Exposing and Eliminating Vulnerabilities to Denial of Service Attacks in Secure Gossip-Based Multicast

Gal Badishi
EE Department, Technion

Idit Keidar
EE Department, Technion

Amir Sasson
CS Department, Technion

Abstract

We propose a framework and methodology for quantifying the effect of denial of service (DoS) attacks on a distributed system. We present a systematic study of the resistance of gossip-based multicast protocols to DoS attacks. We show that even distributed and randomized gossip-based protocols, which eliminate single points of failure, do not necessarily eliminate vulnerabilities to DoS attacks. We propose Drum – a simple gossip-based multicast protocol that eliminates such vulnerabilities. Drum was implemented in Java and tested on a large cluster. We show, using closed-form mathematical analysis, simulations, and empirical tests, that Drum survives severe DoS attacks.

1. Introduction

One of the most devastating security threats faced by a distributed system is a *denial of service* (DoS) attack, in which an attacker makes a system unresponsive by forcing it to handle bogus requests that consume all available resources. In 2003, approximately 42% of U.S. organizations, including government agencies, financial institutions, medical institutions and universities, were faced with DoS attacks [6]. That year, DoS attacks were the second most financially damaging attacks (65 million USD), only short of theft of proprietary information (70 million USD), and far above other attacks ($0.07-27$ million USD) [6]. Therefore, coping with DoS attacks is essential when deploying services in a hostile environment such as the Internet [17].

As a first defense, one may protect a system against DoS attacks using network-level mechanisms [5]. However, network-level filters cannot detect DoS attacks at the application level, when the traffic seems legitimate. Even if means are in place to protect against network-level DoS, an attack can still be performed at the application level, as the bandwidth needed to perform such an attack is usually lower. This is especially true if the application performs intensive computations for each message, as occurs, e.g., with secure protocols based on digital signatures. In this paper, we are concerned with DoS attacks on *secure* application-level multicast protocols (such as, e.g., Spinglass [3]), focusing only on the multicast protocol layer.

A DoS attack that targets every process in a large system inevitably causes performance degradation, but also requires vast resources. In order to be effective even with limited resources, attackers target vulnerable parts of the system. For example, consider a tree-based multicast protocol; by targeting a single inner node in the tree, an attacker can effectively partition the multicast group. Hence, eliminating single points of failure is an essential step in constructing protocols that are less vulnerable to DoS attacks.

We therefore focus on gossip-based (epidemic) multicast protocols, e.g., [7, 2, 8, 10], which eliminate single points of failure using redundancy and random choices. Such protocols are robust and have been shown to provide graceful degradation in the face of amounting failures [9, 11]. One may expect that such a system will not suffer from vulnerabilities to DoS attacks, since it can continue to be effective when many processes fail. Surprisingly, we show that gossip-based protocols can be extremely vulnerable to DoS attacks targeted at a small subset of the processes. This occurs because an attacker can effectively isolate a small set of processes from the rest of the group by attacking this set.

To quantify the effects of DoS attacks, we measure their influence on the time it takes to propagate a message to all the processes in the system, as well as on the average throughput processes can receive. We do this using asymptotic analysis, simulations, and measurements.

Having observed the vulnerabilities of traditional protocols, we turn to search for a protocol that will eliminate these vulnerabilities. Specifically, our goal is to design a protocol that would not allow an attacker to increase the damage it causes by focusing on a subset of the processes. We are not familiar with any previous protocol that achieves this goal.

We present *Drum* (DoS-Resistant Unforgeable Multicast), a gossip-based multicast protocol, which, using a few simple ideas, eliminates common vulnerabilities to DoS attacks. Mathematical analysis and simulations show that Drum indeed achieves our design goal: when an adversary has a large sending capacity, its most effective attack against Drum is an all-out attack that distributes the attacking power as broadly as possible. Obviously, performance degradation due to a broad all-out DoS attack is unavoidable for any multicast protocol, and indeed all the tested protocols exhibit the same performance degradation under such a broad attack.

We have implemented Drum in Java and tested it on a on a cluster of workstations. Our measurements validate the analysis and simulation results, and show that Drum can withstand severe DoS attacks, where naïve protocols that do not take any measures against DoS attacks completely collapse. E.g., under an attack that focuses on 10% of the processes, Drum's latency and throughput remain *constant* as the attack strength increases, whereas in traditional protocols, the latency grows *linearly* with the attack strength, and the throughput continuously degrades.
In summary, this paper makes the following contributions:

- It presents a new framework and methodology for quantifying the effects of DoS attacks. We are not familiar with any previously suggested metrics for DoS-resistance nor with previous attempts to quantify the effect of DoS attacks on a system.
- It uses the new methodology to conduct the first systematic study of the impact of DoS attacks on multicast protocols. This study exposes vulnerabilities in traditional gossip-based protocols.
- It presents Drum, a simple gossip-based multicast protocol that eliminates such vulnerabilities. We believe that the ideas used in Drum can serve to mitigate the effect of DoS attacks on other protocols as well.
- It provides closed-form asymptotic analysis as well as simulations and measurements of gossip-based multicast protocols under DoS attacks varying in strength and extent.

This paper proceeds as follows: Section 2 gives background on gossip-based multicast and related work. Section 3 presents the system model. Section 4 describes Drum. Section 5 presents our evaluation methodology and considered attack models. The following three sections evaluate Drum and compare it to traditional gossip-based protocols using various tools: Section 6 gives closed-form asymptotic latency bounds; Section 7 provides a thorough evaluation using simulations; and Section 8 presents actual latency and throughput measurements. Section 9 concludes.

2. Background and Related Work

Gossip-based dissemination [7] is a leading approach in the design of scalable reliable application-level multicast protocols, e.g., [2, 8, 10]. Our work focuses on symmetric gossip-based multicast protocols that do not rely on external mechanisms such as IP multicast, e.g., lpbcast [8].

Such protocols work roughly as follows: Each process locally divides its time into *gossip rounds*; rounds are not synchronized among the processes. In each round, the process randomly selects a small number of processes to gossip with, and exchanges information with them. Every piece of information is gossiped for a number of rounds. It has been shown that the propagation time of gossip protocols increases logarithmically with the number of processes [19, 10]. There are two methods for information dissemination: (1) *push*, in which the process sends messages to selected processes; and (2) *pull*, in which the process requests messages from selected processes. Both methods are susceptible to DoS attacks: attacking the incoming push channels of a process may prevent it from receiving valid messages, and attacking a process's incoming pull channels may prevent it from sending messages to valid targets. Some protocols use both methods [7, 10]. Karp et al. showed that combining push and pull allows the use of fewer transmissions to ensure data arrival to all group members [10].

Drum utilizes both methods, and in addition, allocates a bounded amount of resources for each operation (push and pull), so that a DoS attack on one operation does not hamper the other. Such a resource separation approach was also used in COCA [22], for the sake of overcoming DoS attacks on authentication servers. Note that Drum deals with DoS attacks at the application-level. Network-level DoS analysis and mitigation has been extensively dealt with (e.g, [20, 4]), but DoS-resistance at the secure multicast service layer has gotten little attention.

Secure gossip-based dissemination protocols were suggested by Malkhi et al. [13, 14, 15]. However, they did not deal with DoS attacks. Follow-up work by Minsky and Schneider [16] suggested a pull-based protocol that can endure limited DoS attacks by bounding the number of accepted requests per round. However, these works solve the *diffusion* problem, in which each message simultaneously originates at more than t correct processes, where up to t processes may suffer Byzantine failures. In contrast, we consider a multicast system where a message originates at a single source. Hence, using a pull-based solution as suggested in [16] does not help in withstanding DoS attacks. Moreover, Minsky and Schneider [16] focus on load rather than DoS attacks; they include only a brief analysis of DoS attacks, under the assumption that no more than t processes perform the attack, and that each of them generates a single message per round (the reception bound is also assumed to be one message per round). In contrast, we focus on substantially more severe attacks, and study how system performance degrades as the attack strength increases.

DoS can also be caused by churn, where processes rapidly join and leave [12], thus reducing availability. In Drum, as in other gossip-based protocols, churn has little effect on availability: even when as many as half the processes fail, such protocols can continue to deliver messages reliably and with good quality of service [9, 11]. A DoS attack of another form can be caused by process perturbations, whereby some processes are intermittently unresponsive. The effect of perturbations is analyzed in [2], where it is shown that probabilistic protocols, e.g., gossip-based protocols, solve this problem. This paper focuses on DoS attacks in which the attacker sends fabricated application messages. We note that our work is the first that we know of that conducts a systematic study of the effect of DoS attacks on message latency.

3. System Model and Architecture

Drum supports probabilistically reliable multicast [2, 8, 10] among processes that are members of a group. Each message is created by exactly one group member (its *source*).

We assume that the underlying network is fully-connected. There are no bounds on message delays, i.e., the communication is asynchronous. The loss rate on the communication links is bounded, uniform, and independent of any other factor. The communication channels are insecure, meaning that senders of incoming messages cannot be reliably identified in a simple manner. However, the data messages' sources (originators) can be identified using standard cryptographic techniques, e.g., [18]. Additionally, some information intended for a specific process may be encrypted using, e.g., a public-key infrastructure.

An adversary can generate fabricated messages and snoop on messages. However, these operations require the adversary to utilize resources. Malicious processes perform DoS attacks on group members. In case these malicious processes are part of the group, they also refrain from forwarding messages.

For simplicity, we consider a static group of *n* processes and assume that every process has complete knowledge of all the other processes in the group. In the full paper [1] we explain how to deal with dynamic membership, i.e., joins and leaves. We note that having incomplete knowledge of current group members in a dynamic setting poses no problem, as long as enough members are known. For more details see [1].

4. DoS-Resistant Gossip-Based Multicast Protocol

Drum is a simple gossip protocol, which achieves DoS-resistance using a combination of pull and push operations, separate resource bounds for different operations, and the use of random ports in order to reduce the chance of a port being attacked.

Each process, p, locally divides its time into rounds. A round is typically in the order of a second, and its duration may vary according to local random choices. Every round, p chooses two small (constant size) random sets of processes, $view_{push}$ and $view_{pull}$, and gossips with them. E.g., when these views consist of two processes each, this corresponds to a combined fan-out of four. In addition, p maintains a message buffer. Process p performs the following operations in each round:

- *Pull-request* – p sends a digest of the messages it has received to the processes in its $view_{pull}$, requesting missing messages. Pull-request messages are sent to a well-known port. The pull-request specifies a randomly selected port on which p will await responses, and p spawns a thread for listening on the chosen port. This thread is terminated after a few rounds.

- *Pull-reply* – in response to pull-request messages arriving on the well-known port, p randomly selects messages that it has and are missing from the received digests, and sends them to the destinations indicated in the requests.

- *Push* – in a traditional push operation, p randomly picks messages from its buffer, and sends them to each target t in its $view_{push}$. In order to avoid wasting bandwidth on messages that t already has, p instead requests t to reply with a message digest, as follows:

  1. p sends a *push-offer* to t, along with a random port on which it waits for a push-reply.
  2. t replies with a *push-reply* to p's random port, containing a digest of the messages t has, and a random port on which t waits for data messages.
  3. If p has messages that are missing from the digest, it chooses a random subset of these, and sends them back to t's randomly chosen port.

  The target process listens on a well-known port for push-offers.

The random ports transmitted during the push and pull operations are encrypted (e.g., using the recipient's public key), in order to prevent an adversary from discovering them. Thus, $|view_{push}| + |view_{pull}|$ encryptions are performed each time these ports are changed.

Upon receiving a new data message, either by push or in response to a pull-request, p first performs some sanity checks. If the message passes these checks, p delivers it to the application and saves it in its message buffer for a number of rounds.

Resource allocation and bounds. In each round, p sends push-offers to all the processes in its $view_{push}$ and pull-requests to all the processes in its $view_{pull}$. If the total number of push-replies and pull-requests that arrive in a round exceeds p's sending capacity, then p equally divides its capacity between sending responses to push-replies and to pull-requests. Likewise, p responds to a bounded number (typically $|view_{push}|$) of push-offers in a round, and if more data messages than it can handle arrive, then p divides its capability for processing incoming data messages equally between messages arriving in response to pull-requests and those arriving in response to push-replies.

At the end of each round, p discards all unread messages from its incoming message buffers. This is important, especially in the presence of DoS attacks, as an attacker can send more messages than p can handle in a round. Since rounds are locally controlled and randomly vary in duration, the attacker cannot "aim" its messages for the beginning of a round. Thus, a bogus message has an equal likelihood of being discarded at the end of the round as an authentic messages does.

Achieving DoS-resistance. We now explain how the combination of push, pull, random port selections, and resource bounds achieves resistance to targeted DoS attacks. A DoS attack can flood a port with fabricated messages. Since the number of messages accepted on each port in a round is bounded, the probability of successfully receiving a given valid message M in a given round is inversely proportional to the total number of messages arriving on the same port as M in that round. Thanks to the separate resource bounds, an attack on one port does not reduce the probability for receiving valid messages on other ports.

In order to prevent a process from *sending* its messages using a *push* operation, one must attack (flood) the push-offer targets, the ports where push-replies are awaited, or the ports where data messages are awaited. However, the push destinations are randomly chosen in each round, and the push-reply and data ports are randomly chosen and encrypted. Thus, the attacker has no way of predicting these choices.

Similarly, in order to prevent a process from *receiving* messages during a *pull* operation, one needs to target the destination of the pull-requests or the ports on which pull-replies arrive. However, the destinations and ports are randomly chosen and the ports are sent encrypted. Thus, using the push operation, Drum achieves resilience to targeted attacks aimed at preventing a process from *sending* messages, and using the pull operation, it withstands attacks that try to prevent a process from *receiving* messages.

5. Evaluation Methodology

The most important contribution of this paper is our thorough evaluation of the impact of various DoS attacks on gossip-based multicast protocols. We evaluate three protocols: (i) Drum, (ii) *Push*, which uses only push operations, and (iii) *Pull*, which uses only pull operations. Pull and Push are implemented the same way Drum is, with the important measures of bounding the number of messages accepted in each round and using random ports. Thus, in comparing the three protocols, we study the effectiveness of combining push and pull operations under the assumption that these other measures are used.

We begin by evaluating the effect that a range of DoS attacks have on message latency using asymptotic mathematical analysis (in Section 6) and simulations (in Section 7). Our simulation results exhibit the trends predicted by the analysis. In the full paper [1], we also present detailed mathematical analysis, with results virtually identical to our simulations.

For these evaluations, we make some simplifying assumptions: We consider the propagation of a single message M, and assume that M is never purged from any process's message buffer. We model the push operation as performed without push-offers (in Drum and in Push). We assume that the rounds are synchronized, and that the message-delivery latency is smaller than half the gossip period; thus, a process that sends a pull-request receives the pull-reply in the same round. All of these assumptions were made in previous analyses of gossip-based protocols, e.g., [2, 8, 13, 16].

The analysis and simulations measure latency in terms of gossip rounds: we measure M's *propagation time*, which is the expected number of rounds it takes a given protocol to propagate *M* to all (in the closed-form analysis) or to 99% (in the simulations) of the correct processes. We chose a threshold of 99% since *M* may fail to reach some of the correct processes. Note that correct processes can be either attacked or non-attacked. In both cases, they should be able to send and receive messages.

Finally, we turn to measure actual performance on a cluster of workstations (in Section 8), and measure the consequences of DoS attacks not only on actual latency (in msecs.), but also on the throughput of a real system, where multiple messages are sent, and old messages are purged from processes' message buffers.

Attacks. In all of our evaluations, we stage various DoS attacks. In each attack, the adversary focuses on a fraction α of the processes ($0 < \alpha \leq 1$), and sends each of them x fabricated messages per round (in Drum, this means $\frac{x}{2}$ push messages and $\frac{x}{2}$ pull-requests). We denote the total attack strength by $B = x \cdot \alpha \cdot n$. We assume that the message source is being attacked (this has no impact on the results of Push). We consider attacks either of a *fixed strength*, where B is fixed and α increases (thus, x decreases); or of *increasing strength*, where either x is fixed and α increases, or vice versa (in both cases, B increases). Examining fixed strength attacks allows us to identify protocol vulnerabilities, e.g., whether an adversary can benefit from targeting a subset of the processes. Increasing strength attacks enable us to assess the protocols' performance degradation due to an increasing attack intensity.

6. Asymptotic Closed-Form Analysis

To simplify the analysis, we assume that all the processes are correct and the DoS attack is launched from outside the system. The protocols use a constant fan-out, F. Every round, each process sends a data message to F processes and accepts data messages from at most F processes. In Drum, F is equally divided between push and pull, e.g., if $F = 4$, then $view_{push} = view_{pull} = 2$, and each process accepts push messages from at most 2 processes and pull-request messages from at most 2 processes in a round.

We denote by p_u the probability of a non-attacked process to accept a valid incoming push or pull-request message sent to it. Similarly, we denote by p_a the probability of an attacked process to accept a valid incoming message. Obviously, p_u is independent of the attack strength. In the full paper [1], we give detailed formulas for p_a and p_u, and show that $p_u > 0.6$ for all $F \geq 3$. Since an attacked process is sent at least x messages in a round, and accepts at most F of them, we get the following coarse bound: $p_a < \frac{F}{x}$.

6.1. Drum

We define the *effective expected fan-in*, *I*, to be the average number of valid data messages a process successfully receives in a round. (If the same data message is received from k processes, we count this as k messages.) Likewise, the *effective expected fan-out*, *O*, is the average number of messages that a process sends and are successfully received by their targets in a round.

Let us examine the effect of a DoS attack on O and I, with respect to the push operation (O_{push} and I_{push}, resp.). The probability of an attacked process to receive a push message is p_a. The probability of a non-attacked process to receive a push message is p_u. Therefore, the effective fan-ins I^a_{push} and I^u_{push} of an attacked and non-attacked process (resp.) are:

$$I^a_{push} = F \cdot p_a \quad \text{and} \quad I^u_{push} = F \cdot p_u \tag{1}$$

When αn processes are attacked, the effective fan-outs are:

$$O^a_{push} = O^u_{push} = F \cdot (\alpha \cdot p_a + (1-\alpha) \cdot p_u) \tag{2}$$

A similar analysis for the pull operation yields the following effective fan-ins and fan-outs:

$$I^a_{pull} = \quad I^u_{pull} = \quad F \cdot (\alpha \cdot p_a + (1-\alpha) \cdot p_u) \tag{3}$$

$$O^a_{pull} = F \cdot p_a \quad \text{and} \quad O^u_{pull} = F \cdot p_u \tag{4}$$

In Drum, $O = \frac{1}{2}(O_{push} + O_{pull})$ and $I = \frac{1}{2}(I_{push} + I_{pull})$. Therefore:

$$O^a = I^a = \tfrac{F}{2} \cdot (\alpha \cdot p_a + (1-\alpha)p_u + p_a) = F \cdot (\tfrac{\alpha+1}{2} \cdot p_a + \tfrac{1-\alpha}{2} \cdot p_u) \tag{5}$$

$$O^u = I^u = \tfrac{F}{2} \cdot (\alpha \cdot p_a + (1-\alpha)p_u + p_u) = F \cdot (\tfrac{\alpha}{2} \cdot p_a + \tfrac{2-\alpha}{2} \cdot p_u) \tag{6}$$

Lemma 1. *Fix α and n. Drum's expected propagation time is bounded from above by a constant independent of x.*

Proof. From Equations (5) and (6) we get that for all x, $O^a = I^a > \frac{1-\alpha}{2} \cdot Fp_u$, and $O^u = I^u > \frac{2-\alpha}{2} \cdot Fp_u$. Since p_u is independent of x, the effective fan-ins and fan-outs of *all* the processes are bounded from below by a constant independent of x. Therefore, the propagation time is inevitably bounded from above by a constant independent of x. □

Figure 1(a) in Section 7.1 illustrates this quality of Drum.

We now consider attacks where the adversary has a fixed attacking power. We denote by $c = \frac{B}{F \cdot n}$ the attack strength divided by the total system capacity.

Lemma 2. *For $c > 5$, Drum's expected propagation time is monotonically increasing with α.*

Proof. We will show that all the processes' effective fan-ins and fan-outs are monotonically decreasing with α. That is, we want to prove that: $\frac{\mathrm{d}O^a}{\mathrm{d}\alpha} < 0$ and $\frac{\mathrm{d}O^u}{\mathrm{d}\alpha} < 0$. We require the following:

$$\frac{\mathrm{d}O^a}{\mathrm{d}\alpha} = \frac{F}{2} \cdot \left(p_a + \alpha\frac{\mathrm{d}p_a}{\mathrm{d}\alpha} + \frac{\mathrm{d}p_a}{\mathrm{d}\alpha} - p_u\right) < 0$$

$$p_a + (\alpha+1)\frac{\mathrm{d}p_a}{\mathrm{d}\alpha} < p_u$$

Recall that $p_a < \frac{F}{x}$. In the full paper [1] we show that $\frac{\mathrm{d}p_a}{\mathrm{d}\alpha} < \frac{F}{\alpha x}$. Bounding the left side of the inequality, we get:

$$p_a + (\alpha+1)\frac{\mathrm{d}p_a}{\mathrm{d}\alpha} < \frac{F}{x} + (\alpha+1)\frac{F}{\alpha x} = \frac{F}{\alpha x} \cdot (\alpha + \alpha + 1) = \frac{2\alpha+1}{c} < \frac{3}{c}$$

Thus, our condition holds when $\frac{3}{c} < p_u$, that is, when $c > \frac{3}{p_u}$. Similarly, for the second derivative we get the condition:

$$\frac{\mathrm{d}O^u}{\mathrm{d}\alpha} = \frac{F}{2} \cdot \left(p_a + \alpha\frac{\mathrm{d}p_a}{\mathrm{d}\alpha} - p_u\right) < 0$$

$$p_a + \alpha\frac{\mathrm{d}p_a}{\mathrm{d}\alpha} < p_u$$

Bounding the left side of the inequality, we get:

$$p_a + \alpha\frac{\mathrm{d}p_a}{\mathrm{d}\alpha} < \frac{F}{x} + \alpha\frac{F}{\alpha x} = \frac{F}{\alpha x} \cdot (\alpha + \alpha) = \frac{2\alpha}{c} < \frac{2}{c}$$

Thus, we require that $\frac{2}{c} < p_u$, or that $c > \frac{2}{p_u}$. This is already inferred from our previous result. The lemma follows since $p_u > 0.6$. □

This behavior is validated in the simulations. Moreover, the simulations show that even for smaller values of c (e.g., 2), Drum's propagation time increases with α (see Figure 3(a)).

6.2. Push

We first prove the following simple lemma.

Lemma 3. $\forall a > 0 \quad a < \frac{1}{\ln(1+\frac{1}{a})} < a + 1$.

Proof. We show that $\forall y > 0 \;\; \frac{1}{y} < \frac{1}{\ln(1+y)} < \frac{1}{y} + 1$. Define $h(y) = \ln(1+y) - \frac{y}{1+y}$ and $g(y) = \ln(1+y) - y$. By taking derivatives we get:

$$h'(y) = \frac{1}{1+y} - (\frac{1}{1+y} - \frac{y}{(y+1)^2}) = \frac{y}{(y+1)^2} > 0, \;\; \forall y > 0,$$

$$g'(y) = \frac{1}{1+y} - 1 < 0, \;\; \forall y > 0.$$

Since $h(0) = g(0) = 0$, $y > \ln(1+y) > \frac{y}{(y+1)}$. Therefore, $\frac{1}{y} < \frac{1}{\ln(1+y)} < \frac{1}{y} + 1$. □

We proceed to show that Push's propagation time is linear in x.

Lemma 4. *The expected propagation time to all processes in Push is bounded from below by:*

$$\frac{\ln n - \ln\left[(1-\alpha)\,n + 1\right]}{\ln\left(1 + F\alpha p_a\right)}$$

Proof. We prove that the given bound holds even for the case where initially all the non-attacked processes have *M*, in addition to the source (which is attacked). The lemma then follows immediately.

Let $M(k)$ denote the expected number of processes that have *M* at the beginning of round k. In round k, each process having M sends it to F other processes. On average, $F\alpha$ of those are attacked, and each attacked process receives the message with probability p_a. Thus, we get the coarse recursive bound $M(k+1) \leq M(k) + M(k) \cdot F\alpha p_a$ with the initial condition $M(0) = (1-\alpha)n + 1$. Thus, $M(k) \leq \left[(1-\alpha)\,n + 1\right](1 + F\alpha p_a)^k$. *M* reaches all the processes when $M(k) \geq n$. The first round number k that satisfies this inequality is the required formula. □

Corollary 1. *Fix α and $n > \frac{1}{\alpha}$. The propagation time of Push increases at least linearly with x.*

Proof. Since α and $n > \frac{1}{\alpha}$ are fixed, the numerator in Lemma 4 is a positive constant. Consider the denominator: since $p_a < \frac{F}{x}$, it holds that $F \cdot \alpha \cdot p_a = O(\frac{1}{x})$. The lemma follows since, by Lemma 3, $\frac{1}{\ln(1+\frac{1}{x})} = \theta(x)$. □

The above corollary explains the trend exhibited by Push in Figure 1(a).

6.3. Pull

Lemma 5. *Fix α and n. The propagation time of Pull grows at least linearly with x.*

Proof. Let Y be the number of correct processes that choose to send a pull-request to the source in a round, then Y is binomially distributed with $\mu = F$. Applying a Chernoff bound for $F \geq 4$, we get that the probability that at most $3F$ other processes choose the source in a round is greater than 0.994. Let $\widetilde{p}$ denote the probability of propagating a message beyond the source in a round. We give a gross over-estimate of $\widetilde{p}$ by assuming that exactly $3F$ other processes choose the source every round. (When fewer processes choose the source, *M* is *less* likely to leave the source.) Since $p_a < \frac{F}{x}$, $\widetilde{p} < (1 - (\frac{x-F}{x})^{3F})$. The number of rounds it takes to propagate a message beyond the message source is geometrically distributed with $\widetilde{p}$. Therefore, its expectation is $\frac{1}{\widetilde{p}} > \frac{x^{3F}}{x^{3F} - (x-F)^{3F}}$. In the full paper [1] we show that $\frac{1}{\widetilde{p}} = \Omega(x)$. □

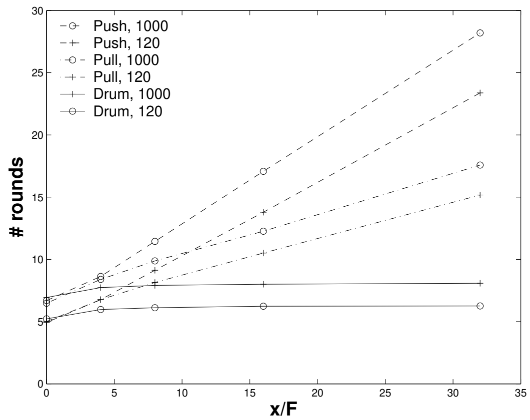
Figure 1(a) illustrates this behavior of Pull.

7. Simulation Results

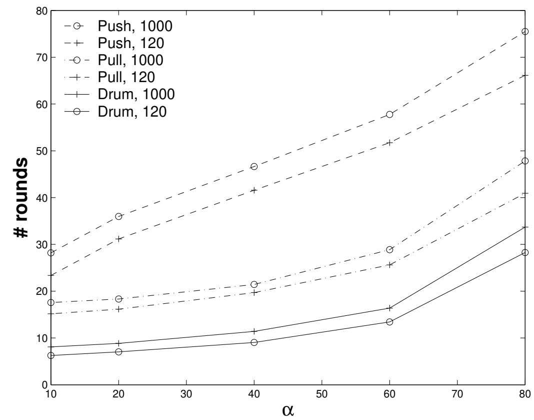
This section presents MATLAB simulations of the three protocols under various DoS attack scenarios. We consider a loss rate of 0.01 on all links and a fan-out of $F = 4$. We assume that 10% of the processes are controlled by the adversary and they do not propagate any valid messages. We note that, according to our model, malicious group members performing a DoS attack are equivalent to group members suffering crash failures, and an externally-sourced DoS attack of the same strength. In the full paper [1] we evaluate the protocols without DoS attacks, and show that they are highly robust to crash failures (cf. [9, 11]). Thus, controlling more group members does not grant the adversary with a significant advantage. We measure the propagation times to the correct processes, both attacked and non-attacked. Each data point is averaged over 1000 runs.

7.1. Targeted DoS Attacks

Figure 1 compares the time it takes *M* to reach 99% of the correct processes for the three protocols under various DoS attacks, with 120 and 1000 processes. Figure 1(a) shows that when 10% of the processes are attacked, the propagation time of both Push and Pull increases linearly with the severity of the attack, while Drum's propagation time is unaffected by the attack strength. This is consistent with the prediction of Lemmas 1 and 5 and Corollary 1. Moreover, the three protocols perform virtually the same without DoS attacks (see the leftmost data point). Figure 1(b) illustrates the propagation time as the percentage of attacked processes (and thus B) increases. Although the protocols exhibit similar trends, Drum propagates messages faster than Push and Pull.

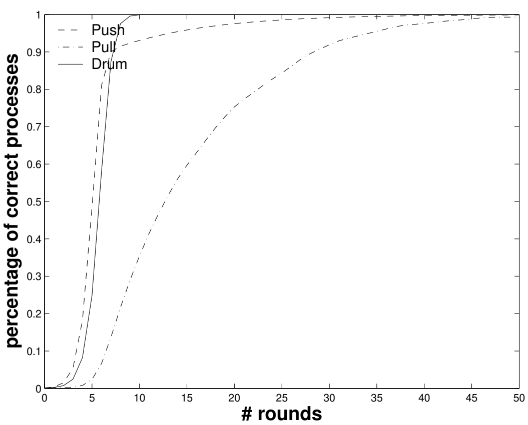


(a) $\alpha = 10\%$.

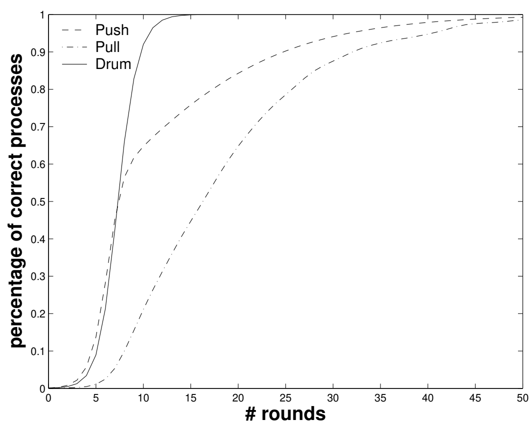


(b) $x = 32F$.

Figure 1. Average propagation time to 99% of the correct processes, $n = 120, 1000$.



(a) $\alpha = 10\%$, $x = 32F$.

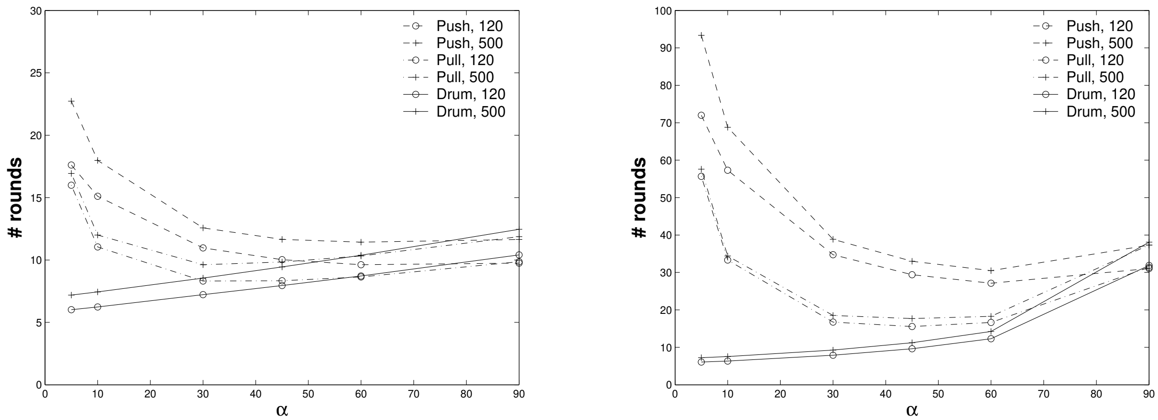


(b) $\alpha = 40\%$, $x = 32F$.

Figure 2. CDF: Average percentage of correct processes that receive *M*, $n = 1000$.

Figure 2 illustrates the cumulative distribution function (CDF) of the percentage of correct processes that receive *M* by a given round, under different DoS attacks. As expected, Push propagates *M* to the non-attacked processes very quickly, but takes much longer to propagate it to the attacked processes. Again, we see that Drum significantly outperforms both Push and Pull when a strict subset of the system is attacked.

Interestingly, on average, Push propagates *M* to more processes per round than Pull does (see Figure 2), although the average number of rounds Pull takes to propagate M to 99% of the correct processes is smaller than that of Push (see Figure 1). This paradox occurs since, with Pull, there is a non-negligible probability that *M* is delayed at the source for a long time. In the full paper [1] we compute that with $F = 4$ and $\frac{x}{F} = 32$, the probability for *M* not being propagated beyond the source in 5, 10, and 15 rounds is 0.54, 0.3, and 0.16 respectively. Once M reaches one non-attacked process, it quickly propagates to the rest of the processes. Therefore, even if by a certain round k, in most runs, a large percentage of the processes have M, there is still a non-negligible number of runs in which Pull does not reach *any* process (other than the source) by round k. This large difference in the percentage of processes reached has a large impact on the average depicted in Figure 2. In contrast, Push, which reaches all the non-attacked processes quickly in all runs, does not have runs with such low percentages factoring into this average. Nevertheless, Push's average propagation time to 99% of the correct processes is much higher than Pull's, because Push has to propagate M to *all* the attacked processes, whereas Pull has to propagate M only out of one attacked process.



(a) $B = 7.2n$ ($c = 2$).

(b) $B = 36n$ ($c = 10$).

Figure 3. Average propagation time to 99% of the correct processes.

7.2. Adversary Strategies

We now evaluate the protocols under a range of attacks with fixed adversary strengths. First, we consider severe attack with $B = 7.2n$ and $B = 36n$ (corresponding to $c = 2$ and $c = 10$, resp.) fabricated messages per round. If the adversary chooses to attack all correct processes, it can send 8 (resp., 40) fabricated messages to each of them in each round, because 90% of the processes are correct. If the adversary instead focuses on 10% of the processes, it can send 72 (resp., 360) fabricated messages per round to each. Figure 3 illustrates the protocols' propagation times with different percentages of attacked processes, for system sizes of 120 and 500. It validates the prediction of Lemma 2, and shows that the most damaging adversary strategy against Drum is to attack all the correct processes. That is, an adversary cannot "benefit" from focusing its capacity on a small subset of the processes. In contrast, the performance of Push and Pull is seriously hampered when a small subset of the processes is targeted. Not surprisingly, the three protocols perform equally when all correct processes are targeted (see the rightmost data point).

8. Implementation and Measurements

We have implemented Drum, Push, and Pull in Java. The implementations are multithreaded. The operations that occur in a round are not synchronized, e.g., one process might send messages before trying to receive messages in that round, while another might first receive a new message, and then propagate it. We run our experiments on 50 machines at the Emulab testbed [21], on a 100Mbit LAN, where a single process is run on each machine (i.e., $n = 50$). We designate 10% of the processes as malicious – they do not propagate any messages, and instead perform DoS attacks on other processes.

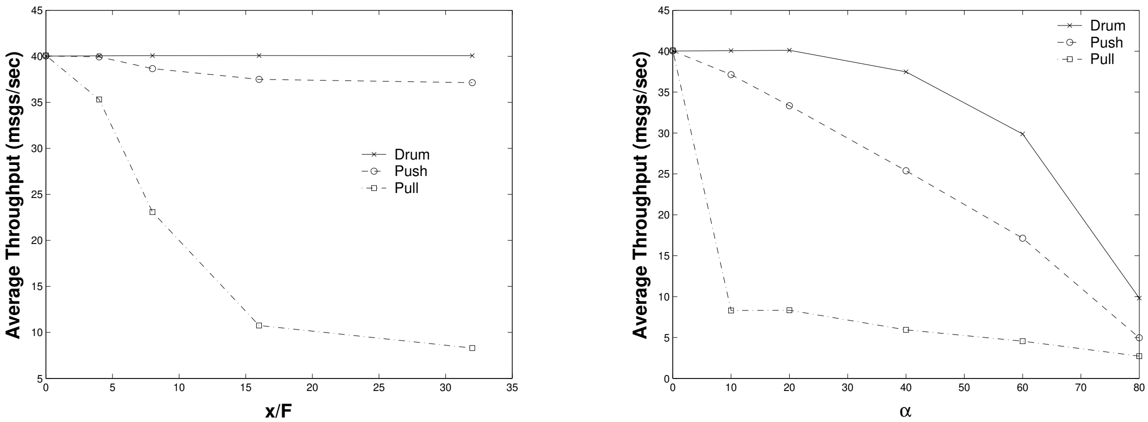
Our first goal for these experiments is to validate the simulation methodology. To this end, we experiment with the same settings that were tested in Section 7. The results are virtually identical to the simulation results, and can be found in the full paper [1].

We proceed to evaluate the protocols in a realistic setting, where multiple messages are sent. By running on a real network, we can faithfully evaluate latency in milliseconds (instead of rounds), as well as throughput.

In each experiment scenario, a total of $10,000$ messages are sent by a single source, at a rate of 40 messages per second. The average received throughput and latency are measured at the remaining 44 correct processes (recall that 5 of the 50 processes are faulty.) The average throughput is calculated ignoring the first and last 5% of the time of each experiment. The round duration is 1 second. Data messages are 50 bytes long (The evaluation of [8] used a similar transmission rate and similar message sizes.)

In a practical system, messages cannot reside in local buffers forever, nor can a process send all the messages it ever received in a single round. In our experiments, messages are purged from processes' buffers after 10 rounds, and each process sends at most 80 messages to each of its gossip partners in a round. These are roughly twice the buffer size and sending rate required for the throughput of 40 messages per round in an ideal attack-free setting, since the propagation time in the absence of attacks is about 5 rounds. Due to purging, some messages may fail to reach all the processes. Since we measure throughput at the receiving end, this is reflected by an average throughput lower than the transmission rate (of 40 messages per second).

Figure 4 shows the throughput at the receiving processes for Drum, Push, and Pull, under the DoS attack scenarios staged in the validation above. Figure 4(a) indicates



(a) $\alpha = 10\%$. (b) $x = 32F$.

Figure 4. Average received throughput.

that, as for latency, Drum's throughput is also unaffected by increasing x, while Push shows a slight degradation of throughput, and Pull's throughput decreases dramatically. Figure 4(b) shows that Drum's throughput gracefully degrades as α increases, while Push exhibits a linear degradation, and Pull's throughput is drastically affected for every $\alpha > 0$.

Figure 5 depicts the CDF of the average latency of *successfully received* messages in two scenarios. Each data point shows, for a given latency l, the percentage of correct processes for which the average latency does not exceed l. We observe that Push is the fastest in delivering messages to non-attacked processes, but suffers from substantial variation in delivery latency, as messages take a long time to reach the attacked processes. E.g., Figure 5(a) shows that the 4 attacked processes (other than the source) measure an average latency 4 times longer than non-attacked processes. While Pull exhibits almost the same average latency for all the processes, this latency is very long. Drum combines the best of Push and Pull: it delivers messages almost as fast as Push, while maintaining a small variation between attacked and non-attacked processes.

9. Conclusions

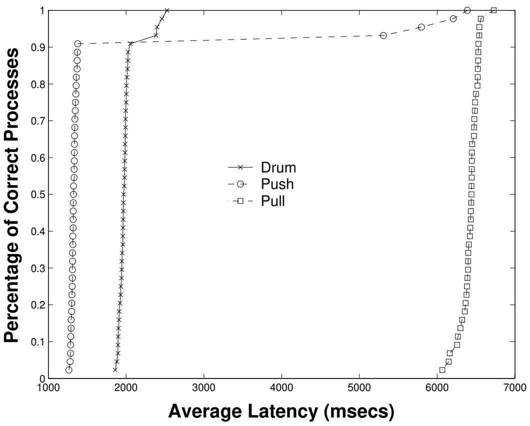
We have conducted the first systematic study of the impact of DoS attacks on multicast protocols, using asymptotic analysis, simulations, and measurements. Our study has exposed weaknesses of traditional gossip-based multicast protocols: Although such protocols are very robust in the face of process crashes, we have shown that they can be extremely vulnerable to DoS attacks. In particular, an attacker with limited attack strength can cause severe performance degradation by focusing on a small subset of the processes.

We have suggested a few simple measures that one can take in order to improve a system's resilience to DoS attacks: (i) combining pull and push operations; (ii) bounding resources separately for each operation; and (iii) random port selection. We have presented Drum, a simple gossip-based multicast protocol that uses these measures in order to eliminate vulnerabilities to DoS attacks. Our closed-form mathematical analysis, simulations, and empirical tests have proven that using both push and pull operations goes a long way in fortifying a system against DoS attacks. We have shown that, as the attack strength increases asymptotically, the most effective attack against Drum is one that targets all the correct processes in the system. As expected, the inevitable performance degradation due to such a broad attack is identical for all the studied protocols. However, protocols that use only pull or only push operations perform much worse under more focused attacks, which have little influence on Drum.

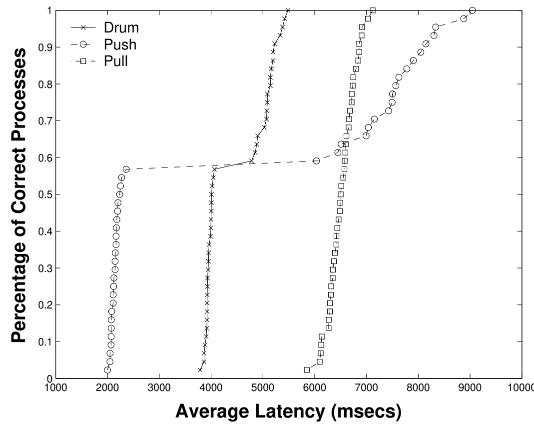
We expect our proposed methods for mitigating the effect of DoS attacks to be applicable to various other systems operating in different contexts. Specifically, the use of well-known ports should be minimized, and each process should be able to choose some of its communication partners by itself. Our analysis process and its corresponding metric can be used to generally quantify the effect of DoS attacks. We hope that other researchers will be able to apply similar techniques in order to quantitatively analyze their system's resilience to DoS attacks.

Acknowledgments

We thank Aran Bergman and Dahlia Malkhi for many helpful comments and suggestions. We are grateful to the Flux research group at the University of Utah, and especially Mac Newbold, for allowing us to use their network emulation testbed and assisting us with our experiments.



(a) $\alpha = 10\%, x = 32F$.



(b) $\alpha = 40\%, x = 32F$.

Figure 5. CDF: average latency of received messages.

References

[1] G. Badishi, I. Keidar, and A. Sasson. Exposing and eliminating vulnerabilities to denial of service attacks in secure gossip-based multicast. TR CCIT 477, Department of Electrical Engineering, Technion, March 2004. http://www.ee.technion.ac.il/∼badishi/papers/drum-tr.ps.

[2] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems (TOCS)*, 17(2):41–88, 1999.

[3] K. P. Birman, R. van Renesse, and W. Vogels. Spinglass: Secure and scalable communications tools for mission-critical computing. In *DARPA International Survivability Conference and Exposition (DISCEX)*, June 2001.

[4] R. K. C. Chang. Defending against flooding-based distributed denial-of-service attacks: A tutorial. *IEEE Communications Magazine*, 40:42–51, October 2002.

[5] Cisco Systems. Defining strategies to protect against TCP SYN denial of service attacks. http://www.cisco.com/warp/public/707/4.html.

[6] CSI/FBI. Computer crime and security survey, 2003. http://www.gocsi.com/forms/fbi/pdf.jhtml.

[7] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Stuygis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC*, pages 1–12, 1987.

[8] P. T. Eugster, R. Guerraoui, S. B. Handurukande, A. M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. In *DSN*, 2001.

[9] I. Gupta, R. van Renesse, and K. P. Birman. Scalable fault-tolerant aggregation in large process groups. In *DSN*, pages 433–442, 2001.

[10] R. M. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *IEEE Symposium on Foundations of Computer Science*, pages 565–574, 2000.

[11] M. J. Lin, K. Marzullo, and S. Masini. Gossip versus deterministically constrained flooding on small networks. In *DISC*, pages 253–267, 2000.

[12] P. Linga, I. Gupta, and K. Birman. A churn-resistant peer-to-peer web caching system. *ACM Workshop on Survivable and Self-Regenerative Systems*, October 2003.

[13] D. Malkhi, Y. Mansour, and M. K. Reiter. Diffusion without false rumors: On propagating updates in a Byzantine environment. *Theoretical Computer Science*, 299(1–3):289–306, April 2003.

[14] D. Malkhi, E. Pavlov, and Y. Sella. Optimal unconditional information diffusion. In *15th International Symposium on DIStributed Computing (DISC)*, 2001.

[15] D. Malkhi, M. K. Reiter, O. Rodeh, and Y. Sella. Efficient update diffusion in Byzantine environments. In *20th IEEE International Symposium on Reliable Distributed Systems (SRDS)*, October 2001.

[16] Y. M. Minsky and F. B. Schneider. Tolerating malicious gossip. *Distributed Computing*, 16(1):49–68, February 2003.

[17] 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.

[18] National Institute for Standards and Technology. Digital Signature Standard (DSS). *FIPS Publication 186-2*, October 2001. http://csrc.nist.gov/publications/fips/.

[19] B. Pittel. On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, February 1987.

[20] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on TCP. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 208–223, May 1997.

[21] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *OSDI*, pages 255–270, Boston, MA, Dec. 2002. USENIX Association.

[22] L. Zhou, F. B. Schneider, and R. van Renesse. COCA: A secure distributed online certification authority. *ACM Transactions on Computer Systems*, 20(4):329–368, 2002.