 0, then perform the state decision procedure for only `ON` interfaces, and, (2) if the reset flag is 1, then perform the state decision procedure for all the interfaces ignoring previously determined states. These rules ensure a transition from `ON` to `OFF` and a transition from `OFF` to `ON`, respectively. A failure in this state transition might eventually be detected by the victim, since attack packets may reach the victim.[‡] Thus, the victim node will periodically send filter request packets with the reset flag set to either 1 or 0 depending on whether it is seeing attack packets, or not, until the attack is suppressed or ends.

4. Tracing

—4.1. Objective—

The ultimate goal of a tracing mechanism is to identify attack nodes, e.g., $\{a_1, a_2, a_3\}$ for the example in Figure 1. However, the goal is usually relaxed to determining the origins of the attack, i.e., $\{r_1, r_2, r_3\}$. This is because tracing-enabled functionality is usually associated with routers, and MAC address spoofing is possible. Most existing tracing approaches[6,11,12] focus on developing mechanisms which can discover an attack graph. Once the attack graph is discovered, the origins of the attack can be pinpointed. However, precisely pinpointing the origins of an attack is not achievable when there is only a partial deployment of tracing nodes—as is likely to be the case in practice. Thus, rather than identifying the exact attack graph, we set up our tracing objective as that of localizing attack nodes by providing sets of *candidate* nodes, possible attack origins. The effect of inaccurate map and dynamic routing of packets will be discussed in Section 6.

—4.2. Our Solution—

4.2.1. Localization—First, we define the notion of 'candidate nodes' used in this paper. Candidate node sets are determined based on the set of boundary tracer nodes. Recall that boundary tracer nodes $BT_{(A,v)}$ are simply tracer nodes which are first met on the path of attack packets associated with given attack incidence (A, v).

Let M_v denote a network map (tree) of upstream nodes rooted at the victim, v. For any node $m \in M_v$, let M_m denote the subtree rooted at m. For each boundary tracer node $n \in BT_{(A,v)}$, let the set of *candidate nodes* of n, C_n, be

$$C_n = M_n \setminus \bigcup_{x \in M_n \cap T} M_x$$

Note that C_n is obtained by subtracting all subtrees rooted at descendant tracer nodes of n from the subtree rooted at n. For example, in Figure 4 where tracer nodes are shown in black, boundary tracer nodes are $\{r_3, r_{11}, r_{17}, r_7\}$ and their sets of candidate nodes are $C_1 = \{r_3, r_2\}$, $C_2 = \{r_{11}, r_{10}, r_{12}\}$, $C_3 = \{r_{17}, r_{15}, r_{16}\}$, $C_4 = \{r_7, r_6, r_4\}$, respectively.

We further define a node c to be a *boundary negative tracer* node with respect to $n \in T_{(A,v)}$ if

1. c is a negative tracer node,
2. n is on the path from the root, v to c, and
3. there are no other nodes in T on the path from n to c.

For example, r_3's boundary negative node is r_1 while r_7 has no boundary negative tracer nodes in Figure 4.

[‡]*Receiving attack packets at the victim side does not necessarily indicate the failure of state transitions. Consider the case where there is no filter node along the way from the attack nodes to the victim. However, we do not assume such a case.*

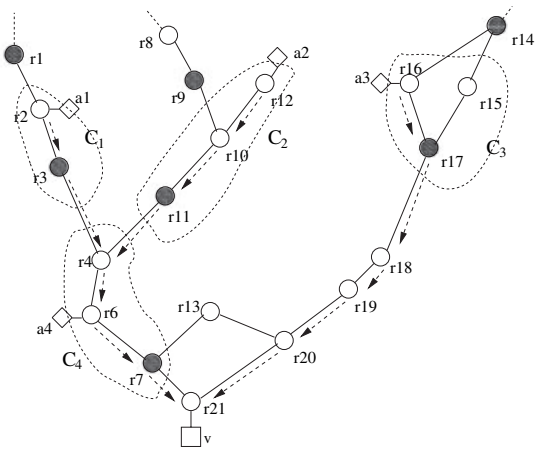

Figure 4. An example of tracer deployed network

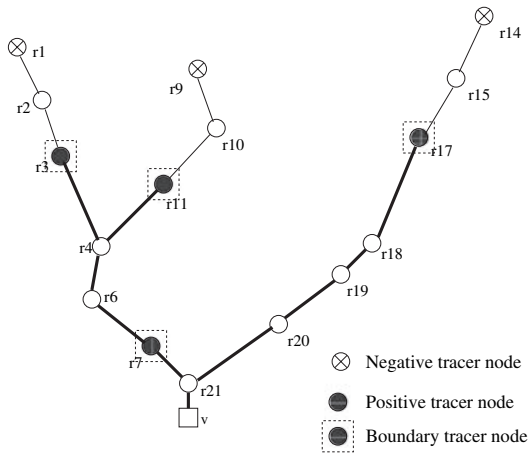

Figure 5. An expanded attack graph

The key idea underlying our tracing approach is that a set of candidate nodes contains at least one origin of an attack. For example, consider the attack node a_1 in the network shown in Figure 4 where tracer nodes are shown in black. With the following information: (1) r_3 is a boundary tracer node, (2) r_1 is a negative tracer node, and (3) the network topology, one can conclude that $\{r_2, r_3\}$ is a set of nodes which contain an attack origin(s).

A *positive* attack graph is defined as the collection of all links and nodes traversed by packets from each node in $T_{(A,v)}$ (i.e., positive tracer nodes), to the victim v. We define an *expanded* attack graph as the positive attach graph plus the collection of all links and nodes traversed by packets from boundary negative nodes to the victim. For the attack incidence in Figure 4, its expanded attack graph is depicted in Figure 5. Thick links and nodes from the victim to positive tracer nodes comprise the positive attack graph.

Note that sets of candidate nodes can be obtained with the following information: (1) a network map of upstream nodes to the victim, (2) an *expanded* attack graph, and (3) boundary tracer nodes. By overlapping the two graphs, i.e., network map and expanded attack graph, we can identify candidate node sets. The network map of upstream nodes to the victim can be obtained using a tool such as Skitter[13] or that developed in Reference 14. Thus the remaining task is to obtain an expanded attack graph and identify the boundary tracer nodes.

4.2.2. Obtaining expanded attack graphs—Note that obtaining a positive attack graph is the goal of existing tracing approaches.[6,11,12] In this section, we propose a new approach to obtaining a positive attack graph and then extend it to determining the expanded attack graph.

As with our filtering approach, we propose to have a multicast session support tracing services, referred to as a *tracer multicast session*. This session is joined by the set of tracer nodes, T. Once an attack incidence, (A, v) is detected at the victim, then v joins the tracer multicast session and sends a *tracing request packet* containing a signature, $AS_{(A,v)}$.[§] Upon receiving the tracing request packet, each tracer node checks whether it has carried attack packets conforming to the attack signature.

In order to obtain path information from tracer nodes to a victim, we propose to use the *traceroute* program. Traceroute provides an executing node with a forward path information from it to a destination. After checking whether it is seeing any attack traffic, each positive tracer node n in $T_{(A,v)}$, performs a traceroute toward the victim, which enables it to identify the forward path from n to v. Then n sends a *tracing report* packet including this path information to the victim. By collecting paths reported by positive tracer nodes, the victim can construct a positive attack graph—a subset of an

[§]*Tracing request packets may contain the attack incidence time for post-mortem tracing if it is supported.*

attack graph. A simple approach to obtain an expanded attack graph is to have every negative tracer node also perform a traceroute to the victim and send a tracing report packet. However, this will not scale when there is a large number of (negative) tracer nodes in a network. Thus, we propose the following mechanism. Once a tracer node is determined to be positive, after some time, the node performs a *scoped* multicast by sending a *tracing solicit packet* to its neighborhood with Time-to-Live (TTL) set to some value, k. The TTL value is first specified in the original tracing request sent by the victim. Upon receiving the tracing solicit packet, a positive tracer node simply ignores it, but a negative tracer node performs a traceroute to the victim and sends a tracing report packet to the victim. Tracing report packets require a flag representing whether they were generated by a positive or negative tracer node. Note that this mechanism makes negative tracer nodes within k TTL distance from positive tracer nodes participate in the tracing operation. However, this mechanism may only produce an *approximate* expanded attack graph. This depends on the scope of the tracing solicit packet and the locations of tracer nodes. For example, if the TTL value is too large or tracer nodes are closely located, unnecessary negative nodes may be included in the graph. However, in this case, one can still identify candidate nodes. In the other case, i.e., where the TTL value is too small or tracer nodes are too far away, one fails to obtain a set of candidate nodes. Reflecting this practical situation, we classify sets of candidate nodes into two classes: *closed* or *open*, i.e., identified or unidentified candidate nodes respectively from the perspective of a victim. This classification depends on the deployment of tracer nodes, i.e., how many and where, as well as which TTL value is used. To reduce the number of open sets, we can envisage the following scheme: after the victim recognizes the existence of sets of candidate nodes which are open, it may perform another tracing operation with a larger TTL value. However, even though this can reduce open sets to closed ones, too large a number of nodes in a closed set requires lots of search effort within the set, which in turn makes localization expensive.

4.2.3. Identification of boundary tracer nodes—Once one obtains an expanded attack graph, it is clear how to identify boundary tracer nodes residing at the end of a positive attack graph, e.g., r_3, r_{11} and r_{17} in Figure 5. However, it is difficult to find boundary tracer nodes residing at internal nodes of a positive attack graph, e.g., r_7 in Figure 5. This is because the packets generated at attack nodes associated with a boundary tracer node cannot be differentiated from packets injected at upstream attack nodes. Note that this problem exists even with networks in which tracers are fully deployed. For example, in Figure 4, a_4's attack origin is invisible unless tracing components have an ability to not only check if an attack packet has passed but also acquire information about which interface(s) attack packets are flowing through. This problem can be handled by the following methods: (1) filtering and tracing can be performed jointly or (2) packets from different attack nodes can be further differentiated, e.g., instead of using the attack signature shared by all attack nodes, using packets themselves may differentiate those attack packets.

5. Performance Evaluation

We conducted three sets of simulations varying topologies and placement strategies. The objective was to explore two questions: how many and where should filters and tracers be deployed in the network to be effective for blocking and localizing attacks? We do not claim these experiments are comprehensive. Instead, our goal is to provide insights on how the proposed framework performs in several representative settings. Below we first discuss performance metrics for the proposed filtering and tracing framework, and then present our simulation results.

—5.1. Performance Metrics—

5.1.1. Filtering metrics—We define the *cost* of an attack node, a_i, as the product of two quantities: the amount of attack traffic injected at a_i, $w(a_i)$ and the distance (i.e., the number of hops) from the attack node to the victim, $d(a_i, v)$. This can be roughly considered as a measure of network resources (e.g., bandwidth, routers' CPU cycles, etc.) that are wasted in processing attack packets from a_i to v. For an attack incidence (A, v), the *total attack cost* $c(A, v)$ is

$$c(A,v) = \sum_{a_i \in A} w(a_i) d(a_i, v)$$

When filter nodes F are deployed and filtering is conducted at the boundary, the total attack cost $c(A, v, F)$ is reduced to

$$c(A,v,F) = \sum_{a_i \in A} w(a_i) d(a_i, bf(a_i))$$

where $bf(a_i)$ is the boundary filter node associated with attack node a_i. To demonstrate the amount of traffic that can be blocked using filtering, we define the *relative attack* cost γ as the ratio of the cost with filters deployed to the cost without, i.e., $\gamma = c(A, v, F)/c(A, v)$. Note that this cost metric is from the perspective of the network rather than the victim. Although the proposed metric is simple, it captures well features associated with attack traffic aggregation. For example, consider a local solution where there is a filter installed at an ingress point only a few hops away from the victim. According to this cost function, γ will be high, capturing the situation where the attack is still effectively disrupting the victim's local network.

5.1.2. Tracing metrics—Let C_i be a closed set of candidate nodes and A_{C_i} be the set of attack nodes in C_i. For a given attack incidence (A, v) and set of tracer nodes T, suppose k of closed sets of candidate nodes are obtained after tracing. In this case, the victim will have to search for attack nodes within $C_1, \ldots, C_k$. We shall define two metrics. The first one, Φ_1, is given by

$$\Phi_1 = \frac{\sum_{i=1}^{k} |A_{C_i}|}{|A|}$$

Note that Φ_1 is the ratio of the number of identified attack nodes to the total number of attack nodes. Here, $(1 - \Phi_1)$ is the portion of attack nodes that lie in open candidate sets, and are assumed to remain undetected. To capture how 'localized' the portion of identified attack nodes is, we define a second metric, Φ_2 as

$$\Phi_2 = \frac{1}{k} \sum_{i=1}^{k} \frac{|C_i|}{|A_{C_i}|}$$

Note that $|C_i| / |A_{C_i}|$ represents the *search effort*, i.e., the average number of nodes to be searched in order to determine attack node(s) in C_i. Thus, Φ_2 is the average search effort over the closed set of candidate nodes. In summary, Φ_1 represents how many attack nodes are identified and Φ_2 captures the search effort or degree of localization achieved by the tracing mechanism.

—5.2. Simulation Results—

For an attack incidence (A, v), we randomly select $|A|$ nodes (excluding the victim) in a given topology to be attack nodes. We assume that attack nodes generate the same amount of attack traffic, i.e., $w(a_i) = c$, $i = 1, \ldots, |A|$. We have built a tool to estimate the performance measures proposed in Sections 5.1.1 and 5.1.2. For all of our results, each performance metric value is the average value of 100 simulation runs (attack incidences). For the results associated with tracing, we set 5 as the TTL value for the solicit tracing packets. For simplicity, we assume that filtering is also performed at boundary tracer nodes, which can identify all boundary tracer nodes. In the simulation, we do not consider dynamic changes in the configuration of attack nodes or routes of attack packets during each attack incidence.

5.2.1. Simulation I—In this simulation we use a real tree topology from Reference 15. The tree was obtained by performing traceroutes at the server, `www.bell-labs.com`, to its clients and consists of around 23,000 distinct nodes. We considered a scenario where randomly placed nodes launch an attack to the server. For the placement of tracer or filter nodes in the network, we choose a random strategy for a given *coverage ratio* β, i.e., β is the portion of total nodes in the network that are filter or tracer nodes.

Figure 6 shows the relative attack cost, i.e., γ, resulting from an attack involving 25 nodes and varying filter coverage ratios. Note that it has a convex shape, i.e., a small increase in coverage ratio can cause a large reduction in the attack traffic when the coverage ratio is small. We observe that one can reduce the attack traffic by 80% (relative attack cost is 0.2), with a filter coverage of 30%.

Figure 7 shows γ for several coverage ratios and a varying number of attack nodes from 25 to 1600. We observe that γ is independent of the number of attack nodes. To explain this result, we notice that $c(A, v, F)$ is roughly given by $|A| w(a_i) E[d^*]$ where

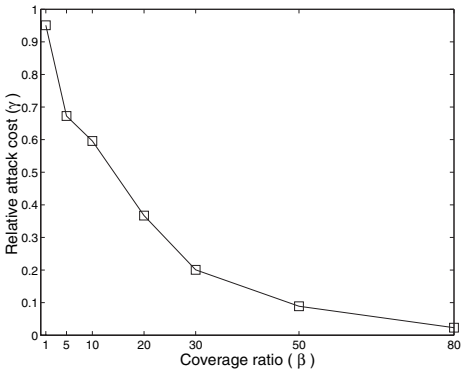

Figure 6. Relative attack cost ($|A| = 25$) in Simulation I

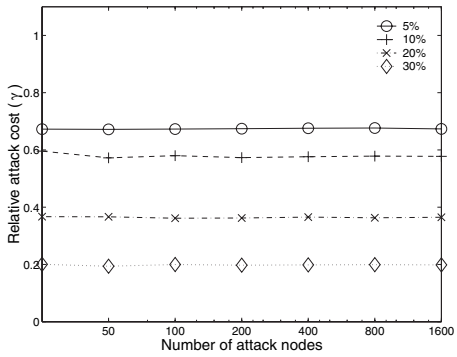

Figure 7. Relative attack cost in Simulation I

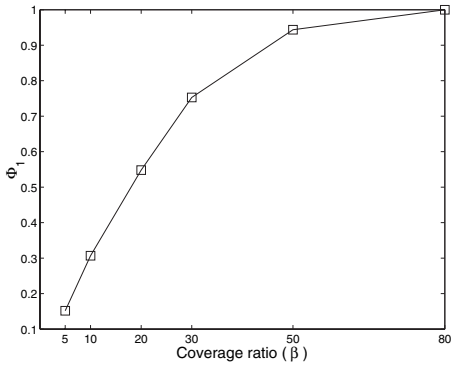

Figure 8. $\Phi_1(|A| = 25)$ in Simulation I

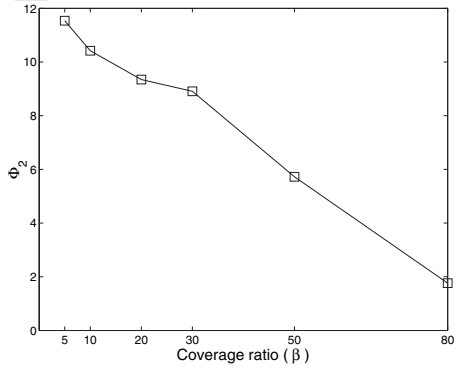

Figure 9. $\Phi_2(|A| = 25)$ in Simulation I

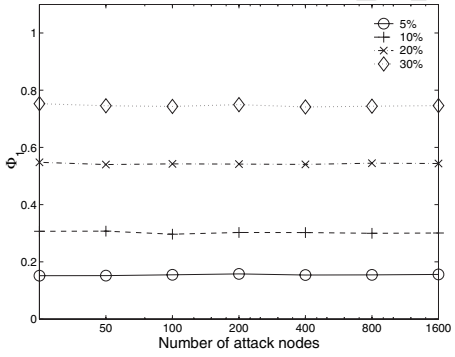

Figure 10. Φ_1 in Simulation I

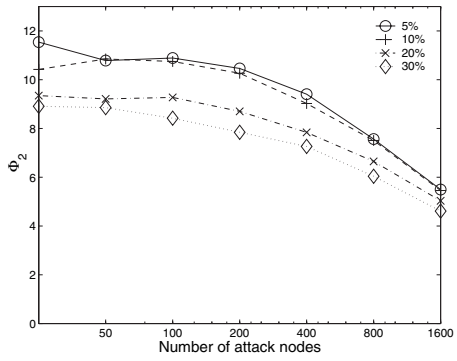

Figure 11. Φ_2 in Simulation I

$E[d^*]$ is the expected distance from attack nodes to their boundary filter nodes. Likewise $c(A, v)$ becomes $|A|w(a_i)E[d]$ where $E[d]$ is an expected distance from attack nodes to the victim. Since we assume $w(a_i)$ is constant, γ simply depends on the ratio of the above two distances.

Figures 8 and 9 show Φ_1 and Φ_2, respectively, for an attack of 25 nodes. At a 30% coverage ratio, around 75% attack nodes can be detected and on average attack nodes can be localized to within 9 nodes. Figures 10 and 11 show results for Φ_1 and Φ_2, respectively, for various coverage ratios

varying the number of attack nodes. We make the following observations. First, Φ_1 is independent of the number of attack nodes. Second, Φ_2 is decreasing as the number of attack nodes increases. These can be explained as follows. As the number of attack nodes increase, one can see more attack nodes will be placed in closed sets of candidate nodes, which ensures more attack nodes will be detected. Thus Φ_1 is likely to be independent of the number of attack nodes. Furthermore, Φ_2 will decrease since there are more attack nodes found in the same closed candidate sets as the number of attack nodes increases.

5.2.2. Simulation II—For the second set of simulations, we generated a 2000-node random transit-stub graph using GT-ITM.[16] This topology is composed of interconnected transit and stub domains where domains are assumed autonomous. In this set-up, we randomly choose a victim node for each attack incidence. To investigate the performance impact of filter or tracer location, we explore a *border* placement strategy, i.e., randomly choose the location of filter or tracer nodes among border nodes of domains versus randomizing over all possible locations. This reflects the case where a network administrator of a domain decides to provide services, the administrator is likely to place filter or tracer nodes at border nodes rather than at random locations within the domain.

The results for different numbers of attack nodes show the same qualitative behavior as those in the previous experiments. We exhibit the results for a 50 node attack. Figure 12 shows γ for various placements. For the same coverage ratio 5% and 10%, border placement performs better than random, and border placement with 15% coverage outperforms random placement with 30% coverage. Figures 13 and 14 show results for Φ_1 and Φ_2, respectively. Here, we obtained an encouraging result that even a 15% deployment of tracer nodes at border nodes can detect more than 90% of the attack nodes and on average attack nodes are localized to within 10 nodes. Additionally, we made the following observations. First, border placement can detect more attack nodes than random placements. Second, all random placements have a lower Φ_2 value than border placement. Indeed even a small coverage ratio of border placement can create a well-balanced number of candidate sets. This reduces the number of open candidate sets, which achieves high Φ_1. Compared to this, with random placement, the number of nodes in the candidate nodes set vary more dramatically. This results in small Φ_1, which identifies the smaller number of open candidate sets.

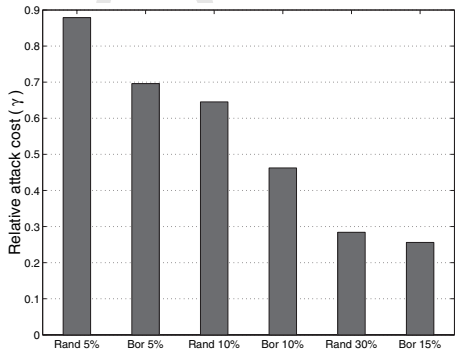

Figure 12. Relative attack cost ($|A| = 50$) in Simulation II

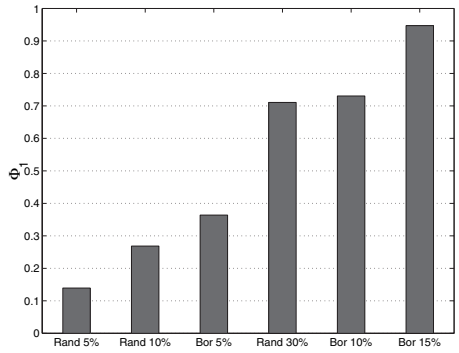

Figure 13. $\Phi_1(|A| = 50)$ in Simulation II

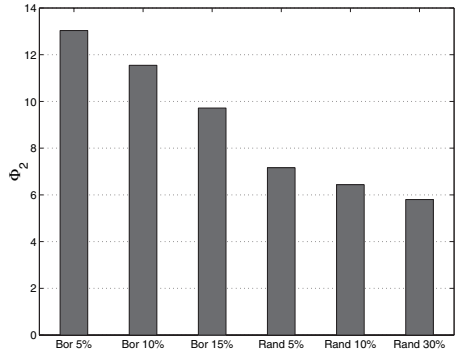

Figure 14. $\Phi_2(|A| = 50)$ in Simulation II

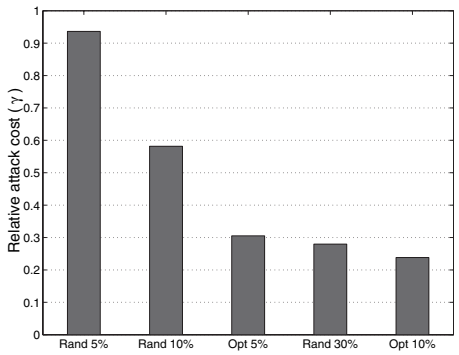

Figure 15. Optimal vs. Random ($|A| = 25$) in Simulation III

However, once it is identified to be open, its search effort becomes less, leading to a smaller Φ_2 value.

5.2.3. Simulation III—Finally, we performed filtering simulations on a 145-node routing tree rooted at the server, `www.bell-labs.com` in Reference 15. The motivation for this set of simulations is to explore the case where the victim (the server) has full control over the placement of filter nodes on a mid-size network.

We considered an *optimal* placement to be one that minimizes the total attack cost under the assumption that each node in the network can be an attack node and generate the same amount of attack packets. This problem is equivalent to the cache location problem studied in References 15 and 17, thus we used their dynamic programming formulation to find optimal placements.

Figure 15 shows the relative attack cost results of 25 attack nodes for random versus optimal placements. As can be seen, optimal placements outperform random. A 10% coverage with optimal placement can reduce attack traffic by 80%.

6. Discussion

—6.1. Implementation Issues—

In this section, we briefly present various implementation methods for the components described in Section 2.2. A number of intrusion detection systems (IDS), or identification algorithms,[18] can serve as detecting components. Note that the ability to obtain attack signatures from attack packets is a critical requirement for our framework and is itself a significant on-going topic of a number of intrusion detection systems. For tracing components, input-debugging[19] can check tracing results only for on-going traffic. By contrast, logging[19] and hash-based logging[6] methods can provide a *post-mortem* tracing service which can check if even past traffic (before a query) contains attack packets. Furthermore, one can envisage that stand-alone data capture devices (e.g., RMON probes[20]) or sniffers (e.g., tcpdump) can be used to provide on-going tracing services in a nonintrusive way, i.e., not affecting routing performance. (See Section 7 for more detailed explanations for each tracing method.) Finally note that there are various filtering services available which operate at different layers of the protocol stack—e.g., IP-level, TCP-level and application level.

—6.2. Inaccurate Map and Dynamic Routing—

As seen in Section 4, our tracing mechanism relies on a map of upstream routers and traceroute results. In this section, we consider the impact of inaccurate information on our tracing mechanism. Even though the network map can be obtained using the tools described in Section 4.2.1, we observe that it is difficult to obtain an updated accurate Internet topology. However, since our goal is not to pinpoint the exact attack origins, such a map does not have to be perfect. Furthermore, note that even without a map, the expanded attack graph itself can be a useful tool for identifying the regions where attacks originated. Also, a map can be obtained in a post-mortem manner after an attack ends.

Note that traceroute may not work as desired due to the manner in which routers are configured along given paths. In addition, traceroute may not provide an accurate attack path due to changes in routing and attack patterns on parallel routes. However, as mentioned above, our goal is to obtain a set of candidate nodes and not to identify the exact attack origins. Thus, inaccurate path information can be still useful. Furthermore, if traceroute is performed while an attack continues, attack packets and traceroute packets will be forwarded based on the same unicast routing table, which produces a more accurate approximate attack graph.

—6.3. Support for Multicasting—

Multicasting can be supported by the following two classes: network-level (IP multicast) and application-level solutions.

6.3.1. IP multicast routing protocol—IP multicast[21] is an efficient one-to-many delivery method which can provide a number of operational advantages for content and network providers by reducing the overall resources consumed to achieve such distribution. A single packet transmitted by the source traverses each link in the multicast distribution tree to all receivers in the multicast group. In this section, we consider which IP multicast routing protocol will be suitable for our framework. Current implementations of multicast routing service, can be classified into two types: source tree and shared tree routing. In source tree routing, the distribution tree is a *reverse shortest-path tree* which is formed by overlaying shortest path from each member to a source. In shared tree routing protocols, e.g., CBT[22] or PIM-SM,[23] the distribution tree is commonly shared by all members irrespective of the sources. Note that in our approach, during a time without attacks, there are no data packets injected into a multicast session. This is because the multicast session is not used for data distribution among members but only for tracing or filtering request distributions from unpredictable victims. Therefore, the cost for maintaining the multicast session becomes an important issue when selecting a multicast routing protocol. In source tree routing, the distribution tree is maintained by periodic reverse-path forwarding and pruning, which incur large overheads. Thus we conclude that a shared tree routing protocol is more suitable than a source tree for our approach.

Shared multicast routing protocols can be further classified into two types: *unidirectional* and *bidirectional* shared tree routing protocols. In a unidirectional shared tree protocol, the sender's packets go to the core first and the core multicasts them to others. By contrast, in bidirectional shared tree routing protocols, members can communicate with each other without going through the core since packets can travel both up toward the core and down from the core. Thus, once a shared tree routing is used, unidirectional routing protocols are inefficient for scoped multicast and communications among neighborhoods. The larger the multicast session and the more demands for scoped multicast, the larger the communication overheads will be in unidirectional shared multicast routing protocols. Reflecting these observations, the long term inter-domain routing solution, Border Gateway Multicast Protocol (BGMP)[24] constructs bidirectional shared trees. Since our tracing mechanism uses scoped multicast to find negative tracer nodes, bidirectional shared tree routing protocols will be more efficient in our framework.

6.3.2. Application-level multicast—Despite its efficient network resource use, the deployment of IP multicast has been hampered by a number of challenges including the need to modify infrastructure and the need to support reliability, flow, and congestion control.

Limited network layer support for multicast in the Internet today, has led to active research on end-system approaches,[25–31] which do not require such infrastructure support, i.e., all multicast related functionalities, including group management and packet forwarding, are implemented at end systems. In this architecture, hosts in the group co-operate to construct an *overlay* structure of unicast connections.

Thus, to remove the hurdle of sluggish IP multicast deployment, we can use multicasting solutions based on end-system approaches. For example, the work in Reference 32 provides a scalable application-level multicast solution.

—6.4. Economic Incentives—

Irrespective of the existence of feasible solutions to mitigate DDoS attacks, a significant hurdle may be the lack of viable economic incentives.[33] For example, installing ingress filters in a domain consumes valuable router resources and reduces the overall routing performance. However, its beneficiaries are likely to be other domains rather than the domain performing ingress filtering. In our framework, we can envisage the following economic model: victims pay a fee for the services provided by tracer and filter nodes. Clearly this gives an incentive to provide tracing and filtering services to victims in other domains. The payment may be dependent on the number of attack packets dropped, the number of tracing or filtering report

packets and so on. One can further consider that victims pay a fee for detecting services. We leave the more detailed pricing mechanisms as future work.

—6.5. Differentiated Services—

As described in Section 6.1, various implementation methods for tracing and filtering components are available with different characteristics. To allow such heterogeneity in implementation in our framework, we can consider providing differentiated services. Instead of having a single multicast session for tracing (or filtering) service, we create different multicast sessions for different services. For example, there could be a multicast session for post-mortem tracing or one for application level filtering.

Note that this scheme allows flexible service requests for victims and heterogeneity in different implementation methods from different domains. Depending on the attack scenarios, a victim can request different services. For example, if a victim detects an attack very late and the attack has already ended, the victim can ask for a post-mortem tracing service rather than tracing for an on-going one. Given its service requirements, each domain may use its own methods. This requires less standardization across network domains, allowing more heterogeneity.

7. Related Work

In this section, we discuss the pros and cons of existing research efforts on defeating DDoS attacks and the proposed approach in this paper.

—7.1. Detection and Mitigating Approach—

There are two types of mitigation mechanisms: host-based and router-based. Host-based approaches[34,35] try to detect and mitigate the impact of attacks by an efficient control of resources from the perspective of the operating system on the victim side. Even though this helps sustain a victim longer, the victim will eventually give in to attacks. By contrast, in the router-based approach,[18] detection and mitigation of attacks are performed at routers. This work defines an *aggregate* as a particular set of packets causing the overload and proposes an identification algorithm for detection and control mechanisms which can reduce such aggregates. A SYN flooding detection mechanism has recently been proposed in Reference 36. It is based on discrepancies between SYN and FIN packets. It is stateless and requires low computational overhead to detect SYN flooding attacks.

—7.2. Proactive Filtering Approach—

Ingress filtering[2] and route-based filtering[3¶] proactively prevent attacks employing spoofing. Routers are configured to drop packets whose source IP addresses are illegitimate based on routing and network topology information. Ingress filtering uses simple direct connectivity information, so it is usually performed at border routers in stub networks.[2] However, in transit networks, it lacks the ability to distinguish between legitimate and illegitimate packets and its effectiveness can only be guaranteed via wide deployment. To overcome these weaknesses, a route-based mechanism[3] performs filtering using source reachability information imposed by routing and network topology. By using additional network topology information, route-based filtering requires less coverage than ingress filtering to be effective.

—7.3. Tracing Approach—

As indicated by recent work on tracing mechanisms,[6,11,12] tracing can be an effective way of discouraging attackers. The identified compromised hosts intentionally or unwillingly participating in attacks can be isolated from the Internet or can provide clues to the real attacker. Existing tracing approaches can be classified into the following two types.

The first type is a query-based approach where trail information is queried to tracing components

¶*Route-based filtering approach also has a tracing by-product, i.e., attack origins can be localized to a set of AS (Autonomous System) sites.*

in the network. The query is usually performed in the reverse direction of the attack packets, i.e., from the victim toward the source(s) of the attack. Checking whether attack packets have been forwarded by given routers can be done by several currently available techniques: logging packets using monitoring tools[19] or input debugging which can identify which ingress port was used by packets departing on a given egress port. However, there may be a high storage overhead for logging methods and some adverse effects on routing functionality when input debugging is used. To overcome these problems, in the hash-based logging approach,[6] routers store packet digests generated by a hash function rather than the packets themselves. Upstream routers to the victim are successively queried for attack packets in a reverse path flooding manner. A key advantage of this approach is that it is capable of tracing a single recently forwarded packet while keeping privacy. However, tracing queries should be initiated early enough that appropriate digest entries have not been overwritten by more recent packets. Note that since queries are sequentially processed in the query-based approach, the malfunctioning of some tracing components may not deliver a query to upstream routers, which result in the failure of the tracing operation.

Another tracing alternative is based on partial path information which is proactively sent to end hosts when packets are forwarded. Once attacks are detected, the victim can reconstruct the routes attack packets took based on stored trail information. In the iTrace method,[37,38] with a low probability routers send extra ICMP messages including their own addresses to the end host. By contrast, IP marking schemes[11,12] can eliminate the extra ICMP messages used in iTrace by having routers probabilistically inscribe *edge* information (represented by two routers at the end of a link) onto a traversing packet. The advantage of this approach is that it enables incremental deployment while keeping the router's overhead low. However, as pointed out in Reference 39, in the presence of multiple attack nodes, the approach suffers from a scalability problem, i.e., uncertainty in identifying origins of attack packets increases proportionally with the number of nodes in a distributed DoS attack. Furthermore, due to its probabilistic character, the solution is confined to tracing attacks associated with large volumes of traffic. Note that in this second approach, end hosts need to proactively save incoming packets irrespective of whether an attack is ongoing, requiring large storage overheads at end hosts.

The work in Reference 7 deals with mitigating DDoS attacks based on IP marking techniques. The key idea was to give blocking components access to attack graph information obtained by IP marking techniques and probabilistically filter out more packets traversed 'infected' edges compared to packets traversed 'clean' edges. This can increase the throughput of legitimate traffic. However, in this approach, dropping legitimate traffic cannot be avoided. Schnackenberg *et al*.[40] propose an IDIP (Intruder Detection and Isolation Protocol) for automated intrusion response systems that can detect a DoS attack and request upstream network elements to block the traffic. Their work focuses on standardization of a set of protocols for interaction among infrastructure components to realize a simple query-based tracing idea.

—7.4. Characteristics of our Solution—

In a proactive filtering approach, filters need to maintain a large number of filtering rule sets and examine every single packet. We envisage a large-scale filtering framework, which suffers from the complexity of managing a large number of filtering rule sets. Furthermore, if attacks are infrequent, valuable resources may be wasted if irrelevant filtering rule sets are applied to packet flows. By contrast, our filtering mechanism has soft-state properties, e.g., renewal and expiration of filtering, which significantly reduces complexity of managing filtering rule sets.

The conventional reactive filtering approach confined to a single administrative domain containing the victim has its limitation. Once attack packets (possibly generated at multiple locations) have been aggregated into large traffic flows, attack traffic can overwhelm the local domain and make local filters inoperable; as a result, the filters are victimized. Note that our filtering mechanism is designed to quickly identify a set of boundary filter locations so that attack packets might be dropped as close as possible to their origins before they are aggregated.

The key requirement for reactive approaches—promptness—can be met by the use of multicast communication in our framework. This is the case when a request packet is distributed in real time via multicast, so filter/tracer nodes can respond concurrently and quickly. The use of multicasting provides the following additional advantages. First, there is no overhead in managing a list of co-operative filter/tracer nodes on detector or filter/tracer nodes, leading to improved scalability. This feature comes from the member abstraction property of multicast service, i.e., members can join and leave a multicast session without explicit knowledge of its membership. Second, the proposed mechanism is robust even in the presence of some malfunctioning filter nodes or packet losses. In a sequential query-based approach, the query process may be terminated due to packet losses or malfunctioning of some filter nodes. In such situations, our approach may not result in optimal filtering/tracing operation, i.e., filtering is performed in non-boundary filter nodes (or attack graph is not accurate), but it may still work well.

8. Conclusion

In this paper, we have presented a new multicast-based filtering and tracing service framework to defeat DDoS attacks. The proposed filtering mechanism pushes filtering operation to boundary nodes so that attack packets might be dropped as close as possible to their origin(s). If we assume that attacks are infrequent, our filtering mechanism can achieve more efficient use of network resources versus proactive solutions. Indeed when no attacks are ongoing, only a multicast session needs to be maintained, without overheads associated with a filtering operation. We also consider the goal of determining sets of candidate nodes for localizing attack origins under a partial deployment of tracing components, and propose a mechanism to achieve this end. Finally note that the proposed tracing mechanism can be used for a network management purpose of monitoring spatial flow of traffics.[41]

A significant challenge of large-scale deployment of both mechanisms is handled by a novel use of multicasting and soft-state. Furthermore the use of multicasting provides a number of desirable characteristics, e.g., fast response—one of the key requirements for reactive solutions, and robustness. Additional contributions of our work include a number of practical considerations: (1) addressing economic incentives, (2) using currently available equipment and technologies without major router modifications, and (3) allowing incremental and partial deployment.

The performance evaluation for the proposed framework shows that a small coverage ratio of well-placed filter or tracer nodes can achieve efficient blocking and localizing of attacks.

Acknowledgements

We wish to thank P. Krishnan, Danny Raz and Yuval Shavitt for providing us with real topology data. We also would like to thank Kyoil Kim for his helpful discussions.

References

1. Computer Emergency Response Team, 'CERT advisory ca-2000-01 denial-of-service developments,' http://www.cert.org/advisories/CA-2000-01.html.
2. Ferguson P, Senie D. *Network ingress filtering: Defeating Denial of Service attacks which employ IP source address spoofing*, RFC 2267, http://www.ietf.org/rfc, Jan. 1998.
3. Park K, Lee H. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets, in *Proc. ACM SIGCOMM*, 2001.
4. Cisco Corporation, Characterizing and tracing packet floods using Cisco routers, www.cisco.com/warp/public/707/22.pdf.
5. Wang H, Bose A, El-Gendy M, Shin KG. IP easy-pass: Edge resource access control, in *Proc. IEEE Infocom*, 2004.
6. Snoeren AC, Partridge C, Sanchez LA, Jones CE, Tchakountio F, Kent ST, Strayer WT. Hash-based IP traceback, in *Proc. ACM SIGCOMM*, 2001.
7. Sung M, Xu J. IP traceback-based intelligent packet filtering: A novel technique for defending against Internet DDoS attacks, in *IEEE International Conference on Network Protocols*, 2002.
8. IETF secure multicast group, http://www.securemulticast.org.
9. Canetti R, Garay J, Itkis G, Micciancio D, Naor M, Pinkas B. Multicast security: A taxonomy and efficient constructions, in *Proc. IEEE Infocom*, 1999.

10. Li J, Reiher P, Popek G. Resilient self-organizing overlay networks for security update delivery. *IEEE Journal on Selected Areas in Communications* 2004.
11. Savage S, Wetherall D, Karlin A, Anderson T. Practical network support for IP traceback, in *Proc. ACM SIGCOMM*, 2000.
12. Song DX, Perrig A. Advanced and authenticated marking schemes for IP traceback, in *Proc. IEEE Infocom*, 2001.
13. Skitter, http: //www.caida.org/tools/measurement/skitter/.
14. Internet mapping, http://cm.bell-labs.com/who/ches/map/dbs/index.html/, 1999.
15. Krishnan P, Raz D, Shavitt Y. The cache location problem, *IEEE/ACM Transactions on Networking* 2000; **8**:
16. Zegura EW, Calvert KL, Bhattacharjee S. How to model an Internet, in *Proc. IEEE Infocom*, 1996.
17. Li B, Golin MJ, Ialiano GF, Deng X. On the optimal placement of web proxies in the Internet, in *Proc. IEEE Infocom*, 1999.
18. Mahajan R, Bellovin SM, Floyd S, Ioannidis J, Paxson V, Shenker S. Controlling high bandwidth aggregates in the network, in http://www.aciri.org/pushback/pushback-toCCR.ps, submitted to CCR, July 2001.
19. Sager G. Security fun with OCxmon and cflowd, in *Internet 2 Working Group Meeting*, Nov. 1998, http://www.caida.org/projects/NGI/content/security/1198.
20. Stallings W, *SNMP, SNMP v2, SNMP v3 and RMON 1 and 2*(Third Edition), Addison-Wesley, Inc., 1999.
21. Deering SE. *Multicast Routing in a Datagram Internetwork*, PhD. thesis, Stanford University, 1991.
22. Ballardie T, Francis P, Crowcroft J, Core based trees(CBT): An architecture for scalable inter-domain multicast routing, in *Proc. ACM SIGCOMM*, 1993.
23. Estrin D, Farinacci D, Helmy A, Thaler D, Deering S, Handley M, Jacobson V, Liu C, Sharma P, Wei L. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification, RFC 2362, http://www.ietf.org/rfc, June, 1998.
24. Thaler D, Estrin D, Meyer D. Border gateway multicast protocol (BGMP): Protocol specification, IETF draft, draft-ietf-bgmp-spec-01.txt, Mar. 2000.
25. Francis P. Yoid: Extending the Internet multicast architecture, in *Tech. reports, ACIRI*, http://www.aciri.org/yoid, 2000.
26. Chawathe Y. *Scattercast: An architecture for Internet broadcast distribution as an infrastructure service*, PhD. thesis, University of California, Berkeley, 2000.
27. Jannotti J, Gifford D, Johnson K, Kasshoek F, O'Toole J. Overcast: Reliable multicasting with an overlay network, in *USENIX OSDI*, 2000.
28. Chu Y, Rao S, Seshan S, Zhang H. Enabling conferencing applications on the Internet using an overlay multicast architecture, in *Proc. ACM SIGCOMM*, 2001.
29. Pendarakis D, Shi S, Verma D, Waldvogel M. ALMI: an application level multicast infrastructure, in *3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, 2001.
30. Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A scalable content-addressable network, in *Proc. ACM SIGCOMM*, 2001.
31. Zhao B, Kubiatowicz J, Joseph A. Tapestry: An infrastructure for fault resilient wide-area location and routing, in *Tech. Report. UCB//CSD-01-1141, U. C. Berkeley*, 2001.
32. Ratnasamy S, Handley M, Karp R, Shenker S. Application-level multicast using content-addressable networks, in *Proc. of Networked Group Communication (NGC)*, 2001.
33. Geng X, Whinston AB. Defeating distributed denial of service attacks, in *IT Pro*, July 2000.
34. Banga G, Druschel P, Mogul J, Resource containers: A new facility for resource management in server systems, in *Proc. of the 1999 USENIX/ACM Symposium on Operating System Design and Implementation*, Feb. 1999.
35. Spatscheck O, Peterson L. Defending against denial of service attacks in Scout, in *Proc. of the 1999 USENIX/ACM Symposium on Operating System Design and Implementation*, Feb. 1999.
36. Wang H, Zhang D, Shin KG. Detecting SYN flooding attacks, in *Proc. IEEE Infocom*, 2002.
37. Bellovin SM. ICMP traceback messages, IETF draft, draft-bellovin-itrace-05.txt, Mar. 2000.
38. Wu SF, Zhang L, Massey D, Mankin A. Intention-driven ICMP trace-back, IETF draft, draft-wu-itrace-00.txt, Feb. 2001.
39. Park K, Lee H. On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack, in *Proc. IEEE Infocom*, 2001.
40. Schnackenberg D, Djahandari K, Sterne D. Infrastructure for intrusion detection and response, in *Proc. First DARPA Information Survivability Conference and Exposition*, 2000.
41. Duffield NG, Grossglauser M. Trajectory sampling for direct traffic observation, *IEEE/ACM Transactions on Networking*, 2001; **9**(4):280–292. ■

AUTHOR QUERY FORM

Dear Author,

During the preparation of your manuscript for publication, the questions listed below have arisen. Please attend to these matters and return this form with your proof.

Many thanks for your assistance.

Query References	Query	Remarks
1	Au? please supply full address	
2	Au? current e-mail address?	
3	Au? please supply fuller addresses and short biographies	
4	Au? ok now?	
5	Au? ok now?	
6	Au? Vol and page?	
7	Au? page?	