# Improving Denial of Service Resistance using Dynamic Local Adaptations

Gal Badishi*
Department of Electrical Engineering
Technion
Haifa, Israel
badishi@ee.technion.ac.il

Idit Keidar
Department of Electrical Engineering
Technion
Haifa, Israel
idish@ee.technion.ac.il

## ABSTRACT

We improve the resistance of gossip-based multicast to (Distributed) Denial of Service (DoS) attacks using dynamic local adaptations at each node. Each node estimates the current state of the attack on the system, and then adapts its behavior according to this local estimation. The adaptation is achieved through modeling the problem of propagating messages under a DoS attack as an optimization problem, and solving it using linear programming, independently at each node. Simulation results show that when the system is under attack, the local decisions each node takes bring the system to a stable point, which is the solution of the linear programming problem. The adaptation leads to propagation times that are 30% faster than those of existing DoS-resistant gossip-based protocols.

## Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

## Keywords

Adaptive denial-of-service resistance, Adaptive gossip-based protocols, Application-level multicast

## 1. INTRODUCTION

Denial of service (DoS) attacks are attacks that usually aim to exhaust resources by overloading an entity with large amounts of bogus messages. The use of armies of infiltrated machines ("zombies") leads to distributed DoS (DDoS), in which the attacker utilizes its set of compromised machines to launch a coordinated attack with massive strength. In this paper, we consider application-level DoS attacks, in which the application is overwhelmed with messages to process even when the network is not congested. This situation is common is applications that require extensive processing for each incoming request, e.g., cryptographic authentication.

*Supported by the Israeli Ministry of Science.


*SAC* '08, March 16-20, 2008 Fortaleza, Brasil


As DoS attacks can cause severe damage and are fairly easy to deploy, it is important to design communication protocols with DoS-mitigation mechanisms in place. However, designing a protocol that performs well under a certain DoS attack does not mean that it still performs well as the attack changes. We believe that a protocol that adapts its parameters to the actual attack taking place can perform better than a static protocol that behaves the same under all DoS attacks. To illustrate this point, we focus on gossip-based multicast protocols as a case study.

We present a novel approach that adapts a gossip-based protocol to the attack currently launched on the system. The adaptation is modeled as an optimization problem, which is solved using linear programming. Every round, each node locally estimates the current state of the attack on the system and feeds it to the linear programming algorithm, which presents the node with the new resource distribution to use. We show, using simulations, that the local resource distribution each node independently calculates improves the global propagation time in the system, i.e., the number of rounds it takes a message generated at the source to reach all nodes with high probability. Our propagation times are better than existing solutions by up to $34\%$.

## 2. BACKGROUND AND RELATED WORK

Gossip-based protocols [1, 2, 3, 5] are round-based randomized multicast protocols that deliver data messages from their sources to all other nodes with high probability. The expected number of rounds it takes a message to reach all nodes in the system is logarithmic in the number of nodes. Each round, every node selects a subset of the nodes in the system uniformly at random and communicates with them. The uniform random selection means that all nodes have an identical part in the system.

Gossip-based communication happens in at least one of the following methods: (1) *Push* – node $A$ sends a push message to node $B$, which replies with its list of data messages, and then $A$ sends $B$ any data messages $B$ does not have; and (2) *Pull* – node $A$ sends a pull message to node $B$ with its list of data messages, and node $B$ replies with any data messages $A$ does not have. Sending lists of messages instead of actual messages minimizes bandwidth and processing times, as we assume that control messages (push/pull) are much shorter than data messages.

The use of randomness makes gossip-based protocols very robust in the face of node failures [3, 5]. However, naïve gossip-based protocols are vulnerable to DoS attacks [**]. Drum [**] is a gossip-based multicast protocol that deals with DoS attacks and uses both push and pull. Drum provides significantly better propagation times than gossip-based protocols that do not take actions to protect themselves from DoS attacks. However, Drum uses a static

allocation of resources at each node – half of the resources are used for pushing, and the other half is used for pulling. In this paper, we propose to dynamically determine the proper resource allocation according to the attack, as locally perceived by each node.

The resources Drum invests in each operation are bounded. The number of nodes a node chooses to talk to each round using push or pull is called the push fan-out ($SO$) or pull fan-out ($LO$), respectively. Similarly, the maximum number of nodes a Drum node answers to in a round using the push channels or pull channels is called the push fan-in ($SI$) or pull fan-in ($LI$), respectively. If the number of push messages that reach a node during a round exceeds that node's $SI$, the node chooses a random subset of $SI$ of them and answers that subset. The same thing happens for pull. When a node is under attack, bogus messages force the node to make such a subset selection, and thus valid messages are dropped.

An adversary can attack Drum on the 2 distinct ports used for incoming push or pull messages. As described above, the push operation is a 3-step process, and the pull operation is a 2-step process. Only the messages from the first step of each operation are sent to these known ports, and each of those messages carries a random port number to reply on. Since the adversary cannot eavesdrop on those messages to find the random port number *and* notify its army of zombies to attack that port fast enough so the attack happens before communication ends, we assume that only the first step in the push and pull operations can be harmed by the adversary.

Adaptation in gossip-based protocols has been explored before, e.g., Rodrigues et. al [6] study adaptation in a gossip-protocol using flow-control to avoid congestion. Kyasanur et al. [4] study adaptive gossip in sensor networks, where the sensors wish to limit their transmission to conserve power consumption. However, we are the first the we know of that provide adaptation to DoS attacks.

# 3. ADVERSARY ASSUMPTIONS

We assume an external attacker that can cause messages to be dropped by overloading the multicast nodes with bogus requests. The attacker is not part of the multicast system, and does not participate in the gossip protocol. All nodes in the system are correct, follow the gossip protocol, and can differentiate between valid and bogus requests, perhaps at the cost of additional work, i.e., authentication.

The attacker has bounded capacity for sending messages in a single round. When mounting an attack, the adversary chooses the nodes and ports to attack, out of the ones it knows, and the number of invalid messages it wishes to send to each attacked node each round. We denote by $\alpha$ the percentage of nodes being attacked, and for simplicity assume that all attacked nodes are attacked with $C_{push}$ bogus push messages and $C_{pull}$ bogus pull messages per round, i.e., every round $C_{push}$ and $C_{pull}$ bogus messages are sent to each attacked node's push and pull ports, respectively.

The attacker uses zombies to leverage its attack, and must communicate with them to update them on the attack strategy. Realizing that the system is adapting itself to the attack, devising a new attack plan and updating all zombies take time. We can therefore assume that by the time the attacker reacts to our adaptation, the system has completely reached its optimized point.

# 4. ADAPTATION

Each node locally adapts its behavior according to its view of the current state of the attack on the system. To perform a useful adaptation, there are two challenges to consider: (1) How to reliably estimate the current state of the attack; and (2) Given the current state of the attack, what is the best strategy to employ. To tackle these challenges, we start by assuming that all nodes know the exact state of the attack, and find a strategy that accommodates the attack and improves propagation time (Section 4.1). We then provide means to estimate the state of the attack (Section 4.2).

## 4.1 Finding the Target Strategy

The only communication elements a node controls are its push and pull channels, whether incoming or outgoing. The distribution of the node's limited resources among these channels constitutes the node's strategy. We want to find the best strategy each node should use to optimize the global propagation time when the system is under a DoS attack. Since gossip-based multicast protocols choose communication partners uniformly at random each round, and since all attacked nodes are attacked in the same manner, it is clear that all attacked nodes should exhibit the same behavior, and all unattacked nodes should use the same strategy. The strategies of the attacked and unattacked nodes will likely not be the same.

Recall that $\alpha$ is the percentage of attacked nodes, and every round the attacker sends each of these nodes $C_{push}$ and $C_{pull}$ bogus messages to their incoming push and pull channels, respectively. For simplicity of analysis, we transform $C_{push}$ and $C_{pull}$ to the concrete damage that they make, and define:

- $p_s$ – the probability of a push message being dropped due to the attack on the push channels (depends on $C_{push}$).
- $p_l$ – the probability of a pull message being dropped due to the attack on the push channels (depends on $C_{pull}$).

We use the following notations for node strategies:

- $ASO$ is an attacked node's push fan-out, i.e., the number of nodes randomly-chosen each round as targets for outgoing push messages. A successful reception of an outgoing push message sent from node $A$ to node $B$ results in transferring data messages from node $A$ to node $B$.
- $ASI$ is an attacked node's push fan-in, i.e., the maximum number of randomly selected incoming push messages (valid or not) that will be processed in a single round.
- $ALO$ and $ALI$ are the same as $ASO$ and $ASI$ (respectively), but for pull. A successful reception of an outgoing pull message sent from node $A$ to node $B$ results in transferring data messages in the *opposite* direction – from node $B$ to node $A$. That is, $ALO$ is responsible for outgoing pull messages, *but incoming data messages*. $ALI$ is responsible for incoming pull messages, *but outgoing data messages*.
- $USO$, $USI$, $ULO$ and $ULI$ are the same as $ASO$, $ASI$, $ALO$ and $ALI$ (respectively), but for an unattacked node.

By definition, all fan-ins and fan-outs are non-negative integers. For instance, in Drum, all fan-ins and fan-outs are equal to $F$, where $F$ is some positive integer, e.g., 4. In a push protocol, all push fan-ins and fan-outs are equal to $2F$, and all pull fan-ins and fan-outs are equal to $0$. $F$ is bounded from above due to the limited resources the node can allocate for the communication.

We now solve an optimization problem to find the nodes' best strategy under attack. We start by describing a set of constraints that each node must adhere to. All constraints are normalized by $F$, our basic unit of reference:

CONSTRAINT 1. $ALI + ASO = 2$
$ULI + USO = 2$

*Reasoning.* Receiving pull messages and sending push messages provide the same functionality – sending data messages from the node to the nodes it communicates with. The resources are thus bound by 2 units, as we are essentially bounding two communication channels (push and pull) together.

CONSTRAINT 2. $ASI + ALO = 2$
$USI + ULO = 2$

*Reasoning.* Both receiving push messages and sending pull message allow the node to receive data messages from nodes it communicates with. Once again, the total amount of resources allocated for this purpose is 2 units.
Additionally, we have some constraints on the system as a whole:

CONSTRAINT 3.
$$\alpha ASI + (1-\alpha)USI = \alpha ASO + (1-\alpha)USO$$
$$\alpha ALI + (1-\alpha)ULI = \alpha ALO + (1-\alpha)ULO$$

*Reasoning.* The total amount of resources allocated for outgoing push messages should be equivalent to the total amount of resources allocated for incoming push messages, otherwise resources are wasted. This is true for pull as well.

CONSTRAINT 4. $\alpha ASO + (1-\alpha)USO = 2 \cdot \left(1 - \frac{p_s}{p_s+p_l}\right)$
$$\alpha ALO + (1-\alpha)ULO = 2 \cdot \left(1 - \frac{p_l}{p_s+p_l}\right)$$

*Reasoning.* It is important to have both the push and pull operations. The push operation allows an attacked source to propagate its message quickly via its outgoing push channel. It has been proven that it takes a time linear in $C_{pull}$ to retrieve a message from an attacked source, when exclusively using the pull protocol [**]. Pull allows an attacked node to receive data messages easily from an unattacked node, through the outgoing pull channel. Using push alone to deliver messages to attacked nodes takes a time linear in $C_{push}$ [**].

Obviously, the more a channel is attacked, the less we want to use it – hence the ratio. Note that in case only one channel is attacked, it is closed and the attack has no influence. Obviously, this is the best strategy for such a case. Additionally, when both channels are attacked at the same strength, it is clear that the amount of resources allocated for push and pull should be equal (from symmetry).
Finally, we have the boundary conditions:

CONSTRAINT 5. $0 \leq ASO,\ ASI,\ ALO,\ ALI \leq 2$
$0 \leq USO,\ USI,\ ULO,\ ULI \leq 2$

To complete the optimization-problem statement, we still need to define the cost function to minimize. We want to minimize losses in the system, so that more messages can be processed by nodes, and thus data messages will be transferred faster. All messages lost due to the attack are dropped at the incoming channels of the attacked nodes. Assuming that attacked nodes are sent at least $ASI$ and $ALI$ valid messages for their incoming push and pull ports, respectively, the attack-induced losses in the system are defined by the following function:

$$f(\text{fan-outs and fan-ins}) = \alpha p_s ASI + \alpha p_l ALI$$

We have completed the definition of the optimization problem, and can now turn to solving it. Since we assume that we know $\alpha$, $p_s$ and $p_l$, we get a set of linear equations and inequalities in 8 variables (the fan-outs and fan-ins). The function to minimize, $f$, is also linear. Thus, we can solve this optimization problem using linear programming.

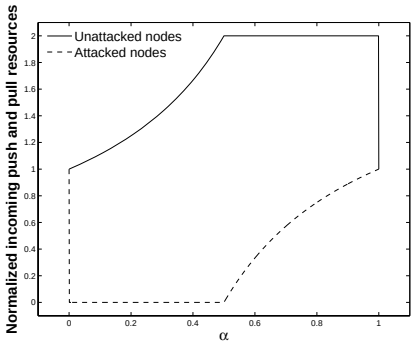
Figures 1 and 2 show some solutions to the optimization problem for different scenarios, as calculated using MATLAB.

Figure 1(a) shows the change in the resources allocated for the incoming push channels, $ASI$ and $USI$, as a function of the percentage of the attacked nodes, $\alpha$. The system is attacked on both push and pull channels, and the probability of a valid message being dropped due to the attack is greater than 0 and equal for both channels, i.e., $p_s = p_l > 0$. Due to this symmetry, exactly the same resources are allocated for the incoming pull channels, i.e., $ALI$ and $USI$. The actual values of $p_s$ and $p_l$ do not matter, as long as they are equal and positive. We can see that as soon as the attack begins ($\alpha > 0$), the attacked nodes deallocate all resources used for the incoming push channels, which minimizes our cost function $f$. From Constraint 2 we can tell that these resources are diverted to the outgoing pull channels (not shown on figure). This is a good adaptation, since the attacked nodes experience problems receiving data messages via their incoming push channels due to the attack, and it is best if they concentrate more resources on receiving data messages using their outgoing pull channels, which do not directly suffer from the attack. Figure 1(b) shows the actual fan-ins the nodes should use (whole numbers), for $F = 4$.

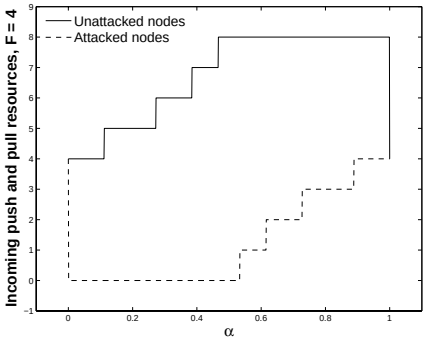
From Figure 1(a) and Constraints 1 and 2, we get that as more nodes are attacked (up to $50\%$ of the nodes), the total amount of resources allocated by attacked nodes for outgoing channels increases, since each attacked node directs all its resources to its outgoing channels. To accommodate this increase in the total amount of resources allocated for outgoing channels, the unattacked nodes increase the amount of resources allocated for their incoming channels. This conforms to Constraint 3. When more than $50\%$ of the nodes are attacked, the unattacked nodes can no longer compensate for the increase in the incoming-channels' resources, as they have already exhausted all their available resources. Consequently, the attacked nodes change their behavior and direct some resources from the outgoing channels to the incoming channels. Finally, a node's strategy in a system where all nodes are attacked, and both push and pull channels are attacked at the same strength, is equal to the node's strategy in a system in which no node is attacked at all. This is a consequence of all nodes experiencing the same environment, and the equal allocation of resources to the push and pull channels, as per Constraint 4, since both of them exhibit the same loss rate.

Figure 1(c) shows the nodes' behavior when only the push channels are attacked. The figure shows that the attacked nodes invest all their resources for outgoing data messages in the incoming pull channels, and thus, by Constraint 1, do not use outgoing push at all. We can see that the unattacked nodes do the same thing, as the resources they allocate for outgoing push messages immediately drop to 0 when the attack commences. It is easy to see that Constraint 3 means that no resources are allocated for incoming push messages as well, and all resources are diverted to outgoing pull messages (by Constraint 2). The resulting strategy is the exclusive use of pull in the system. The dual case of an attack solely on the pull channels exhibits the opposite results, where only push is used in the system, and is omitted here for brevity.

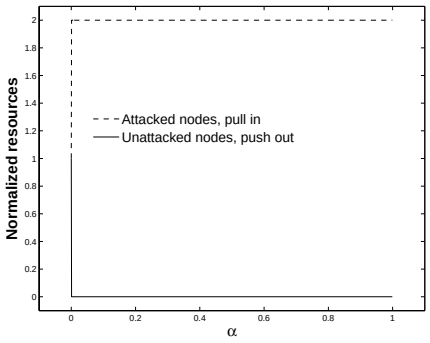
Figures 2(a) and 2(b) show the nodes' behavior when the attack on push is stronger than the attack on pull, such that the loss probability for push, $p_s$, is 1, and the loss probability for pull, $p_l$, is $0.5$. From Constraint 4 we get that the system will try to divide the total amount of resources allocated in the system for outgoing channels to $\frac{2}{3}$ for push, and $\frac{4}{3}$ for pull (out of a combined total of 2 normalized resources). Figure 2(a) shows that the attacked nodes immediately cease to use the incoming push channels as the attack begins (to minimize the cost function $f$), and shift the deallocated




(a) Linear programming fan-ins, $p_s = p_l > 0$.

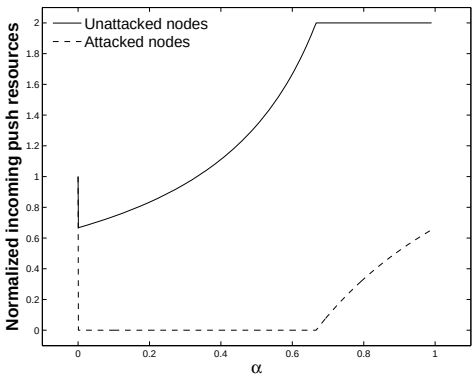



(b) Concrete fan-ins for $F = 4$, $p_s = p_l > 0$.

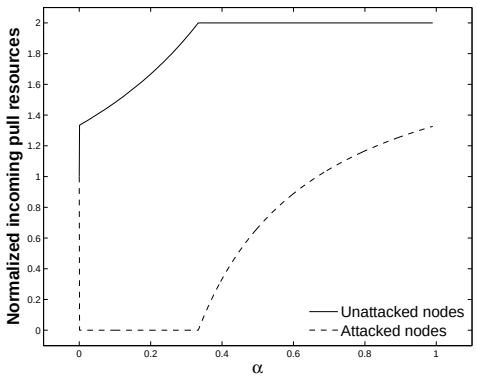



(a) Linear programming fan-ins, $p_s = 1$, $p_l = 0$.

**Figure 1: Target strategies as a function of $\alpha$.**




(a) Push resources.




(b) Pull resources.

**Figure 2: Target strategies for $p_s = 1$, $p_l = 0.5$, as a function of $\alpha$.**

resources to the outgoing pull channels (Constraint 2). Similarly, Figure 2(b) shows that the attacked nodes also reduce to 0 the resources they allocate for the incoming pull channels, which means that their outgoing push channels are at full capacity (Constraint 1).

Back to Figure 2(a), the unattacked nodes continue to use push, to support the outgoing push channels of the attacked nodes, but reduce the resource allocation to $\frac{2}{3}$ of the basic unit. Similarly, in Figure 2(b) we can see the unattacked nodes using pull with an allocation of $\frac{4}{3}$ of the basic unit. As the percentage of attacked nodes increases, the unattacked nodes need to compensate for the increase in the total amount of resources allocated for the outgoing channels in the system, so they start increasing the resource allocation for their incoming channels, at the expense of their outgoing channels (cf. Figure 2). Once the unattacked nodes cannot allocate more resources for the incoming channels, the attacked nodes start allocating resources for the incoming channels, to support Constraint 3. This happens earlier for pull than for push.

## 4.2 Attack Estimation

Now that each node knows what strategy to employ based on the system's state, we need to devise a way for a node to estimate that state through local observations. According to our adaptation algorithm, a state is completely defined by $\alpha$, $p_s$ and $p_l$, so we need to find a way to estimate those variables.

A node's perception of the system comes from its interaction with other nodes. Each round the node performs push and pull communication with a different random subset of nodes, and can use this communication to evaluate the system's state.

An attacked node knows that it is being attacked, as it receives many bogus messages each round. Each time an attacked node communicates with other nodes, it informs them that it is being attacked. The estimation of $\alpha$ is based on the information gained from communicating with attacked nodes. Additionally, $p_s$ and $p_l$ are estimated based on percentage of outgoing push and pull messages that were not replied to. This factor also contributes to the calculation of $\alpha$, as we assume that messages that were not replied to were dropped due to an attack. This method of estimating $p_s$ and $p_l$ works well as long as the outgoing channels of the node performing the estimation and the incoming channels of the nodes being estimated have fan-outs and fan-ins (respectively) greater than 0. Otherwise, no meaningful data will be gathered.

To ensure that estimation is performed regardless of the fan-ins and fan-outs, each node allocates static resources for incoming and outgoing special *probe* messages. These messages are sent to the push and pull channels of other nodes. A node that receives a probe message, simply replies with an empty message to indicate that it is able to receive messages. This mechanism is light-weight, does not impose any limitations on the nodes, and is used solely to evaluate $p_s$ and $p_l$, and not for answering push/pull messages.

We use the following notations when considering some node $A$'s outgoing communication in a single round:

- $SO$, $LO$ – the number of nodes $A$ sent messages (including probes) to via the push or pull channels, respectively.
- $SOA$, $LOA$ – the number of nodes $A$ sent messages (including probes) to via the push or pull channels, respectively, and the nodes replied and indicated that they were attacked.

- $SOD$, $LOD$ – the number of nodes $A$ sent messages (including probes) to via the push or pull channels, respectively, and got no reply back. These nodes are also presumed to be under attack.

Each round $r$, every node performs its local estimations as follows:

$$\alpha(r) = \frac{SOA + SOD + LOA + LOD}{SO + LO}$$

$$p_s(r) = \frac{SOD}{SOA + SOD}$$

$$p_l(r) = \frac{LOD}{LOA + LOD}$$

Estimations are subject to fluctuations, since the choice of nodes to communicate with is random. In order to prevent the nodes from constantly changing their strategies even when the system's state remains intact, the nodes do not use single-round estimations in the calculation of their strategies, but rather use the average of the last $k$ estimations, e.g., $AVERAGE(\alpha(r-k+1),\ \alpha(r-k+2),\ \ldots,\ \alpha(r-1),\ \alpha(r))$. The last $k$ estimations are set to 0 when a node first joins the system. When an estimation cannot be performed, because of the denominator being 0 in some round, that round's estimation is chosen to be the average of the last $k$ estimations. Choosing a small $k$ means that nodes are able to respond more rapidly to a change in the system state (a change in the attack strength/distribution). Choosing a large $k$ means that there are no fluctuations in the fan-ins and fan-outs as long as the system's state does not change.

# 5. SIMULATION RESULTS

We test our adaptation mechanism through MATLAB simulations. Our system consists of a 1,000 nodes, communicating using a gossip-based push/pull multicast protocol. The simulation progresses in synchronous rounds. In each round, all nodes send push/pull messages to randomly-selected nodes. Push/pull messages that do not get dropped due to limited incoming resources or due to an attack get answered, and then data messages are transferred. Finally, the nodes perform any calculations they may have for that round. All rounds begin and end at the same time in all nodes. A round is finished when all operations for that round end at all the nodes. In all experiments, $F = 4$.

Section 5.1 tests the effectiveness of the strategies computed in Section 4.1. $\alpha$, $p_s$ and $p_l$ are assumed to be known, and the propagation time of our adaptive protocol is compared with 3 other protocols. Section 5.2 evaluates the estimation procedure described in Section 4.2. The nodes constantly estimate the state of the attack and change their strategies according to the solution to the minimization problem with the perceived $\alpha$, $p_s$ and $p_l$.

## 5.1 Strategy Evaluation

We start by evaluating our solution to the adaptation problem, as described in Section 4.1. We assume that $\alpha$, $p_s$ and $p_l$ are known in advance to all nodes that use adaptation, and compare 4 gossip-based multicast protocols: (1) Push – only uses the push channels. (2) Pull – only uses the pull channels. (3) Drum – divides its resources equally between push and pull. (4) Adaptive Drum – divides its resources according to the adaptation strategy described in Section 4.1.

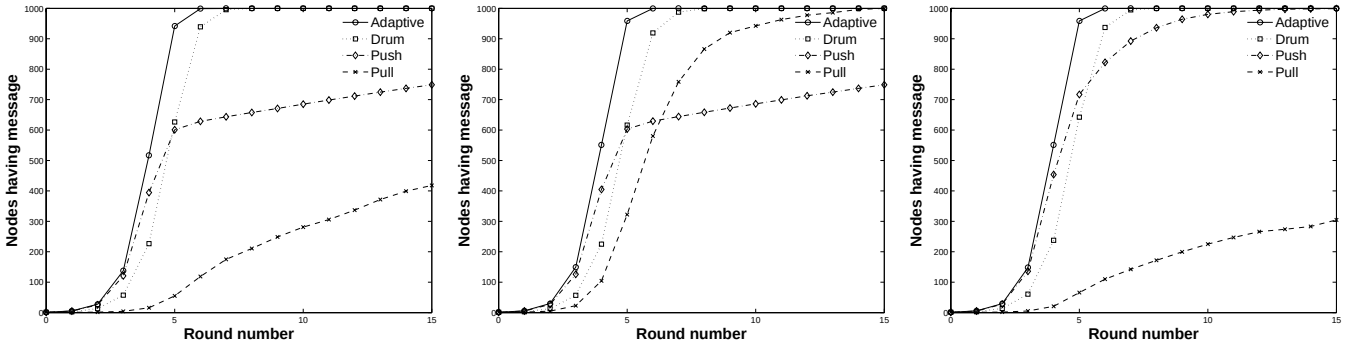
To determine the exact strategy Adaptive Drum uses, we need to determine the exact values of $p_s$ and $p_l$ before running the simulation. $p_s$ depends on $C_{push}$ and on $ASI$. Similarly, $p_l$ depends on $C_{pull}$ and $ALI$. We first assume that all fan-ins and fan-outs equal to $F$ (as in Drum), and calculate $p_s$ and $p_l$. Then, we solve the optimization problem and get the adapted $ASI$ and $ALI$. This might change the values of $p_s$ and $p_l$, so we recalculate them, and so on. When we are finished, we have the values of $p_s$ and $p_l$ after stabilization, and the proper strategies for all nodes. These are the strategies we use for Adaptive Drum.

We assume that new data messages are constantly generated in the system, and examine the propagation of one of those messages, i.e., the number of nodes that have the message as the rounds progress. The message originates at an attacked node in round 0. For simplicity, we assume that whenever nodes send data messages to one another, they send all the data messages they know of. This is consistent with the assumptions used in [1, 2]. Due to the random nature of gossip-based multicast protocols, each data point represents an average of 100 independent experiments.

Figure 3(a) shows the message propagation times for all 4 protocols, where $40\%$ of the nodes are attacked on both their push and pull incoming channels, with 1,000 bogus messages per channel per round ($p_s = p_l \approx 1$). The optimized strategy for this scenario was shown in Figure 1. In both cases, we can see that Adaptive Drum propagates the message to all nodes faster than the rest of the protocols. Push quickly propagates the message to the unattacked nodes, but then takes time to deliver it to the attacked nodes. Pull experiences problems getting the message out of its source, since the source is attacked. Drum starts propagating the message slower than Push, since it also uses the pull channels, but then continues to propagate the message faster than Push, as Drum has little trouble to propagate the message to the attacked nodes. The decision of the attacked nodes to allocate all their resources to the outgoing channels means that Adaptive Drum's propagation times are similar to Push's propagation times, at the beginning of the dissemination. However, Adaptive Drum's robustness is soon realized, as it continues to propagate the message at the same good pace, while Push's propagation speed is significantly slowed when it is time to deliver the message to the attacked processes. The case of $\alpha = 20\%$ was also tested and provided similar results.

Figures 3(a) and 3(b) show propagation times when the attack is uneven on the push channels and the pull channels. Although the attack is uneven, we still get that both $p_s$ and $p_l$ are very close to 1, due to the adaptation of the attacked nodes. Figure 3(b) depicts a scenario where push is attacked in a 10-times stronger attack than pull. We can see that indeed Pull performs better than Push, but still, Adaptive Drum provides the fastest propagation time. Figure 3(c) shows the opposite case, where the pull channels are attacked more severely than the push channels. In this case, we can see that the propagation time of Push improves. Nevertheless, Adaptive Drum still achieves the best propagation time. These results stem from the fact that when both channels are attacked, relying on just one of them is not enough when it comes to delivering messages to attacked nodes (push) or receiving messages from attacked nodes (pull).

Figure 4 compares the propagation times for Drum and Adaptive Drum. The figure shows the number of rounds it takes a message to reach all the nodes in the system for the worst experiment. That is, each data point is the minimal round number for which in all 100 experiments all nodes had the message. We can see that Adaptive Drum is constantly better than Drum, when there is an attack (recall that when there is no attack present, Adaptive Drum and Drum are exactly the same). Adaptive Drum improves the propagation time by $13\%$ to $34\%$ compared to Drum. Also, the improvement becomes more significant as the percentage of attacked nodes increases.




(a) $C_{push} = 1{,}000$, $C_{pull} = 1{,}000$. (b) $C_{push} = 1{,}000$, $C_{pull} = 100$. (c) $C_{push} = 100$, $C_{pull} = 1{,}000$.

**Figure 3: Message propagation under attack, $\alpha = 40\%$.**

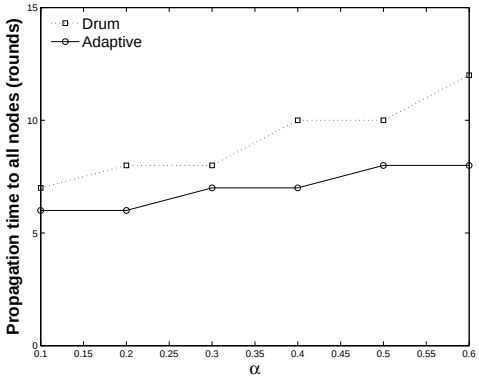



**Figure 4: Propagation times of Drum vs. Adaptive Drum, $C_{push} = C_{pull} = 1{,}000$.**

## 5.2 Estimation Evaluation

We proceed to evaluate the estimation mechanisms. In this set of experiments, nodes estimate $\alpha$, $p_s$ and $p_l$ each round, and adjust their fan-ins and fan-outs for the next round according to the average of the last $50$ estimations. The adversary attacks $40\%$ of the nodes, with $C_{push} = C_{pull} = 1{,}000$. The attack begins in round $0$. It is reasonable to assume that only a portion of the nodes is attacked, as the adversary may not even know about most of the nodes. All results presented are of a single experiment, for two nodes chosen at random. $100$ experiments and several nodes were tested, and all provided similar results.

Figures 5(a) and 5(b) show the estimation of $\alpha$ as performed by a randomly-selected attacked node (Figure 5(a)), and a randomly-selected unattacked node (Figure 5(b)). We can see that in both cases the nodes get a very close average estimation of $\alpha$. Once $50$ rounds pass and there are $50$ estimations of the attack, the average estimation virtually stays the same.

Figure 5(c) shows the estimation of $p_s$ as performed by a randomly-selected attacked node (the results for an unattacked node are similar). The node estimates that $p_s \approx 1$, which fits the calculation of $p_s$ performed in Section 5.1 for Figure 3(a). The results for $p_l$ are similar, and thus we do not show them here.

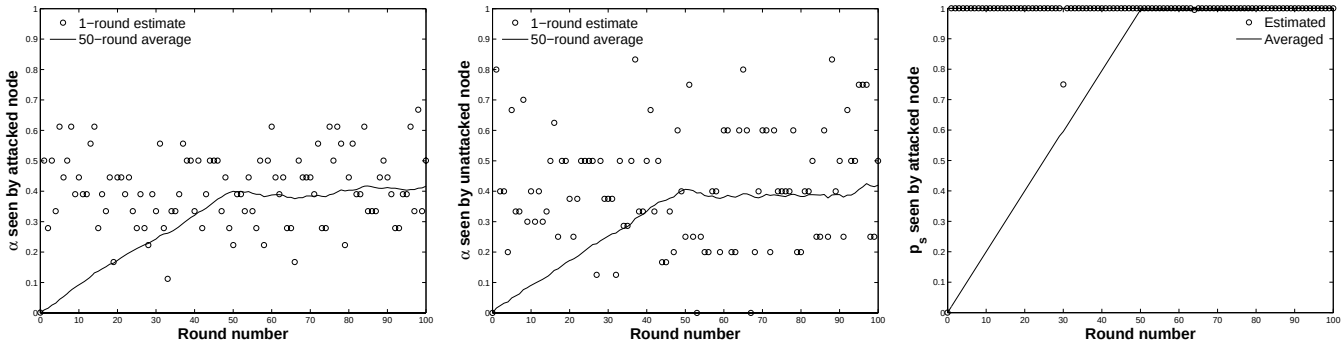
The application of the average estimations shown in Figure 5 is presented in Figure 6(a), which shows the push fan-ins resulting from solving the optimization problem using the average estimations (the results for the pull fan-ins are similar). The fan-ins presented are used by a randomly-selected attacked node, and a randomly-selected unattacked node. Since the average estimations were fairly accurate, the resulting fan-ins are the same ones used when the attack is known (cf. data point for $\alpha = 0.4$ in Figure 1(b)). Thus, we get that using local decisions and incomplete knowledge at each node, the whole system adapts itself to using the fan-ins (and thus also the fan-outs) that solve the optimization problem when the attack parameters are fully known.

Since the adversary takes time to realize that the system has adapted its behavior and inform all the zombies to change the attack strategy, our protocol should resist even attackers that change their attack strategy. We chose to measure the average for the last $50$ rounds, and indeed, after $50$ rounds the system stabilizes. Essentially, parts of the system may stabilize before others do, e.g., the attacked nodes reach their final fan-ins immediately, since as soon as they sense an attack they drop the allocated resources for their incoming channels to $0$ (see in Figure 1). Thus, the propagation time can be improved even before $50$ rounds pass. Averaging can also be made on less than $50$ rounds, to reach the final strategy faster. Either way, rounds are short in nature (may be less than a second), and $50$ rounds only take several seconds.

Figures 6(b) and 6(c) examine the use of various averages for the estimation of $\alpha$. The figure shows the average estimated value of $\alpha$ for different numbers of data points per average. Figure 6(b) shows the averages as calculated for a randomly-selected attacked node, and Figure 6(c) shows the calculated averages for an unattacked node. These figures correspond to Figures 5(a) and 5(b), respectively. We can see that the smaller the length of the average, the more it fluctuates, although it reaches the area of $\alpha = 0.4$ more rapidly. The fluctuations are less evident for the attacked node, since the attacked node has its outgoing channels at full capacity, and thus gets more samples for the estimation. In contrast, the unattacked node is mainly focusing on the incoming channels, so the little resources it uses for the outgoing channels provide him with little information for the estimation.

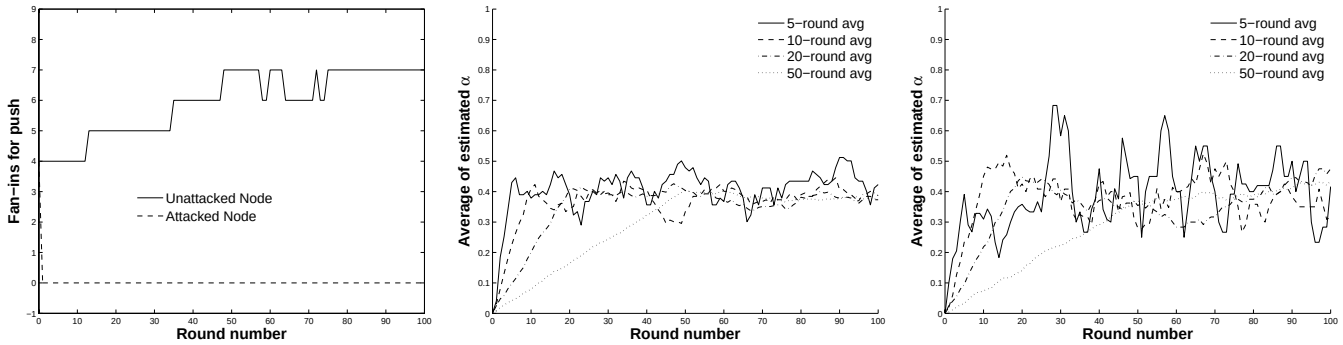
# 6. CONCLUSIONS

We presented a novel approach to dealing with DoS attacks – adapting the protocol's behavior according to the perceived attack. Adaptation is done locally at each node, but a global improvement is achieved. The adaptation is based on a set of constraints that compose an optimization problem, which is solved using linear programming. Our simulations showed that in our case study adaptation increases performance by up to $34\%$. We believe that our work is the first step in designing adaptive protocols that deal with DoS attack better than static protocols.




(a) Attacked node's estimation of $\alpha$. (b) Unattacked node's estimation of $\alpha$. (c) Attacked node's estimation of $p_s$.

**Figure 5: Estimations of $\alpha$ and $p_s$, $C_{push} = C_{pull} = 1{,}000$, $\alpha = 0.4$.**




(a) Push fan-ins. (b) Attacked node's estimation of $\alpha$. (c) Unattacked node's estimation of $\alpha$.

**Figure 6: Estimation/Adaptation results, $C_{push} = C_{pull} = 1{,}000$, $\alpha = 0.4$.**

# 7. REFERENCES


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

[2] P. T. Eugster, R. Guerraoui, S. B. Handurukande, A.-M. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. *TOCS*, 21(4):341–374, 2003.

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

[4] P. Kyasanur, R. R. Choudhury, and I. Gupta. Smart gossip: an adaptive gossip-based broadcasting service for sensor networks. In *MobiHoc*, pages 91–100, 2006.

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

[6] L. Rodrigues, S. Handurukande, J. P. U. do Minho, R. Guerraoui, and A.-M. Kermarrec. Adaptive gossip-based broadcast. In *DSN*, pages 47–56, 2003.