Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 66 (2006) 1152–1164
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Honeypot back-propagation for mitigating spoofing distributed Denial-of-Service attacks☆

Sherif Khattab[a], Rami Melhem[a,*], Daniel Mossé[a], Taieb Znati[a,b]

[a]*Department of Computer Science, University of Pittsburgh, PA 15260, USA*
[b]*Department of Information Science and Telecommunications, University of Pittsburgh, PA 15260, USA*

Received 17 December 2005; received in revised form 31 March 2006; accepted 10 April 2006
Available online 7 July 2006

Abstract

The Denial-of-Service (DoS) attack is a challenging problem in the current Internet. Many schemes have been proposed to trace spoofed (forged) attack packets back to their sources. Among them, hop-by-hop schemes are less vulnerable to router compromise than packet marking schemes, but they require accurate attack signatures, high storage or bandwidth overhead, and cooperation of many ISPs.

In this paper, we propose *honeypot back-propagation*, an efficient hop-by-hop traceback mechanism, in which accurate attack signatures are obtained by a novel leverage of the roaming honeypots scheme. The reception of attack packets by a roaming honeypot (a decoy machine camouflaged within a server pool) triggers the activation of a tree of *honeypot sessions* rooted at the honeypot under attack toward attack sources. The tree is formed hierarchically, first at Autonomous system (AS) level and then at router level. Honeypot back-propagation supports incremental deployment by providing incentives for ISPs even with partial deployment.

Against low-rate attackers, most traceback schemes would take a long time to collect the needed number of packets. To address this problem, we also propose *progressive back-propagation* to handle low-rate attacks, such as on-off attacks with short bursts. Analytical and simulation results demonstrate the effectiveness of the proposed schemes under a variety of DDoS attack scenarios.

Keywords: Network security; Denial-of-Service attacks; Honeypots; Traceback

1. Introduction

The Internet has witnessed a proliferation of services, some are publicly accessible, such as google.com and DNS, whereas others, such as subscription-based services, are private to specific user communities. Meanwhile, Distributed Denial-of-Service (DDoS) attacks continue to pose a real threat to Internet services [5–7], and the arsenal of DDoS attackers is full of different mechanisms [28]. For instance, certain fields of attack packets may be spoofed, or forged, to complicate detection and isolation or to hide attack sources.

The focus of this paper is on defending *private* services against DDoS attacks employing source-address spoofing. In private services, legitimate clients are either known a priori, such as in telecommuting to an enterprise network, or they subscribe to the private service to acquire access to the network. In either scenario, legitimate clients should be able to access the service from *anywhere* in the Internet [17].

Many schemes have been proposed to defend against source-address spoofing [13,27,38,2,43,31,32,11,3,50,41,26]. Spoofing prevention methods [13,32,3] may interfere with the operation of some Internet protocols, such as mobile IP [33], that utilize spoofing "legitimately". Traceback schemes [38,2,43,31,11,50,41,26] identify the real sources of spoofed attack packets. However, they require an accurate identification of attack packets, and a complete defense would need to take an action against the attack sources, to stop them for instance. The Pushback mechanism automatically takes such an action by enforcing aggregate-based congestion control (ACC). Without accurate attack signatures, however, some attack scenarios, such as highly dispersed DDoS attacks, cause Pushback to collaterally damage legitimate traffic [27].

☆ A preliminary version of this paper was presented at SSN'06 [18].

* Corresponding author.
E-mail addresses: skhattab@cs.pitt.edu (S. Khattab), melhem@cs.pitt.edu (R. Melhem), mosse@cs.pitt.edu (D. Mossé), znati@cs.pitt.edu (T. Znati).

0743-7315/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2006.04.016

Honeypots are a promising approach for obtaining accurate attack signatures. They can be physical or virtual machines and have been successfully used to detect worm signatures [35,25,44]. The *Roaming honeypots* scheme camouflages honeypot locations within a pool of server replicas by randomly designating a set of servers to be active over a period of time and using the remaining servers as honeypots [21]. In other words, each server in the pool, in coordination with legitimate clients and remaining peer replicas, assumes the role of a honeypot for specific intervals of time. Roaming makes it difficult for attackers to identify active servers, thereby causing them to be trapped in the honeypots.

In this paper, we propose *honeypot back-propagation*, a hierarchical traceback scheme, which effectively traces back to and *stops* sources of attack streams without significant impact on the performance of legitimate traffic streams. It achieves these properties by combining the effectiveness of the Pushback mechanism for tracing back and controlling attack traffic, and the ability of the roaming honeypots to accurately and promptly detect attack signatures.

The main idea of the proposed scheme is that when a roaming honeypot receives packets, it initiates a traceback process by alerting autonomous systems (ASs) across the path(s) towards attack sources. The alert triggers an AS-level input-debugging-like [45] process on traffic *destined* for the honeypot, and further propagates honeypot activations upstream towards attack sources. Moreover, within each AS, attack hosts are identified and filtering rules are installed to block their network access. The ability of honeypot back-propagation to accurately distinguish attack packets from legitimate ones enables this aggressive action, as opposed to rate limiting, against attack traffic without penalizing legitimate traffic.

When a very large number of hosts participate in a DDoS attack, widespread deployment and cooperation among ISPs are required for traceback to be effective. Honeypot back-propagation provides a high payoff in this regard. First, it uses accurate attack signatures, and thus, reduces collateral damage. Second, it helps ISPs to accurately locate compromised hosts on their networks. This information is helpful, because these hosts may be involved in spreading viruses and spam to other hosts within the ISP. Third, incremental benefits are possible with partial deployment of honeypot back-propagation, because network messages involved in the scheme can be piggybacked on BGP messages to traverse legacy networks. Another implication of an attack launched from a large number of machines is that the attack rate per machine can be reduced while achieving the same damage. We address low-rate attacks by a *progressive honeypot back-propagation* scheme.

We evaluated our schemes analytically and using ns-2 [47] simulations. The analytical model estimates the average time to reach and stop an attacker, while the simulations study the effect of different attack parameters on the effectiveness of the scheme. The results show that attacks can be stopped within seconds under many scenarios.

The rest of the paper proceeds as follows. In the next section we review some of the related DDoS defenses. Section 3 presents service and attack models. For completeness, we describe the roaming honeypots framework, on which we build honeypot back-propagation, in Section 4. Section 5 describes the hierarchical honeypot back-propagation scheme with inter-AS and intra-AS components and discusses some design issues. We present the progressive back-propagation scheme to handle low-rate attacks in Section 6. In Section 7, we present the analytical models, and in Section 8, we describe our simulation models and results. Section 9 concludes the paper.

2. Related work

Many schemes have been proposed to address the source-address spoofing problem in the Internet. The approaches behind these schemes can be classified into prevention [13,16,32,3,8,48,23], traceback [38,2,43,31,11,50,41,26,45,4, 36,30,27], and mitigation [17,34,51].

Spoofing prevention: In the context of private services that we consider in this paper, IPsec [16] is a potential defense alternative. However, IPsec is not available everywhere and its performance overhead can be prohibitive [17]. Ingress filtering [13], SPM [3], and DPF [32] interfere with the operation of Internet protocols, such as mobile IP [33], which use spoofing "legitimately". IP puzzles represent a promising direction in suppressing DDoS attacks [8,48]. Optimal tuning of puzzle difficulty remains, however, an open problem. The Mohonk, or mobile honeypots, scheme propagates unused addresses using BGP options [23], so that (spoofed) packets with matching source addresses can be safely dropped. Our scheme makes it difficult for attackers to discover and avoid sending traffic to unused addresses.

Traceback: We classify traceback schemes into packet marking schemes and hop-by-hop schemes. *Packet marking* schemes (e.g., [38,43,31,11,50,30]) construct attack paths locally at the victim by collecting markings stamped into packets by intermediate routers. However, these schemes are vulnerable to compromised routers, which can inject forged markings to increase the number of false positives [43]. Authentication schemes have been proposed to solve this problem [43], but they require high computational overhead at the routers and high storage overhead at the victims to maintain router keys.

Hop-by-hop traceback: The basic idea of hop-by-hop schemes is that traceback starts at the router next to the attack victim, where neighboring routers upstream on the attack paths are identified using a signature of attack packets. This process continues until the attack machines are reached. CenterTrack uses an overlay network to determine ingress points of attack traffic into a network cloud (an AS for instance) [45], and controlled flooding injects packet floods into the network and detects attack paths based on traffic perturbations [4]. Pushback provides an automation of this process, whereby aggregate-based rate-limiting filters are propagated upstream from congested routers [15]. Both detection of misbehaving aggregates and assignment of rate limits are done using the ACC mechanism [27]. At each Pushback-enabled router, the rate limit of an aggregate is shared in a max–min fairness

fashion among input ports on which traffic matching the aggregate signature is received. To estimate the arrival rate of each input port, a feature similar to input debugging is used to map each packet at the output queue to its corresponding input port. Pushback accepts misbehaving aggregate signatures through special request messages. Honeypot back-propagation scheme can be viewed as a realization of this feature; when a server takes the role of a honeypot, the server's destination address defines the malicious aggregate.

Hop-by-hop schemes are less vulnerable to false positives caused by compromised routers; if a compromised router drives the traceback process into an upstream router not on the attack path, traceback will stop at that router because the attack signature will not be matched by any packets. Hop-by-hop schemes, however, require the identification of an accurate attack signature, that is, one that leads only to attack sources and remains unchanged until the attack sources are reached. An exception is the single-packet traceback scheme [41,26], which can use a single attack packet as the signature. However, it requires high storage overhead at routers or high bandwidth overhead. Hop-by-hop schemes require cooperation of many ISPs as well. Our honeypot back-propagation is an efficient hop-by-hop scheme that extracts accurate attack signatures and provides deployment incentives for ISPs.

Against a large number of low-rate attackers launching an on–off attack with short bursts [24], most traceback schemes take a long time to collect the number of packets needed to construct each attack path, and they produce a large number of false positives. To address this limitation, we present a progressive scheme that can trace back to low-rate attackers within a reasonable time and with a low false positive rate.

Mitigation: The SOS architecture [17] tackles the same problem as ours: DoS attack in the context of a private service with predetermined clients. However, the latency caused by the hash-based routing in SOS can be up to 10 times the direct communication latency [17]. Our work aims at providing a more efficient solution by avoiding hash-based routing and by taking actions only when attacks occur. Traceback information can be used for filtering at the victim. For instance, StackPi is a deterministic packet marking scheme that allows the victim to locally filter attack packets based on the mark field [34]. However, the scheme's accuracy, in terms of false positive and false negative rates, deteriorates with a large number of dispersed attackers, and the scheme is vulnerable to compromised routers. Level-k max–min fairness [51] addresses the drawbacks of the hop-by-hop application of max–min fairness in Pushback, which can severely punish legitimate traffic sharing aggregate signatures with misbehaving traffic (see Section 8.4.1). However, it is still ineffective against highly dispersed attackers.

3. The spoofing DDoS attack

We consider a private service deployed over a pool of replicated servers as depicted in Fig. 1, whereby legitimate clients are either known a priori or acquire access to the service through

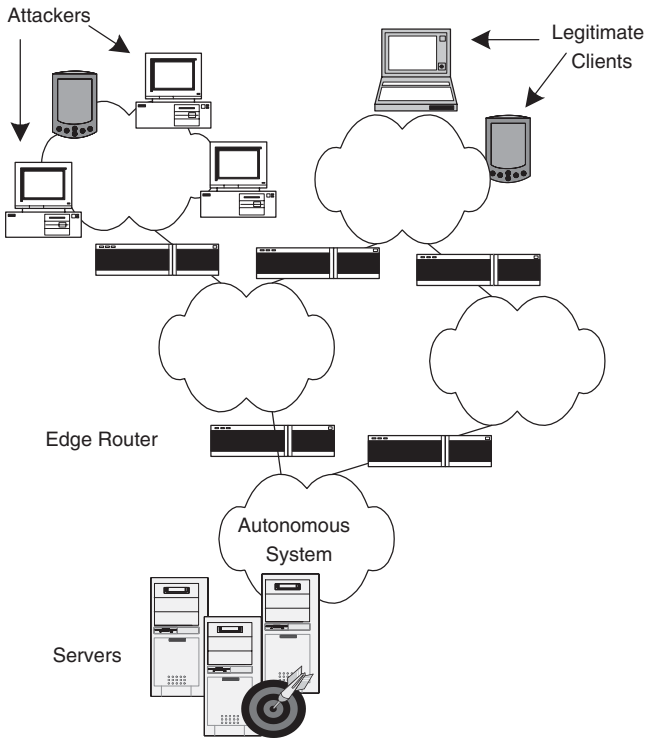

Fig. 1. The service is provided by a pool of replicated servers. Legitimate clients access the servers from anywhere in the Internet. Attackers send spoofed attack traffic to the servers.

on-line subscription. Legitimate clients should be able to access the service from anywhere in the Internet.

A variety of attacks can be staged against the private service. In this study, we assume that attacks can be launched from a number of attack hosts, or zombies, by sending spoofed packets destined for the servers. [1] The attack may result in the following: (1) blocking of legitimate connection requests from reaching the servers and (2) degrading the throughput of both TCP flows from servers to clients as well as data flows from clients into servers. For example, if TCP ACK packets from clients to servers get dropped due to the attack, the throughput of TCP flows is degraded.

4. Roaming honeypots background

The proactive server roaming scheme has been proposed in [19], where a prototype of the scheme is evaluated, and has been studied through simulation in [37]. The roaming honeypots scheme leverages proactive server roaming to camouflage honeypot locations within a pool of servers, so as to make it difficult for attackers to direct their traffic away from the honeypots and to avoid detection [21]. The scheme allows for k out of N servers to be concurrently active, whereas the remaining $N-k$ act as honeypots. The locations of the current active servers, and, thus, the honeypots, are changed according to a pseudo-random schedule shared among the servers and legit-

[1] We assume that legitimate clients are not compromised.

imate clients. Therefore, legitimate clients always send their service requests to active servers, whereas attack requests may reach the honeypots. The source address of any request that hits a honeypot is blacklisted, so that all future requests from this source are subsequently dropped. The source address is not blacklisted unless a full handshake is recorded to ensure that it is not spoofed.

The pseudo-random schedule divides time into *epochs*, whereby at the end of each epoch, the set of active servers changes. A long hash chain is generated using a one-way hash function, and used in a backward fashion. The last key in the chain, K_n, is randomly generated and each key, K_i $(0 < i < n)$, in the chain is computed as $H(K_{i+1})$ and used to determine the active servers during epoch i. Upon subscription to the service, each legitimate client is assigned a roaming key, K_t from the hash chain, with a varying value of t according to each client's trust level and/or other policies. K_t acts as a *time-based* token, whereby it allows the client to track the service up to and including epoch t. Clients also receive the list of servers. When subscription expires, that is, the current service epoch exceeds t, the client may contact the subscription service to acquire a new key.

Roaming honeypots require loose clock synchronization of servers and legitimate clients. That is, the clock shift among system components is bounded by a constant, δ. To maintain loose clock synchronization, clients synchronize their clocks with servers at each service request. However, if a client remains inactive for a long period of time, it contacts the subscription service for resynchronization. At the server side, each service epoch starts earlier by δ at the new servers and ends later by $\delta + \gamma$ at the active servers of the previous epoch, where γ is estimated communication delay from clients to servers.

When server switching occurs in the middle of a connection, the connection is migrated to another active server where it is resumed. Roaming honeypots adopts an approach similar to [46,40], whereby each active server periodically checkpoints per-connection state of current connections and sends the checkpoints to the corresponding clients. Clients send the checkpoints to the new servers to resume their connections. More details can be found in [21,22].

Different deployment approaches have been proposed to implement roaming honeypots [22]. In one approach, clients are responsible for tracking epoch lengths and active servers across service epochs. Another approach makes the roaming honeypots scheme transparent to clients by employing an intermediate proxy network. In this paper, we assume the first deployment approach because of its simplicity.

5. Honeypot back-propagation

Our honeypot back-propagation extends the roaming honeypots scheme to defend against source-address spoofing DDoS attacks. As described in the previous section, servers alternate between providing service and acting as honeypots according to a shared pseudo-random schedule [21]. Each server S enters a *honeypot epoch* as soon as it is scheduled to be inactive. During a honeypot epoch, S expects to receive no legitimate traffic, therefore, any packet destined for S is most likely an attack packet. A honeypot epoch ends once S becomes active again. As described earlier, these honeypot epochs are selected in coordination among servers and legitimate clients so as not to cause service interruption.

In other words, honeypot epochs are time windows, in which a server receives a stream of pure attack packets, which we term *honeypot traffic*. In honeypot back-propagation, we trace the honeypot traffic back to its origin(s) and install filters as close as possible to the attack sources. The back-propagation process is achieved hierarchically, first between ASs to reach ASs hosting attack machines and then among routers within these ASs to reach the attack machines and block their network access.

5.1. Inter-AS propagation

The basic idea of inter-AS honeypot back-propagation is that, during the honeypot epochs of a server S, back-propagating *honeypot sessions* are created in *ASs* upstream from S towards attack sources. A honeypot session is a data structure with a set of associated actions. The data structure is a record of the IP address of S and the set of upstream ASs from which honeypot traffic was received. The actions of a honeypot session are triggered by the reception of network packets as detailed below. While a honeypot session is active at an AS, packets entering the AS destined for S trigger further propagation of honeypot sessions into the neighboring upstream ASs from which the packets are received. The back-propagation process stops if no more attack packets are received or when *non-transit* ASs are reached. A non-transit AS does not allow transit traffic from other ASs to pass through. Honeypot sessions trigger intra-AS back-propagation at their hosting ASs to reach and stop attack hosts as will be described in the next subsection. Except for the honeypot sessions at non-transit ASs, all other honeypot sessions are torn down at the end of honeypot epochs.

To implement honeypot back-propagation, two main mechanisms are required. The first is needed to identify the ingress points of honeypot traffic so that honeypot sessions get propagated to the corresponding upstream ASs. The second is needed for propagating and tearing down the honeypot sessions.

The first mechanism uses a *honeypot session manager (HSM)*, which is a host in the AS network that maintains honeypot sessions and identifies the AS edge routers from which honeypot traffic enters the AS. Upon receiving a honeypot session, ingress honeypot traffic is diverted into the HSM by sending iBGP route announcement declaring the HSM as the next-hop for ingress traffic destined to S [1]. Upon receiving this route announcement, edge routers forward honeypot traffic into the HSM. However, it is still needed to identify the honeypot traffic ingress points. To achieve this goal, two methods can be used: tunneling and packet marking.

In the tunneling approach, generic routing encapsulation tunnels [12] are setup between each edge router and the HSM; the HSM identifies the ingress points by inspecting from which tunnels the diverted traffic is received. In packet marking, each

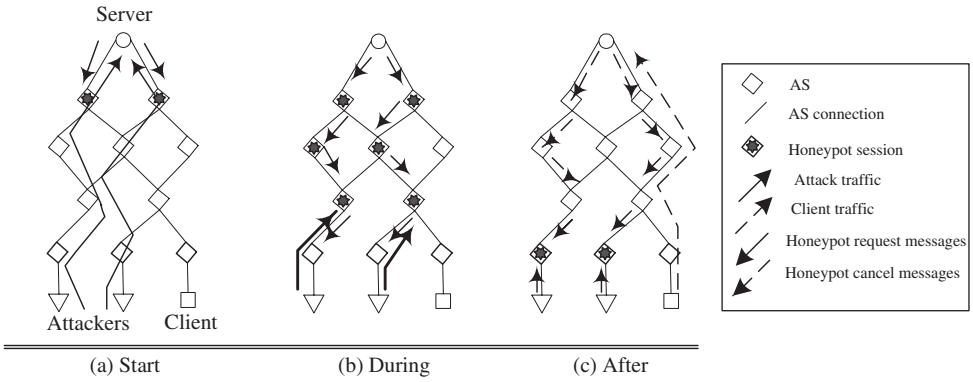

Fig. 2. The operation of inter-AS honeypot back-propagation at the start of, during, and at the end of each honeypot epoch. Honeypot sessions are maintained and propagated by honeypot session managers (HSM) inside ASs. (a) While acting as a honeypot and upon reception of attack packets, a server S sends honeypot request messages to the HSM(s) in its home AS(s). When a HSM receives a honeypot request message, it creates a honeypot session. (b) During honeypot epochs, HSMs propagate honeypot sessions upstream towards attack sources. (c) At the end of honeypot epochs, S sends honeypot cancel messages to tear down honeypot sessions except at stub ASs hosting attackers.

edge router is assigned a unique identifier such that if there are n such routers, $\lg n$ bits are needed. Edge routers stamp their IDs into the diverted honeypot traffic. Because only the honeypot traffic, which will be discarded anyway, is marked, the ID field in the IP header can be safely used. This packet marking scheme is similar to the destination-end provider marking scheme in [39]. We note that in the packet marking we propose, a compromised edge router cannot cause false positives by driving the back-propagation process into ASs not on the attack path. Back-propagation will stop at these ASs, as no packets will match the attack signature (the address of the honeypot). Once an ingress point is determined by the HSM, a honeypot session is propagated to HSM of the corresponding neighbor AS.

To achieve honeypot session propagation and tear-down, we define two messages: *honeypot request* and *honeypot cancel* messages. Fig. 2 illustrates the propagation and tear-down of honeypot sessions along a tree of ASs routed at the server's AS. As depicted in Fig. 2(a), whenever the server S starts a honeypot epoch, it sends a honeypot request message to the HSM(s) of its AS(s). Upon reception of a honeypot request message, the HSM creates a honeypot session, during which the honeypot traffic ingress points are identified as described earlier. When an ingress point x is identified, a honeypot request message is sent to the HSM responsible for the upstream AS connected to x. Fig. 2(b) illustrates a snapshot of the scheme *during* a honeypot epoch.

At the end of the honeypot epoch (Fig. 2(c)), S sends a honeypot cancel message to the HSM(s) of its AS(s). When a HSM receives a honeypot cancel message, it sends a cancel message to all upstream HSMs to whom it has previously sent a honeypot request message (this information is stored in the honeypot session) and cancels traffic diversion at edge routers, so that S can receive traffic. The HSM of a non-transit AS retains the honeypot session until intra-AS back-propagation is performed to reach and stop attack hosts; otherwise the honeypot session is removed.

5.2. Intra-AS propagation

Intra-AS back-propagation reaches and stops attack hosts within each AS hosting a honeypot session. This step is necessary when both legitimate clients and attackers are hosted on the same AS, and it provides incentives for ISPs by identifying attack hosts within their networks; the attack traffic sent by these hosts may backfire at the ISP by causing performance degradation inside the ISP network or by being directed at the ISP's own customers.

In intra-AS honeypot back-propagation, honeypot sessions at the HSMs are used to further pin down attack hosts. The basic tenet of the intra-AS back-propagation is the use of hop-by-hop traceback within the AS to locate access routers, that is, first-hop routers, of attack hosts. The access routers then identify the MAC addresses of attack hosts and inform the network switches to close the ports connected to the identified MAC addresses. This process effectively blocks compromised machines from accessing the network. We note that a similar process is already implemented in current network switches [9].

We now describe the intra-AS hop-by-hop traceback process. Recall that honeypot traffic is diverted from edge routers, both ingress and egress, into the HSM during honeypot epochs. Ingress routers are upstream towards attack sources and egress routers are downstream towards the servers. The HSM sends *local* honeypot request messages to the *egress* routers from which it received diverted traffic. Upon reception of a honeypot request message, the router creates a honeypot session, during which the honeypot traffic input ports are identified using router-level input debugging [45]. When an input port x is identified, a local honeypot request message is sent to the upstream router connected to x provided that local honeypot messages do not cross AS boundaries. At the end of the honeypot epoch, the HSM sends honeypot cancel messages to the egress routers. When a router receives a honeypot cancel message, it sends a cancel message to all upstream routers to whom it has previously relayed a honeypot request message. Recall that if

an access router is reached by the back-propagation process, it is because it is connected to an attack host. All honeypot sessions are removed except for the MAC-address-based filters installed at switch ports of attack hosts (as described above).

5.3. Design issues

Here, we discuss some design issues involved in the honeypot back-propagation scheme.

Message security: To prevent forging of honeypot request and cancel messages, which in itself can lead to DoS attacks, inter-AS honeypot request and cancel messages are encrypted and authenticated using shared keys between ASs, in a similar way to securing BGP sessions. Moreover, in intra-AS back-propagation, messages are sent hop-by-hop and, thus, are authenticated using the TTL field in the same way as in ACC/Pushback [27] (routers accept only messages from sources one-hop away, that is, with a TTL value of 255).

False positives: Packet marking traceback schemes can incur a high rate of false positives with compromised routers [43]. Honeypot back-propagation avoids these false positives. However, other sources of false positives may exist, and here we show how they can be avoided as well. Longitudinal studies of honeypots (e.g., [49]) show that honeypots receive a large amount of benign traffic, such as non-malicious probing. Activating honeypot back-propagation in response to these false positives causes high unnecessary overhead of installing and tearing down honeypot sessions. Server-to-server control traffic is another source of false positives. To tolerate false positives, honeypot request messages are sent only when the rate of received traffic exceeds a threshold or an attack is detected using other methods (e.g., [42]). Selection of an appropriate threshold depends on the type of the protected service and is a subject of future work.

Incremental deployment: In order for honeypot back-propagation to reach attack hosts and stop the attack, all ASs along the attack path as well as all routers inside attack-hosting ASs must support the scheme. However, partial deployment is possible with some benefits. Partial deployment may create gaps between ASs (or routers for intra-AS back-propagation) implementing the scheme, which would prevent further propagation of honeypot sessions. To bypass these deployment gaps, we use routing options to piggyback request and cancel messages over routing protocol messages. This is similar to the way mobile honeypots are propagated using BGP options [23]. When a HSM (router) fails to propagate honeypot sessions to upstream ASs (routers) from which honeypot traffic is received, the HSM (router) broadcasts the honeypot requests over routing announcements to all upstream ASs (routers). These announcements are propagated until they reach a deploying AS (router) from which point normal propagation is resumed.

Protection of the honeypot session manager (*HSM*): The HSM within an AS is a critical resource in the honeypot back-propagation defense. Therefore, it should be protected from targeted attacks. This protection can be achieved by replication and load balancing. Moreover, external attack traffic targeted at the HSM can be prevented by assigning a private IP address (e.g., 10.0.0.1) to the HSM. This way the HSM is not accessible from outside the AS. For HSMs to be able to communicate, BGP can be used, whereby control messages are carried over BGP messages between ASs and then distributed to the local HSM address within each AS.

Overhead of the scheme: Honeypot back-propagation achieves substantial performance gains under attack (see the next sections for an evaluation of this performance gain) at the cost of a small overhead. Three components contribute to this overhead. First, under no attacks the underlying roaming honeypots scheme causes performance degradation ranging from 14% [19] to as high as 100% [21] depending on legitimate client load. As described in [21], this degradation is caused by three factors: (1) the offered load, both legitimate and illegitimate, is distributed over $k < n$, instead of n servers; (2) when a server switches from active to idle, all its current legitimate connections move to another server, re-establish TCP connections and re-enter TCP slow-start, losing their current TCP throughput; and (3) legitimate connections flock simultaneously into new servers distorting the otherwise smoother request arrival distribution. However, by triggering the scheme only when attacks are detected, this performance degradation can be avoided. The second component of overhead is the computational and storage overhead inside routers. Other mechanisms require the same overhead (e.g. [27]), and the impact of this overhead is mitigated by activating the scheme only when attacks are detected. The third component is the network message overhead caused by the honeypot request and cancel messages exchanged over the attack tree. Although, the number of messages is linear in the number of attackers, the number of attack messages suppressed by the scheme is much higher.

6. Progressive honeypot back-propagation

In the honeypot back-propagation scheme described above, if an attack host sends few attack packets, honeypot sessions may not reach the attack host's AS (and access router in the intra-AS scheme) during a single honeypot epoch. This happens, for example, if the attack host launches an on–off attack with a short on-burst. To solve this problem, we propose progressive honeypot back-propagation, in which the attack path is discovered over more than one honeypot epoch as follows.

Let τ be the average time required for the honeypot request message to propagate one AS hop upstream and to setup a honeypot session as described in Section 5. The server S maintains a list of intermediate ASs to keep track of the last transit ASs at which no further propagation was possible at the last honeypot epoch. When a cancel message reaches a transit AS A, such as A_1 and A_2 in Fig. 3(a), the AS checks if it has sent any requests upstream. If it has not done so, that is, request propagation stopped at A, the HSM of A sends its identity A and a time stamp to S, which in turn calculates t_A, A's time distance in seconds from S and stores this information in the list of intermediate ASs. At $t_A + \tau$ seconds before the next honeypot epoch, a request message (Fig. 3(b)) is sent to each AS A in the

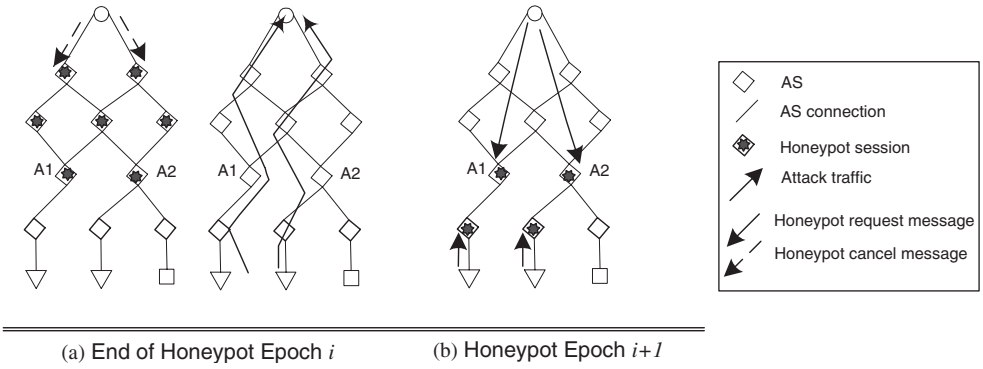

Fig. 3. An example of the operation of progressive inter-AS honeypot back-propagation in two consecutive honeypot epochs. (a) During epoch i, back-propagation stops at two transit ASs, A_1 and A_2, and honeypot sessions are canceled at epoch end. (b) In the next honeypot epoch, however, the server sends honeypot request messages directly to A_1 and A_2, and back-propagation resumes from there.

intermediate-AS list, allowing the back-propagation process to resume.

In order to maintain the intermediate-AS list and prevent explosion of its size, we apply the following two rules. First, we remove from the list any AS A added at honeypot epoch i if A did not send a message to S at the next honeypot epoch, $i+1$; as A does not send a message at honeypot epoch $i+1$, it either has propagated a honeypot request upstream or the report message was lost. In the former, A should be removed, and in the latter, which is a rare situation, propagation is restarted. The second rule is that S removes an intermediate-AS entry A if it receives reports from A in ρ consecutive honeypot epochs.

To implement these two rules, S keeps a flag as well as a counter for each intermediate-AS entry A in the list. The flag is used to detect if A did not send a message in the current honeypot epoch, in which case A should be removed according to the first rule. S increments the counter every time A reports its identity, and when the counter reaches the threshold ρ, A is removed from the list.

7. Analysis

In this section we study the expected time it takes to capture, that is, to reach and stop, the DDoS attack hosts using the honeypot back-propagation schemes. We also discuss some parameter selection guidelines.

7.1. Overview

We consider n_a DDoS attack hosts, with attack host i ($0 \leqslant i < n_a$) h_i hops away from the victim server S. Let CT_i be the capture-time of attack host i, that is the time needed to reach and stop host i. In general, the expected time to stop n_a attack hosts is

$$CT = \max_{0 \leqslant i < n_a} CT_i.$$

We assume the time is divided into epochs with constant length, m seconds. S is designated to act as a honeypot during each epoch with a *honeypot probability*, p. In the roaming honeypots scheme [21], $p = \frac{N-k}{N}$, where N is the total number of servers and k is the number of concurrent active servers. We assume full deployment, that is, all ASs (and routers within each AS hosting attackers) in the attack tree support the honeypot back-propagation scheme, and, we consider progressive honeypot back-propagation unless stated otherwise.

We use probabilistic analysis to derive conservative expressions of the average capture-time, $E[CT_i]$, of an attack host in the case of continuous and on–off attacks. We define Bernoulli trials so that a success occurs with probability p and results in overlapping of a honeypot epoch with the attack stream. During this time overlap, honeypot sessions are propagated a number of hops (ASs in inter-AS back-propagation and routers in intra-AS back-propagation) toward the attack host. The number of hops progressed per success is related to the amount of the attack-honeypot time overlap by the following equation: $\#\_hops\_per\_success = \frac{attack-honeypot\_overlap\_time\_per\_success}{time\_to\_propagate\_one\_hop}$. In order to guarantee progress, the attack-honeypot overlap time has to be long enough to propagate at least one hop for the progressive scheme and h_i hops for the basic scheme toward the attacker. Moreover, for progressive honeypot back-propagation, the threshold, ρ, for maintaining ASs in the intermediate-AS list must be large enough (see Section 6). Reaching the attack host needs a certain number of successes $n_{hp_i} = \frac{h_i}{\#\_hops\_per\_success}$, and $average\_\#\_trials\_needed = \frac{n_{hp_i}}{p}$. The average capture time is the average number of trials needed multiplied by the time between trials, $E[CT_i] = (average\_\#\_trials\_needed)(time\_between\_trials)$. Hence, for basic honeypot back-propagation, given that $attack - honeypot\_overlap\_time\_per\_success \geqslant (time\_to\_propagate\_one\_hop) \cdot h_i$,

$$E\left[CT_i^{\text{basic}}\right] = \left(\frac{1}{p}\right)(time\_between\_trials). \quad (1)$$

For progressive honeypot back-propagation, provided that $attack - honeypot\_overlap\_time\_per\_success \geqslant time\_to\_$

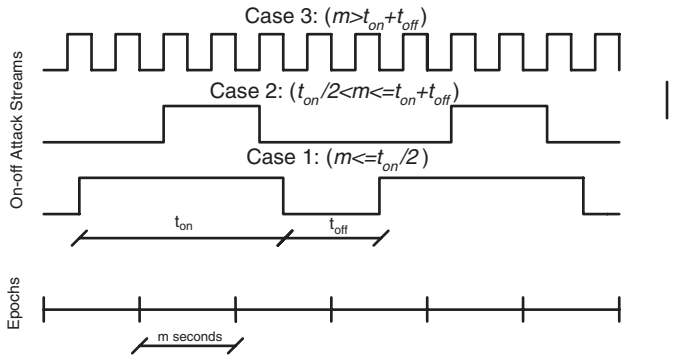

Fig. 4. Three different cases of on–off attacks.

propagate_one_hop,

$$E\left[CT_i^{\text{progressive}}\right] = \left(\frac{h_i}{p \cdot \#\_hops\_per\_success}\right) \times (time\_between\_trials). \quad (2)$$

In the next subsections, we detail the analysis for continuous and on–off attacks.

7.2. Continuous attack

In the continuous attack, each attack host, i, continuously injects packets at a constant rate, r_i packets/s. We define each time epoch as a Bernoulli trial with a probability p of success, that is, being a honeypot epoch. The attack-honeypot overlapping time is a full epoch, that is, m seconds and so is the time between trials. Also, the time needed to propagate one hop is $\frac{1}{r_i} + \tau$ seconds. This is because it takes about $\frac{1}{r_i}$ seconds [2] for a honeypot session to receive an attack packet from attack host i, and it takes on average τ seconds to propagate a honeypot session one hop upstream. Thus, the average number of upstream hops propagated per success (honeypot epoch) is $\geqslant \left\lfloor \frac{m}{\frac{1}{r_i}+\tau} \right\rfloor$. Substituting in Eqs. (1) and (2) yields:

$$E\left[CT_{i,\text{continuous}}^{\text{basic}}\right] \leqslant \frac{m}{p}, \quad \text{if } m \geqslant h_i\left(\frac{1}{r_i} + \tau\right), \quad (3)$$

$$E\left[CT_{i,\text{continuous}}^{\text{progressive}}\right] \leqslant \frac{m}{p}\frac{h_i}{\left\lfloor \frac{m}{\frac{1}{r_i}+\tau} \right\rfloor}, \quad \text{if } m \geqslant \frac{1}{r_i} + \tau. \quad (4)$$

7.3. On–off attack

Attack hosts launching on–off attacks alternate between sending packets at a rate of r_i packets/s during on-bursts for t_{on} seconds and stopping for t_{off} seconds. We consider three cases depending on the relation of m with t_{on} and t_{off}. These cases are illustrated in Fig. 4.

[2] To be more accurate, it takes at most $\frac{1}{r_i} + J_i$, where J_i is the jitter of the path from host i into S

Case 1 ($m \leqslant \frac{t_{\text{on}}}{2}$): In this case, the Bernoulli trial is each attack on-burst, which overlaps at least $\frac{t_{\text{on}}-m}{m}$ time epochs. On average $p\frac{t_{\text{on}}-m}{m}$ of these are honeypot epochs. Thus, the average attack/honeypot time overlap per on-burst is $p(t_{\text{on}}-m)$ seconds and the average number of hops per on-burst is $\left\lfloor \frac{p(t_{\text{on}}-m)}{\frac{1}{r_i}+\tau} \right\rfloor$.

Eqs. (1) and (2) yield:

$$E\left[CT_{i,\text{on–off}_1}^{\text{basic}}\right] \leqslant (t_{\text{on}} + t_{\text{off}}), \quad \text{if } m \geqslant h_i\left(\frac{1}{r_i} + \tau\right), \quad (5)$$

$$E\left[CT_{i,\text{on–off}_1}^{\text{progressive}}\right] \leqslant (t_{\text{on}} + t_{\text{off}})\frac{h_i}{\left\lfloor \frac{p(t_{\text{on}}-m)}{\frac{1}{r_i}+\tau} \right\rfloor}, \quad \text{if } m \geqslant \frac{1}{r_i} + \tau. \quad (6)$$

Case 2 ($\frac{t_{\text{on}}}{2} < m \leqslant (t_{\text{on}} + t_{\text{off}})$): In this case, each attack on-burst overlaps with exactly one epoch e for $t_{\text{on}}/2$ seconds or more. The Bernoulli trial is each attack on-burst, with success probability p that epoch e is a honeypot epoch and time between consecutive trials equal to $t_{\text{on}} + t_{\text{off}}$. Substitutions in Eqs. (1) and (2) yield:

$$E\left[CT_{i,\text{on–off}_2}^{\text{basic}}\right] \leqslant \frac{(t_{\text{on}} + t_{\text{off}})}{p}, \quad \text{if } \frac{t_{\text{on}}}{2} \geqslant h_i\left(\frac{1}{r_i} + \tau\right),$$
$$E\left[CT_{i,\text{on–off}_2}^{\text{progressive}}\right] \leqslant \frac{(t_{\text{on}} + t_{\text{off}})}{p}\frac{h_i}{\left\lfloor \frac{\frac{t_{\text{on}}}{2}}{\frac{1}{r_i}+\tau} \right\rfloor}, \quad \text{if } \frac{t_{\text{on}}}{2} \geqslant \frac{1}{r_i} + \tau. \quad (7)$$

A special case of Case 2 (*very small* t_{on}): The best attack strategy, in terms of increased attacker capture-time, fits in Case 2 and decreases t_{on} so that

$$\frac{t_{\text{on}}}{2} < 2\left(\frac{1}{r_i} + \tau\right) \quad \text{or} \quad \left\lfloor \frac{\frac{t_{\text{on}}}{2}}{\frac{1}{r_i} + \tau} \right\rfloor = 1 \quad (8)$$

and increases t_{off} as much as possible. In this case, Eq. (7) reduces to

$$E\left[CT_{i,\text{on–off}_{2*}}^{\text{progressive}}\right] = h_i\frac{(t_{\text{on}} + t_{\text{off}})}{p}. \quad (9)$$

Case 3 ($m > (t_{\text{on}} + t_{\text{off}})$): In this case, we let $T_m = t_{\text{on}} \cdot \left\lfloor \frac{m}{t_{\text{on}}+t_{\text{off}}} \right\rfloor$, so that each epoch overlaps with attack on-bursts for T_m or more seconds. We define each epoch as a Bernoulli trial with success probability p of being a honeypot epoch. Eqs. (1) and (2) yield:

$$E\left[CT_{i,\text{on–off}_3}^{\text{basic}}\right] \leqslant \frac{m}{p}, \quad \text{if } T_m \geqslant h_i\left(\frac{1}{r_i} + \tau\right), \quad (10)$$

$$E\left[CT_{i,\text{on–off}_3}^{\text{progressive}}\right] \leqslant \frac{m}{p}\frac{h_i}{\left\lfloor \frac{t_{\text{on}}\lfloor \frac{m}{t_{\text{on}}+t_{\text{off}}} \rfloor}{\frac{1}{r_i}+\tau} \right\rfloor}, \quad \text{if } T_m \geqslant \frac{1}{r_i} + \tau. \quad (11)$$

Follower attack: In the above analysis, we assumed that the attack traffic is independent of the selection of honeypot epochs. An interesting attack type that depends on the honeypot epochs is the follower attack, in which the attack host stops sending

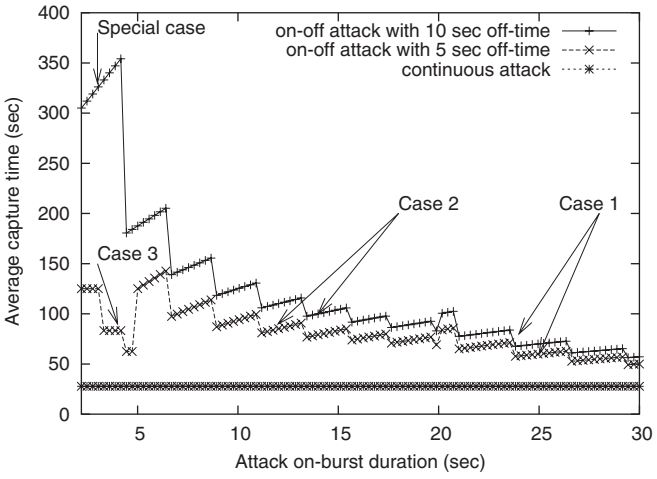

Fig. 5. Performance of progressive back-propagation against continuous and on–off attacks

traffic after the start of a honeypot epoch with a delay $d_{\text{follow}} > 0$ [21]. The average time to capture a follower attacker is

$$E\left[CT_{i,\text{follower}}^{\text{progressive}}\right] \leqslant \frac{m}{p}\frac{h_i}{max\left(1, \left\lfloor \frac{d_{\text{follow}}}{\frac{1}{r_i'}+\tau} \right\rfloor\right)}, \text{ if } d_{\text{follow}} \geqslant \frac{1}{r_i'} + \tau.$$

7.4. Comparison

We compare the performance of progressive honeypot back-propagation against continuous (Eq. (4)) and on–off (Eqs. (6), (7), (9) and (11)) attacks in Fig. 5. We plot the equations derived above against t_{on} with two values of t_{off}, namely 5 and 10 s. We use the parameters suggested in [21], $m = 10s$, $N = 5$, $k = 3$. The attack rate $r = 10$ packets/s, and the attack host is 10 hops away, that is, $h = 10$. Eq. (6) holds for $t_{\text{on}} \geqslant 20$ and Eq. (7) holds for $5 \leqslant t_{\text{on}} < 20$. For $t_{\text{off}} = 5$, Eq. (11) holds for $t_{\text{on}} < 5$. For $t_{\text{off}} = 10$, Eq. (9) holds for $2.2 \leqslant t_{\text{on}} < 4.4$. The plot validates our argument that the best attack strategy falls within Eq. (9). Having illustrated the relation between continuous and on–off attacks for progressive honeypot back-propagation, we henceforth consider only continuous attacks and basic honeypot back-propagation.

8. Simulation results

The main advantage of honeypot back-propagation stems from obtaining accurate and prompt attack signatures from the roaming honeypots. To analyze the effect of honeypot usage, we augment the Pushback scheme [27] with honeypot back-propagation and compare the performance of plain and augmented Pushback under different attack scenarios. We note that the Pushback framework is flexible enough to allow for the plugging of better attack detection, which our honeypot back-propagation scheme provides. Due to space limitation, we report some of our results; more detailed results can be found in [20].

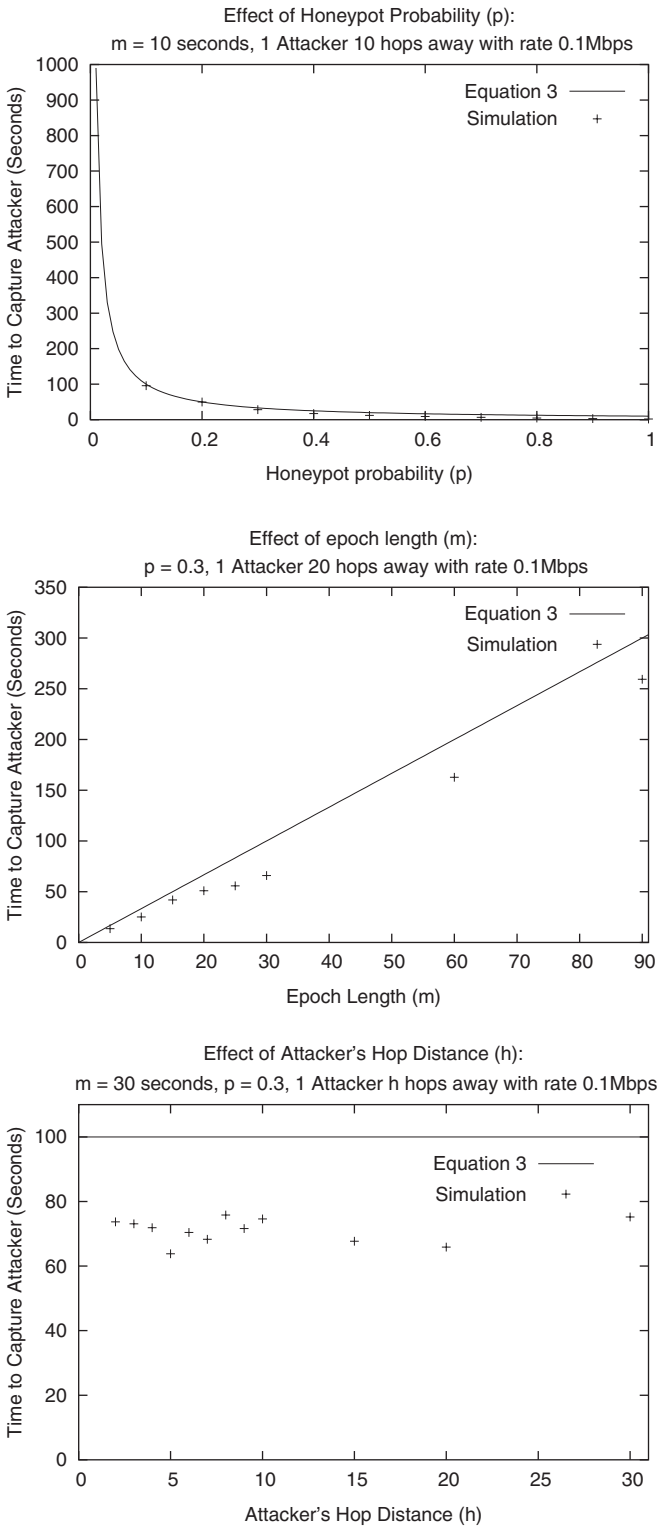

Fig. 6. Validation of Eq. (3).

8.1. ns-2 model

We modified the Pushback ns-2 module to implement intra-AS honeypot back-propagation. First, because we do not depend on the ACC, we disabled this feature. We defined a new message type for honeypot requests, and a new session type for

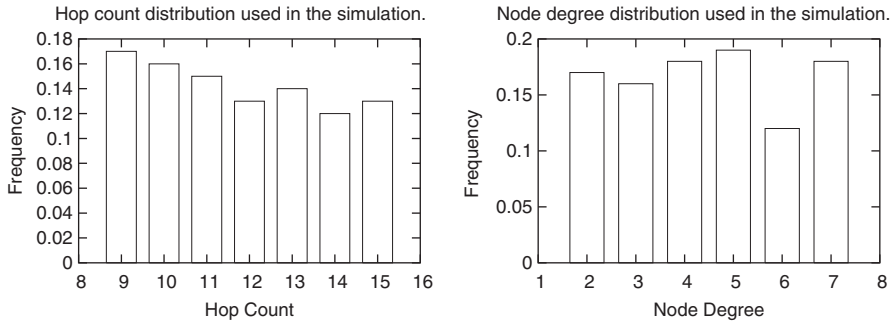

Fig. 7. Hop count and node degree distributions for the simulated topology.

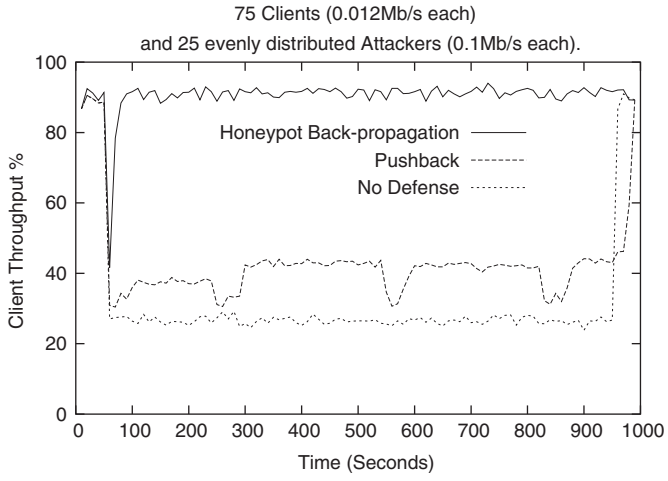

Fig. 8. Time plot of one simulation run. Attack is between 50 and 950 s.

honeypot sessions, and we modified the Pushback response to cancel messages in access routers to comply with our scheme. The roaming honeypots module [29] is also extended to send honeypot request and cancel messages, to support roaming with legitimate UDP traffic, and to start and end each honeypot epoch a little bit later and earlier, respectively, to accommodate in-transit legitimate traffic and clock synchronization.

8.2. Model validation

The first set of experiments validate the analysis presented in Section 7 for continuous attacks. To focus on the attack path, we use a string topology with one server at one end and an attacker at the other end. We vary the epoch length, m, the honeypot probability, p, and the hop distance between the attacker and the server, h. We measure the average time needed to capture the attacker over 100 simulation runs using basic honeypot back-propagation and plot this average against Eq. (3). The results, shown in Fig. 6 and in [20], support that Eq. (3) serves as an upper-bound of the average capture-time.

8.3. Simulation topology and parameters

Our simulation topology is a tree with hop-count and router-degree distributions shown in Fig. 7. This tree represents the network paths from legitimate clients and attack hosts to the servers. Although a tree collected using Internet measurements is more realistic, we use distributions roughly matching those of measured trees (e.g., [10,14]).

We model five servers behind a bottleneck link of 1 Mb/s capacity. The bottleneck represents the root of the tree topology. Links incident on leaf nodes have a 1 Mb/s capacity and 1 ms propagation delay, whereas links incident on servers are 10 Mb/s in capacity and 1 ms in propagation delay. All other links have a capacity of 10 Mb/s and a propagation delay of 10 ms. Although these capacities and propagation delays are not real, their relative values roughly represent relations between access and core links and were used to expedite the simulations.

Results with a network of 100 leaf nodes are reported in this paper, from which we select legitimate clients and attack hosts. Both legitimate clients and attackers send CBR traffic destined for the servers. In the results shown, we vary the number of legitimate nodes while keeping the total legitimate rate at about 90% of the bottleneck capacity (similar results were obtained with lower legitimate loads).

At the start of each periodic epoch, each legitimate client selects one of the three active servers uniformly at random and directs its traffic into it. In the case of Pushback and no-defense experiments, legitimate traffic is uniformly distributed over all five servers. Each attack host picks a server among the five servers uniformly at random and keeps on attacking it. Each simulation run lasts for 1000 s. Legitimate traffic starts at time 0, while attack traffic is from 50 to 950 s. We measure the throughput of legitimate traffic as a percentage of the total bottleneck link capacity. Fig. 8 shows an example of how the legitimate throughput changes with time during one simulation run. At 50 s, legitimate throughput suffers from a drop for all schemes. However, for honeypot back-propagation, the legitimate throughput quickly recovers after attack hosts are captured.

8.4. Effect of attack parameters

We study the effect of the location of attack nodes in terms of hop distance from victim servers, the number of attackers, and the attack rate per attack host for. Because we are interested in the system behavior under attack, in the next figures we average

Parameter(s)	Simulated Values
Bottleneck Link Capacity	1.0 Mb/s
Total Legitimate Rate	90% of bottleneck capacity
(Number of servers (N), Number of active servers (k))	(5, 3)
Epoch length	10 seconds
Total Number of Leaf Nodes	100
Number of Attackers	25, 50, 75
Attacker Locations	Far, Close, Evenly Distributed
Rate Per Attacker	(0.01, 0.05, 0.1, 0.5) Mb/s

Fig. 9. Simulation parameters.

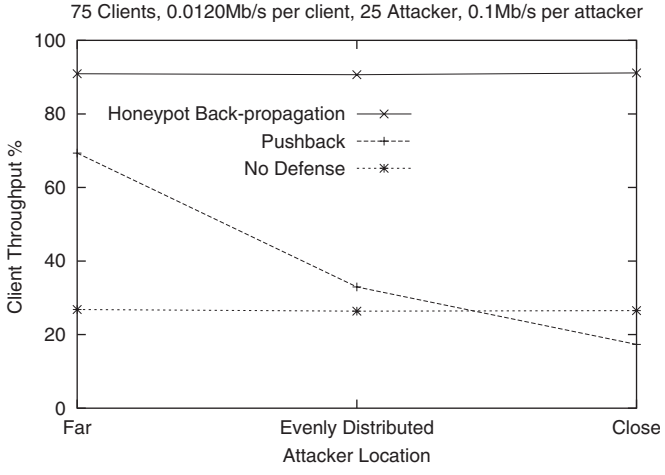

Fig. 10. Effect of attacker locations.

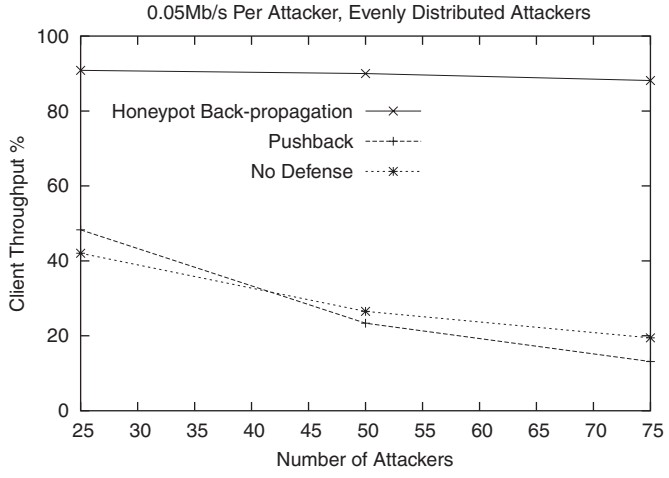

Fig. 11. Effect of number of attackers.

the client throughput during the attack time (i.e., from 50 to 950 s) of each simulation run. Fig. 9 summarizes the studied parameters and the values we experiment with.

8.4.1. Location of attackers

We consider three scenarios of attacker locations: (a) close attackers, where the attackers are assigned to the closest leaves to the victim servers in the topology tree, (b) far attackers, in which attackers are assigned to the furthest leaves from the servers, and (c) evenly distributed attackers, in which the location of attack nodes are selected uniformly at random over all leaf nodes. In all scenarios, legitimate clients occupy the remaining leaf nodes.

As depicted in Fig. 10, as the locations of the attackers get closer to the servers, ACC/Pushback punishes legitimate traffic more. The reason for this behavior is that ACC/Pushback adopts a hop-by-hop max–min fairness allocation of the rate limit among upstream routers without taking into consideration the number of end hosts behind each upstream router. So, for example, the fair share of an end-host connected to a router with another two upstream routers is one-third of the rate limit irrespective of the fact that the two routers may have many upstream end-hosts connected through them. The result of this behavior is that as attackers get closer to the victim, their fair share increases, reducing the residual rate for legitimate clients. For close attackers, ACC/Pushback is even worse than no defense at all because it actually protects attack traffic.

8.4.2. Number of attackers

Fig. 11 shows that, for ACC/Pushback and for evenly distributed attackers, as the number of attackers increases, the number of attackers close to the victim increases, leading to an increase of the (protected) attack rate, and thus, the negative impact on legitimate throughput consequently increases. However, for far attackers, that is, close legitimate clients (not shown), client throughput is independent of the number of attackers, because in this case ACC/Pushback protects the legitimate traffic of the clients.

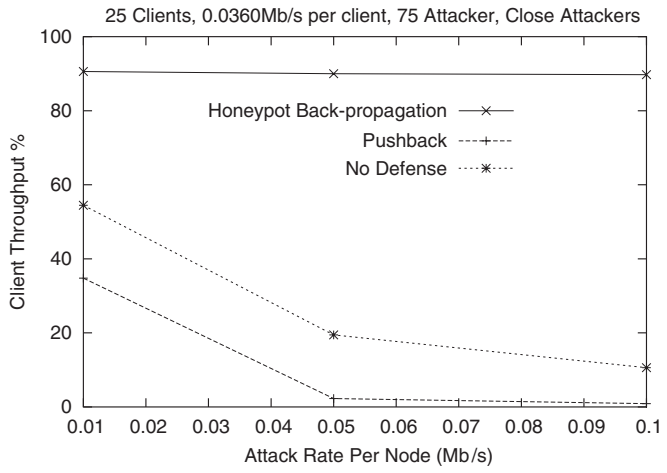

Fig. 12. Effect of attack rate per node.

As long as attacks last for enough time for honeypot back-propagation to reach their sources, which is the case in Fig. 11, honeypot back-propagation is independent of the number of attackers. However, for higher number of attackers, some attack hosts may not be reached by honeypot back-propagation with our parameter values. For example, with 500 attackers, if each attacker sends at a rate of 5 kb/s, the 1 Mb/s bottleneck link will be clogged. With a packet size of 1000 bytes basic scheme, epoch length has to be at least $\geqslant \frac{1000*8\,\text{bits}}{5\,\text{kb/s}} \cdot h_i \approx 1.6 \cdot h_i$ seconds to reach an attacker h_i hops away from the server, without accounting of the delay of honeypot session propagation. For instance, to reach an attacker 10 hops away, the epoch length has to be at least 16 s.

8.4.3. Rate per attacker

As shown in Fig. 12, for ACC/Pushback and close attackers, as the attack rate per attacker increases, the attack can grab more of the ACC/Pushback-enforced rate limit, resulting in reduction of legitimate throughput. For far attackers, however, legitimate throughput is independent of the individual rate per far attacker, because of ACC/Pushback protection of legitimate traffic of the close clients [20].

9. Conclusion

In this paper we presented *honeypot back-propagation*, a hierarchical traceback defense against DDoS attacks with spoofed source addresses. Each server acts as a honeypot for specific time intervals, honeypot epochs, whose durations are unpredictable to attackers. During these epochs, the server receives pure attack streams, which trigger back-propagation of honeypot sessions towards attackers. Honeypot back-propagation supports incremental deployment and incurs a small overhead, since it is activated only during attacks.

We analytically derived expressions for the expected time to stop a DDoS attack. Through ns-2 simulations, we validated our models, showed the feasibility of the honeypot back-propagation scheme, and confirmed its added benefit to the ACC/Pushback defense.

Acknowledgments

The authors were supported in part by NSF under Grant ANI-0087609.

References

[1] S. Agarwal, T. Dawson, C. Tryfonas, DDoS mitigation via regional cleaning centers, Technical Report, RR04-ATL-013177, SPRINT ATL Research (January 2004).
[2] S.M. Bellovin, M. Leech, T. Taylor, ICMP traceback messages, in: draft-ietf-itrace-01.txt, internet-draft, October 2001. Expired draft.
[3] A. Bremler-Barr, H. Levy, Spoofing prevention method, in: IEEE INFOCOM, 2005.
[4] H. Burch, B. Cheswisk, Tracing anonymous packets to their approximate source, in: Proceedings of the 14th Systems Administration Conference (LISA), 2000.
[5] CAIDA, Nameserver DoS Attack, ⟨http://www.caida.org/funding/dns-analysis/oct02dos.xml⟩ (October 2002).
[6] CAIDA, SCO Offline from Denial-of-Service Attack, ⟨http://www.caida.org/analysis/security/sco-dos/⟩ (December 2003).
[7] CERT, MS-SQL Server Worm, Advisory CA-2003-04, ⟨http://www.cert.org/advisories/CA-2003-04.html⟩ (January 2003).
[8] W. chang Feng, The case for TCP/IP puzzles, in: Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture, 2003, pp. 322–327.
[9] CISCO, CISCO Group Management Protocol (CGMP), available at: ⟨http://www.cisco.com/warp/public/473/22.html⟩ (2006).
[10] Cooperative Association for Internet Data Analysis, ⟨http://www.caida.org/⟩.
[11] D. Dean, M. Franklin, A. Stubblefield, An algebraic approach to IP traceback, ACM Trans. Inf. Syst. Secur. 5 (2) (2002) 119–137.
[12] D. Farinacci, T. Li, S. Hanks, D. Meyer, P. Traina, Generic routing encapsulation (GRE), in: RFC 2784, 2000.
[13] P. Ferguson, D. Senie, Network ingress filtering: defeating denial of service attacks which employ IP Source Address Spoofing, in: RFC 2827, 2001.
[14] Internet Mapping Project, ⟨http://research.lumeta.com/ches/map/⟩ (2004).
[15] J. Ioannidis, S.M. Bellovin, Implementing pushback: router-based defense against DDoS attacks, in: Proceedings of Network and Distributed System Security Symposium (NDSS), 2002.
[16] S. Kent, R. Atkinson, Security architecture for the Internet protocol, in: RFC 2401, 1998.
[17] A. Keromytis, V. Misra, D. Rubenstein, SOS: secure overlay services, in: ACM SIGCOMM, 2002.
[18] S.M. Khattab, R. Melhem, D. Mossé, T. Znati, Honeypot back-propagation for mitigating spoofing distributed denial-of-service attacks, in: Proceedings of the Second International Workshop on Security in Systems and Networks (SSN'06) (April 2006).
[19] S.M. Khattab, C. Sangpachatanaruk, R. Melhem, D. Mossé, T. Znati, Proactive server roaming for mitigating denial-of-service attacks, in: Proceedings of the International Conference on Information Technology: Research and Education (ITRE), 2003.
[20] S.M. Khattab, C. Sangpachatanaruk, R. Melhem, D. Mossé, T. Znati, Honeypot back-propagation for mitigating spoofing distributed denial-of-service attacks, Technical Report, TR-04-111, Department of Computer Science, University of Pittsburgh (September 2004).
[21] S.M. Khattab, C. Sangpachatanaruk, D. Mossé, R. Melhem, T. Znati, Roaming honeypots for mitigating service-level denial-of-service attacks, in: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04), March 2004.
[22] S.M. Khattab, C. Sangpachatanaruk, D. Mossé, R. Melhem, T. Znati, Roaming honeypots for mitigating service-level denial-of-service attacks, under submission, available at: ⟨http://www.cs.pitt.edu/skhattab/roaming.pdf⟩ (2006).
[23] B. Krishnamurthy, Mohonk: mobile honeypots to trace unwanted traffic early, in: Proceedings of the ACM SIGCOMM Workshop on Network Troubleshooting (NetT '04), 2004, pp. 277–282.

[24] A. Kuzmanovic, E. W. Knightly, Low-rate TCP-targeted denial of service attacks. (The Shrew vs. the Mice and Elephants), in: ACM SIGCOMM, 2003.
[25] J. Levine, R. LaBella, H. Owen, D. Contis, B. Culver, The use of honeynets to detect exploited systems across large enterprise networks, in: Proceedings of the IEEE Workshop on Information Assurance and Security, 2002.
[26] J. Li, M. Sung, J. Xu, L. Li, Large-Scale IP traceback in high-speed internet: practical techniques and theoretical foundation, in: Proceedings of IEEE Symposium on Security and Privacy, 2004.
[27] R. Mahajan, S.M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, S. Shenker, Controlling high bandwidth aggregates in the network, ACM SIGCOMM CCR 32 (3) (2002) 62–73.
[28] J. Mirkovic, P. Reiher, A taxonomy of DDoS attack and DDoS defense mechanisms, ACM SIGCOMM CCR 34 (2) (2004) 39–53.
[29] NetSec Group, ⟨http://www.cs.pitt.edu/NETSEC⟩.
[30] M. Oe, Y. Kadobayashi, S. Yamaguchi, An implementation of a hierarchical IP traceback architecture, in: Proceedings of the Symposium on Applications and the Internet Workshops (SAINT) Workshop on IPv6 and Applications, 2003.
[31] K. Park, H. Lee, On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack, in: IEEE INFOCOM, 2001, pp. 338–347.
[32] K. Park, H. Lee, On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets, in: ACM SIGCOMM, 2001.
[33] C. Perkins, IP mobility support, in: RFC 2002, 1996.
[34] A. Perrig, D. Song, A. Yaar, StackPi: a new defense mechanism against IP spoofing and DDoS attacks, Technical Report, CMU-CS-02-208, School of Computer Science, Carnegie Mellon, Pittsburgh, PA, USA (December 2002).
[35] T.H. Project, Know Your Enemy, Addison-Wisley, Indianapolis, IN, 2002.
[36] G. Sager, Security fun with OCxmon and cflowd, Internet2 working group meeting, November 1998.
[37] C. Sangpachatanaruk, S.M. Khattab, T. Znati, R. Melhem, D. Mossé, A simulation study of the proactive server roaming for mitigating denial of service attacks, in: Proceedings of the 36th Annual Simulation Symposium (ANSS), 2003.
[38] S. Savage, D. Wetherall, A. Karlin, T. Anderson, Practical network support for IP traceback, in: ACM SIGCOMM, 2000.
[39] V. Siris, I. Stavrakis, Provider-based deterministic packet marking against distributed DoS attacks, in: Proceedings of the International Workshop on Security in Systems and Networks (SSN), 2005.
[40] A.C. Snoeren, H. Balakrishnan, M.F. Kaashoek, The migrate approach to Internet mobility, in: Proceedings of the Oxygen Student Workshop, 2001.
[41] A.C. Snoeren, C. Partridge, L.A. Sanchez, C.E. Jones, F. Tchakountio, B. Schwartz, S.T. Kent, W.T. Strayer, Single-packet IP traceback, IEEE/ACM Transactions on Networking 10 (6) (2002) 721–734.
[42] Snort, ⟨http://www.snort.com⟩ (2004).
[43] D.X. Song, A. Perrig, Advanced and authenticated marking schemes for IP traceback, in: IEEE INFOCOM, 2001.
[44] S. Staniford, V. Paxson, N. Weaver, How to own the Internet in your spare time, in: Proceedings of the 11th USENIX Security Symposium, 2002.
[45] R. Stone, CenterTrack: An IP overlay network for tracking DoS floods, in: Proceedings of the Ninth USENIX Security Symposium, 2000.
[46] F. Sultan, K. Srinivasan, D. Iyer, L. Iftode, Migratory TCP: connection migration for service continuity in the Internet, in: Proceedings of the International Conference on Distributed Computing Systems (ICDCS), 2002.
[47] The Network Simulator-ns-2, ⟨http://www.isi.edu/nsnam/ns/⟩.
[48] X. Wang, M. Reiter, Mitigating bandwidth-exhaustion attacks using congestion puzzles, in: Proceedings of the ACM Conference on Computer and Communications Security (CCS), 2004.
[49] N. Weiler, Honeypots for distributed denial-of-service attacks, in: Proceedings of the IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2002.
[50] A. Yaar, A. Perrig, D. Song, FIT: fast internet traceback, in: IEEE INFOCOM, 2005.
[51] D.K.Y. Yau, J.C.S. Lui, F. Liang, Defending against distributed denial-of-service attacks with max–min fair server-centric router throttles, in: Proceedings of the IEEE International Workshop on Quality of Service (IWQoS), 2002.

Sherif Khattab received a B.E. in Computer Engineering from Cairo University in 1998 and an M.S. degree in Computer Science from the University of Pittsburgh in 2004. He is currently pursuing his Ph.D. in Computer Science at the University of Pittsburgh. His research focuses on network security in both wired and wireless networks.

Rami Melhem received a B.E. in Electrical Engineering from Cairo University in 1976, an M.A. degree in Mathematics and an M.S. degree in Computer Science from the University of Pittsburgh in 1981, and a Ph.D. degree in Computer Science from the University of Pittsburgh in 1983. He was an Assistant Professor at Purdue University prior to joining the faculty of The University of Pittsburgh in 1986, where he is currently a Professor of Computer Science and Electrical Engineering and the Chair of the Computer Science Department. His research interests include Real-Time and Fault-Tolerant Systems, Optical Networks, High Performance Computing and Parallel Computer Architectures. Dr. Melhem served on program committees of numerous conferences and workshops. He was on the editorial board of the IEEE Transactions on Computers and the IEEE Transactions on Parallel and Distributed systems. He is serving on the advisory boards of the IEEE technical committees on Computer Architecture. He is the editor for the Springer Book Series in Computer Science and is on the editorial board of the Computer Architecture Letters, The International Journal of Embedded Systems and the Journal of Parallel and Distributed Computing. Dr. Melhem is a fellow of IEEE and a member of the ACM.

Daniel Mosse received a B.S. in Mathematics from the University of Brasilia in 1986, and M.S. and Ph.D. degrees in Computer Science from the University of Maryland in 1990 and 1993, respectively. He joined the faculty of The University of Pittsburgh in 1992, where he is currently a Professor. His research interest include fault-tolerant and real-time systems, as well as networking. The major thrust of his research in the new millennium is real-time systems, power management issues, and networks (wireless and security). Typically funded by NSF and DARPA, Dr. Mosse's projects combine theoretical results and actual implementations.
Dr. Mosse has served on program committees for all major IEEE- and ACM-sponsored real-time related conferences and as the program and general chairs for RTSS, RTAS, Brazilian RT Workshp, and RT Education Workshop. Dr. Mosse has been an associate editor of IEEE Transactions on Computers and is currently on the editorial board of the Journal of Real-Time Systems. He is currently a member of IEEE Computer Society, the Association for Computing Machinery, and the American Society for Engineering Education.

Dr. Znati is currently a Professor in the Department of Computer Science, with a joint appointment in Telecommunications in the Department of Information Science at the University of Pittsburgh. He also served as a Senior Program Director for networking research at the National Science Foundation, in the Division of Advanced Networking Infrastructure and Research, and later in the Division of Computer and Network Systems within the Computer Information Systems and Engineering Directorate. In the fourth year of his tenure at NSF, he served as the Chair of the Information Technology Research Initiative (ITR), an NSF cross-directorate program.
Dr. Znati's current research interests focus on the design of network architectures and protocols for wired and wireless communication networks to support applications' QoS and security requirements. Dr. Znati served as the General Chair of IEEE INFOCOM 2005, SECON 2004, the first IEEE conference on Sensor and Ad Hoc Communications and Networks, the Annual Simulation Symposium, and UbiCare'06 the first workshop on ubiquitous and pervasive healthcare. He is a member of the Editorial Board of the International Journal of Parallel and Distributed Systems and Networks, the Pervasive and Mobile Computing Journal, the Journal on Wireless Communications and Mobile Computing.