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

Gal Badishi, Idit Keidar, and Amir Sasson

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.

Index Terms

C.2.4.b Distributed applications, C.4.f Reliability, availability, and serviceability. D.1.8 Distributed programming.

I. 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 a *distributed denial of service* (DDoS) attack, the attacker utilizes multiple computers as the source of a DoS attack, in order to increase the attack strength. Since a DDoS attack is essentially a strong DoS attack, we will consider them to be the same. In 2003, approximately 42% of U.S. organizations, including government agencies, financial institutions, medical institutions and universities, were faced with DoS attacks [4]. That year, DoS attacks were the second most financially damaging attacks, only short of theft of proprietary information, and far above other attacks [4]. Therefore, coping with DoS attacks is essential when deploying services in a hostile environment such as the Internet [20].

As a first defense, one may protect a system against DoS attacks using network-level mechanisms [25], [22], [23]. These mechanisms involve rate-limiting incoming traffic, and filtering packets according to their headers. 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.

As network-level DoS-mitigation solutions are increasingly available, application level DoS attacks are becoming a major concern [29]. Consequently, vendors have begun employing some measures against DoS attacks at the application layer [10], [21]. Such solutions are commonly deployed at the network/firewall level, although they are application-specific. However, these measures are usually just hard-coded validity

In IEEE Transactions on Dependable and Secure Computing (TDSC), 3:1, March 2006. A preliminary version of this paper appeared in The IEEE International Conference on Dependable Systems and Networks (DSN) 2004.

Gal Badishi and Idit Keidar are with the Electrical Engineering Department, Technion – I.I.T.

Amir Sasson was with the Computer Science Department, Technion – I.I.T. when the work was conducted.

Gal Badishi is supported by the Israeli Ministry of Science.

checks for well-known protocols, and do not contain means to deal with resource exhaustion caused by the application. In this paper, we are concerned with coping with DoS attacks in application-level multicast protocols. The basic idea is to assume simple and general mechanisms at the network/firewall level and to exploit them at the application (multicast protocol) level.

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.

We focus on large-scale distributed systems (e.g., 1000 processes). 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 [5], [1], [6], [8], [12], [13], [11], 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], [14]. As in previous work, e.g., [1], [13], we assume that the gossip-based multicast system is deployed in a WAN environment, and as such, its nodes suffer from DoS attacks launched from outside the system. 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.

Having observed the vulnerabilities of traditional protocols, we turn to search for ways to eliminate these vulnerabilities. Specifically, our goal is to design a protocol that does 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 are familiar with only one previous work, by Minsky and Schneider [19], that addresses DoS attacks on a gossip-based protocol. However, the problem they consider differs from ours in a way that renders their approach inapplicable to our setting (see Section II), and moreover, they only deal with limited attack strengths.

We present *Drum* (DoS-Resistant Unforgeable Multicast), a gossip-based multicast protocol, which, using a few simple ideas, eliminates common vulnerabilities to DoS attacks: the best attack against Drum requires the attacker to target the entire system. The 3 main ideas used in Drum are:

1) Simultaneously using two gossiping techniques, *push* and *pull*.
2) Allocating separate resources for each operation.
3) Using random ports whenever possible, for each communication channel.

Mathematical analysis and simulations show that Drum indeed achieves our design goal: an attacker cannot substantially hinder Drum's performance by targeting a small subset of the processes. 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. (We concentrate on heavy attacks since they are the most damaging, and one can expect them to happen in actual scenarios [28].) Obviously, performance degradation due to a broad all-out DDoS attack is unavoidable for any multicast protocol, and indeed all the tested protocols exhibit the same performance degradation under such a broad attack. In contrast, under an attack that focuses on a strict subset of the processes, Drum's latency remains *constant* as the attack strength increases, whereas in traditional protocols, the latency grows *linearly* with the attack strength.

We have implemented Drum in Java and tested it 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 in terms of latency and throughput.

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 fault-tolerant protocols, showing that robustness, although necessary, is not sufficient for DoS-mitigation.
- 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 effects of DoS attacks on other protocols as well.
- It provides closed-form asymptotic analyses 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 II gives background and related work. Section III presents the system model. Section IV describes Drum. Section V 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 VI gives closed-form asymptotic latency bounds; Section VII provides a thorough evaluation using simulations; and Section VIII presents latency and throughput measurements. Section IX evaluates the usefulness of two specific DoS-mitigation techniques used in Drum. Section X concludes. The appendices contain some derivations for the analysis.

II. Background and Related Work

Gossip-based dissemination [5] is a leading approach in the design of scalable reliable application-level multicast protocols, e.g., [1], [6], [8], [12], [13], [11]. Our work focuses on symmetric gossip-based multicast protocols like lpbcast [6]. We consider protocols that do not rely on external mechanisms such as IP multicast.

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 tries to exchange 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 [24], [11]. There are two methods for information dissemination: (1) *push*, in which the process sends messages to randomly selected processes; and (2) *pull*, in which the process requests messages from randomly selected processes. We show that 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 [5], [11]. Karp et al. showed that combining push and pull allows the use of fewer transmissions to ensure data arrival to all group members [11].

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. Similar resource separation was also used in COCA [33], for the sake of overcoming DoS attacks on authentication servers. Drum further utilizes randomly selected ports for data transmission, thus making it difficult for an attacker to target these ports.

Secure gossip-based dissemination protocols were previously suggested by Malkhi et al. [16], [17], [18]. However, they did not deal with DoS attacks. Follow-up work by Minsky and Schneider [19] 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 that utilizes $t+1$ disjoint paths, as suggested in [19], does not help in withstanding DoS attacks in the multicast system we consider. Moreover, Minsky and Schneider [19] focus on load rather than on 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.

Drum deals with DoS attacks at the application-level, assuming network-level defenses are already in place. Network-level DoS analysis and mitigation has been extensively dealt with [27], [2], [7], [30], [3], [25], but DoS-resistance at the secure multicast service layer has gotten little attention. 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.

Here, we focus on DoS attacks in which the attacker sends fabricated application messages. DoS can also be caused by churn, where processes rapidly join and leave [15], thus reducing availability. In Drum, as in other gossip-based protocols, churn has little effect on availability: even when as many as half of the processes fail, such protocols can continue to deliver messages reliably and with good quality of service [14]. 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 [1], where it is shown that probabilistic protocols, e.g., gossip-based protocols, solve this problem.

III. System Model

Drum supports probabilistically reliable multicast [1], [6], [11] among processes that are members of a group. Each message is created by exactly one group member (its *source*). Throughout this paper we assume that the multicast group is static. There are n members in the group, and each process p has a list of the other $n-1$ group members.

Like previous gossip protocols [1], [6], we assume that the underlying network is fully-connected. The message latency varies, but it is bounded. The link-loss probability is constant, equal for all links, 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.

An adversary can generate fabricated messages. However, this requires the adversary to utilize resources. Malicious processes can perform DoS attacks on group members. We note that authenticating messages, e.g., using digital signatures, does not solve the DoS problem, as fabricated messages must be invalidated using a costly operation.

We assume that communication can take place on ports that change on demand, and that the multicast protocol can randomly choose to process a subset of the messages that arrive to a designated port, and ignore messages that arrive to other ports. We further assume that a DoS attack that does not specifically target the designated port does not affect the reception on this port (i.e., the application-level DoS attack does not cause a network-level DoS attack as well). This can be achieved using available network-level products [25], [22], [23].

We assume that a process can choose a random port for communication that the adversary cannot predict. We assume that the adversary only attacks ports it knows of. In our protocol, the use of a random port is limited in time, and the process notifies another process of this new communication port by sending it a message stating the port number. We assume that it takes the adversary considerable time to react to this message, so that it cannot attack this random port while it is still in use. This assumption is justified, since an attacker that has significant strength is probably employing a DDoS attack and needs to notify its subordinates whenever it wishes to change targets.

IV. 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. The rounds are not synchronized among the processes. 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 (group members), $view_{push}$ 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.

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. The sanity checks employ cryptographic mechanisms, which ensure that the attacker has negligible probability of fabricating a message that passes these checks. Consequently, bogus messages impact only their first recipient. However, the sanity checks are costly in terms of execution time (e.g., verifying digital signatures). Thus, performing sanity checks at a high rate effectively causes DoS.

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. The messages are randomly chosen from the incoming message buffers.

At the end of each round, p flushes 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.

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, as are the push-reply and data ports. 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. 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.

V. 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. In addition to examining the effect of DoS on Drum, we also measure the effectiveness of the DoS-mitigating techniques employed by it. We mostly concern ourselves with the benefits of combining both the push and pull methods. 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. Subsequently, Section IX evaluates the effectiveness of Drum's other DoS-mitigation concepts, by contrasting Drum's performance against that of two modified versions of Drum: one without resource separation, and a second without using random ports.

We begin by evaluating the effect that a range of DoS attacks have on message latency using asymptotic mathematical analysis (in Section VI) and simulations (in Section VII). Our simulation results exhibit the trends predicted by the analysis.

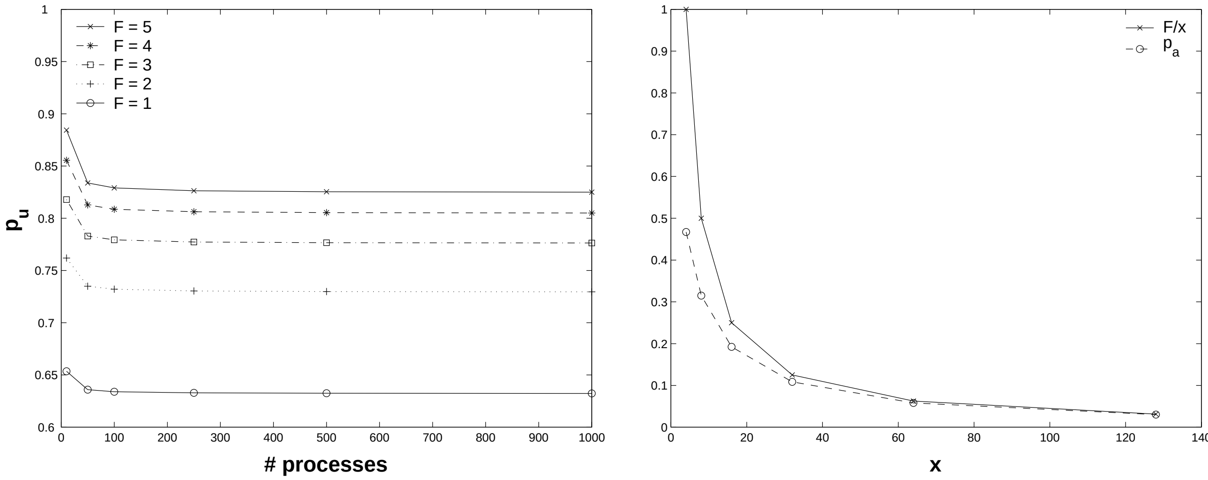
For these evaluations, we make some simplifying assumptions: We assume no message is ever purged from any process's message buffer, and that all processes have some messages in their buffers (from previous multicast sessions). We also assume that when processes send a data message, they send the complete contents of their buffer in a single operation. 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., [1], [6], [16], [19].

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

We turn to measure actual performance on a cluster of workstations (in Section VIII). Our goal for this evaluation is twofold: First, we wish to ensure that the simplifying assumptions made in the analysis and simulations have little impact on their results. E.g., in the implementation, rounds are not synchronized and the push-offer mechanism is used (in Drum and in Push). Second, we seek to 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. We assume that the DoS attacks are launched from outside the system. DoS from inside the group is essentially just one source (or more) generating excessive traffic. This can happen regardless of any malicious nodes being part of the multicast group, e.g., in a heterogenous system. Consequently, this is in fact a flow-control problem, as one cannot differentiate between a malicious attack and legitimate excessive traffic. Flow control in gossip-based multicast has been dealt with in [26].

In each DoS 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 note that randomly choosing the attack targets every round does not make any difference, as the communication partners are re-chosen uniformly at random each round. We denote the total attack strength by $B = x{\cdot}\alpha{\cdot}n$.



(a) p_u as a function of n, for various fan-outs.

(b) p_a vs. $\frac{F}{x}$ for $F = 4$, $n = 1000$.

Fig. 1. Actual values of p_u and p_a.

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.

VI. Asymptotic Closed-Form Analysis

In this section we assume that all the processes are correct. The protocols use a constant fan-out, F. Every round, each process sends messages to F processes and accepts 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 analyze Drum in Section VI-A, Push in Section VI-B, and Pull in Section VI-C.

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 Appendix I, we give detailed formulas for p_a and p_u, and Lemma 8 proves that $p_u > 0.6$ for all $F \geq 3$. Numerical calculations using the formula in Appendix I show that $p_u > 0.6$ for all $F \geq 1$, as can be seen in Figure 1(a). When at least one valid message is sent, an attacked process is sent at least $x + 1$ messages in a round, and accepts at most F of them. We get the following coarse bound: $p_a < \frac{F}{x}$. Figure 1(b) shows an example of the numerical calculation of p_a versus $\frac{F}{x}$.

A. Drum

We begin by considering increasing strength attacks. We show that in Drum, an adversary does not gain any significant advantage by increasing its attack strength while focusing on a fixed strict subset of the processes.

Lemma 1: Fix $\alpha < 1$ and n. Drum's expected propagation time is bounded from above by a constant independent of x.

Proof: Since $\alpha < 1$, some processes are not attacked at all. Let us look at a two-stage propagation scheme that works as follows: At the first stage, only the source propagates the message. The expected propagation time from the source via push to all the non-attacked processes is independent of x and bounded, since n is fixed. At the next stage, the non-attacked processes constitute non-attacked sources

for the rest of the group via pull. The expected propagation time of the second stage is again independent of x and bounded. Since n is fixed, this two-stage expected propagation time is constant. The two-stage propagation from the source to all of the destinations is obviously not faster than Drum's propagation. Thus, Drum's expected propagation time is bounded from above by a constant independent of x. ■
Figure 3(a) in Section VII-B illustrates this quality of Drum, using simulations.

We now consider attacks where the adversary has a fixed attacking power. Thus, the attacker can intensely attack a small group of processes, or perform a moderate attack on a large number of processes. We would like to see which strategy is more beneficial to the attacker. We denote by $c = \frac{B}{F \cdot n} = \frac{\alpha x}{F}$ the attack strength divided by the total system capacity. We show that the adversary's best strategy against Drum is to attack as many processes as it can, i.e., increase α.

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:

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

Similar arguments apply for the pull operation. The probability of an attacked process to receive a pull-request is p_a. The same probability for a non-attacked process is p_u. Receiving pull-requests allows a process to send data messages, and on average, each process receives F pull-requests. Due to the use of random ports, we assume that each pull-reply is actually being received, and thus, the effective fan-outs are:

$$O^a_{pull} = \quad F \cdot p_a \quad \text{and} \qquad O^u_{pull} = F \cdot p_u \tag{3}$$

Receiving data messages requires sending pull-requests. Each round, F pull-requests are being sent. On average, αF of them reach an attacked process and are successfully read with probability p_a, and $(1-\alpha)F$ of those reach a non-attacked process and are successfully read with probability p_u. Due to the use of random ports, we can assume it makes no difference whether the requesting process is attacked or not. We get the following fan-ins:

$$I^a_{pull} = \quad I^u_{pull} = \quad F \cdot (\alpha \cdot p_a + (1 - \alpha) \cdot p_u) \tag{4}$$

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

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

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

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:

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

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

Recall that $p_a < \frac{F}{x}$. In Lemma 7 in Appendix I 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, when applying the derivative to the second term we get the condition:

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

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 in Section VII-C. Moreover, the simulations show that even for much smaller values of c (ranging from 0.25 to 2), Drum's propagation time increases with α (see Figures 7–8).

B. 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 $\forall y > 0 \quad \frac{1}{y} < \frac{1}{\ln(1+y)} < \frac{1}{y} + 1$.

Define $h(y) = \ln(1+y) - \frac{y}{1+y}$ and $g(y) = \ln(1+y) - y$. By taking derivatives we get:

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

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 the message (denoted by *M*), in addition to the source (which is attacked). The lemma then follows immediately.

Let the random variable $M(k)$ denote the number of processes that have *M* at the beginning of round k, and let $E\left[M(k)\right]$ denote its expectation. 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 $E\left[M(k+1)\right] \le E\left[M(k)\right] + E\left[M(k)\right] \cdot F\alpha p_a$ with the initial condition $E\left[M(0)\right] = M(0) = (1-\alpha)n + 1$. Thus, $E\left[M(k)\right] \le \left[(1-\alpha)\,n+1\right]\left(1+F\alpha p_a\right)^k$. *M* reaches all the processes when $E\left[M(k)\right] \ge n$. To bound k from below we use the fact that having $\left[(1-\alpha)\,n+1\right]\left(1+F\alpha p_a\right)^k < n$ implies that $E\left[M(k)\right] < n$. Thus, the first round number k that may satisfy the inequality $E\left[M(k)\right] \ge n$ 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 $F \cdot \alpha \cdot p_a$ is $O(\frac{1}{x})$. The lemma follows since, by Lemma 3, $\frac{1}{\ln(1+\frac{1}{x})}$ is $\Theta(x)$. ■

The above corollary explains the trend exhibited by Push in Figure 3(a).

C. Pull

We begin by proving the following lemma.

Lemma 5: $\forall b \in \mathbb{N}$ $\frac{x^b}{x^b-(x-F)^b}$ is $\Omega(x)$.

Proof: We first show that $\frac{a-1}{b} \leq \frac{a^b}{a^b-(a-1)^b}$ for every $a > 1$, $b \in \mathbb{N}$.

We prove by induction on b that $\frac{b}{a-1} \geq \frac{a^b-(a-1)^b}{a^b}$. For $b = 1$, $\frac{1}{a-1} \geq \frac{1}{a}$ for every $a > 1$. The inductive Step: $\frac{a^{b+1}-(a-1)^{b+1}}{a^{b+1}} = \frac{a(a)^b-(a-1)(a-1)^b}{a(a)^b} = \frac{a^b}{a(a)^b} + \frac{a-1}{a}\frac{a^b-(a-1)^b}{a^b} \leq \frac{1}{a} + \frac{a-1}{a}\frac{b}{a-1} = \frac{1}{a} + \frac{b}{a} = \frac{b+1}{a} \leq \frac{b+1}{a-1}$.

By substituting $\frac{x}{F}$ for a in the proven inequality, we get that $\frac{x-F}{bF} \leq \frac{x^b}{x^b-(x-F)^b}$ for every $x > F$. Therefore, $\frac{x^b}{x^b-(x-F)^b}$ is $\Omega(x)$. ∎

We define $\widetilde{p}$ as probability that the message M is propagated from the source in a round.

Lemma 6: Fix α and n. The number of rounds it takes a message to leave the source in Pull grows at least linearly with x.

Proof: We give a gross over-estimate of $\widetilde{p}$ by assuming that all the other $n-1$ processes choose the source every round. (When fewer processes choose the source, M is *less* likely to leave the source.) Since $p_a < \frac{F}{x}$, $\widetilde{p} < (1 - (\frac{x-F}{x})^{n-1})$. The number of rounds it takes to propagate a message beyond the message source is geometrically distributed with $\widetilde{p}$. Therefore, its expectation is $\frac{1}{\widetilde{p}} > \frac{x^{n-1}}{x^{n-1}-(x-F)^{n-1}}$. Substituting $n-1$ for b in Lemma 5, we get that $\frac{1}{\widetilde{p}}$ is $\Omega(x)$. ∎

Corollary 2: Fix α and n. The propagation time of Pull grows at least linearly with x.

Figure 3(a) illustrates this behavior of Pull.

VII. Simulation Results

This section presents MATLAB simulations of the three protocols under various DoS attack scenarios. All group members constantly have messages to send, and we track the propagation of one of these messages, M, from its source. Each process receives messages from at most $F = 4$ other processes each round (disregarding pull-replies). If more than F processes try to access this process's incoming channels, a random F-sized subset of them is chosen. We consider a link-loss probability of 0.01 on all links and a fan-out of $F = 4$. Rounds are synchronized among all processes. Each data point is averaged over 1000 runs, where in each run the number of rounds it takes the message to reach 99% of the processes is measured.

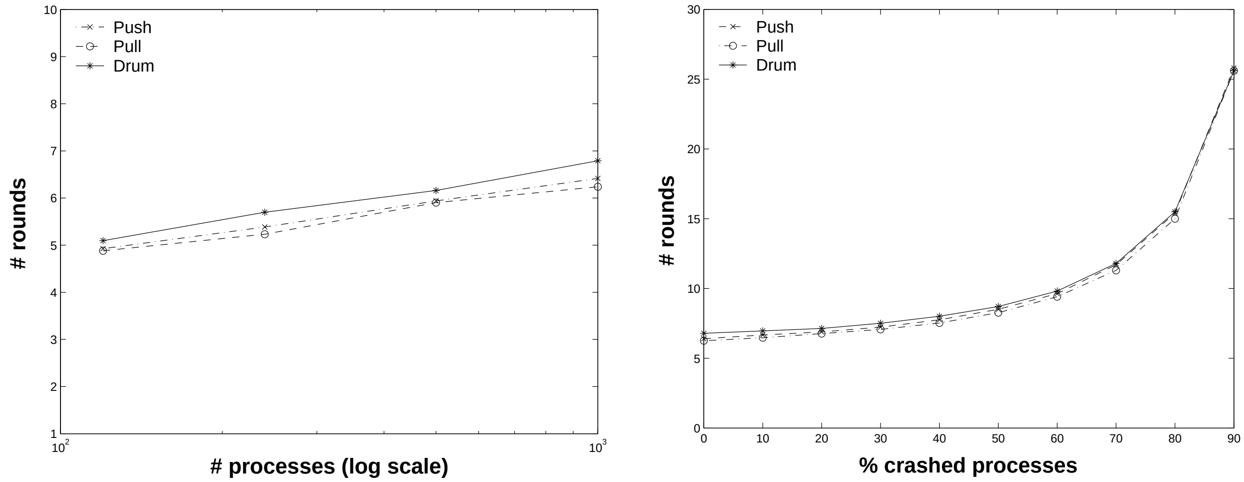
In Section VII-A we consider situations with no DoS attack (either no failures or only crash failures), and validate known results about gossip protocols. We continue in Sections VII-B and VII-C by measuring the effect of DoS attacks on the system. In these studies, we assume that 10% of the processes have crashed when the system started (we assume that no failure detectors are being used), and that the DoS attack is launched from outside the system. Since we do not assume that nodes can detect that their gossip partners are down, assuming that nodes crash right when the system starts has no special effect on the results. If nodes crash later on, the system will operate as usual until the processes crash. After that, the system will operate as analyzed with processes that have crashed right from the start.

We measure the propagation times to the correct processes, both attacked and non-attacked. In Section VII-B we measure the impact of targeted DoS attacks, and in Section VII-C we examine fixed strength attacks and adversary strategies.

A. Validating Known Results

We begin by evaluating the three protocols in a failure-free scenario, and in situations where crash failures occur. We assume that the crashes occur before M is generated, and that the source does not crash. We also assume that the crashes are not detected by the correct processes, i.e., they try to gossip with crashed processes as well.

Our aim is to validate two known results: (1) the propagation time of gossip-based multicast protocols is $O(\log n)$ [24], [11], as can be seen in Figure 2(a), with a logarithmic x-axis; and (2) the performance

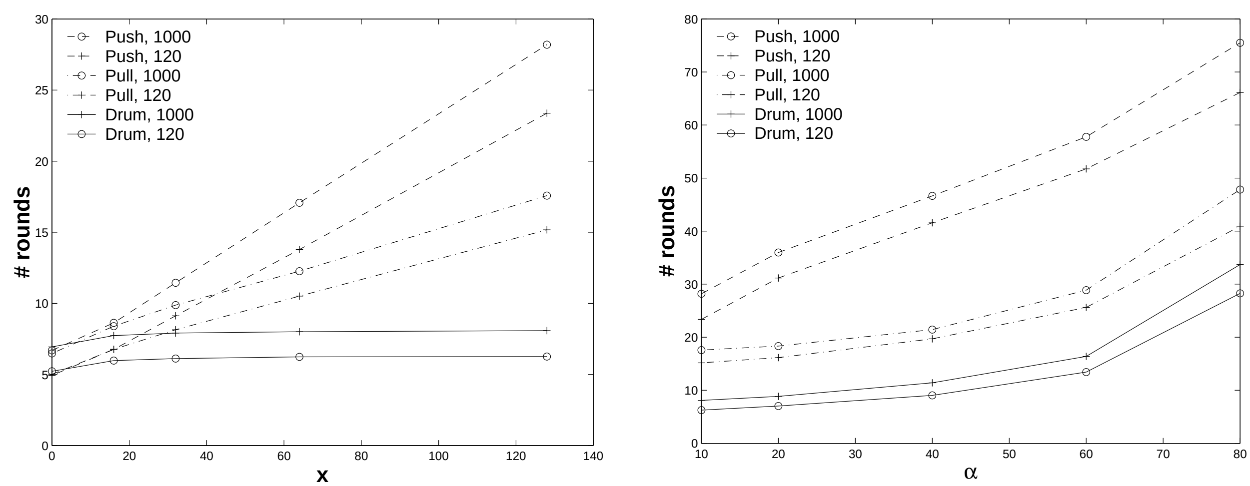


(a) Failure-free operation. (b) Operation with crashed processes, $n = 1000$.

Fig. 2. Runs without DoS attack: Average propagation time to 99% of the correct processes (simulations).

of such protocols degrades gracefully as crash failures amount [9], [14], as depicted in Figure 2(b)). We can see that Push and Pull slightly outperform Drum in these experiments. This is due to the fact that the bounds on the pull and push channels in Drum are strict, i.e., even if in a specific round no messages have arrived via the push channels, only requests from at most two distinct processes will be handled, although the process is capable of handling four such requests. Conversely, Push and Pull have only one bound, which guarantees that messages won't be discarded if they can be processed. The ability to perform well even when many processes crash stems from the random choice of communication partners each round.

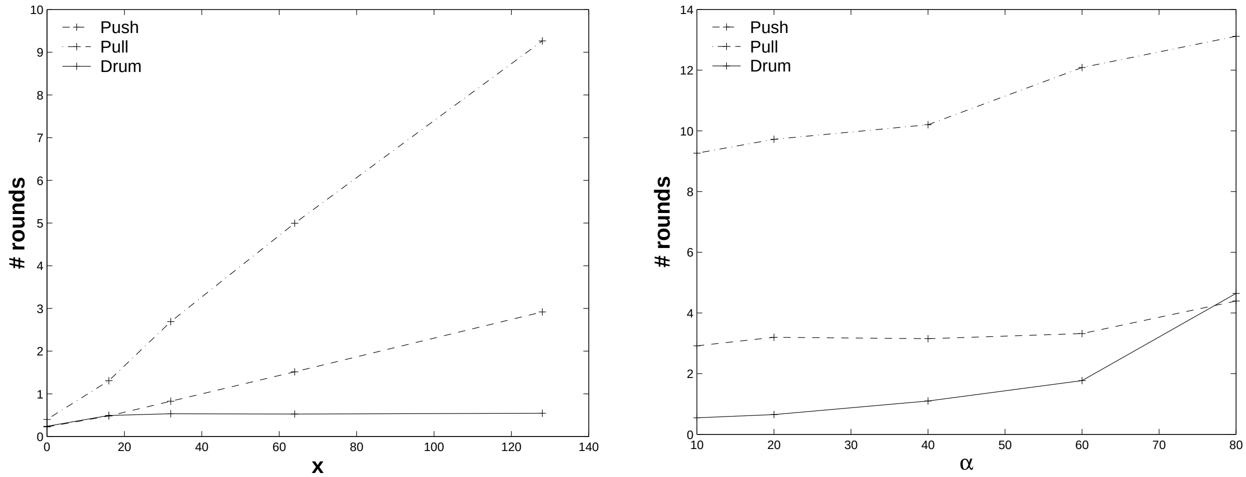
B. Targeted DoS Attacks



(a) $\alpha = 10\%$. (b) $x = 128$.

Fig. 3. Increasing attack strength: Average propagation time to 99% of the correct processes, $n = 120, 1000$ (simulations).

In this section we consider targeted attacks, where a subset of size αn of the processes is attacked. Figure 3 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 3(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 Lemma 1 and Corollaries 1 and 2. Moreover, the three protocols perform virtually the same without



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

(b) $x = 128$.

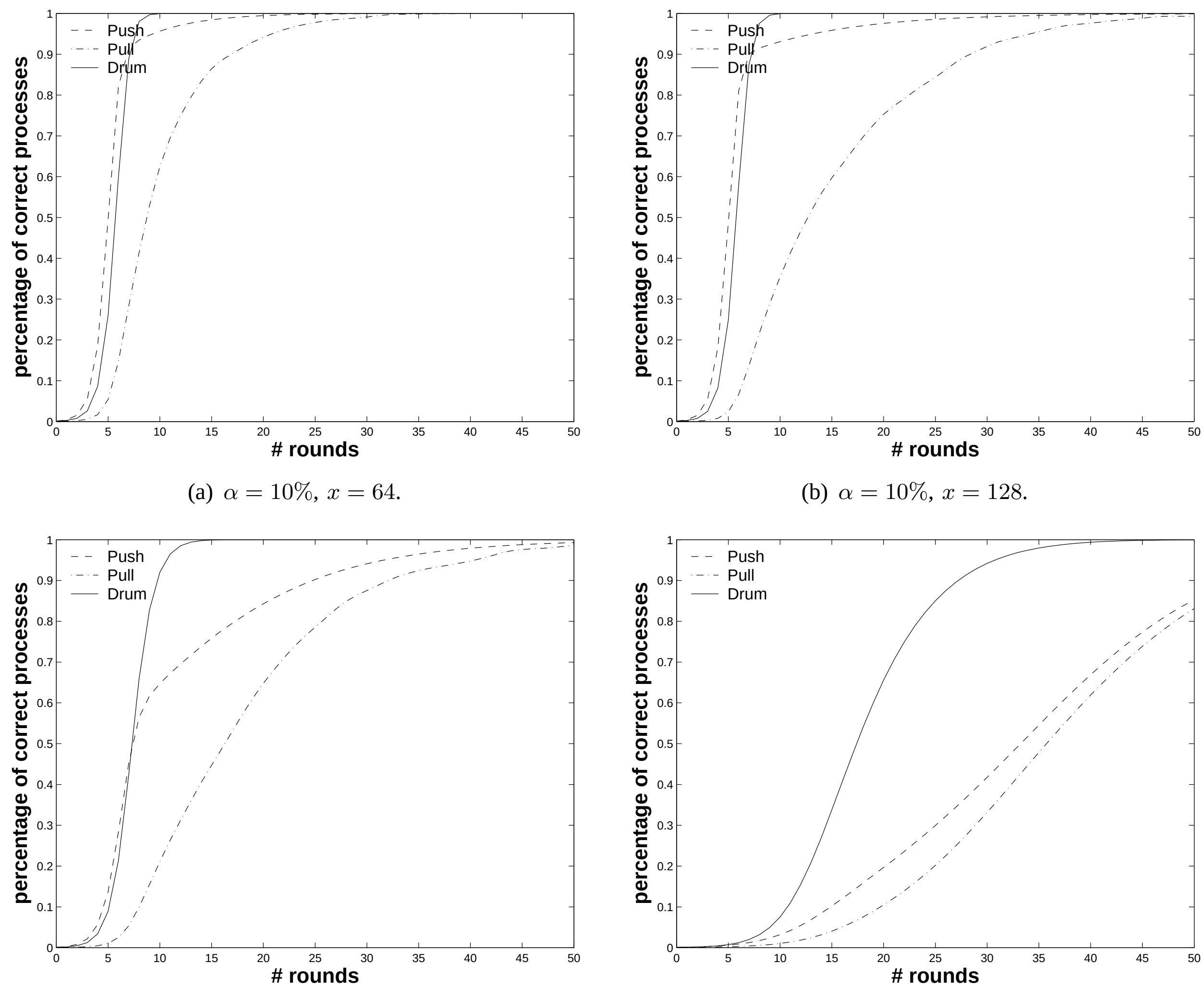
Fig. 4. Increasing attack strength: STD of the propagation time to 99% of the correct processes, $n = 1000$ (simulations).

DoS attacks (see the leftmost data point). Figure 3(b) illustrates the propagation time as the percentage of attacked processes (and thus B) increases. The rightmost data point in this figure matches a scenario where only 10% of the processes are both correct non-attacked. Although the protocols exhibit similar trends, Drum propagates messages much faster than Push and Pull.

Figure 4 illustrates the *standard deviation* (STD) of the propagation times presented in Figure 3 for $n = 1000$. It shows that for a fixed α, Drum's STD is not affected by the attack strength, whereas the other protocols' STD increases linearly. Furthermore, both Drum and Push exhibit a small STD compared to Pull. E.g., for $\alpha = 10\%$ and $x = 128$, the STDs of Drum and Push are 0.5 and 2.9 rounds (resp.), whereas Pull's STD is 9.3 rounds. Therefore, the behavior of Drum and Push is more predictable. The high STD of Pull's propagation time is mainly due to the large STD of the number of rounds it takes to propagate *M* beyond the source. The number of rounds it takes to propagate *M* beyond the source is geometrically distributed with $\widetilde{p}$, where $\widetilde{p}$ is the probability to propagate *M* beyond the source in a round. Thus, the STD of the number of rounds it takes to propagate *M* beyond the source is $\frac{\sqrt{1-\widetilde{p}}}{\widetilde{p}}$. A numerical calculation of $\widetilde{p}$ according to the formula in Appendix II, with $F = 4$ and $x = 128$, yields an STD of 8.17 rounds, which explains Pull's measured STD of 9.3 rounds mentioned above.

Figure 5 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 5), 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 3). This paradox occurs since, with Pull, there is a non-negligible probability that *M* is delayed at the source for a long time. With $F = 4$ and $x = 128$, the probability of *M* not being propagated beyond the source in 5, 10, and 15 rounds is 0.54, 0.3, and 0.16 resp. (as computed using the formula for $\widetilde{p}$ in Appendix II). 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 significant impact on the average depicted in Figure 5. 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



(a) $\alpha = 10\%$, $x = 64$.

(b) $\alpha = 10\%$, $x = 128$.

(c) $\alpha = 40\%$, $x = 128$.

(d) $\alpha = 80\%$, $x = 128$.

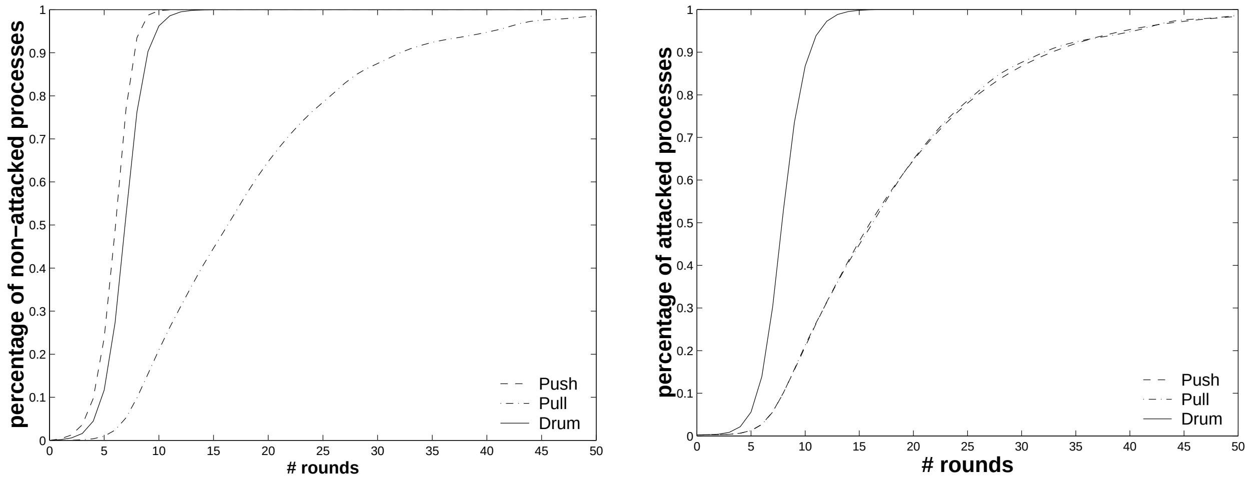
Fig. 5. Targeted DoS attacks: CDF: Average percentage of correct processes that receive *M*, $n = 1000$ (simulations).

M only out of one attacked process.

Figure 6 illustrates this behavior: Figure 6(a) shows that Push propagates M much faster than Pull to the non-attacked processes, while Figure 6(b) indicates that Push and Pull take the same time to propagate M to the attacked processes. Conversely, Drum exhibits fast propagation times both to attacked and non-attacked processes.

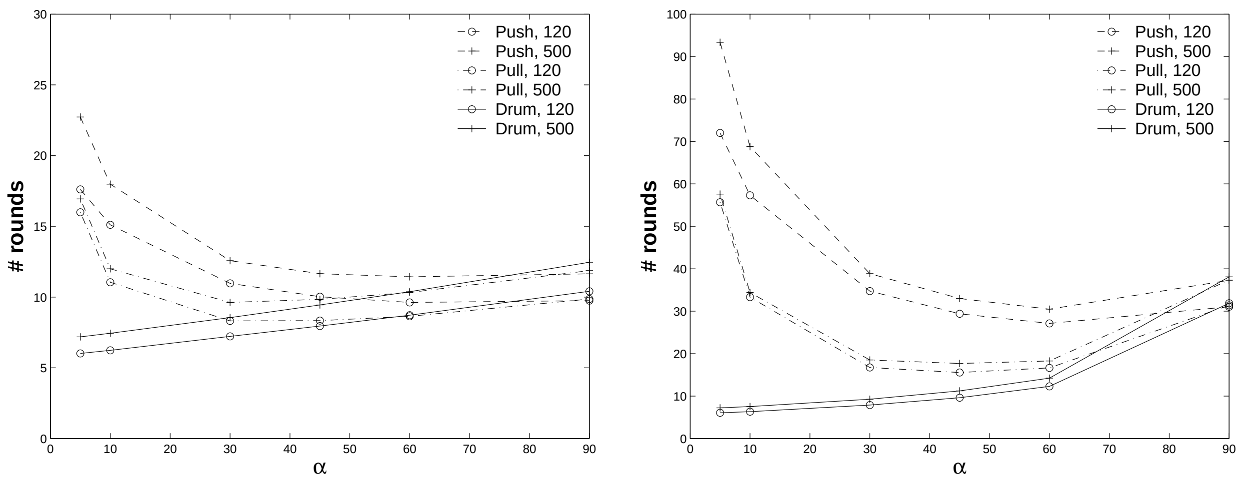
C. Adversary Strategies

We now evaluate the protocols under a range of attacks with fixed adversary strengths. First, we consider severe attacks 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 of them. Figure 7 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).



(a) Non-attacked processes. (b) Attacked processes.

Fig. 6. Propagation to attacked vs. non-attacked processes: CDF: Average percentage of attacked versus non-attacked processes that receive *M*, $n = 1000$, $\alpha = 40\%$, $x = 128$ (simulations).



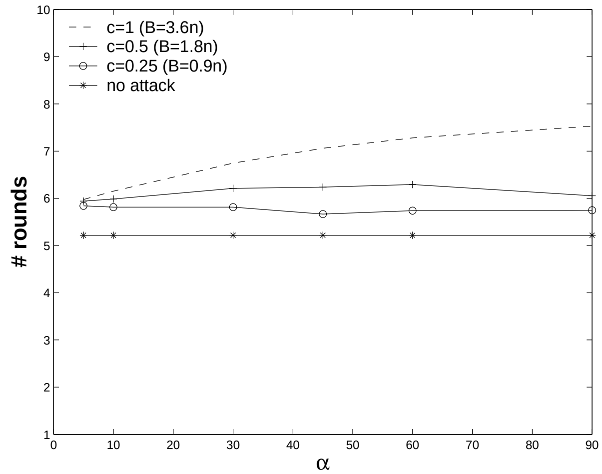
(a) $B = 7.2n$ ($c = 2$). (b) $B = 36n$ ($c = 10$).

Fig. 7. Strong fixed strength attacks: Average propagation time to 99% of the correct processes (simulations).

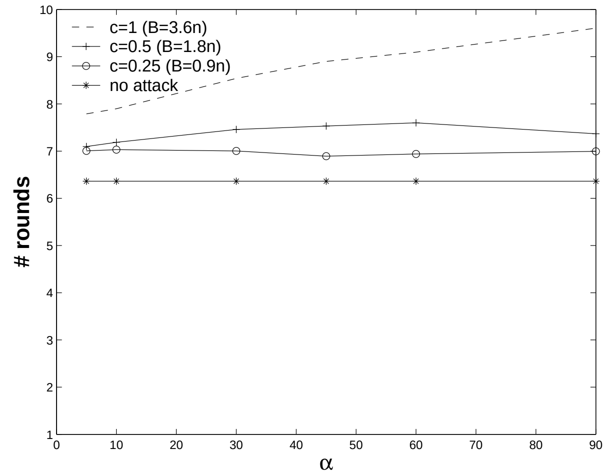
Next, we evaluate Drum under attacks with relatively small adversary powers of $B = 0.9n$, $B = 1.8n$ and $B = 3.6n$ ($c = 0.25$, $c = 0.5$, and $c = 1$, resp.) and also without an attack (as a baseline). As Figure 8 shows, such attacks have little impact on Drum's propagation time.

VIII. 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 [32], on a 100Mbit LAN, where a single process is run on each machine (i.e., $n = 50$). As in the simulations, 10% of the processes have crashed when the system started (these crashes go undetected), and the DoS attack is launched from outside the system. Since we do not have a router/firewall that randomly selects messages according to the protocol's needs, we have implemented the selection of messages by sequentially reading messages from the port at random times within the round, and discarding all messages at the end of the round. Since rounds are locally controlled and randomly vary in duration, the attacker cannot "aim" its messages for the beginning of a round.

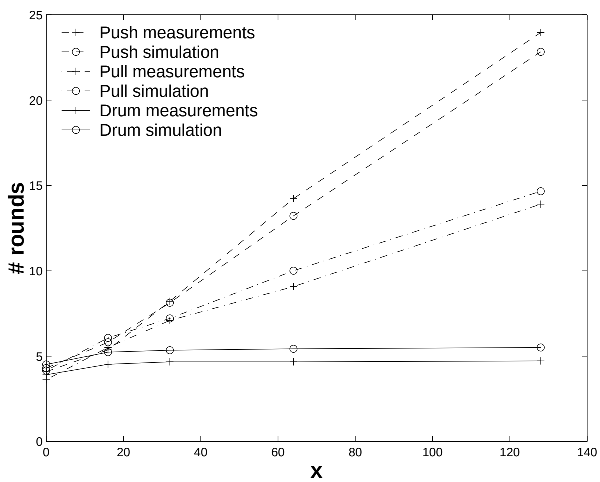


(a) $n = 120$.

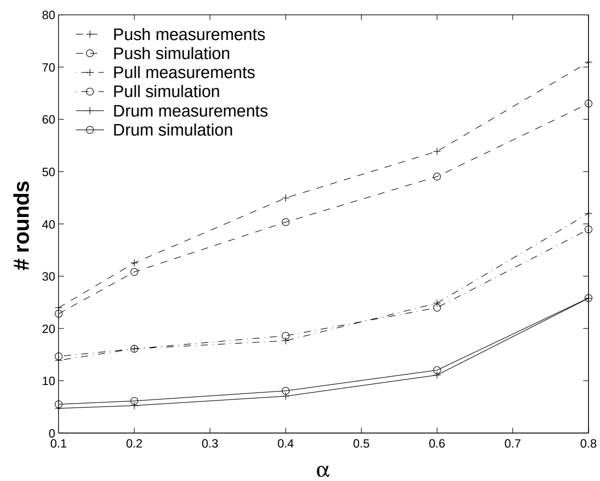


(b) $n = 500$.

Fig. 8. Weak fixed strength attacks: Drum, average propagation time to 99% of the correct processes (simulations).



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



(b) $x = 128$.

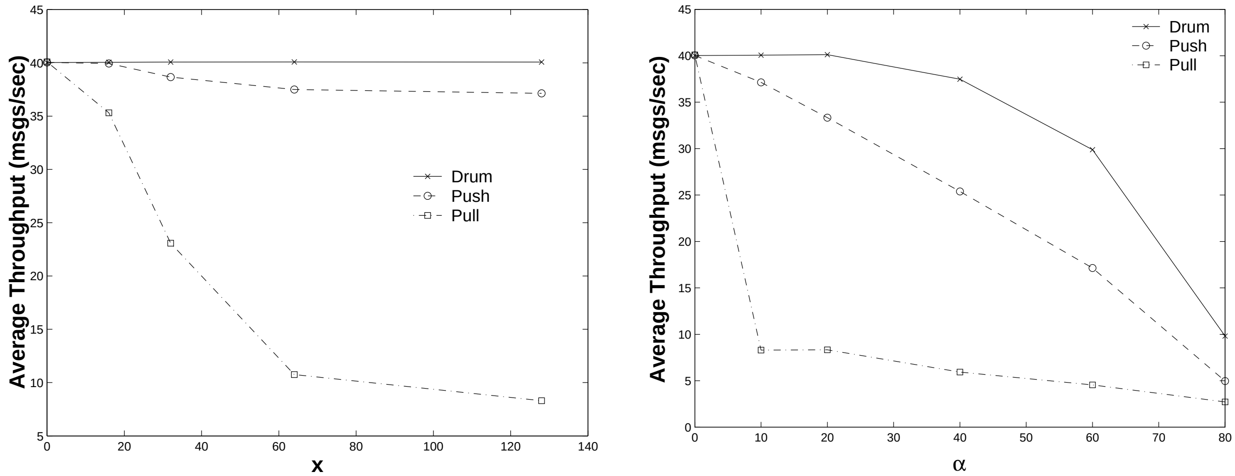
Fig. 9. Simulations vs. measurements: Average propagation time to 99% of the correct processes, $n = 50$.

A. Validating the Simulation Methodology

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 VII, first for increasing values of x and $\alpha = 10\%$, and then for $x = 128$ and increasing values of α. As in the simulations, every process has messages to send, and we track the propagation of one of those messages. Each data point is averaged over 1000 runs, again, as in the simulations.

Due to the lack of synchronization, messages can be propagated multiple hops in a single round in some situations. We use the following method to count the number of rounds it takes to propagate a message: when a message is created, a round counter is attached to it and initialized to 0. The message source logs the value 0, and immediately increases the round counter to 1. Whenever a process receives a new message, it logs the message's current round counter. Every round, each process increments the round counters of all the messages in its local buffer.

Figure 9 depicts the results of these experiments, and compares them with the corresponding simulation results. It shows that the experimental results are consistent with the simulation results, indicating that the simplifying assumptions made in the analysis and simulations have negligible effect on the results.



(a) $\alpha = 10\%$. (b) $x = 128$.

Fig. 10. Increasing attack strength: Average received throughput (measurements).

B. High Throughput Experiments

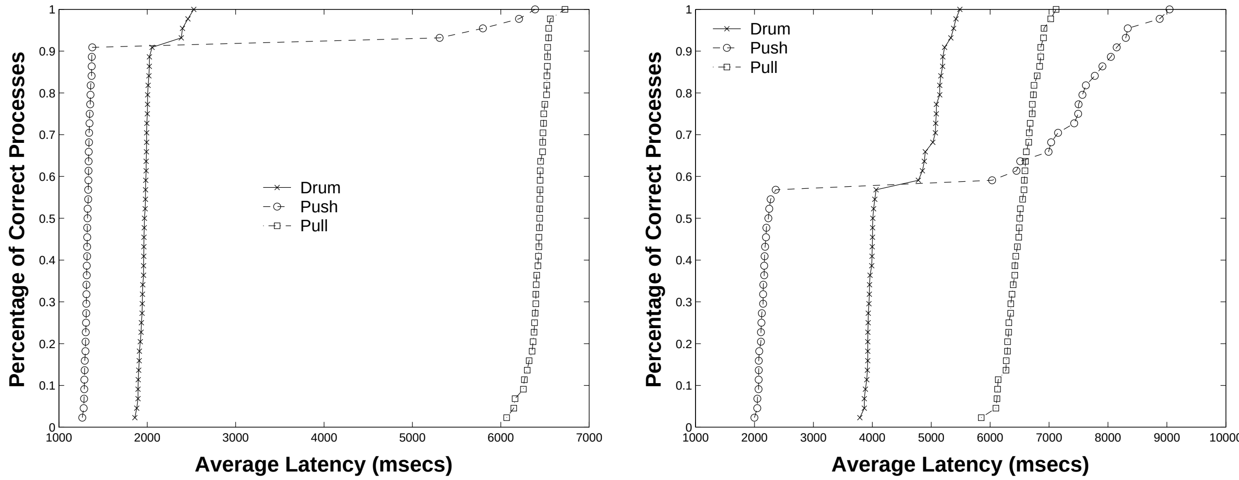
We proceed to evaluate the protocols in a realistic setting, where multiple messages are sent, and old messages are purged from processes' buffers. By running on a real network, we can faithfully evaluate latency in milliseconds (instead of rounds), as well as throughput.

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 in [6] 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 randomly chosen *new* 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 an attack 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).

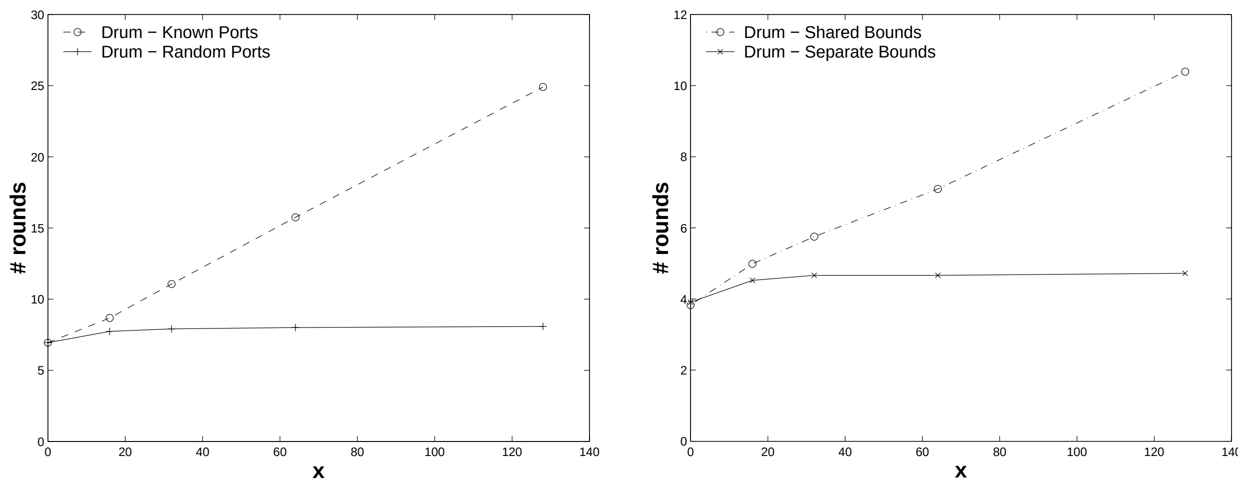
Figure 10 shows the throughput at the receiving processes for Drum, Push, and Pull, under the same DoS attack scenarios staged above. Figure 10(a) indicates that, as for latency, Drum's throughput is also unaffected by increasing x, while Push shows a slight degradation of throughput, and Pull's throughput decreases dramatically. Figure 10(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 11 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 11(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.



(a) $\alpha = 10\%$, $x = 128$. (b) $\alpha = 40\%$, $x = 128$.

Fig. 11. CDF: average latency of received messages (measurements).



(a) Random ports, $n = 1000$ (simulations). (b) Separate bounds, $n = 50$ (measurements).

Fig. 12. The effect of random ports and separate bounds on Drum's performance, $\alpha = 10\%$.

IX. Other DoS-Mitigation Methods

Until now, we have evaluated the advantage of combining both the push and pull techniques as a way to mitigate DoS attacks, in the context of a protocol that also employs resource bounds and random ports. We now turn to examine the importance of using the other two techniques: utilizing random ports whenever possible, and allocating separate resources for orthogonal operations.

In order to evaluate the effectiveness of random ports, we simulate Drum as described in Section VII, with the difference that pull-replies are sent to a well-known port instead of to a random one. The adversary attacks this port by equally dividing its attack strength for the pull channels between the pull-request port and the pull-reply port (i.e., each pull port is attacked with a quarter of the total attack strength). Figure 12(a) presents simulation results comparing Drum's performance with and without the use of random ports, when 10% of the processes are attacked. The results show a linear increase in propagation time for the well-known ports variation of Drum, as the rate of bogus messages each attacked process receives in a round increases. This is in contrast to the propagation time of Drum using random ports, which is bounded by a constant.

When solely using well-known ports, the adversary can attack both pull ports, as well as the push port. A process under attack experiences difficulty receiving messages both via push and through the pull channels, since the push and pull-reply ports are attacked. The same process's ability to send messages is only partly hampered. Although the pull-request port is attacked, the adversary cannot directly affect the process's outgoing push channels.

Next, we measure the effect of resource separation on Drum's performance. To this end, we change Drum's implementation detailed in Section VIII. Resources are now combined (i.e., a joint bound on the maximum number of processed messages per round is used) for receiving control messages: pull-requests, push-offers, and push-replies. We do not include the reception of data messages in this bound, since this bound may differ greatly from the bound on control messages in actual scenarios. Figure 12(b) contrasts the measurements of Drum's propagation time with shared bounds against those with separate bounds, when 10% of the processes are attacked. The results indicate a linear degradation of performance as the attack rate increases, when bounds are shared. On the other hand, the unmodified version of Drum is virtually indifferent to the increase in attack strength.

Shared bounds degrade Drum's performance under a DoS attack, since the fabricated control messages sent by the adversary to the well-known push-offer and pull-request ports consume resources that should be used for reading pull-requests, push-offers, and push-replies. The valid control messages are then discarded when resources are exhausted, and the attacked process becomes less responsive.

We conclude that random ports and separate resource bounds are crucial to Drum's ability to cope with DoS attacks.

X. 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 for each communication channel. 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 these measures go 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 divides the attack power among 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.

Appendix I
Calculating p_u and p_a

Suppose process p_i sends a message to process p_j, we want to calculate the probability that process p_j accepts this message. Denote the event "process p_i sends a message to process p_j" by S_{ij}. Assume $n > F$, and define q as the probability that process p_j appears in process p_i's *view*, then:

$$q = 1 - \frac{n-2}{n-1} \cdot \frac{n-3}{n-2} \cdots \frac{n-1-F}{n-F} = 1 - \frac{n-1-F}{n-1} = \frac{F}{n-1}$$

Let Y be the number of valid messages received by p_j in a single round, then:

$$\begin{array}{rrcl} & \Pr(Y \leq 0 \mid S_{ij}) & = & \Pr(Y \geq n \mid S_{ij}) = 0 \\ 0 < y < n & \Pr(Y = y \mid S_{ij}) & = & \binom{n-2}{y-1} q^{y-1}(1-q)^{n-1-y} \end{array}$$

Let p_Y be the probability that a non-attacked process, p_j, discards the message sent by p_i, given S_{ij}, then:

$$p_Y = \begin{cases} 0 & Y \leq F \\ \frac{Y-1}{Y} \cdot \frac{Y-2}{Y-1} \cdots \frac{Y-F}{Y-F+1} = \frac{Y-F}{Y} & Y > F \end{cases}$$

Calculating p_u gives:

$$\begin{aligned} p_u \;=\; & 1 - \sum_{y=-\infty}^{\infty} p_y \cdot Pr(Y = y \mid S_{ij}) = \\ & 1 - \sum_{y=F+1}^{n-1} \frac{y-F}{y} \cdot \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} = \\ & \sum_{y=1}^{F} \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} + \\ & \sum_{y=F+1}^{n-1} \frac{F}{y} \cdot \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} \end{aligned} \tag{7}$$

If p_j is attacked with $x \geq F$ messages, we get:

$$p_Y = \frac{Y+x-1}{Y+x} \cdot \frac{Y+x-2}{Y+x-1} \cdots \frac{Y+x-F}{Y+x-F+1} = \frac{Y+x-F}{Y+x}$$

And thus:

$$\begin{aligned} p_a \;=\; & 1 - \sum_{y=-\infty}^{\infty} p_y \cdot Pr(Y = y \mid S_{ij}) = \\ & 1 - \sum_{y=1}^{n-1} \frac{y+x-F}{y+x} \cdot \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} = \\ & \sum_{y=1}^{n-1} \frac{F}{y+x} \cdot \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} < \\ & \sum_{y=1}^{n-1} \frac{F}{x} \cdot \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} = \frac{F}{x} \end{aligned}$$

Lemma 7: $\frac{\mathrm{d}p_a}{\mathrm{d}\alpha} < \frac{F}{\alpha x}$.

Proof: Calculating the derivatives, we get:

$$\begin{aligned}
\frac{\mathrm{d}p_a}{\mathrm{d}x} &= \sum_{y=1}^{n-1} \frac{\mathrm{d}\frac{F}{y+x}}{\mathrm{d}x} \cdot \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} = \\
& \sum_{y=1}^{n-1} \frac{-F}{(y+x)^2} \cdot \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} \\
\frac{\mathrm{d}x}{\mathrm{d}\alpha} &= \frac{\mathrm{d}\frac{B}{\alpha n}}{\mathrm{d}\alpha} = \frac{-B}{\alpha^2 n} \\
\frac{\mathrm{d}p_a}{\mathrm{d}\alpha} &= \frac{\mathrm{d}p_a}{\mathrm{d}x} \cdot \frac{\mathrm{d}x}{\mathrm{d}\alpha} = \\
& \sum_{y=1}^{n-1} \frac{FB}{\alpha^2 n (y+x)^2} \cdot \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} = \\
& \sum_{y=1}^{n-1} \frac{Fx}{\alpha (y+x)^2} \cdot \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} < \\
& \sum_{y=1}^{n-1} \frac{Fx}{\alpha x^2} \cdot \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} = \frac{F}{\alpha x}
\end{aligned}$$

■

We now give a bound on p_u.

Lemma 8: $p_u > 0.6$.

Proof: Define:

$$\mu \triangleq E[Y \mid S_{ij}] = \textstyle\sum_{y=1}^{n-1} y \cdot \binom{n-2}{y-1} q^{y-1}(1-q)^{n-1-y} = \frac{n-2}{n-1} \cdot F + 1$$

$$E[Y^2 \mid S_{ij}] = \textstyle\sum_{y=1}^{n-1} y^2 \cdot \binom{n-2}{y-1} q^{y-1}(1-q)^{n-1-y} = \frac{(n-2)(n-3)}{(n-1)^2} \cdot F^2 + 3 \cdot \frac{n-2}{n-1} \cdot F + 1$$

$$\sigma^2 \triangleq Var(Y \mid S_{ij}) = \frac{(n-2)(n-3)}{(n-1)^2} \cdot F^2 + 3 \cdot \frac{n-2}{n-1} \cdot F + 1 - \left(\frac{n-2}{n-1} \cdot F + 1\right)^2 = \frac{n-2}{n-1} \cdot F - \frac{n-2}{(n-1)^2} \cdot F^2$$

By [31], for $n \gg 1$ we get that Y given S_{ij} can be approximated using a normal distribution function, with $\mu = F + 1$ and $\sigma^2 = F$. The cumulative distribution function $D(x)$ is thus:

$$D(x) = \tfrac{1}{2} \cdot \left(1 + \mathrm{erf}\left(\tfrac{x-\mu}{\sqrt{2}\sigma}\right)\right) = \tfrac{1}{2} \cdot \left(1 + \mathrm{erf}\left(\tfrac{x-F-1}{\sqrt{2F}}\right)\right) \quad \text{where} \quad \mathrm{erf}(z) = 1 - \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-t^2} \mathrm{d}t$$

From [31] we get the following:

$$\frac{1}{x+\sqrt{x^2+2}} < e^{x^2} \int_x^\infty e^{-t^2} \mathrm{d}t < \frac{1}{x+\sqrt{x^2+\frac{4}{\pi}}}$$

Concluding that:

$$\mathrm{erf}(z) = 1 - \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-t^2} \mathrm{d}t > 1 - \frac{2}{\sqrt{\pi}} \cdot \frac{e^{-z^2}}{z + \sqrt{z^2 + \frac{4}{\pi}}}$$

The first sum in formula 7 is approximated by $D(F)$. Calculating $D(F)$ gives:

$$\begin{aligned}
D(F) &= \frac{1}{2} \cdot \left(1 + \mathrm{erf}\left(\frac{-1}{\sqrt{2F}}\right)\right) > \frac{1}{2} + \frac{1}{2} \cdot \left(1 - \frac{2}{\sqrt{\pi}} \cdot \frac{e^{-\frac{1}{2F}}}{\sqrt{\frac{1}{2F} + \frac{4}{\pi}} - \frac{1}{\sqrt{2F}}}\right) = \\
& 1 - \frac{1}{\sqrt{\pi}} \cdot \frac{e^{-\frac{1}{2F}}}{\frac{\sqrt{\pi+8F}}{\sqrt{2\pi F}} - \frac{1}{\sqrt{2F}}} = 1 - \sqrt{2} \cdot \frac{\sqrt{F} \cdot e^{-\frac{1}{2F}}}{\sqrt{\pi + 8F} - \sqrt{\pi}}
\end{aligned}$$

Define:

$$g(F) = \frac{\sqrt{F} \cdot e^{-\frac{1}{2F}}}{\sqrt{\pi + 8F} + \sqrt{\pi}}$$

We want to bound D(x) from above by finding for which values of F, $g'(F) < 0$. The denominator of $g'(F)$ is always positive, so we ignore it when calculating the derivative:

$$\left(\frac{e^{-\frac{1}{2F}}}{2\sqrt{F}} + \frac{\sqrt{F}e^{-\frac{1}{2F}}}{2F^2}\right)\left(\sqrt{\pi + 8F} - \sqrt{\pi}\right) - \frac{8\sqrt{F}e^{-\frac{1}{2F}}}{2\sqrt{\pi+8F}} < 0$$

$$\frac{F^{\frac{3}{2}} + F^{\frac{1}{2}}}{2F^2}\left(\sqrt{\pi + 8F} - \sqrt{\pi}\right) - \frac{8\sqrt{F}e^{-\frac{1}{2F}}}{2\sqrt{\pi+8F}} < 0$$

$$\frac{\left(F^{\frac{3}{2}} + F^{\frac{1}{2}}\right)\left(\sqrt{\pi+8F} - \sqrt{\pi}\right)\sqrt{\pi+8F} - 8F^{\frac{5}{2}}}{2F^2\sqrt{\pi+8F}} < 0$$

Once again, the denominator is positive, and we get:

$$\left(F^{\frac{3}{2}} + F^{\frac{1}{2}}\right)\left(\sqrt{\pi + 8F} - \sqrt{\pi}\right)\sqrt{\pi + 8F} - 8F^{\frac{5}{2}} < 0$$

$$\pi + 8F - \sqrt{\pi^2 + 8\pi F} - 8F \cdot \left(1 - \frac{1}{F+1}\right) < 0$$

$$\frac{8F}{\sqrt{\pi}(F+1)} < \sqrt{\pi + 8F} - \sqrt{\pi}$$

Taking derivatives we get:

$$\frac{8}{\sqrt{\pi}(F+1)^2} \overset{?}{<} \frac{8}{2\sqrt{\pi + 8F}}$$

$$2\sqrt{\pi + 8F} \overset{?}{<} \sqrt{\pi}(F+1)^2$$

Clearly, $(F+1)^2$ grows faster than $2\sqrt{\pi + 8F}$. Numerically solving for $F = 1$ shows that the inequality holds. Thus, it holds for every $F \in \mathbb{N}$. Consequently, we only need to find the first F for which:

$$\frac{8F}{\sqrt{\pi}(F+1)} < \sqrt{\pi + 8F} - \sqrt{\pi}$$

A numerical solution for this inequality shows that it first holds for $F = 3$. Thus, for $F \geq 3$ we get that $g'(F) < 0$, and thus $D(F+1) > D(F)$. Assigning $F = 3$ in our previous bound for D(F), we get that for all $F \geq 3$, $D(F) \geq D(3) > 0.3968 \approx 0.4$. Assuming $F \geq 3$, we get:

$$\sum_{y=1}^{F} \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} > 0.4$$

Since $D(x)$ is maximal at $x = \mu = F + 1$ and symmetric around it, we get the approximation:

$$\sum_{y=F+1}^{2F} \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} > \sum_{y=1}^{F} \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y}$$

And finally, we conclude that:

$$
\begin{aligned}
p_u \quad = \quad & \sum_{y=1}^{F} \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} + \\
& \sum_{y=F+1}^{n-1} \frac{F}{y} \cdot \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} > \\
& \frac{2}{5} + \sum_{y=F+1}^{2F} \frac{F}{2F} \cdot \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} > \\
& \frac{2}{5} + \frac{1}{2} \cdot \sum_{y=1}^{F} \binom{n-2}{y-1} \left(\frac{F}{n-1}\right)^{y-1} \left(\frac{n-1-F}{n-1}\right)^{n-1-y} > \\
& \frac{2}{5} + \frac{1}{2} \cdot \frac{2}{5} = \frac{3}{5}
\end{aligned}
$$

■

APPENDIX II
CALCULATING $\widetilde{p}$

We now compute $\widetilde{p}$, the probability that M is propagated from the source in a round in Pull. Assume $n > F$, and define q as the probability that process p_2 appears in process p_1's $view_{pull}$, then $q = \frac{F}{n-1}$. Let Y be the number of valid pull-requests received in a single round, then:

$$
\begin{aligned}
& \Pr(Y < 0) &=& \Pr(Y \geq n) = 0 \\
0 \leq y < n \quad & \Pr(Y = y) &=& \binom{n-1}{y} q^y (1-q)^{n-1-y}
\end{aligned}
$$

Assume $x \geq F$, and define p_Y as the probability that a valid pull-request is read from the buffer, then:

$$
p_Y = 1 - \left(1 - \frac{Y}{Y+x}\right)\left(1 - \frac{Y}{Y+x-1}\right)\ldots\left(1 - \frac{Y}{Y+x-F+1}\right) = 1 - \frac{x! \cdot (Y+x-F)!}{(x-F)! \cdot (Y+x)!}
$$

The probability $\widetilde{p}$ that a valid pull-request is read from the buffer, independent of Y, is:

$$
\widetilde{p} = \sum_{y=-\infty}^{\infty} p_y \cdot \Pr(Y = y) = \sum_{y=0}^{n-1} \left(1 - \frac{x! \cdot (y+x-F)!}{(x-F)! \cdot (y+x)!}\right) \binom{n-1}{y} \left(\frac{F}{n-1}\right)^y \left(\frac{n-1-F}{n-1}\right)^{n-1-y}
$$

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.

REFERENCES

[1] 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.

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

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

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

[5] 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 *6th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 1–12, 1987.

[6] P. T. Eugster, R. Guerraoui, S. B. Handurukande, A. M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. In *The International Conference on Dependable Systems and Networks (DSN)*, 2001.
[7] X. Geng and A. B. Whinston. Defeating distributed denial of service attacks. *IEEE IT Professional*, pages 46–51, July/August 2000.
[8] I. Gupta, A.-M. Kermarrec, and A. J. Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. In *21st IEEE International Symposium on Reliable Distributed Systems (SRDS)*, pages 180–189, October 2002.
[9] I. Gupta, R. van Renesse, and K. P. Birman. Scalable fault-tolerant aggregation in large process groups. In *The International Conference on Dependable Systems and Networks (DSN)*, pages 433–442, 2001.
[10] Juniper Networks. The Need for Pervasive Application-Level Attack Protection. `http://itresearch.forbes.com/detail/RES/1067617`
[11] 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.
[12] A.-M. Kermarrec, L. Massouli, and A. J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):248–258, March 2003.
[13] M. J. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. In *European Dependable Computing Conference (EDCC)*, pages 364–379, 1999.
[14] M. J. Lin, K. Marzullo, and S. Masini. Gossip versus deterministically constrained flooding on small networks. In *14th International Symposium on DIStributed Computing (DISC)*, pages 253–267, 2000.
[15] 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.
[16] 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.
[17] D. Malkhi, E. Pavlov, and Y. Sella. Optimal unconditional information diffusion. In *15th International Symposium on DIStributed Computing (DISC)*, 2001.
[18] 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.
[19] Y. M. Minsky and F. B. Schneider. Tolerating malicious gossip. *Distributed Computing*, 16(1):49–68, February 2003.
[20] 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.
[21] NetContinuum. Web Application Firewall: How NetContinuum Stops the 21 Classes of Web Application Threats. `http://www.netcontinuum.com/products/whitePapers/getPDF.cfm?n=NC_WhitePaper_WebFirewall.pdf`.
[22] P-Cube. Dos protection. `http://www.p-cube.com/new_solutions/service_DoS.shtml`.
[23] P-Cube. Minimizing the effects of dos attacks. `http://www.juniper.net/solutions/literature/app_note/350001.pdf`.
[24] B. Pittel. On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, February 1987.
[25] Riverhead Networks. Products overview. `http://www.riverhead.com/pr/index.html`.
[26] L. Rodrigues and R. G. A.-M. K. S. Handurukande, J. Pereira. Adaptive gossip-based broadcast. In *The International Conference on Dependable Systems and Networks (DSN)*, pages 47–56, June 2003.
[27] 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.
[28] S. Staniford, V. Paxson, and N. Weaver. How to own the Internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, August 2002.
[29] Stephen de Vries. Application Denial of Service Attacks. `http://www.corsaire.com/white-papers/040405-application-level-d`
[30] J. Wang, L. Lu, and A. A. Chien. Tolerating denial-of-service attacks using overlay networks – impact of overlay network topology. *ACM Workshop on Survivable and Self-Regenerative Systems*, October 2003.
[31] E. W. Weisstein. *CRC Concise Encyclopedia of Mathematics*.
[32] 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 *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002. USENIX Association.
[33] 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.