Available online at www.sciencedirect.com

ScienceDirect

Computer Networks 51 (2007) 4252–4269

Computer Networks

www.elsevier.com/locate/comnet

AID: A global anti-DoS service

Shigang Chen [a,*], Yibei Ling [b], Randy Chow [a], Ye Xia [a]

[a] *Department of Computer and Information Science and Engineering, University of Florida, United States*
[b] *Applied Research Laboratories, Telcordia Technologies, United States*

Received 4 September 2006; received in revised form 14 January 2007; accepted 22 May 2007
Available online 5 June 2007

Responsible Editor: R. Molva

Abstract

Distributed denial of service (DDoS) has long been an open security problem of the Internet. Most proposed solutions require the upgrade of routers across the Internet, which is extremely difficult to realize, considering that the Internet consists of a very large number of autonomous systems with routers from different vendors deployed over decades. A promising alternative strategy is to avoid the universal upgrade of router infrastructure and instead rely on an overlay of end systems. The prior anti-DoS overlays were designed to protect emergency services for authorized clients. They assume that trust exists between authorized clients and a private server. Only authenticated traffic can pass through the overlay network to reach the server, while the attack traffic is not admitted without passing the authentication. The follow-up extension of the anti-DoS overlays for web service has other serious limitations. This paper attempts to solve an important problem. How to design an anti-DoS overlay service (called AID) that protects general-purpose public servers while overcoming the limitations of the existing systems? Anyone, including the attackers, should be able to access the server. Authentication can no longer be the means of defense. While both normal and malicious clients are given the access, AID is designed to fend off attack traffic while letting legitimate-traffic through. Its operations are completely transparent to the users (humans or hosts), the client/server software, and the internal/core routers. To connect the AID service nodes (which are end systems), we choose a random overlay network for its rich, unpredictable connectivity, short diameter, and ease of management. We use a distributed virtual-clock packet scheduling algorithm to restrict the amount of data any client can impose on AID. We analyze the properties of the AID service based on probabilistic models. Our simulations demonstrate that AID can effectively protect legitimate-traffic from attack traffic. Even when 10% of all clients attack, just 1.4% of legitimate-traffic is mistakenly blocked, no matter how aggressive the attackers are.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Network-level security and protection; Denial of service attacks; Overlay networks

1. Introduction

The goal of a DoS (denial of service) attack is to completely tie up certain resources so that legitimate users are not able to access a server. A successful DoS attack achieves two objectives: overpowering

* Corresponding author. Tel.: +1 352 392 2713; fax: +1 352 392 1220.
E-mail address: sgchen@cise.ufl.edu (S. Chen).

1389-1286/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2007.05.005

the victim and concealing the offender's identity. In a DDoS (distributed denial of service) attack, multiple attack sources launch a coordinated offense against one victim, which increases the resources for the offense while making it harder to track down the attackers.

There are two types of DoS attacks. In a *high-rate attack*, each malicious client aggressively sends data to the server. Even when the number of malicious clients is smaller than the number of normal clients, the attack traffic can still overwhelm the legitimate traffic.[1] In a *low-rate attack*, the number of malicious clients is far greater than the number of normal clients. The aggregate attack traffic is overwhelming even when each malicious client sends data at a low-rate, making it indistinguishable from a normal client. High-rate DoS attacks are more common because it is not always possible for an attacker to acquire tens of thousands of compromised, geographically dispersed computers that are needed to launch low-rate attacks against popular servers on the Internet. This paper focuses on high-rate attacks.

1.1. Background

1.1.1. Router-based defense

Much work against DoS is on spoofing prevention. Ferguson and Senie proposed *ingress filtering* [1], which requires the edge routers of stub networks to inspect outbound packets and discard those packets whose source addresses do not belong to the stub networks. Park and Lee pioneered the concept of *route-based distributed packet filtering* [2]. The basic idea is for a router to drop a packet if the packet is received from an adjacent link that is not on any routing path from the packet's source to the packet's destination. Wang et al. proposed *SYN-dog* [3], a software agent installed at the edge routers of stub networks. The agent detects SYN-flooding from the attached networks by monitoring the difference between outbound SYN packets and inbound SYN/ACK packets. For all above approaches, the effectiveness of preventing DoS comes only after the deployment has been carried out widely across the Internet.

[1] Assume a server can handle all legitimate-traffic when it is not under attack. For a DoS attack to succeed, the volume of attack traffic must be far greater than the volume of the legitimate traffic. For example, if the former is nine times of the later, then it can cause up to 90% of legitimate-traffic to be dropped, as the server has to discard those requests when its capacity is largely consumed by the attack traffic.

Observing that not all routers in a network are capable of performing hop-by-hop traffic diagnostics to find flooding sources, Stone proposed CenterTrack [4] to connect all edge routers via IP tunnels to a mesh of special tracking routers, forming an IP overlay. Selected packets are rerouted through these tracking routers, which perform hop-by-hop traffic analysis towards the flooding sources.

In recent years, there has been a flourish of research work on *IP traceback* based on packet audit or route inference, whose goal is to find the origins of packets with spoofed source addresses [5–8]. IP traceback is a reactive approach, which does not prevent spoofed packets from harming their victims [9]. Sung and Xu proposed to use IP traceback to identify the network links that carry attack traffic and to preferentially filter out packets that are inscribed with the marks of those links [9]. Yaar et al. proposed *path identifier (Pi)*, a novel approach that assigns the same mark to packets traversing the same path and different marks to packets traversing different paths. Because the attack packets from the same source always carry the same mark, the victim is able to filter out the packets with identified attack marks. The effectiveness of these approaches also require wide deployment in order for most legitimate-traffic to be marked differently from the attack traffic.

Mahajan et al. proposed aggregate-based congestion control (ACC) to rate-limit the identified attack traffic [10]. A congested router starts with local rate-limit, and progressively pushes the rate limit to some neighbor routers and further out, forming a dynamic rate-limit tree. Routers in the tree perform filtering based on their shares of the rate-limit.

Anderson et al. proposed a capability-based approach to prevent DoS attacks [11]. The basic idea is to augment each participating BGP router with a RTS server that issues authorizing tokens and a VP function that verifies tokens. A source first acquires a capability token from the RTS server at the destination site. The token is cached by the intermediate VPs on the path from the source to the destination. The subsequent packets will carry the token and only certain number of packets with the token will be permitted by the VPs. This approach deposits per-flow soft state (e.g., token, source/destination addresses, etc.) at the intermediate VPs. Similar to other router-based solutions, the deployment involves routers of different ASs (Autonomous Systems) from all over the Internet.

The recent spoofing prevention method (SPM) [12] by Bremler-Barr and Levy allows step-wise deployment by combining *packet tagging* at the egress edge router of the source AS and *tag verification* at the ingress edge router of the destination AS. The method works at the AS level. However, some ASs are very big. Because a malicious host within an AS can forge any source addresses belonging to the AS, it is desirable for a defense system to have the flexibility of working at arbitrary network-levels, down to the edge routers of LANs. For SPM to achieve such flexibility, each participating edge router must explicitly learn the address ranges behind all other participating edge routers, which is hard to achieve at the Internet scale.

1.1.2. Host-based defense

The host-based solutions require modifications to servers, e.g., SYN cookies [13] and SYN cache [14], which handle SYN-flooding attacks only, or require modifications to both servers and client hosts, e.g., client puzzles [15].

1.1.3. Overlay-based defense

Keromytis et al. proposed a novel architecture called *Secure Overlay Services* (SOS) [16], which proactively prevents DoS attacks. It is designed for emergency services. A certificate for accessing a protected server must be issued to each authorized client. Client requests are first authenticated and then routed via a Chord overlay network [17] to one of the servlets, which forward the requests to the target site. The defense against DoS relies on client authentication and the secrecy of the servlets' locations. Mayday [18] by Andersen is a generalization of SOS. It studies a variety of choices for authentication, routing, and filtering. It also assumes a closed group of trusted clients that can be authenticated. Neither SOS nor Mayday handles attacks from compromised clients. Since authentication is fundamental to SoS and Mayday, they are not suitable for protecting a general public server (such as Yahoo or Google), because all users (including the attackers) are automatically authorized, which makes the authentication itself meaningless.

In general, SOS requires special client software. WebSOS applies the SOS architecture to web service. Using the web proxy feature and TLS, it requires no modification to web browsers and web servers. However, the first version [19] continued assuming a group of trusted clients, each having a public key certificate from the WebSOS administrator. The second version [20] attempted to solve this problem by using Graphic Turing Tests to separate human users from automated attackers. Before granting the access to a protected web server, the system presents a distorted image of an arbitrary word on the browser of a user. It relies on the fact that humans can read the word within the distorted image and the current automated tools cannot. In reply, the human "authenticates" himself/herself by entering the word from the image in ASCII. This approach has a number of limitations. First, it is designed only for web service because the browser can support such exchange without modification. The same thing is not true for other applications, particularly, text/voice applications whose client software does not handle images. Second, even for web browsing, the change of user experience can be undesirable. Imagine that you have to view and reply a distorted image each time you access Google. Third, it does not protect applications that do not always involve human users. Even for web browsing, some legitimate software may be developed to automatically mine data on the web.

1.2. Our contributions

We study the problem of providing an *anti-DoS service* (called AID) for public servers. The practical advantages of such a service are apparent. It meets the need of the enterprises (e.g., retailers, banks, libraries, web portals, etc.) that do not have security expertise but want to outsource their anti-DoS operations. It supports incremental deployment on a registration basis. A registered network (at any level) receives immediate protection, independent of what others do. Below are the design features of the AID service:

1. AID is open to all clients and does not require authentication. Any client, including an attacker, can register for the service *without any restriction*. AID is able to mitigate DoS attacks launched from both unregistered attackers and registered attackers. This property overcomes the limitation of SOS and Mayday whose defense relies on client authentication.
2. AID does not require the upgrade of client/server software, OS, or protocols at the hosts. It is transparent to human users, which overcomes the limitation of WebSOS. Its service nodes are end systems. This allows great flexibility in deployment locations. This property distinguishes

AID from most host-based or router-based solutions.
3. AID is scalable, efficient, and light-weighted. The defense is activated only when a DoS attack occurs.

AID uses a random overlay network that connects registered client networks via service nodes to registered servers. The overlay topology is simple and easy to manage, with each node having a small number of randomly-selected nodes as its neighbors. The maximum routing distance is three hops with the majority node pairs just two hops away. We design a distributed virtual-clock packet scheduling algorithm, which is executed on the overlay network to mitigate DoS attacks. It ensures that most attack packets are dropped while most legitimate packets are delivered. We analyze the properties of the AID service, including how fast AID mitigates an attack and what percentage of legitimate-traffic will be mistakenly blocked. Such analysis is often lacked in the prior works. Our simulations show that most legitimate-traffic survives even when the majority of all clients attack. We also discuss the deployment/implementation/robustness issues and how to minimize the traffic carried by the overlay network.

The rest of the paper is organized as follows. Section 2 motivates the concept of self-complete defense systems. Section 3 describes the AID service. Section 4 discusses various issues. Section 5 analyzes the property of the system. Section 6 presents the simulation results. Section 7 draws the conclusion.

2. Motivation

2.1. Self-complete defense systems

Consider a networked system of $S + C$, where S is a set of servers and C is a set of client networks. In a DDoS attack, a group of attackers from some client networks A ($\subset C$) flood a server s ($\in S$), such that the legitimate clients from $C - A$ are not able to access s. In this paper, it is not important to distinguish between multiple attackers and a single attacker launching the offense from multiple Zombies. When we say "multiple attackers", we mean either of the two cases.

On the Internet, any practical defense system must support incremental deployment. Suppose a defense system is installed on $S' + C'$, where $S' \subset S$ and $C' \subset C$. The system is said to be *self-complete* if a client network in C' is able to access any server in S' during a DoS attack as long as the client itself does not participate in the attack. A self-complete system must defeat both the "internal" attackers of $A \cap C'$, which are within the defense coverage, and the "external" attackers of $A \cap (C - C')$, which are outside of the defense coverage. Most existing defense systems are not self-complete. A few examples are given below.

Example 1 (*Ingress filtering* [1]). Suppose all networks in C' perform ingress filtering and those in $C - C'$ do not. The attackers from $C - C'$ can forge the source addresses and pretend that the requests are from a client network in C'. This attack traffic is not filtered. The server cannot distinguish the attack traffic from the legitimate-traffic, and has to drop both. Hence, ingress filtering is not self-complete. SYN-dog [3] is also not self-complete by a similar analysis.

Example 2 (*Route-based distributed packet filtering* [2]). The original paper demonstrated that a partial deployment (on 18% of Internet ASs) can effectively prevent spoofed packets from reaching their destinations. However, 18% of the Internet consists of a large number of ASs, and consequently the system may not be effective even after C' contains hundreds of ASs. Before C' is sufficiently large to cover most Internet routing paths, attackers from $C - C'$ can still flood a server if the routing paths from some attackers to the server do not contain routers from C'. Therefore, the route-based filtering is not self-complete.

Similar to [2], the IP traceback systems are not self-complete because, during a partial deployment, the ASs that deploy the traceback system may be denied of accessing a remote server, if there exists a routing path from an attacker to the server and the traceback is not implemented on the path, or it is partially implemented on the path such that some legitimate-traffic is mixed with and indistinguishable from the attack traffic. Take Pushback [10] as another example. In order to ensure a group of remote networks to access a server, all intermediate routers, or at least the edge routers of all intermediate ASs, must implement Pushback, such that the attack traffic can be rate-limited before entering the routing paths from those networks to the server. Hence, the protection of a remote network depends on the actions of all intermediate ASs.

When an organization deploys a system that is not self-complete such as ingress filtering, it essentially takes a *good-citizen* strategy in helping the global effort of defeating DDoS attacks.[2] But the benefits for its own hosts arrive only after other organizations on the Internet are also good-citizens and, moreover, implementing the same defense. On the other hand, if an organization joins a self-complete system such as AID proposed in this paper, its hosts are immediately protected with only one exception – the protection is voided if some of its hosts participate in the attack. In other words, a self-complete system makes sure (1) that the external attackers outside of the system have no impact on the clients in the system, and (2) that the internal attackers in the system will only harm themselves but not others that behave normally.

For a self-complete system to defend *most of the Internet*, it also requires *wide deployment*. This is obvious because the system only protects the networks that it *covers*. But a self-complete system does not require wide deployment before being effective, whereas most current systems do require that, which has been explained above. Even at the initial phase of incremental deployment, a self-complete system provides immediate protection among those (maybe a small number of) networks that join the system.

In the following, we motivate how to design a self-complete system that defeats attackers from both within and outside the system coverage.

2.2. Overlay defense systems

What causes many existing defense systems not self-complete? Because they cannot handle the external attackers not covered by the systems. Unless universally deployed, a defense system covers only a portion of the Internet. It effectively handles the internal attacks launched from this portion, but cannot always fend off the external attacks launched from outside. For example, networks that do not implement ingress filtering can forge arbitrary traffic to overwhelm a server.

To solve this problem, we must have a way to differentiate the traffic within the system's coverage from the traffic outside. One solution is to create an overlay network to carry the former while letting the Internet to carry the latter. When a DoS attack occurs, the traffic from the overlay network is given a reserved portion of the server's capacity, which effectively eliminates the impact of the external attackers. Moreover, the overlay connections bypass the requirement of infrastructure support at the network layer, and the defense can be expanded across the Internet to reach the edge of the customer networks. In comparison, a router-based solution (e.g., IPtraceback or pushback) is limited within the boundary of the AS that deploys it.

The existing overlay defense systems either require client authentication or have other serious limitations. We attempt to remove these limitations, which requires new architecture as well as new techniques.

3. A global anti-DoS service

We describe a global anti-DoS service, called *AID*, which is a self-complete defense system. Anyone (including attackers) can register, but only the well-behaved are protected.

3.1. Overall system architecture

The system architecture is shown in Fig. 1. Illustrated as a cloud, the AID service is implemented as a distributed overlay system, consisting of geographically dispersed AID stations that carry out service registration and anti-DoS operations. Unlike many P2P overlay systems where nodes can join from anywhere, we envision the dedicated AID stations are owned, deployed, and managed by a single or a few trusted entities. Because the AID stations are end systems (like CDN servers), they can be deployed anywhere with high-speed access to local ISPs. The throughput of the overlay network can be increased by adding more AID stations, whose

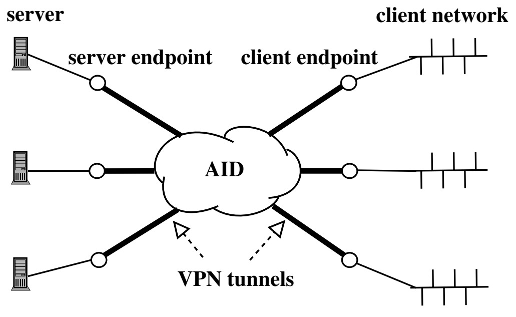


Fig. 1. AID system architecture.

[2] Similar observations were made in a recent paper by Bremler-Barr and Levy [12].

number can be in thousands or tens of thousands. While these stations may be “widely spread across the Internet”, unlike router-based solutions, the AID deployment does not require the modification of Internet router infrastructure (which is a much higher demand, requiring the concerted effort of the ISPs).

3.1.1. Secure VPN overlay

A client (or server) network may register to a nearby AID station. As part of the registration process, a hardware device (or a software module) is installed at the edge of the client network (behind the BGP router) to support secure VPN.[3] AID stations should also support secure VPN. The registration establishes a VPN tunnel between the edge device and an AID station. It is also possible for a registered network to establish more than one tunnel with different AID stations. This is especially true for a server network having multiple access links with different ISPs. Each access link has its own edge device for AID registration.

While the AID stations each connect to a subset of registered networks, they also form an overlay network amongst themselves via secure VPN tunnels, whose purpose is to prevent external injection of false traffic. In the rest of the paper, we often refer to “a client network” simply as “a client”. The implementation of secure VPN tunnels is discussed in Section 4.3.

3.1.2. Operation overview

The client/server traffic is always routed through the Internet unless there is a DoS attack. Once a DoS attack is detected by an overloaded server (Section 4.1), a notification is sent to the AID station that the server is registered to. The AID station triggers the defense by forming a tunnel tree from all clients to the server under attack and allowing the client requests to be redirected via the overlay network to the server (Section 3.4). Efforts are made to minimize the traffic that the overlay network carries (Section 4.2).

3.1.3. Handling attacks

When a registered server is under attack, the goal of AID is to make sure that the clients within its coverage, i.e., the registered clients, can access the server. From the server’s point of view, attack traffic may come from the Internet (if some attackers do not register) or from the overlay network via VPN tunnel (if some attackers register):

- To handle the unregistered attackers, the server’s edge device gives higher priority to the tunnel traffic and blocks excessive Internet traffic.[4] This may seem unfair to the unregistered clients who do not attack. However, even if the unregistered clients were given all the server’s capacity, they would still not be able to access the server because their traffic was mixed with the huge volume of attack traffic from the Internet. It is natural that some clients are not protected during incremental deployment because they are not covered by the defense yet.
- Now how to handle the registered attackers? They seem indistinguishable from the other registered clients. The idea is simple: because they are within the AID’s coverage and their traffic will be carried by the overlay network, we can design distributed algorithms on the overlay network to restrict the amount of traffic each registered client may send. The challenges are how to form an *efficient* overlay topology with a routing structure and how to enforce the *appropriate* data rate for each registered client, which are solved in the rest of the section.

3.2. Topology requirements

1. *Small diameter:* The diameter is defined as the number of hops on the longest routing path of the overlay topology. Each “hop” potentially requires a packet to travel across the Internet. Hence, comparing with the underlying Internet, an m-hop routing path in the overlay network increases the delay as well as the aggregate traffic volume by m times in the worst-case. It is highly desirable that all routing paths are bounded by two or three hops.

[3] Secure VPN is a matured technology that is widely used on the Internet as a replacement of leased lines to securely connect remote networks (or hosts). It has become a feature of today’s firewalls and routers such as the market-dominating Cisco IOS routers [21]. VPN software is also widely available for hosts of most OS platforms. Both AID and [1,3] require the modification of edge devices. The difference is that a self-complete system only modifies the edge devices of the participating networks, whereas a self-incomplete system has to modify the edge devices of all networks [1,3].

[4] In addition, a service contract may be established with the ISP to forward the tunnel traffic ahead of Internet traffic.

2. *Small nodal degree:* Each AID station can take a limited number of VPN tunnels. For example, the high-end Cisco PIX 535 Firewall supports up to 2000 VPN connections [22]. Much of this capacity should be reserved for the registration of clients. The number of tunnels used for connecting peer AID stations should be kept modest.

The first requirement of bounding the path length by two to three excludes most existing overlay topologies. A full mesh topology has the smallest diameter (one) but the largest nodal degree, which violates the second requirement, given that the number of AID stations may grow into thousands. A spanning tree is also problematic because any internal node failure breaks the topology. Tree is a good routing structure, but not a good topology structure, which requires a high degree of redundancy to prevent graph partition. Furthermore, it is hard to design a load-balanced tree with diameter 3 and no apparent root.[5] There are numerous DHT-based P2P networks [17,23–25] that are proposed for distributed file sharing. Their restrictions (and consequently complexities) of maintaining numerical relationships between neighbors' identifiers (after hashing) are unnecessary in our context, where there is no need to look up any specific identifier or route packets based on identifiers (as SOS does). This will become clear in Section 3.4.

3.3. Random overlay network

There is a tradeoff to be made between the two requirements. We propose to form a random overlay network (RONet) among the AID stations. RONet is constructed by every station randomly selecting a number of other stations to be its neighbors. The tunnel between two neighbors is bi-directional. A great advantage of RONet is that it is extremely simple and easy to manage. Yet, it is meshy with rich connectivity and short in diameter. The unpredictable nature of a random topology prevents an attacker from designing an attack strategy based on specific topological characteristics.

Let N be the set of n AID stations. Each $x \in N$ connects with a subset N_x of k randomly-selected stations via VPN tunnels. Consider an arbitrary station $y \in N$. Let P_l be the probability for x to reach y in l or less hops. $P_0 = \frac{1}{n}$, which happens only when $y = x$. $P_1 = \frac{k+1}{n}$, which happens when $y \in N_x + \{x\}$. We derive P_2 below.

x **cannot** reach y in two or less hops if and only if $(N_y + \{y\}) \cap (N_x + \{x\}) = \emptyset$. The number of possible selections for $N_y + \{y\}$ from N is $\binom{n}{k+1}$. The number of possible selections for $N_y + \{y\}$ from $N - N_x - \{x\}$ is $\binom{n-k-1}{k+1}$. The probability of $(N_y + \{y\}) \cap (N_x + \{x\}) = \emptyset$ is:

$$\frac{\binom{n-k-1}{k+1}}{\binom{n}{k+1}} = \frac{(n-k-1)!(n-k-1)!}{n!(n-2k-2)!},$$

Hence,

$$P_2 = 1 - \frac{(n-k-1)!(n-k-1)!}{n!(n-2k-2)!}. \tag{1}$$

Theorem 1. *If $n > 2$ and $k = \sqrt{n}$, then $P_2 > 1 - \frac{1}{e}$, where e is the natural constant.*

The proofs for all theorems and lemmas of this paper can be found in Appendix A. Next we derive P_3. If x cannot reach y in two or less hops, the conditional probability for x **not** reaching y in three hops is $p_3 = \left(\frac{(n-k-2)!(n-k-2)!}{(n-3)!(n-2k-1)!}\right)^k$, which is derived in Appendix B. Hence, we have:

$$\begin{aligned} P_3 &= 1 - (1 - P_2)p_3 \\ &= 1 - \frac{(n-k-1)!(n-k-1)!}{n!(n-2k-2)!}\left(\frac{(n-k-2)!(n-k-2)!}{(n-3)!(n-2k-1)!}\right)^k. \end{aligned} \tag{2}$$

Consider an example where $k = 100$, i.e., each station has 100 neighbors. When n is as large as 10,000, $P_2 = 0.63$ and $P_3 = 1 - 1.33 \times 10^{-44}$, which means with almost perfect certainty that the distance between any two nodes does not exceed three hops.

Given any two nodes, let ϕ be the target probability for them to be three or less hops away. Based on (2), there exists a functional relationship between n and the lower bound of k:

$$\phi \leqslant 1 - \frac{(n-k-1)!(n-k-1)!}{n!(n-2k-2)!}\left(\frac{(n-k-2)!(n-k-2)!}{(n-3)!(n-2k-1)!}\right)^k. \tag{3}$$

From the above relation, we can numerically calculate k's lower bound for any network size. For

[5] Ideally all AID stations should be treated equally.

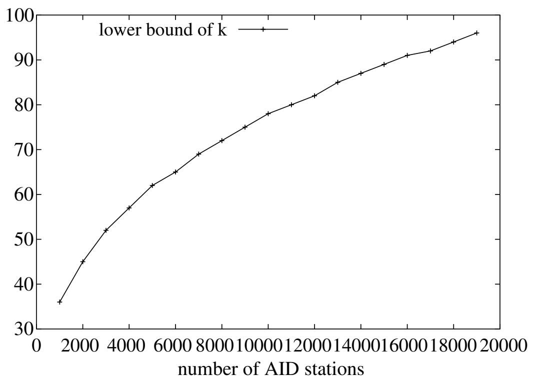


Fig. 2. Minimum number of neighbors per station (k), such that the probability of any two stations to be three or less hops away is bounded by $\phi = 1 - 10^{-20}$.

example, when $\phi = 1 - 10^{-20}$, the lower bound of k with respect to n is shown in Fig. 2.

Even when ϕ is extremely close to one, it remains possible for two nodes to be more than three hops apart. Although such a rare event is practically insignificant, if one still wants to eliminate it, a backup mechanism may be used, with each station reporting its neighbor set to a management server.[6] The server will detect any long path and adjust the topology, e.g., by instructing the two stations at the path ends to add a link between them. We want to stress that, working on the side line as a topology checker, this optional server is not a critical element of RONet. It is *not needed* in any operation performed on RONet to be described. Even its interaction with RONet is extremely rare. Therefore, it neither constitutes a single point of failure, nor degrade RONet's performance and robustness in any manner.[7]

3.4. Constructing tunnel tree from clients to server under attack

Our next task is to establish a routing structure on RONet. Let A_s be the AID station that a server s registers to. There is a VPN tunnel between A_s and the edge device of s. When s is under attack, A_s is signaled with the type of attack traffic, denoted as π. An example is "All SYN packets to s" for a SYN-flooding attack. Attack detection and type determination are discussed in Section 4.1.

Each AID station x periodically queries its neighbors for detected attacks. A_s's neighbors learn the attack from A_s. Their neighbors learn the attack from them, and so on. When a station x learns the attack from another station y, it informs the adjacent tunnels to admit client traffic of type π with destination s, and it forwards the traffic to y, which is one hop closer to the server. All AID stations will eventually form a tunnel tree. The specified traffic π will then flow up the tunnel tree from the registered clients to the server.[8] The query period represents a tradeoff between control overhead and responsiveness. If the topology diameter is three, then the tunnel tree is constructed after just three query periods and the tree depth is three. In order to construct the tunnel tree more quickly, A_s may directly broadcast π to all AID stations. This design is volunerable when the attackers initiate many short-lived DoS attacks and cause the overlay to perform excessive broadcasts. In comparison, the scheme based on periodic queries has constant control overhead and reasonable response time.

The above routing protocol is resilient to node failure. When an internal node of the tunnel tree fails, its child nodes are disconnected from the tree. As long as the node failure does not partition the overlay network, the next queries will rejoin all nodes to the tree. Erdos and Renyi [26] showed that, for a random graph to be connected with a high probability p, the expected node degree d should be:

$$d \geqslant \frac{n-1}{n}(\ln n - \ln(-\ln p)). \tag{4}$$

Suppose the overlay topology is designed to withstand the failure of a percentage α of all AID stations. After α percentage of all stations fails (for example, due to the outage of an ISP), the expected number of neighbors for each remaining node is $(1-\alpha)k$. From (4), we derive a requirement for k:

[6] As a bonus, it presents a global picture of the overlay topology to the AID service provider. This may be useful in assisting other management functions.

[7] Just like RONet, most highly-reliable distributed systems (such as Chord [17] and other P2P systems) have a non-zero probability of failure (e.g., network partition due to massive node departure). It is normally ignored in the system design; to guard against it, a centralized backup mechanism is likely to be needed in one way or another.

[8] Traffic other than π flows through the Internet as normal (Section 4.2). The purpose is to minimize the traffic carried by the overlay.

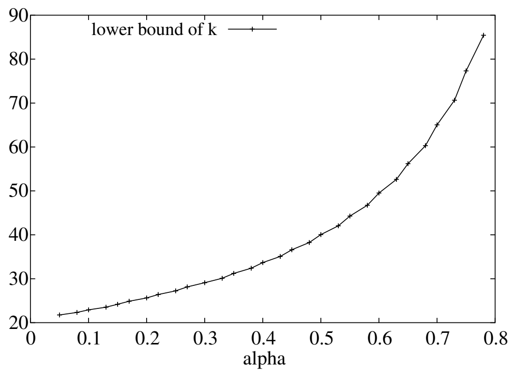


Fig. 3. Minimum number of neighbors per station (k), such that the system will remain functional after α percentage of all AID stations fails.

$$
\begin{aligned}
&(1-\alpha)k \geqslant \frac{(1-\alpha)n-1}{(1-\alpha)n}(\ln(1-\alpha)n - \ln(-\ln p)),\\
&k \geqslant \frac{(1-\alpha)n-1}{(1-\alpha)^2 n}(\ln(1-\alpha)n - \ln(-\ln p)).
\end{aligned} \tag{5}
$$

For example, if $p = 0.99999$ and $n = 10{,}000$, the minimum value of k with respect to α is shown in Fig. 3. If $k = 85$, then even after 78% of all AID stations fails, the system is still functional.

DoS attacks may congest Internet links or overload specific AID stations. When an AID station detects performance degradation of a neighbor station, it may simply treat the neighbor as having failed. The routing protocol will automatically reroute traffic to a different neighbor. As analyzed above, even when the overlay paths to a large number of AID stations are congested, by temporarily disabling those links, the system automatically adapts to less loaded paths. Additional robustness mechanisms are discussed in Section 4.4, including redundant backup tunnels and parallel tunnel trees, where the server registers to multiple AID stations to improve resiliency.

It should be emphasized that, according to the routing protocol, a separate tunnel tree will be constructed for each server under attack. The traffic in different tunnel trees is processed independently based on destination addresses.

3.5. Distributed virtual-clock packet scheduling

Given that the registered attackers have access to the overlay network, the next problem is how to filter the attack traffic in order to protect the server as well as the overlay infrastructure. We use the concept of virtual-clock packet scheduling, which was originally designed for a router to guarantee certain shares of its bandwidth to the passing flows [27]. This capability matches our anti-DoS goal, which is to prevent the aggressive malicious clients from consuming most of the service's capacity and starving the legitimate clients. The idea is that, if we can guarantee each client a fair share of the server's capacity, all clients will be immune from DoS attacks because their access to the server is guaranteed. Below we design a distributed virtual-clock scheduling scheme, which allows the AID system to assign fair shares of the server's capacity to the registered clients. This scheme is designed to operate on the "distributed" AID system, instead of on a single router as the original virtual-clock algorithm does. Moreover, with an adaptive mechanism described in the next subsection, the scheme assigns a "fair" share, instead of a "fixed" share, of the server's capacity to each client. Note that a fair share is a variable quantity inversely proportional to the number of active clients, which changes over time.

The data traffic is forwarded in the tunnel tree upstream from the clients to the server's edge A_s and then to the server s. Each AID station receives data packets from downstream tunnels and multiplexes them into the upstream tunnel. The control traffic travels in the opposite direction. A_s periodically broadcasts a CONTROL(T) message to all AID stations in the tree, where T is a control parameter called the *waiting interval*. The goal is to introduce tunable waiting time between consecutive data packets from a client. For now let us assume A_s has some way to determine the best value for T.

Assume the system clocks of all AID stations are loosely synchronized. Each AID station x maintains a virtual-clock VC_u (initialized to be the local system clock) for every tunnel u connecting to a client network. VC_u advances by T for every byte transmitted.

1. When x receives a data packet from a client network via tunnel u, it updates VC_u as follows:
$$\mathrm{VC}_u = \max\{\mathrm{VC}_u, \mathrm{current\_time}\} + T \times L, \tag{6}$$
where L is the length of the packet. The packet is then labeled with a timestamp equal to VC_u.
2. When x receives a data packet from a neighboring station, the packet already has a timestamp assigned by the first station receiving it.

An AID station stores all received data packets in a shared heap array [28] in ascending order based on their timestamps. A packet is scheduled to enter the upstream tunnel only when the current system clock is larger than the timestamp of the packet plus the maximum clock skew on any branch of the tunnel tree. Because the timestamps of a virtual-clock advance at the speed of T per byte, the maximum sustainable rate that the overlay network will deliver for a client is $\frac{1}{T}$ bytes per unit of time.

The timestamp-based scheduling performed by all AID stations collectively ensures that each client receives a fair share of the server's capacity. By tuning the value of T, we can also make sure that the unused bandwidth left by some clients are picked up by the other clients.

If a registered client hosts an attacker, the high volume of attack traffic will quickly advance its virtual-clock, which results in large timestamps for its packets. The packets will be pushed to the end of the heap array and dropped if the maximum size of the array is reached. Consequently, most excessive attack packets will be blocked at the AID station connecting to the offending client. They do not have a chance to traverse the overlay network and further consume resources.

3.6. Determining T

How to determine the best value for T? Suppose we want the total rate of tunnel traffic to be roughly bounded by a target rate c_s. A_s starts by broadcasting CONTROL(T) in the tunnel tree with $T = \frac{1}{c_s}$. It then tunes the value of T in the subsequent broadcasts. There are two phases. The first is an exponential phase, where A_s doubles T at each subsequent broadcast. Doubling T makes the virtual-clocks run twice as fast, which effectively reduces the maximum rate per client by half. Once the arrival rate of tunnel traffic at A_s is below c_s, a linear phase starts for fine tuning. Suppose T is changed from I to $2I$ by the last update of the exponential phase. $\varepsilon \in (0,1)$ is a system parameter. The linear phase will decrease T by εI periodically until the arrival rate is above c_s, at which moment we call the system converges. It is obvious that $T \geqslant I$.

Let T_1 and T_2 be the waiting intervals before and after the last update of the linear phase, respectively. $T_2 = T_1 - \varepsilon I \geqslant (1-\varepsilon)T_1$. Hence, $\frac{1}{T_2} \leqslant \frac{1}{1-\varepsilon}\frac{1}{T_1}$, where $\frac{1}{T_1}$ and $\frac{1}{T_2}$ are the traffic rates allowed from each registered client before and after the last update, respectively. Therefore, the total arrival rate at A_s is improved at most by a factor of $\frac{1}{1-\varepsilon}$ at the last update. Because the arrival rate is below c_s before this update, it must be below $\frac{1}{1-\varepsilon}c_s$ after the update. Hence, the arrival rate of tunnel traffic at A_s is in the range of $\left[c_s, \frac{1}{1-\varepsilon}c_s\right]$ when the system converges. After converging, the updates of T may be restarted if the arrival rate goes beyond the range.

It is required that the server's capacity is at least $\frac{1}{1-\varepsilon}c_s$. The edge router must ensure that the tunnel traffic is fully admitted up to a rate of $\frac{1}{1-\varepsilon}c_s$ and the Internet traffic is admitted to fill the remaining server's capacity. For simplicity, we have made all virtual-clocks run at the same speed. A weight can be introduced in (6) to give some clients more bandwidth than others.

Persistent overloading of a server by aggressive behaviors of some normal clients will also trigger the same defense process described above, which will drop the excessive traffic from aggressive clients. This is a QoS property that protects clients with normal usage from being "squeezed" by aggressive users.

4. Discussions

4.1. Attack detection

The DoS attack may be detected by an overloaded server which starts to drop packets or by the edge device based on a preconfigured policy, e.g., the rate of connections exceeds 10,000 s^{-1}or the traffic volume exceeds 10 Mb/s for certain period of time. The type π of client traffic to be regulated can be as straightforward as "all IP packets to the server (or the subnet) under attack". More specific types can be identified for specific attacks. For example, the traffic type for a SYN attack is "all SYN packets to the server" and the type for a smurf attack is "all ICMP echo-reply packets to the server".

4.2. Minimizing traffic on overlay

In order to prevent the overlay network from being overloaded, we must minimize the traffic it carries. AID kicks in when there is an attack. It is likely that only a small portion of servers will be under DoS attack at any single time. For the vast majority of servers that are not under attack, AID is not on their communication paths. Furthermore, AID only carries some traffic (type π) from the registered clients to the attacked servers. It *does not* *carry* the reverse traffic from the servers to the clients, which accounts for the majority of the total traffic volume for many applications such as web browsing. All traffic that is not carried by the overlay network will be delivered via the Internet as usual.

The registered attackers may generate a huge number of attack packets. The overlay network will not carry all of them; the distributed virtual-clock scheduling algorithm (Section 3.5) makes sure that the total traffic carried by the overlay network from all registered clients to a server does not exceed the server's capacity. For example, suppose a web server can process 10^4 new connections per second. When the server is under a SYN-flooding attack, the AID system will be activated. The SYN packets from the registered clients to the server will now go through the overlay network, while all other packets still go through the Internet. Moreover, only up to 10^4 SYN packets per second will be delivered through the numerous AID stations of the overlay network. The excessive SYN packets will be automatically dropped at the edge of the overlay. Consider bandwidth-flooding attacks. For most common services on the Internet, a server sends much more than it receives. For example, if the outbound capacity of a web server is one Gigabit per second, then an inbound request rate of 10 Mb/s may already cause a DoS attack. In this case, the load on our overlay system will be no more than 10 Mb/s during the attack. In yet another example, suppose AID protects 100,000 registered servers and the clients' request traffic is one hundredth of the servers' reply traffic. If there are 1000 servers under attack at a given time, then the AID overlay carries a small fraction of 1/10,000 of the total traffic between the clients and the servers.

It is possible to design an overlay network to carry all client traffic even without attacks, as SOS [16] and Mayday [18] do. This would however require a larger transmission capacity and thus more AID stations.

4.3. Implementation

The edge device of a registered client may need to pass traffic of type π into a secure VPN tunnel and the rest traffic to the Internet. We adopt the standard implementation used by a firewall [22] or a router with IPSec capability [21]. The edge device is configured with regular ACL (access control list) and crypto ACL. A received packet is first matched against the regular ACL, which decides whether the packet should be dropped or forwarded. If the packet should be forwarded, it is matched against the crypto ACL, which decides whether the packet should be sent in clear or in a tunnel. Under a DoS attack, as the tunnel tree is formed, the edge device is informed to add π to the crypto ACL and consequently the traffic is redirected into the VPN tunnel, instead of going directly to the Internet.

The purpose of our secure VPN is to prevent injection of false packets. There is no privacy requirement. We can conveniently rely on the IPSec/AH standard for trust management and communication integrity, which only requires a hash function to generate/verify MAC (message authentication code) for tunnel packets. The registration process requires the domain administrative privilege of a client/server network. It should not accept registration requests from a host or repeated requests from an unverified domain, which prevents registration-based attacks.

It is practically feasible to gather the necessary resources for building such an overlay network. Today's content distribution networks (CDN) consist of hundreds or thousands of machines with the capacity to deliver the multimedia content from a large number of content providers. In comparison, the demand for the overlay network in AID is much smaller because it supports only those servers *currently under attack* and involves only the traffic *from clients to the servers*. Moreover, it has a mechanism to regulate the traffic volume it carries.

4.4. Robustness of the AID system

It is extremely hard to overwhelm the entire overlay network due to the sheer number of AID stations and the rich connectivity between them. The damage that the attackers can inflict on a particular AID station is also limited. First, unregistered attackers cannot place their traffic in the overlay network. Second, even though the registered attackers are able to place their traffic in the tunnels, the distributed virtual-clock packet scheduling algorithm ensures that they can only impose limited traffic on the overlay network.

Having these protection measures, if an AID station is still overloaded, it simply reassigns its client/server networks to other randomly-selected AID stations. To avoid service disruption, a client/server network may pre-establish some backup tunnels with different AID stations. When the AID station of one tunnel is overloaded, the traffic will switch to a different tunnel. Additional tunnels are dynamically established under the extreme case that all AID stations for the backup tunnels are overloaded. For a client, typically only one tunnel will be active at any given time; more than one active tunnels give the client a larger share of the server's capacity. For a server, one or more tunnels may be active depending on the service contract. Multiple server tunnels result in multiple tunnel trees, whose combined traffic is bounded by the server's capacity. Multiple trees serve not only the resilience purpose but also the load-balancing purpose because they spread the traffic more evenly across the overlay network.

5. Analysis

We study two important performance metrics: *convergence time* and *misblocking percentage*. The former answers the questions of whether the system will stabilize and how long it takes to block out the attack traffic. The latter answers the questions of how much legitimate traffic from registered clients is mistakenly blocked and whether the system can control the impact of the collateral damage.

5.1. Convergence time

The following theorem shows that the worst-case convergence time is logarithmic in the number of registered client networks, denoted as m. It does not consider the three query periods for constructing a tunnel tree, which is a constant time. The notations, including those defined later, can be found in Table 1 for quick reference.

Table 1
Notations

T	Speed of virtual-clock, a parameter of distributed virtual-clock algorithm
ε	Convergence threshold, a parameter of distributed virtual-clock algorithm
c_s	Target rate of the total tunnel traffic delivered via AID to s
m	Number of registered clients
a	Number of registered clients that attack
r	Maximum data rate allowed for a client after AID converges
β	Ratio of legitimate-traffic rate to c_s
$f(x)$	Probability density function for legitimate-traffic rate distribution over all registered clients
Δ	Misblocking percentage

Theorem 2. *Given any set of steady traffic from all clients, it takes no more than* $\left(\log_2 m + \frac{1}{\varepsilon}\right)$ *updates of T before the AID system converges.*

Example 3. Suppose $m = 100{,}000$ and $\varepsilon = 0.1$. It takes at most 27 updates of T for the AID system to converge. If the update period is 10 s, the convergence time will be at most 4.5 min.

5.2. Misblocking percentage

In the following, we will establish an upper bound for misblocking percentage. Suppose AID converges at a certain value of T, which allows each registered client to send data at a rate up to $r = \frac{1}{T}$. Let β be the ratio of the total legitimate-traffic rate (from all registered clients) to the target rate c_s for tunnel traffic. β is expected to be within $(0,1)$. If not, c_s should be increased with more server's capacity reserved for tunnel traffic. The total legitimate-traffic rate is thus βc_s.

Consider the probability distribution of the legitimate-traffic rates generated by individual registered clients. Let $f(x)$ be the probability density function. The average rate of legitimate-traffic generated by a registered client is $\beta c_s/m$. Therefore,

$$\int_0^\infty x f(x)\,\mathrm{d}x = \frac{\beta c_s}{m}. \tag{7}$$

Let a be the number of registered client networks that attack. These client networks are called *attack clients*, and the rest $(m-a)$ client networks are called *normal clients*. Suppose the attack clients are randomly distributed among all clients. The probability density function of the traffic rates from the normal clients remains to be $f(x)$. We have the following lemmas.

Lemma 1. *The misblocking percentage for all legitimate-traffic is*:

$$\Delta = \frac{m}{\beta c_s}\int_r^\infty (x-r)\cdot f(x)\,\mathrm{d}x. \tag{8}$$

Lemma 2. *The rate at which each registered client is allowed to send is*:

$$r \geqslant \frac{\beta c_s}{m} + \frac{(1-\beta)c_s}{a}.$$

In the following, we analyze the misblocking percentage (Δ) based on three specific distribution functions of $f(x)$.

5.2.1. Exponential distribution

$f(x)=\lambda e^{-\lambda x}$. It models the case where most clients access the server at low-rates but a small number of clients access the server at high-rates. The mean is $\frac{1}{\lambda}$, which is equal to $\frac{\beta c_s}{m}$ by (7). Hence, $\lambda=\frac{m}{\beta c_s}$. Applying $f(x)=\lambda e^{-\lambda x}$ to (8), we have:

$$\Delta=\frac{m}{\beta c_s}\int_r^\infty (x-r)\lambda e^{-\lambda x}\,dx=\frac{m}{\beta c_s\lambda}e^{-r\lambda}=e^{-\frac{rm}{\beta c_s}}.$$

By Lemma 2, we have:

$$\Delta\leqslant e^{-1-\frac{m}{a}\left(\frac{1}{\beta}-1\right)}. \tag{9}$$

Interestingly, the upper bound of Δ depends on the total rate of legitimate-traffic, characterized by β, but does not depend on the total rate of attack traffic.[9] Fig. 4 shows the upper bound of Δ with respect to β and a. Given a target value of Δ, the number of attack clients that the system can tolerate is:

$$a\leqslant m\left(\frac{1}{\beta}-1\right)\Big/(-\ln\Delta-1). \tag{10}$$

Consider $\Delta=0.0001$ and $m=100{,}000$. If $\beta=0.8$, then the system can tolerate 3044 attack clients. If $\beta=0.2$, then the system can tolerate 48,719 attack clients; even when 48.7% of all clients attack, the remaining clients can still access the server with a misblocking percentage of just 0.0001.

5.2.2. Normal distribution

$f(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(x-\mu)^2/\sigma^2}$. It models the case where the registered clients are similar but with certain variations. The mean is μ, which is equal to $\beta c_s/m$ by (7). From Lemma 1,

$$\begin{aligned}\Delta&=\frac{1}{\mu}\int_r^\infty (x-r)\cdot\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(x-\mu)^2/\sigma^2}\,dx\\&=\frac{1}{\mu}\int_{\frac{r-\mu}{\sigma}}^\infty(\sigma y-r+\mu)\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}y^2}\,dy\\&=\frac{\sigma}{\mu\sqrt{2\pi}}e^{-\frac{1}{2}(r-\mu)^2/\sigma^2}-\frac{r-\mu}{\mu}\left(1-\Phi\left(\frac{r-\mu}{\sigma}\right)\right),\end{aligned}$$

where $\Phi(x)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^x e^{-\frac{1}{2}t^2}\,dt$, whose value can be found in almost any statistics book in tabular format. To simplify the analysis, we assume $\sigma=\mu/2$:

[9] The upper bound depends on the number of attackers (a) but not on the total rate of attack traffic, which means that the attackers will not be better off by increasing their individual attacking rates.

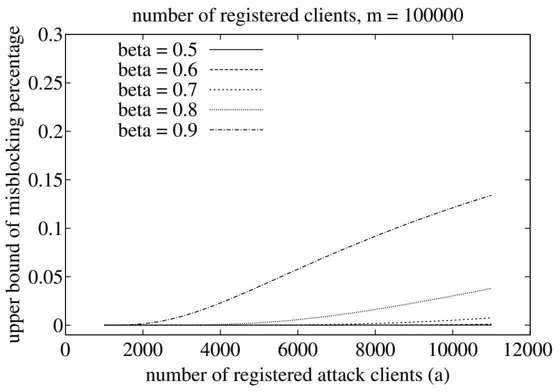


Fig. 4. Upper bound of misblocking percentage (Δ) with respect to number of attackers (a) as well as the total rate of legitimate traffic (characterized by β).

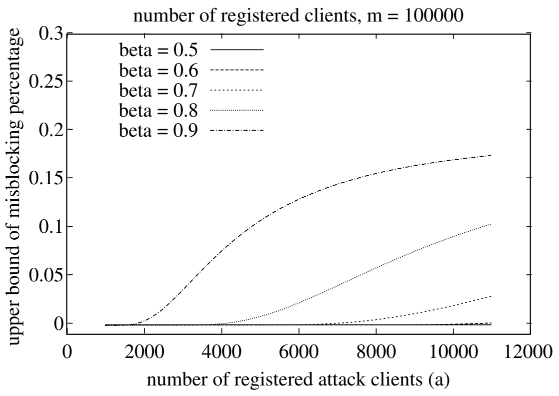


Fig. 5. Upper bound on Δ with respect to different β and a.

$$\begin{aligned}\Delta&=\frac{1}{2\sqrt{2\pi}}e^{-\frac{1}{8}(r-\mu)^2/\mu^2}-\frac{r-\mu}{\mu}\left(1-\Phi\left(\frac{2(r-\mu)}{\mu}\right)\right)\\&\leqslant\frac{1}{2\sqrt{2\pi}}e^{-\frac{1}{8}(r-\mu)^2/\mu^2}.\end{aligned}$$

Applying Lemma 2, we have:

$$\Delta\leqslant\frac{1}{2\sqrt{2\pi}}e^{-\frac{1}{8}\left[\left(\frac{1}{\beta}-1\right)\frac{m}{a}\right]^2}.$$

Fig. 5 shows the upper bound of Δ with respect to β and a. The upper bounds shown in Figs. 4 and 5 are not tight due to the relaxations we made. Our simulation shows better results.

5.2.3. Singular distribution

The rates of legitimate-traffic from all clients are the same, $\beta c_s/m$, which is a special case of the normal distribution with $\sigma\to 0$. By Lemma 2, we know $r\geqslant\frac{\beta c_s}{m}+\frac{(1-\beta)c_s}{a}$. Because r is no less than the data

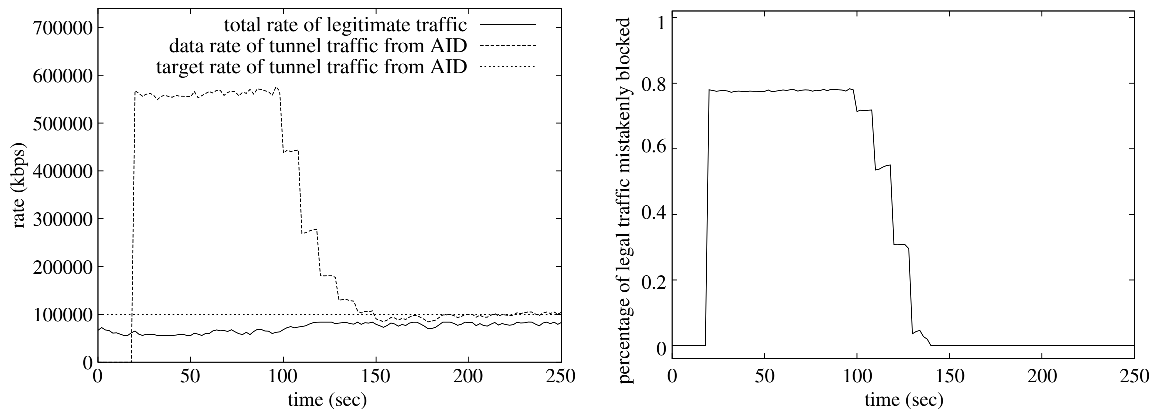


Fig. 6. Left-hand plot: AID brings the tunnel traffic rate around the target rate; right-hand plot: misblocking percentage drops to almost zero after AID converges.

rate of any normal client, no legitimate data from normal clients will be lost.

6. Simulation results

In the previous section, we established the upper bounds for the worst-case convergence time and the worst-case misblocking percentage. In this section, we use simulations to evaluate AID in terms of average convergence time and average misblocking percentage. The default simulation parameters are given as follows. There are 1000 AID stations, each having 100 registered client networks. Consider a registered server under a DDoS attack. The target rate of tunnel traffic from AID is 100,000 kbps (kilobits per second), which the edge device (A_s) will forward to the server. The initial legitimate-traffic rates from all registered clients follow an exponential distribution with an average rate of 7 kbps. The total legitimate-traffic rate from the clients is thus 70,000 kbps. $\beta = 0.7$. The simulation is performed at ticks of two seconds each. The arrival rate of legitimate-traffic from a client fluctuates up to $\pm 50\%$ over each tick. Assume 1000 randomly-selected client networks have compromised machines.[10] The attack rates from those client networks follow an exponential distribution with an average attack rate of 500 kbps.[11] The waiting interval T is updated every 10 s. $\varepsilon = 5\%$. The default parameters are always assumed unless the figures indicate otherwise.

We use simulations to show that AID is practical, but we do not compare the performance of AID with that of the existing overlay defense systems, SOS, Mayday, and WebSOS. The reason is that their difference is not quantitatively in performance. The means of defense by SOS/Mayday are client authentication and servlet secrecy, which makes them unsuitable for public servers. WebSOS is only for web service and requires human interaction. The means of defense by AID are distributed filtering and topology randomization, and it is designed for all public servers with or without human interaction, which remove the limitations of SOS/Mayday/WebSOS.

6.1. *Effectiveness of AID*

Fig. 6 shows how AID reacts to a DDoS attack. The three curves of the left-hand plot are (1) the total rate of legitimate traffic sent by the registered clients, which fluctuates over time, (2) the total data rate received from AID by the server's edge device, which jumps up after the attack begins and is reduced over time by the distributed virtual-clock algorithm, and (3) the target rate of tunnel traffic from AID. Before the attack, all client traffic goes through the Internet and thus the server receives no traffic from AID. After the attack starts at time = 20, legitimate/attack traffic from registered clients is routed through AID. Consequently, the traffic received by the server's edge device from AID shoots up. When the edge device drops the portion above the target rate, it drops legitimate-traffic proportionally,

[10] The total number of compromised machines may exceed 1000 because each attack client may have multiple compromised machines.

[11] Some clients may have more compromised machines or higher-speed Internet connections than others.

causing a large misblocking percentage. The distributed virtual-clock algorithm iteratively refines the waiting interval. At time = 100, T is large enough to cause some attack traffic to be blocked. The exponential phase terminates at time = 150, and then the linear phase improves the data rate gradually above c_s. The right-hand plot shows the fraction of legitimate-traffic mistakenly blocked by AID. When the attack occurs, a significant portion of legitimate-traffic (almost 80%) is blocked. The misblocking percentage is reduced over time and reaches almost zero after time = 140.

6.2. Against a large number of attackers

Fig. 7 demonstrates that AID is able to handle a large number of attack clients. The left-hand plot shows that the convergence time increases at a modest (sublinear) pace with the number of attack clients. The time increase for the exponential phase is steady, while the time for the linear phase changes up and down significantly. That is because the linear phase is greatly affected by the value of the waiting interval after the last update of the exponential phase, which is not monotonically related to the number of attack clients. The right-hand plot shows the misblocking percentage, which increases with the number of attack clients. When there are 2000 attack clients, virtually no legitimate-traffic is blocked. Remarkably, even when the number of client networks hosting attackers reaches 10,000, just 1.4% of legitimate packets are lost.

6.3. Impact of β

Fig. 8 shows how the total rate of legitimate-traffic, characterized by β, will affect the misblocking percentage. As expected, a larger value of β causes more legitimate packets to be dropped. On the other hand, when $\beta = 0.7$, there is no observable misblocking when the number of attack clients is 4000 or less. When $\beta = 0.6$, there is no observable misblocking when the number of attack clients is 7000 or less. When $\beta = 0.5$, there is no observable misblocking when the number of attack clients is 10,000 or less. Another interpretation of data is that, if the number of attack clients is 4000 (7000 or 10,000), the legitimate clients can send their traffic at least at 70% (60% or 50%) of the maximum possible speed.

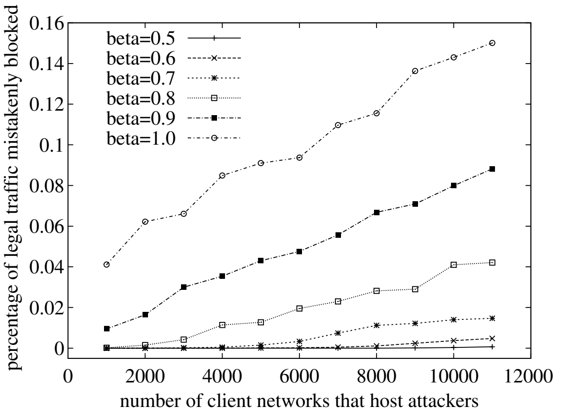
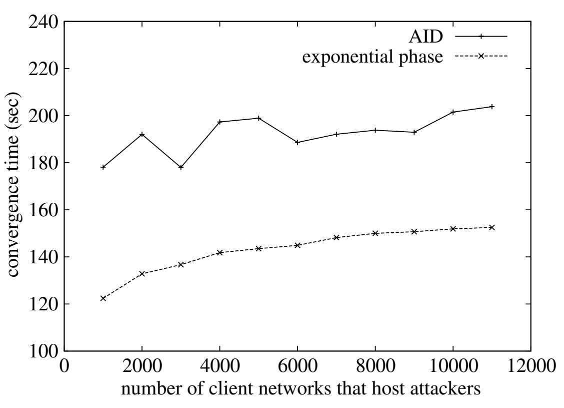


Fig. 8. Impact of the total rate of legitimate-traffic (characterized by β) on the misblocking percentage.

7. Conclusion

The traditional anti-DoS strategy is mainly along the line of modifying the OS, protocol stacks, or other software of end hosts or routers. The resulting



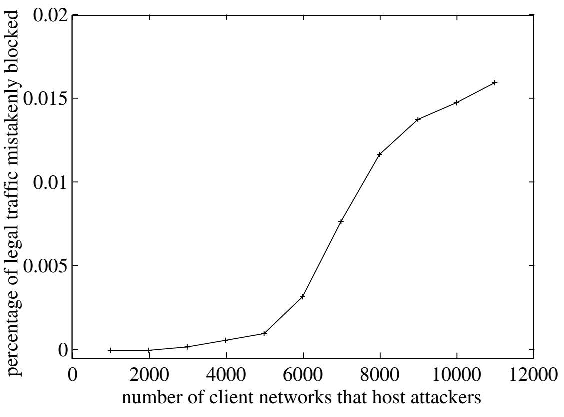


Fig. 7. Scalability with increasing number of registered clients hosting attackers.

defense systems are often not self-complete and require widespread deployment. In this paper, we take a new defense strategy of providing a self-complete global anti-DoS service (AID) for general-purpose public servers. Differing from SOS and Mayday, AID is open to all clients instead of a group of authorized ones, and it handles both external and internal attacks. The primary defensive means are no longer authentication but distributed filtering and topology randomization. Differing from WebSOS, AID is designed for applications beyond web service and it does not require human interaction.

Appendix A. Proofs

Proof of Theorem 1. By Eq. (1), we have:

$$\begin{aligned}
P_2 &= 1 - \frac{(n-k-1)!(n-k-1)!}{n!(n-2k-2)!} \\
&= 1 - \frac{n-k-1}{n} \times \frac{n-k-2}{n-1} \times \cdots \times \frac{n-2k-1}{n-k} \\
&> 1 - \left(\frac{n-k}{n}\right)^k = 1 - \left(1 - \frac{k}{n}\right)^k \\
&= 1 - \left(1 - \frac{1}{\sqrt{n}}\right)^{\sqrt{n}} \quad \text{by the theorem assumption} \\
&\quad k = \sqrt{n},
\end{aligned}$$

$\left(1 - \frac{1}{\sqrt{n}}\right)^{\sqrt{n}}$ is a monotonically-increasing function and,

$$\lim_{n\to\infty}\left(1 - \frac{1}{\sqrt{n}}\right)^{\sqrt{n}} = \lim_{n\to\infty}\left(1 - \frac{1}{n}\right)^{n} = \frac{1}{e}.$$

Hence, we must have $P_2 > 1 - \frac{1}{e}$. □

Proof of Theorem 2. The waiting interval T is initially $\frac{1}{c_s}$, and each update in the exponential phase doubles T. The sustainable rate that a client can send via the AID system to a server s is bounded by $\frac{1}{T}$. When $T \geqslant \frac{m}{c_s}$, each registered client can send at a rate no more than $\frac{c_s}{m}$, and the total rate of all registered clients is no more than c_s, which terminates the exponential phase. It takes $\log_2 m$ updates to increase T from $\frac{1}{c_s}$ to $\frac{m}{c_s}$. Suppose the last update in the exponential phase changes T from I to $2I$. Each update in the linear phase decreases T by εI. The total decrease will not exceed I; otherwise, the last update of the exponential phase would have been unnecessary. Hence the number of updates in the linear phase is bounded by $\frac{1}{\varepsilon}$. Therefore, the total number of updates is bounded by $\left(\log_2 m + \frac{1}{\varepsilon}\right)$. □

Proof of Lemma 1. The combined data rate generated by all normal clients is $\int_0^\infty x \cdot (m-a)f(x)\,\mathrm{d}x$, among which the portion that is admitted by AID after convergence is $\int_0^\infty \min\{x,r\} \cdot (m-a)f(x)\,\mathrm{d}x$ and the portion that is blocked by AID is $\int_r^\infty (x-r) \cdot (m-a)f(x)\,\mathrm{d}x$.

For traffic from normal clients, the percentage that is mistakenly blocked by AID is:

$$\begin{aligned}
\Delta &= \frac{\int_r^\infty (x-r) \cdot (m-a)f(x)\,\mathrm{d}x}{\int_0^\infty x \cdot (m-a)f(x)\,\mathrm{d}x} \\
&= \frac{m}{\beta c_s}\int_r^\infty (x-r) \cdot f(x)\,\mathrm{d}x. \qquad \square
\end{aligned}$$

Proof of Lemma 2. The combined data rate generated by all normal clients is $\int_0^\infty x \cdot (m-a)f(x)\,\mathrm{d}x$. For each attack client, the data rate admitted by AID after convergence is bounded by r. Hence, the combined attack rate admitted by AID is bounded ar. When the linear phase terminates, the total arrival rate at A_s is expected in the range of $\left[c_s, \frac{1}{1-\varepsilon}c_s\right]$. We must have:

$$\int_0^\infty \min\{x,r\} \cdot (m-a)f(x)\,\mathrm{d}x + ar \geqslant c_s. \tag{11}$$

Simplifying (11), we can derive a bound for r:

$$\begin{aligned}
&\int_0^\infty x \cdot (m-a)f(x)\,\mathrm{d}x + ar \geqslant c_s, \\
&\frac{\beta c_s}{m}(m-a) + ar \geqslant c_s \quad \text{by (7)}, \\
&r \geqslant \frac{\beta c_s}{m} + \frac{(1-\beta)c_s}{a}. \qquad \square
\end{aligned}$$

Appendix B. Deriving p_3

Consider two arbitrary stations x and y. If x cannot reach y in two or less hops, the conditional probability for x not reaching y in three hops is derived as follows: Consider arbitrary station $z \in N_x$. x does not have a three hop path via z to y if and only if $N_z \cap N_y = \emptyset$. Other than x, z has $k-1$ neighbors, which can be selected from $N - \{x,z,y\}$. The number of possible selections is:

$$\binom{n-3}{k-1} = \frac{(n-3)!}{(k-1)!(n-k-2)!}.$$

Other than y, there are $k-1$ nodes in N_y. The number of possible selections of N_z that do not share a station with N_y is:

$$\binom{n-3-(k-1)}{k-1} = \binom{n-k-2}{k-1} = \frac{(n-k-2)!}{(k-1)!(n-2k-1)!}.$$

The conditional probability for x not reaching y in three hops via z is:

$$\frac{\binom{n-k-2}{k-1}}{\binom{n-3}{k-1}} = \frac{(n-k-2)!(n-k-2)!}{(n-3)!(n-2k-1)!}.$$

There are k nodes in N_x. Hence, The conditional probability for x not reaching y in three hops via $\forall z \in N_x$ is:

$$p_3 = \left(\frac{(n-k-2)!(n-k-2)!}{(n-3)!(n-2k-1)!}\right)^k.$$

References

[1] P. Ferguson, D. Senie, Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing, in: IETF, RFC 2267, Janurary 1998.

[2] K. Park, H. Lee, On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets, in: Proceedings of the ACM SIGCOMM'01, August 2001.

[3] H. Wang, D. Zhang, K.G. Shin, SYN-dog: sniffing SYN flooding sources, in: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), July 2002.

[4] R. Stone, CenterTrack: an IP overlay network for tracking DoS floods, in: Proceedings of the Ninth USENIX Security Symposium, August 2000.

[5] S. Savage, D. Wetherall, A. Karlin, T. Anderson, Practical network support for IP traceback, in: Proceedings of the ACM SIGCOMM'00, August 2000.

[6] A.C. Snoren, C. Partridge, L.A. Sanchez, C.E. Jones, F. Tchakountio, S.T. Kent, W.T. Strayer, Hash-based IP traceback, in: Proceedings of the ACM SIGCOMM'01, August 2001.

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

[8] J. Li, M. Sung, J. Xu, L. Li, Large-scale IP traceback in high-speed Internet: practical techniques and theoretical foundation, in: IEEE Symposium on Security and Privacy, May 2004.

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

[10] P. Mahajan, S.M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, S. Shenker, Controlling high bandwidth aggregates in the network, Computer Communications Review 32 (3) (2002) 62–73.

[11] T. Anderson, T. Roscoe, D. Wetherall, Preventing internet denial-of-service with capabilities, in: Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II), November 2003.

[12] A. Bremler-Barr, H. Levy, Spoofing prevention method, in: Proceedings of the INFOCOM'05, March 2005.

[13] D.J. Bernstein, SYN cookies, 1997, http://cr.yp.to/syncookies.html.

[14] J. Lemon, Resisting SYN flood DoS attacks with a SYN cache, in: Proceedings of the USENIX BSDCON'2002, February 2002.

[15] A. Juel, J. Brainard, Client puzzles: a cryptographic countermeasure against connection depletion attacks, in: Proceedings of the Network and Distributed System Security Symposium (NDSS'99), February 1999.

[16] A.D. Keromytis, V. Misra, D. Rubenstein, SOS: Secure Overlay Services, in: Proceedings of the ACM SIGCOMM'02, August 2002.

[17] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for Internet applications, in: ACM SIGCOMM'01, 2001.

[18] D.G. Andersen, Mayday: distributed filtering for Internet services, in: Proceedings of the Fourth USENIX Symposium on Internet Technologies and Systems, March 2003.

[19] D.L. Cook, W.G. Morein, A.D. Keromytis, V. Misra, D. Rubenstein, WebSOS: protecting web servers from DDoS attacks, in: Proceedings of the 11th IEEE International Conference on Networks (ICON), September/October 2003.

[20] W.G. Morein, A. Stavrou, D.L. Cook, A.D. Keromytis, V. Misra, D. Rubenstein, Using graphic turing tests to counter automated DDoS attacks against web servers, in: Proceedings of the 10th ACM International Conference on Computer and Communications Security (CCS), October 2003.

[21] Cisco Systems, Cisco IOS Network Security, Cisco Press, 1998.

[22] Cisco Systems, Cisco PIX 500 Series Firewalls, http://www.cisco.com/warp/public/cc/pd/fw/sqfw500/.

[23] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: ACM SIGCOMM'01, 2001.

[24] D. Malkhi, M. Naor, D. Ratajczak, Viceroy: a scalable and dynamic emulation of the butterfly, in: Proceedings of the ACM PODC'02, July 2002.

[25] F. Kaashoek, D.R. Karger, Koorde: a simple degree-optimal Hash table, in: Proceedings of the Second International Workshop on Peer-to-Peer Systems (IPTPS'03), Feburary 2003.

[26] J. Spencer, The strange logic of random graphs, Number 22 in Algorithms and Combinatorics, Springer-Verlag, 2000, ISBN 3-540-41654-4.

[27] L. Zhang, VirtualClock: a new traffic control algorithm for packet switching networks, ACM Transactions on Computer Systems 9 (2) (1991) 101–124.

[28] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, The MIT Press and McGraw-Hill Book Company, 1989.

Shigang Chen received his B.S. degree in computer science from University of Science and Technology of China in 1993. He received M.S. and Ph.D. degrees in computer science from University of Illinois at Urbana-Champaign in 1996 and 1999, respectively. After graduation, he had worked with Cisco Systems for 3 years before joining University of Florida as an assistant professor in the Department of Computer and Information Science and Engineering. His research interests include quality of service, wireless networks, Internet security, and overlay networks. He received IEEE Communications Society Best Tutorial Paper Award in 1999. He was a guest editor for ACM/Baltzer Journal of Wireless Networks (WINET) and IEEE Transactions on Vehicle Technologies. He served as a TPC co-chair for the Computer and Network Security Symposium of IEEE IWCCC 2006, a vice TPC chair for IEEE MASS 2005, a vice general chair for QShine 2005, a TPC co-chair for QShine 2004, and a TPC member for many conferences including IEEE ICNP, IEEE INFOCOM, IEEE SANS, IEEE ISCC, IEEE Globecom, etc.

Yibei Ling is research scientist at applied research, Telcordia technologies (formerly Bellcore). His research interests include distributed computing, query optimization in database management system, scheduling, checkpointing, system performance, fault localization and self-healing in mobile ad hoc network and power-aware routing in mobile ad hoc network. He has published several papers in IEEE Transactions on computers, IEEE Transactions on Knowledge and Data Engineering, IEEE Transactions on Biomedical Engineering, SIGMOD, ICDE, PODC, Information system. He is the architect, as well as developer, of the voice subsystem of Telcordia Notification System. He received his B.S. in EE from Zhejiang University in 1982, his M.S. in statistics from Shanghai medical university (now Fuda University) in 1988, and his Ph.D. in CS from Florida State University at Miami in 1995. He is a member of the IEEE.

Randy Chow received the B.S. degree in electronic engineering from National Chiao-Tung University, Taiwan, in 1968, and the M.S. and Ph.D. degrees from the Department of Computer and Information Science at the University of Massachusetts in 1974 and 1977, respectively. He has been on the faculty in the Computer and Information Science and Engineering Department at the University of Florida since 1981, where he is currently a full professor. His research interests are distributed systems, computer networks, and security with a focus on ontology-based information access models for workflow systems. His recent affiliations include serving as a program director for the Distributed Systems and Compiler Program at NSF from 2001 to 2003 and a visiting professor at National Chiao-Tung University form 2003 to 2004. He is a member of IEEE, ACM, and editorial board of the Journal of Information Science and Engineering.

Ye Xia is an assistant professor at the Computer and Information Science and Engineering department at the University of Florida, starting in August 2003. He has a Ph.D. degree from the University of California, Berkeley, in 2003, an M.S. degree in 1995 from Columbia University, and a B.A. degree in 1993 from Harvard University, all in Electrical Engineering. Between June 1994 and August 1996, he was a member of the technical staff at Bell Laboratories, Lucent Technologies in New Jersey. His research interests are in computer networking area, including performance evaluation of network protocols and algorithms, congestion control, resource allocation, and load-balancing on peer-to-peer networks. He is also interested in probability theory, stochastic processes and queueing theory.