Keeping Denial-of-Service Attackers in the Dark

Gal Badishi[1], Amir Herzberg[2], and Idit Keidar[1]

[1] The Technion Department of Electrical Engineering
[2] Bar Ilan University, Department of Computer Science

Abstract. We consider the problem of overcoming (Distributed) Denial of Service (DoS) attacks by realistic adversaries that can eavesdrop on messages, or parts thereof, but with some delay. We show a protocol that mitigates DoS attacks by eavesdropping adversaries, using only available, efficient packet filtering mechanisms based mainly on (addresses and) port numbers. Our protocol avoids the use of fixed ports, and instead performs 'pseudo-random port hopping'. We model the underlying packet-filtering services and define measures for the capabilities of the adversary and for the success rate of the protocol. Using these, we analyze the proposed protocol, and show that it provides effective DoS prevention for realistic attack and deployment scenarios.

1 Introduction

Denial of service (DoS) attacks have proliferated in recent years, causing severe service disruptions [7]. The most devastating attacks stem from *distributed denial of service (DDoS)*, where an attacker utilizes multiple machines (often thousands) to generate excessive traffic [15]. Due to the acuteness of such attacks, various commercial solutions and off-the-shelf products addressing this problem have emerged.

The most common solution uses an existing firewall/router (or protocol stack) to perform *rate-limiting* of traffic, and to *filter* messages according to header fields like address and port number. Such mechanisms are cheap and readily available, and are therefore very appealing. Nevertheless, rate-limiting indiscriminately discards messages, and it is easy to spoof (fake) headers that match the filtering criteria: an attacker can often generate spoofed packets containing correct source and destination IP addresses, and arbitrarily chosen values for almost all fields used for filtering[3]. Therefore, the only hope in using such efficient filtering mechanisms to overcome DoS attacks lies in choosing values that are *unknown to the adversary*. E.g., TCP's use of a random initial sequence number is a simple version of this approach, but is inadequate if the attacker has some (even limited) eavesdropping capability.

More effective DoS solutions are provided by expensive commercial devices that perform stateful filtering [17–19]. These solutions specialize in protecting a handful of commonly-used stateful protocols, e.g., TCP; they are less effective for stateless traffic such as UDP [19]. Such expensive solutions are not suitable for all organizations.

[3] An exception is the TTL field of IP packets, which is automatically decremented by each router. This is used by some filtering mechanisms, e.g. BGP routers that receive only packets with maximal TTL value (255) to ensure the packets were sent by a neighboring router, and the Hop Counter Filtering proposal.

Finally, the most effective way to filter out offending traffic is using secure source authentication with message authentication codes (MAC), as in, e.g., IP-Sec [3]. However, this requires computing a MAC for every packet, which can induce significant overhead, and thus, this approach may be even more vulnerable to DoS attacks.

Our goal is to address DoS attacks on end hosts, e.g., in corporate networks, assuming the network leading to the hosts is functional. (A complementary solution protecting the end network can be deployed at the ISP.) In this paper, we focus on fortifying the basic building block of two-party communication. Specifically, we develop a DoS-resistant datagram protocol, similar to UDP or raw IP. Our protocol has promising properties, especially in overcoming realistic attack scenarios where attackers can discover some of the control information included in protocol packets [1]. Nevertheless, additional work is required in order to devise a practical system based on our ideas.

Our solution uses a *dual-layer approach*: On the one hand, we exploit cheap, simple, and readily-available measures at the network layer. On the other hand, we leverage these network mechanisms at the application layer. The latter allows for more complex algorithms as it has to deal with significantly fewer packets than the network layer, and may have closer interaction with the application. The higher layer dynamically changes the filtering criteria used by the underlying layer, e.g., by closing certain ports and opening others for communication. It is important to note that the use of dynamically changing ports instead of a single well-known port does not increase the chance of a security breach, as a *single* application is listening on *all* open ports.

The main contribution of our work is in presenting a *formal framework* for understanding and analyzing the effects of proposed solutions to the DoS problem. The main challenges in attempting to formalize DoS-resistance for the first time are: coming up with appropriate models for the attacker and the environment, modeling the functionality that can be provided by underlying mechanisms such as firewalls, and defining meaningful metrics for evaluating suggested solutions. We capture the functionality of a simple network-level DoS-mitigation solution by introducing the abstraction of a *port-based rationing channel*. Our primary metric of an end-to-end communication protocol's resistance to DoS attacks is *success rate*, which is the worst-case expected portion of valid application messages that successfully reach their destination, under a defined adversary class.

Having defined our model and metrics, we proceed to give a generic analysis of the communication success rate over a port-based rationing channel in different attack scenarios. We distinguish between *directed* attacks, where the adversary knows the port used, and *blind* attacks, in which the adversary does not know the port. Not surprisingly, we show that directed attacks are extremely harmful: with as little as 100 machines (or a sending capacity 100 times that of the protocol) the success rate is virtually zero. On the other hand, the worst-case success rate that an attacker can cause in blind attacks in realistic scenarios is well over 90% even with $10{,}000$ machines.

Our goal is therefore to "keep the attacker in the dark", so that it will have to resort to blind attacks. Our basic idea is to change the filtering criteria (i.e., ports) in a manner that cannot be predicted by the attacker. This *port-hopping* approach mimics the technique of frequency hopping spread spectrum in radio communication [20]. We assume that the communicating parties share a secret key unknown to the attacker; they apply a pseudo-random function [8] to this key in order to select the sequence of ports they will use. Note that such port-hopping has negligible effect on the communication overhead for realistic intervals between hops. The remaining challenge is synchronizing the processes, so that the recipient opens the port currently used by the sender. We present a protocol for doing so in a realistic partially synchronous model, where processes are equipped with bounded-drift bounded-skew clocks, and message latency is bounded.

1.1 Related Work

Our work continues the main line of research on prevention of Distributed Denial of Service attacks, which focuses on filtering mechanisms to block and discard the offending traffic. Our work is unique in providing rigorous model and analysis. Necessarily, we focus on what we consider as the most basic and common attack scenarios and on a simple network architecture, protecting the communication between two agents, both using our protocol. We hope that future work will extend our protocols, model and analysis to more complex attack scenarios and network architectures, as investigated in some of the related work.

Most closely related is the work presented in [1, 12], which proposes realistic and efficient mechanisms that do not require global adoption, yet allow a server to provide services immune to DDoS attacks. These solutions, like ours, utilize efficient packet-filtering mechanisms between the server and predefined, trusted ‘access point’ hosts, and use an overlay network to allow clients to forward messages via the overlay network to the access points and then to the server. Our protocol is appropriate for use between the access points and the server, utilizing efficient packet filtering mechanisms (as available in current routers). The basic ideas of filtering based on ports or other simple identifiers (‘keys’), and even of changing them, already appear in [1, 12], but without analysis and details. On the other hand, [1] provides a discussion of attack types and limitations, justifying much of our model, including the assumption that the exposure of the identifier (port) number may be possible but not immediate.

There are other several proposed methods to filter offending DoS traffic. Some proposals, e.g., [10, 13], filter according to the source IP address. This is convenient and efficient, allowing implementation in existing packet filtering routers. However, IP addresses are subject to spoofing; furthermore, using a white-list of source addresses of legitimate clients/peers is difficult, since many hosts may have dynamic IP addresses due to the use of NAT, DHCP and mobile-IP. Some proposals try to detect spoofed sender, using new routing mechanisms such as ‘path markers’ supported by some or all of the routers en route [21, 22, 2, 14]; notice that global router modification is difficult to achieve. Few proposals try to detect spoofed senders using only existing mechanisms, such as the hop count (TTL) [9]. However, empirical evaluation of these approaches show rather disappointing results [6].

A different approach is to perform application-specific filtering for pre-defined protocols [11, 16]. Such protection schemes are cumbersome, only work for a handful of well-known protocols, and are usually restricted to attackers that transmit invalid protocol packets.

In earlier work, we have presented Drum [5], a gossip-based multicast protocol resistant to DoS attacks at the application level. Drum does not make use of pseudo-random port-hopping, and it heavily relies on well-known ports that can be easily attacked. Therefore, Drum is far less resistant to DoS attacks than the protocol we present here. Finally, Drum focuses on multicast only, and as a gossip-based protocol, it relies on a high level of redundancy, whereas the protocol presented herein sends very little redundant information.

2 Model and Definitions

We consider a realistic *semi-synchronous* model, where processes have continuously-increasing local clocks with bounded drift Φ from real time. Each party may schedule events to occur when its local clock reaches a specific value (time). There is a bound Δ on the transmission delay, i.e., every packet sent either arrives within Δ time units, or is considered lost.

Our goal is to send messages from a sender A to a recipient B, in spite of attempts to disrupt this communication by an adversary. The basic technique available to the adversary is to clog the recipient by sending many packets. The standard defense deployed by most corporations is to rate-limit and filter packets, typically by a firewall. We capture this type of defense mechanism using a *port-based rationing channel* machine, which models the communication channel between A and B as well as the filtering mechanism. To send a message, A invokes a *ch_send(m)* event, a message is received by the channel in a *net_recv(m)* event, and B receives messages via *ch_recv(m)* events. We assume that the adversary cannot clog the communication to the channel, and that there is no message loss other than in the channel. The channel discards messages when it performs rate-limiting and filtering.

The channel machine is formally defined in [4]. We now provide an intuitive description of its functionality. Since we assume that the attacker can spoof packets with valid addresses, we cannot use these addresses for filtering. Instead, the channel filters packets using port numbers, allowing deployment using existing, efficient filtering mechanisms. Specifically, let the set of port numbers be $\{1, \ldots, \psi\}$. We are particularly interested in the case $\psi = 65536$, since this is the number of ports available for UDP recipients, and using them requires minimal changes on the sender side. The buffer space of the channel is a critical resource. The channel's interface includes the *alloc* action, which allows B to break the total buffer space of R messages into a separate allocation of R_i messages per port $i \in \{1, \ldots, \psi\}$, as long as $R \geq \sum_{i=1}^{\psi} R_i$. For simplicity, we assume that the buffers are read and cleared together in a single *deliver* event, which occurs exactly once on every integer time unit. If the number of packets sent to port i since the last *deliver* exceeds R_i, a uniformly distributed random subset of R_i of them is delivered.

We define several parameters that constrain the adversary's strength. The most important parameter is the *attack strength*, C, which is the maximal number of messages that the adversary may inject to the channel between two *deliver* events.

As shown in [1], attackers can utilize different techniques to try to learn the ports numbers expected by the filters (and used in packets sent by the sender). However, these techniques usually require considerable communication and time. To simplify, we allow the adversary to eavesdrop by exposing messages, but we assume that the adversary can expose packets no earlier than $\mathcal{E}$ time after they are sent, where $\mathcal{E}$ is the *exposure delay* parameter. The exposure delay reflects the time it takes an attacker to expose the relevant information, as well as to distribute it to the (many) attacking nodes, possibly using very limited bandwidth (e.g., if sending from a firewalled network). Our protocol works well with as little as $\mathcal{E} > 5\Delta$. Notice that while we assumed messages *always* arrive within Δ time, this is only a simplification, and our results are valid even if a few messages arrive later than that; therefore, Δ should really be thought of as the typical maximal round trip time, and not as an absolute bound on a message's lifetime (e.g., a second rather than 60 seconds).

Since the adversary may control some behavior of the parties, we take a conservative approach and let the adversary schedule the *app_send(m)* events in which the application (at A) asks to send m to B. To prevent the adversary from abusing these abilities by simply invoking too many *app_send* events before a *deliver* event, we define the *throughput*, $T \geq 1$, as the maximal number of *app_send* events in a single time unit. We further assume that $R \geq \Delta T$, i.e. that the capacity of the channel is sufficient to handle the maximal rate of *app_send* events.

Since we focus on connectionless communication such as UDP, our main metric for a system's resiliency to DoS attacks is its *success rate*, namely the probability that a message sent by A is received by B.

Definition 1 (Success rate μ). *Let E be any execution of a given two-party protocol operating over a given port-based rationing channel with parameters $\Psi, R, C, \Phi, \Delta, \mathcal{E}$ and T, with adversary ADV. Let $end(E)$ be the time of the last* deliver *event in E. Let $sent(E)$ ($recv(E)$) be the number of messages sent (resp., received) by the application, in app_send (resp., app_recv) events during E, prior to $end(E) - \Delta$ (resp, $end(E)$). The* success rate *μ of E is defined as $\mu(E) = \frac{recv(E)}{sent(E)}$. The success rate of adversary ADV is the average success rate over all executions of ADV. The success rate of the protocol, denoted $\mu(\Psi, R, C, \Phi, \Delta, \mathcal{E}, T)$, is the worst success rate over all adversaries ADV.*

Finally, a protocol can increase its success rate by sending redundant information, e.g., multiple copies or error-correcting codes. We therefore also consider a system's *message* (*bit*) *complexity*, which is the number of messages (resp. redundant bits) sent on the channel per each application message.

3 Analyzing the Channel's Success Rate in a Single Slot with a Single Port

This section provides generic analysis of the probability of successfully communicating over a port-based rationing channel under different attacks, when messages are sent to a single open port, p. This analysis is independent of the timing model and the particular protocol using the channel, and can therefore serve to analyze different protocols that use such channels, e.g., the one we present in the ensuing section. We focus on a single $deliver$ event, and analyze the *channel's delivery probability*, which is the probability for a valid message in the channel's buffer to be delivered, in that event. Since every *ch_send(m)* event eventually results in m being added to the channel's buffer, we can use the channel's delivery probability to analyze the success rates of higher level protocols.

Let R_p denote the ration allocated to port p in the last *alloc* event, and let $In(p)$ be the contents of the channel's buffer for port p (see [4] for more details). Consider a $deliver$ event of a channel from A to B, when A sends messages only to port p. We introduce some notations:

$R_p = R$ is the value of the channel's R_p when $deliver$ occurs.
$a_p = a$ is the number of messages whose source is A in the channel's $In(p)$ when $deliver$ occurs. We assume $a \leq R$. If $a_p < R_p$ (i.e., $a < R$), we say that there is *over-provisioning* on port p.
c_p is the number of messages whose source is *not* A in $In(p)$ when $deliver$ occurs.

Assume that $1 \leq a \leq R$. If $c_p < R - a + 1$ then B receives A's messages, and the attack does not affect the communication from A to B on port p. Let us now examine what happens when $c_p \geq R - a + 1$.

Lemma 1 *If $c_p \geq R - a + 1$, then the channel's delivery probability is $\frac{R}{c_p+a}$.*

Proof. The channel delivers $m \in In(p)$ if it is part of the R messages read uniformly at random from the $c_p + a$ available messages. Thus, the delivery probability is $\frac{R}{c_p+a}$.

If the attacker knows that B has opened port p, it can direct all of its power to that port, i.e., $c_p = C$, where we assume $C \geq R - a + 1$. We call this a *directed attack*.

Corollary 1 *In a directed attack at rate C on B's port p, the delivery probability on the attacked port is $\frac{R}{C+a}$, assuming $1 \leq a \leq R$ and $C \geq R - a + 1$.*

Lemma 2 *For fixed R and c_p such that $1 \leq a \leq R$ and $c_p \geq R-a+1$, the probability of B receiving only invalid messages on port p decreases as a increases.*

The proof of this lemma is simple and is omitted due to space considerations.

3.1 Blind Attack

We define a *blind attack* as a scenario where A sends messages to a single open port, p, and the adversary cannot distinguish this port from a random one. We now analyze the worst-case delivery probability under a blind attack.

In general, an adversary's strategy is composed both of timing decisions and injected messages. The timing decisions affect a, the number of messages from A that are in the channel at a given delivery slot. Given that a is already decided, we define the set of all strategies of an attacker with sending rate C as:

$$S(C) \triangleq \Big\{\{c_i\}_{i\in\psi} \ | \ \forall i \in \psi: \ c_i \in \mathbb{N} \cup \{0\} \ \wedge \ \sum_{i=1}^{\psi} c_i = C\Big\}$$

Each strategy $s \in S$ is composed of the number of messages the attacker sends to each port. Note that since the adversary wishes to minimize the delivery probability, we restrict the discussion to the set of attacks that fully utilize the attacker's capacity for sending messages. We denote by $\mu_B(a, C, R) : S(C) \rightarrow [0, 1]$ the channel's delivery probability under all possible blind attack strategies with the given a, C, and R. Since S is a finite set, μ_B has at least one minimum point, and we define the delivery probability to be that minimum:

$$\mu_B(a, C, R) \triangleq \min_{s \in S(C)} \mu_B(a, C, R, s)$$

We sometimes use μ_B instead of $\mu_B(a, C, R)$ when a, C, and R are clear from context. We want to find lower bounds on μ_B, depending on the attacker's strength. We say that port p_i is attacked in strategy s if $c_{p_i} > 0$. We partition $S(C)$ according to the number of ports being attacked, as follows:

$$S_k \triangleq \{s \in S(C) \mid \text{Exactly } k \text{ ports are being attacked in } s\}$$

In [4] we find a lower bound on μ_B as follows: We first derive a lower bound on $\{\mu_B(a, C, R, s_k) | s_k \in S_k\}$; this lower bound is given as a function of k. Incidently, the worst depredation occurs when the attacker divides its power equally among the attacked ports, i.e., when it sends $\frac{C}{k}$ messages to each attacked port. Then, we show lower bounds on $\mu_B(a, C, R)$ by finding the k that yields the minimum value. In [4], we prove the following lemma:

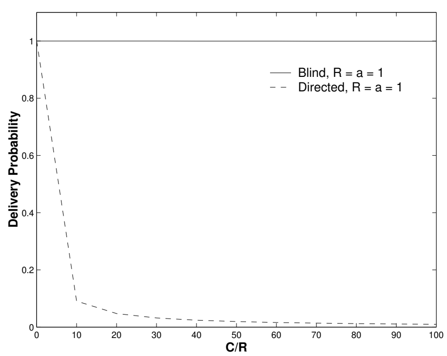
Lemma 3 *$\mu_B(a, C, R)$ is bounded from below by the following function $f(a, C, R)$:*

$$f(a, C, R) = \begin{cases} \frac{\psi a}{C + \psi a} & \text{if } R = a \text{ and } C \geq \psi \\ 1 - \frac{C}{\psi(1+a)} & \text{if } R = a \text{ and } C < \psi \\ \frac{\psi R}{C + \psi a} & \text{if } R > a \text{ and } C \geq \frac{\psi a}{\sqrt{\frac{R}{R-a}} - 1} \\ \frac{\psi a - C\left(\sqrt{\frac{R}{R-a}} - 1\right)}{\psi a} + \frac{R}{\psi} \cdot \frac{C\left(\sqrt{\frac{R}{R-a}} - 1\right)^2}{a^2 \sqrt{\frac{R}{R-a}}} & \text{if } R > a \text{ and } C < \frac{\psi a}{\sqrt{\frac{R}{R-a}} - 1} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

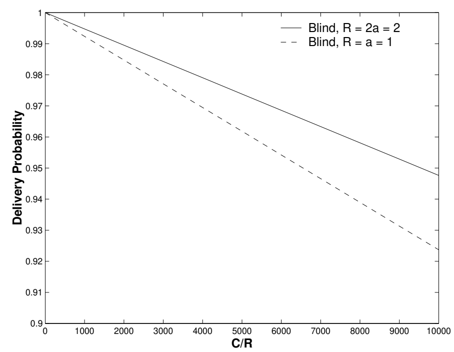
Lemma 3 provides us with some insights of the adversary's best strategy and of the expected degradation in delivery probability. If no over-provisioning is used (i.e., $R = a$), then the adversary's best strategy is to attack as many ports as possible. This is due to the fact that even a single bogus message to the correct port degrades the expected delivery probability. When the adversary has enough power to target all of the available ports with at least one message, it can attack with more messages per attacked port, and the delivery probability asymptotically degrades much like the function $\frac{1}{C}$. When not all ports are attacked, the adversary would like to use its remaining resources to attack more ports rather than target a strict subset of the ports with more than one bogus message per port. The degradation of the expected delivery probability is then linear as the attacker's strength increases.

When over-provisioning is used ($R > a$), it affects the attack and its result in two ways. First, the attacker's best strategy may not be to attack as many ports as it can, since a single bogus message per port does not do any harm now. Second, for an adversary with a given strength, the degradation in delivery probability is lower when over-provisioning is used than when it is not employed. We can see in Equation 1 that if the attacker has enough power to attack all the ports, the over-provisioning ratio $\frac{R}{a}$ is also the ratio by which the delivery probability is increased, compared to the case where $R = a$.

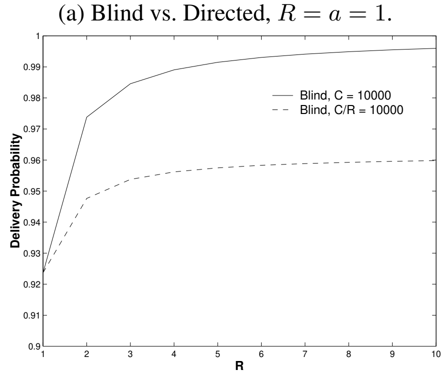
3.2 Actual Values



(a) Blind vs. Directed, $R = a = 1$.



(b) Blind, $a = 1$.



(c) Blind, $a = 1$.

Fig. 1. Delivery probability per slot in various attack scenarios on a single port, $\psi = 65536$.

Figure 1 shows the expected worst-case delivery probabilities for various attack scenarios on a single port. For directed attacks, we show the actual delivery probability, and for blind attacks, the lower bound $f(a, C, R)$ is shown. We chose $\psi = 65536$, the number of ports in common Internet protocols, e.g., UDP. Figure 1(a) illustrates the major difference between a directed attack and a blind one: even for a relatively weak attacker ($C \leq 100$), the delivery probability under a directed attack approaches 0, whereas under a blind attack, it virtually remains 1.

Figure 1(b) examines blind attacks by much stronger adversaries (with C up to $10{,}000$ for $R = 1$, and up to 20,000 for $R = 2$). We see that the delivery probability gradually degrades down to a low of 92.5% when $R = 1$. If we use an over-provisioned channel, i.e., have $a = 1$ (one message from A) when $R = 2$, the delivery probability improves to almost 95% for $C = 20{,}000$. (The ratio $\frac{C}{R}$ is the same for both curves). Figure 1(c) shows the effect of larger over-provisioning. We see that the cost-effectiveness of over-provisioning diminishes as $\frac{R}{a}$ increases.

4 DoS-Resistant Communication

We now describe a protocol that allows for DoS-resistant communication in a partially-synchronous environment. The protocol's main component is an ack-based protocol. B sends acknowledgments for messages it receives from A, and these acks allow the parties to hop through ports together. However, although the ack-based protocol works well as long as the adversary fails to attack the correct port, once the adversary gets a hold of the port used, it can perform a directed attack that renders the protocol useless. By attacking the found data port, or simultaneously attacking the found data and ack ports, the adversary can effectively drop the success rate to 0, and no port hopping will occur. To solve this matter, there is a time-based proactive reinitialization of the ports used for the ack-based protocol, independent of any messages passed in the system.

4.1 Ack-Based Port Hopping

We present a port-hopping communication protocol that has no indication of time. In the absence of a clock, A and B must rely on message-passing in order to synchronize their port-hopping. We present an *ack-based port-hopping* protocol, which uses two port-based rationing channels, from B to A (with ration R_{BA}) and vice versa (with ration R_{AB}). For simplicity we assume $R_{AB} = 2R_{BA} = 2R$. B always keeps two open ports for data reception from A, and A keeps one port open for acknowledgments (acks) from B. The protocol hops ports upon a successful round-trip on the most recent port used, using a *pseudo-random function*, PRF^*[4]. In order to avoid hopping upon adversary messages, all protocol messages carry authentication information, using a second pseudo-random function, PRF, on $\{0,1\}^\kappa$. (We assume that PRF and PRF^* use different parts of A and B's shared secret key.)

The protocol's pseudocode appears in Figure 2. Both A and B hold a port counter P, initialized to some $seed$ (e.g., 1). Each party uses its counter P in order to determine which ports should be open, and which ports to send messages to. B opens port p_{old} using the $(P-1)^{th}$ element in the pseudo-random sequence, and p_{new}, using P. A sends data messages to the P^{th} port in the sequence, and opens the P^{th} port in a second pseudo-random sequence designated for acks. When B receives a valid data message

[4] Intuitively, we say that $f_{key}(data)$ is *pseudo-random function (PRF^*)* if for inputs of sufficient length, it cannot be distinguished efficiently from a truly random function r over the same domain and range, by a PPT adversary which can receive $g(x)$ for any values of x, where $g = r$ with probability half and $g = f$ with probability half. For definition and construction, see [8].

from A on port p_{old}, it sends an ack to the old ack port. When it receives a valid message on port p_{new}, it sends an ack to the P^{th} ack port, and then increases P. When A receives a valid ack on port p_{ack}, it increases P. We note that several data messages may be in transit before a port hop takes place, since it takes at least one round-trip time for a port hop to take effect. The proof of the next theorem is given in [4].

PROTOCOL FOR RECIPIENT B:

On $ch\_recv(m, p_{new})$**:**
  if $m.auth = PRF_{S_{AB}}(P|\text{"}data\text{"})$ **then**
    $app\_recv(m.data)$
    $alloc(p_{old}, 0)$
    $p_{old} = p_{new}$
    $p_{new} = PRF^*_{S_{AB}}(P+1|\text{"}data\text{"})$
    $alloc(p_{new}, R_{AB}/2)$
    $ack = PRF_{S_{AB}}(P|\text{"}ack\text{"})$
    $ch\_send(ack, PRF^*_{S_{AB}}(P|\text{"}ack\text{"}))$
    $P = P + 1$

On $ack\_init(seed)$**:**
    $P = seed$
    $p_{old} = PRF^*_{S_{AB}}(P-1|\text{"}data\text{"})$
    $p_{new} = PRF^*_{S_{AB}}(P|\text{"}data\text{"})$
    $alloc(p_{old}, R_{AB}/2)$
    $alloc(p_{new}, R_{AB}/2)$

On $ch\_recv(m, p_{old})$**:**
  if $m.auth = PRF_{S_{AB}}(P-1|\text{"}data\text{"})$ **then**
    $app\_recv(m.data)$
    $ack = PRF_{S_{AB}}(P-1|\text{"}ack\text{"})$
    $ch\_send(ack, PRF^*_{S_{AB}}(P-1|\text{"}ack\text{"}))$

PROTOCOL FOR SENDER A:

On $ack\_init(seed)$**:**
    $P = seed$
    $p_{ack} = PRF^*_{S_{AB}}(P|\text{"}ack\text{"})$
    $alloc(p_{ack}, R_{BA})$

On $app\_send(data)$**:**
    $m = data|PRF_{S_{AB}}(P|\text{"}data\text{"})$
    $ch\_send(m, PRF^*_{S_{AB}}(P|\text{"}data\text{"}))$

On $ch\_recv(ack, p_{ack})$**:**
  if $ack.auth = PRF_{S_{AB}}(P|\text{"}ack\text{"})$ **then**
    $alloc(p_{ack}, 0)$
    $p_{ack} = PRF^*_{S_{AB}}(P+1|\text{"}ack\text{"})$
    $alloc(p_{ack}, R_{BA})$
    $P = P + 1$

Fig. 2. Two-party ack-based port-hopping.

Theorem 1 *When using the ack-based protocol, the probability that a data message that A sends to port p arrives when p is open is 1 up to a polynomially-negligible factor*[5].

In order to compute the throughput that the protocol can support in the absence of a DoS attack (i.e., when $C = 0$), we need to take latency variations into consideration. Since messages sent up to Δ time apart can arrive in the same delivery slot, a throughput $T \leq R/\Delta$ ensures $a \leq R$. For the rest of this section, we assume $T \leq R/\Delta$.

We now analyze the protocol's success rate under DoS attacks. We say that the adversary is in *blind mode* if it does not know the ports used by the protocol. We first

[5] Namely, for every polynomial $g > 0$, there is some κ_g s.t. when $\kappa \geq \kappa_g$, then the success rate $\mu(t) \geq f(R, C, R) - g(\kappa)$.

give a lower bound on the success rate in blind mode, and then give a lower bound on the probability to be in blind mode at a given time t. Finally, μ is bounded by the probability to be in blind mode throughout the execution of the protocol, times the success rate in blind mode.

Suppose B opens port p with reception rate R_p, and that $a \leq R_p$ messages from A are waiting in its channel, along with c_p messages from the adversary ($c_p \geq 0$). By Lemma 1, the success rate monotonically non-increases with a. Since the adversary can control a by varying the network delays, it can set a as high as possible for a delivery slot. Therefore the worst case success rate occurs when $a = T\Delta$. Using Equation 1 in Section 3, we get that the success rate in blind mode is bounded from below by $f(T\Delta, C, R)$.

Note that the protocol begins in blind mode. We now analyze the probability that the protocol can keep the adversary in blind mode. The only way the adversary can learn of a port used by the protocol is using an *expose* event $\mathcal{E}$ time after a message is sent to this port. This information is only useful to the adversary if the port is still in use. Let us trace the periodic sequence of events that causes the data port to change (once the data port changes, the acks for the old port are useless). Assume that A continuously sends messages $m_1, m_2, \ldots$ to B starting at time 0, and consider an execution without an attack: (1) By time Δ, B receives a valid message from the channel and sends an ack to A; (2) By time 2Δ, A receives the ack and changes the sending port; (3) B gets the last message destined for the old port at most at time 3Δ.

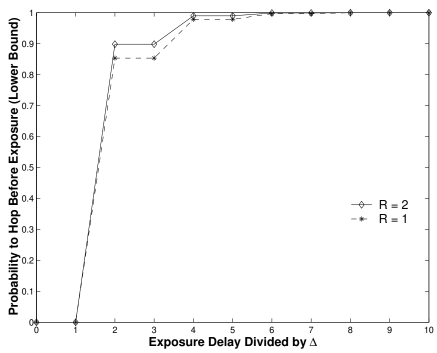
If $\mathcal{E} \geq 3\Delta$, the adversary remains in blind mode. Now let us examine what happens under attack. In order to prevent the port from changing, the adversary must either prevent B from getting valid data messages or prevent A from receiving acks. By Lemma 2, the probability that all valid messages are dropped decreases when a increases. Thus, (as opposed to the previous analysis), in order to increase the probability that all valid messages are dropped, the adversary should set $a \leq 1$ whenever possible. Denote $\mu_B = f(1, C, R)$, the lower bound on the probability of a single message to be received on a single port given in Section 3.1.

Lemma 4 *If $\mathcal{E} = 2k\Delta$ for $k > 0$, and A sends messages to B at least every 2Δ time units, then the probability that the port changes while the attacker is still blind is at least $1 - (1 - \mu_B^2)^k$.*

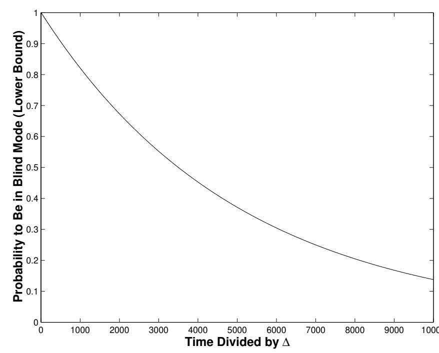
Proof. The probability that the port does not change in a single round-trip is at most $1 - \mu_B^2$. Since A sends messages to B every 2Δ time units, at the conclusion of each maximal time round-trip, there is at least one new message on its own round-trip. In order for the port not to change while the adversary is still blind, every round-trip needs to fail. Since the attacker can react only after $2k\Delta$ time, there is time for k round-trips in which the attacker is blind, even if none of them succeed. The probability that all of them fail is less than $(1 - \mu_B^2)^k$. If one succeeds, the port changes. And so, the probability that the port changes is at least $1 - (1 - \mu_B^2)^k$.

The lower bound above is illustrated in Figure 3(a).

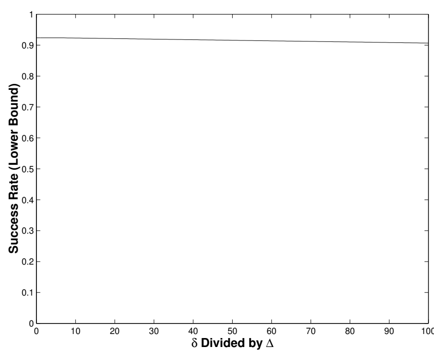
We now bound the probability to be in blind mode at time t, by assuming that once the attacker leaves the blind mode it never returns to it. The bound is computed using a



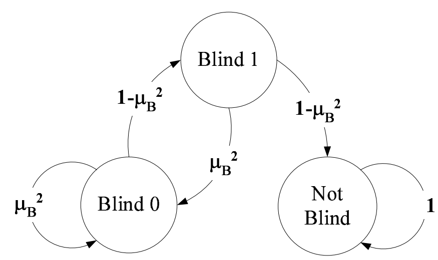
(a) Bound on prob. to hop before exposure, $\frac{C}{R} = 10{,}000$.



(b) Bound (g) on prob. to stay in blind mode, $R = 1$, $C = 10{,}000$, $\mathcal{E} = 4\Delta$.



(c) Bound on protocol's success rate, $\Delta = T = R = 1$, $C = 10{,}000$, $\mathcal{E} = 5\Delta$.



(d) Markov chain for computing the lower bound in Figure 3(b).

Fig. 3. The effect of $\mathcal{E}$ on the ack-based protocol, $\psi = 65536$.

Markov chain, where each state is the number of round-trips that have failed since the last port change. In the last state, all round-trips have failed before the exposure, and thus the attacker is no longer blind. The Markov chain for $\mathcal{E} = 4\Delta$ is shown in Figure 3(d). We use the chain's transition matrix to compute the probability $g(t, \mathcal{E}, C, R)$ for remaining in blind mode at time t. Figure 3(b) shows values of g for $\mathcal{E} = 4\Delta$. We can see that the protocol works well only for a limited time.

Finally, we note that the protocol's message complexity is 2, since it sends an ack for each message, and its bit complexity is constant: $\log_2(\psi)$ bits for the port plus κ bits for the authentication code.

4.2 Adding Proactive Reinitializations

We now introduce a proactive reinitialization mechanism that allows choosing new seeds for the ack-based protocol depending on time and not on the messages passed in the system. We denote by $t_A(t)$ and $t_B(t)$ the local clocks of A and B, resp., where t

is the real time. From Section 2 we get that $0 \leq |t_A(t) - t| \leq \Phi$, $0 \leq |t_B(t) - t| \leq \Phi$. We also assume $t_A, t_B \geq 0$.

If A reinitializes the ack-based protocol and then sends a message to B at time $t_A(t_0)$, this message can reach B anywhere in the real time interval $(t_0, t_0 + \Delta]$. Therefore, the port used by A at $t_A(t_0)$ must be open by B at least throughout this interval. To handle the extreme case where A sends a message at the moment of reinitialization, B must use the appropriate port starting at time $t_B(t_0) - \Phi$. (We note that t_0 may also be Φ time units apart from $t_A(t_0)$.) We define δ as the number of time units between reinitializations of the protocol, and assume for simplicity and effectiveness of resource consumption that $\delta > 4\Phi + \Delta$ (see Figure 4 for more details).

Every δ time units, A feeds a new seed to the ack-based protocol, and B anticipates it by creating a new instance of the protocol, which waits on the new expected ports. Once communication is established using the new protocol instance, or once it is clear that the old instance is not going to be used anymore, the old instance is terminated. The pseudocode for the proactive reinitialization mechanism can be found in Figure 4 (where ABPI stands for *ack-based protocol instance*). Due to space considerations we do not detail the change in port rations at the recipient's side as protocol instances are created or terminated. We also note that there is a negligible probability that more than one ack-based protocol instance will share the same port. Even if this happens, differentiating between instances can be easily done by adding the instance number (i.e., the total number of times a reinitialization was performed) to each message. The proof of the next theorem is given in [4].

ADD-ON FOR SENDER A:

Whenever $t_A(t) \in \{0, \delta, 2\delta, \ldots\}$**:**
  $ack\_init(t_A(t)/\delta)$

ADD-ON FOR RECIPIENT B:

When $t_B(t) = 0$**:**
  Create 1^{st} ABPI
  For that ABPI, $ack\_init(0)$

ADD-ON FOR RECIPIENT B (CONT'D):

Whenever $(t_B(t) + 2\Phi) \in \{\delta, 2\delta, 3\delta, \ldots\}$**:**
  Create a new ABPI
  For that ABPI, $ack\_init((t_B(t) + 2\Phi)/\delta)$

$4\Phi + \Delta$ time after creating a new ABPI
or Δ time after receiving 1^{st} msg for that ABPI**:**
  Terminate all older ABPIs

Fig. 4. Proactive reinitialization of the ack-based protocol.

Theorem 2 *When using the ack-based protocol with proactive reinitializations, the probability that a data message that* A *sends to port* p *arrives when* p *is open is* 1 *up to a polynomially-negligible factor.*

Proactive reinitialization every δ time units allows us to limit the expected degradation in success rate for a single ack-based protocol instance. Choosing δ is therefore an important part of the combined protocol. A small δ allows us to maintain high success rate in the ack-based protocol, but increases the average number of ports that are open in every time unit (due to running several protocol instances in parallel). When several ports are used the ration for each one of them is decreased, and so might the success rate. On the other hand, choosing a high δ entails lower success rate between reinitializations. We conclude the discussion above and the results presented in Section 4.1 with the following theorem:

Theorem 3 *The success rate of the proactively reinitialized ack-based protocol with throughput* $T \le R/\Delta$ *and reinitialization periods of length* δ *is bounded from below by:* $g(\delta + \Delta, \mathcal{E}, C, R) \cdot f(T\Delta, C, R)$ *up to a polynomially-negligible factor.*

Figure 3(c) shows the value of $g(\delta + 1, \mathcal{E}, 10000, 1) \cdot f(1, 10000, 1, 1)$.

5 Conclusions and Directions for Future Work

We have presented a model for port-based rationing channels, and a protocol robust to DoS attacks, for communication over such channels. The protocol is simple, efficient and effective, as shown by our analysis. While this worst case analysis is valuable, it can be followed by simulations, experiments, and common case analysis. Moreover, the system aspects of deploying such a protocol in today's Internet are yet to be dealt with.

As the important field of application-level DoS mitigation is relatively new, there is much research space to explore. We now describe several future research directions. Clearly, many more directions can be sought.

Our model is realistic, as it only requires the underlying channel to provide port-based filtering; therefore, it can be efficiently implemented using existing mechanisms, typically at a firewall or router between the victim and the attacker. This raises an interesting question regarding the trade-off between the cost and the possible added value of implementing additional functionality by the channel (e.g., at the firewall).

This work has focused on two parties only. It would be interesting to extend it to multiparty scenarios, such as client-sever and multicast. These scenarios may require a somewhat different approach, and will obviously necessitate analyses of their own. Furthermore, we required the parties to share a secret key; we believe we can extend the solution to establish this key using additional parties, e.g., a key distribution center. Another approach is to use 'proof of work' (computational puzzles) techniques to establish the shared key, by sending an encrypted key together with a 'proof of work' that ensures that the sender worked much more than the recipient.

Our work has focused on resisting DoS attacks; however, it could impact the performance and reliability properties of the connection; in fact, it is interesting to explore combinations between our model and problem, and the classical problems of reliable communication over unreliable channels and networks. Furthermore, since our work requires a shared secret key, it may be desirable to merge it with protocols using shared secret key for confidentiality and authentication, such as SSL/TLS and IP-Sec.

One of the most important issues when designing a secure and robust protocol is to quantify its effectiveness under attack. This calls for a clear definition of a metric for that protocol. But even more importantly, such an analysis requires a clear definition of the adversary and the environment it operates in. The challenge in modeling the adversary and the environment rises because the model should reflect reality. Good models allow to test the protocol as if it deployed "in the wild", where real adversaries give it their best shot. Such models enable us to find the attacker's optimal strategy, or provide a bound thereof, thus permitting analytical analysis of the protocol. We hope that future work will take further strides towards defining realistic yet tractable models.

References

1. D. G. Andersen. Mayday: Distributed filtering for internet services. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.
2. K. Argyraki and D. R. Cheriton. Active internet traffic filtering: Real-time response to denial-of-service attacks. In *Proceedings of the USENIX Annual Technical Conference*, April 2005.
3. R. Atkinson. Security architecture for the internet protocol. RFC 2401, IETF, 1998.
4. G. Badishi, A. Herzberg, and I. Keidar. Keeping denial-of-service attackers in the dark. TR CCIT 541, Department of Electrical Engineering, Technion, July 2005.
5. G. Badishi, I. Keidar, and A. Sasson. Exposing and eliminating vulnerabilities to denial of service attacks in secure gossip-based multicast. In *The International Conference on Dependable Systems and Networks (DSN)*, pages 223–232, June.July 2004.
6. M. Collins and M. K. Reiter. An empirical analysis of target-resident dos filters. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 103–114, May 2004.
7. CSI/FBI. Computer crime and security survey, 2003.
8. O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the Association for Computing Machinery*, 33(4):792–807, 1986.
9. C. Jin, H. Wang, and K. G. Shin. Hop-count filtering: an effective defense against spoofed DDoS traffic. In V. Atluri and P. Liu, editors, *Proceedings of the 10th ACM Conference on Computer and Communication Security (CCS-03)*, pages 30–41, New York, Oct. 27–30 2003. ACM Press.
10. J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *Proceedings of the International World Wide Web Conference*, pages 252–262. IEEE, May 2002.
11. Juniper Networks. The need for pervasive application-level attack protection.
12. A. D. Keromytis, V. Misra, and D. Rubenstein. Sos: An architecture for mitigating ddos attacks. *Journal on Selected Areas in Communications*, 21(1):176–188, 2004.
13. B. Krishnamurthy and J. Wang. On network-aware clustering of Web clients. In *Proceedings of the SIGCOMM*, Aug. 2000.
14. P. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *Computer Communications Review*, 32(3):62–73, July 2002.
15. D. Moore, G. Voelker, and S. Savage. Inferring Internet denial-of-service activity. In *Proceedings of the 10th USENIX Security Symposium*, pages 9–22, August 2001.
16. NetContinuum. Web application firewall: How netcontinuum stops the 21 classes of web application threats.
17. P-Cube. DoS protection.
18. P-Cube. Minimizing the effects of DoS attacks.
19. Riverhead Networks. Defeating DDoS attacks.
20. S. M. Schwartz. Frequency hopping spread spectrum (fhss).
21. A. Yaar, A. Perrig, and D. Song. Pi: A path identification mechanism to defend against DDoS attacks. In *IEEE Symposium on Security and Privacy*, May 2003.
22. A. Yaar, A. Perrig, and D. Song. An endhost capability mechanism to mitigate DDoS flooding attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.