2008

'Won't You Be My Neighbor?' Neighbor Selection Attacks in Mesh-based Peer-to-Peer Streaming

Jeff Seibert

Davied Zae

Sonia Fahmy
Purdue University, fahmy@cs.purdue.edu

Cristina Nita-Rotaru
Purdue University, crisn@cs.purdue.edu

Report Number:
08-004

WON'T YOU BE MY NEIGHBOR? NEIGHBOR SELECTION ATTACKS IN MESH-BASED PEER-TO-PEER STREAMING

Jeff Seibert
David Zage
Cristina Nita-Rotaru

Department of Computer Science
Purdue University
West Lafayette, In 47907

CSD TR #08-004
January 2008

"Won't You Be My Neighbor?" Neighbor Selection Attacks in Mesh-based Peer-to-Peer Streaming

Jeff Seibert, David Zage, Cristina Nita-Rotaru
Department of Computer Science, Purdue University
E-mail:{jcseiber, zagedj, crisn}@cs.purdue.edu

Abstract

P2P streaming has grown in popularity, allowing people in many places to benefit from live audio and television services. Mesh-based P2P streaming has emerged as the predominant architecture in real-world use because of its resilience to churn and node failures, scalability, and ease of maintenance. The proliferation of these applications on the public Internet raises questions about how they can be deployed in a secure and robust manner. Failing to address security vulnerabilities could facilitate attacks with significant consequences such as content censorship, unfair business competition, or external impact on the Internet itself. In this paper, we identify and evaluate neighbor selection attacks against mesh-based P2P streaming which allow insider attackers to control the mesh overlay formation and maintenance. We demonstrate the effect of the attacks against a mesh-based P2P streaming system and propose a solution to mitigate the attacks. Our solution is scalable, has low overhead, and works in realistic heterogeneous networks. We evaluate our solution using a mesh-based P2P streaming system with real-world experiments on the PlanetLab Internet testbed and simulations using the OverSim P2P simulator.

1 Introduction

Peer-to-peer streaming is rapidly maturing into a technology that offers scalable delivery of audio and video content over the Internet. Several commercial efforts are exploring the use of peer-to-peer systems for live media streaming [46, 40, 42, 53, 43, 23, 34, 38, 41, 55, 57, 19, 50, 54, 27, 62]. High user demand for these systems is demonstrated by their deployment in several countries [62, 59] and their increasingly large user base. For example, studies conducted for PPLive [40], one of the most popular systems today with over 350 channels, showed that it has been experiencing up to 400,000 simultaneous users, even having 200,000 simultaneous users watching a single channel [26]. Recent Internet studies also indicate that over 60% of traffic is dominated by P2P systems [13], with video accounting for more than one-third of all Internet traffic today [32].

Significant research has been conducted on the design and implementation of multicast overlay architectures as a platform for P2P streaming [11, 37, 56, 63, 28]. Typically, such architectures have a neighbor selection component that enables new nodes to join the overlay and a data plane component that facilitates the flow of data. Based primarily on the structure of the data plane, two architectures for Internet P2P streaming have emerged in recent years: tree-based and mesh-based architectures.

A tree-based streaming overlay constructs a tree in which the source broadcasting the stream is the root of the tree and every other peer in the network is a child of either the source or another peer. While single tree-based routing topologies are simple and efficient, they may experience an imbalance in the load on the peers as leaf nodes do not forward data. Multi-tree overlays were introduced to distribute bandwidth costs across participants by disseminating the data on multiple separate trees. Examples of single-tree multicast overlays include ESM [15] and multi-tree multicast overlays include Chunkyspread [56] and SplitStream [11].

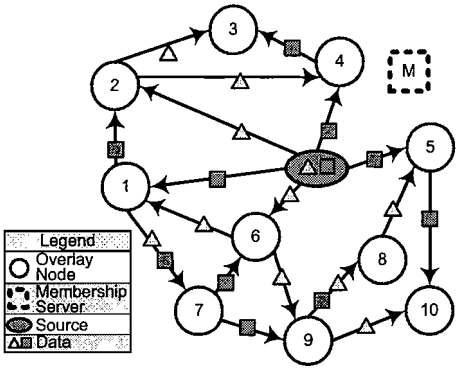


Figure 1: Example of a unidirectional mesh-based streaming overlay in which the source sends two different data chunks as denoted by the gray triangle and orange square. Each node has an *in-neighbors* set and an *out-neighbors* set. For example, for node 6, the *in-neighbors* set consists of node 7 and the source, while the *out-neighbors* set consists of nodes 1 and 9.

A mesh-based streaming overlay facilitates data dissemination in a less structured manner where peers exchange data with a subset of the nodes in the network which is initially obtained from some *membership server*. The major difference between tree-based and mesh-based systems is that in mesh-based systems there is no predefined route in which data flows. Initial mesh-based systems used bidirectional meshes where data can be sent and received in any direction. To capitalize on heterogeneous bandwidth capabilities and at the same time not penalize low bandwidth nodes, recent systems use unidirectional meshes where data is sent in only one direction (as seen in Figure 1). In unidirectional meshes each node maintains two sets of neighbors, an *in-neighbors* set and an *out-neighbors* set. Nodes receive data from nodes in their *in-neighbors* set and send data to nodes in their *out-neighbors* set. A node decides on the number of *out-neighbors* it can accept based on its available bandwidth. The number of *in-neighbors* for a node is usually a system constant. Examples of mesh-based multicast overlays include Chainsaw [37] and CoolStreaming/DONet [63]. Several highly popular P2P streaming systems, such as PPLive [40] and PPStream [42], extend ideas from the mesh-based BitTorrent [17] system for real-time streaming. Examples of unidirectional meshes include Meshcast [10], and Prime [31]. The mesh-based architecture has emerged as the predominant architecture today because meshes are characteristically resilient to churn and node failures, while unidirectional meshes efficiently use the heterogeneous bandwidth capabilities existent in real-world P2P deployments. Furthermore, studies have shown in simulations [31] and real-world experiments [44] that mesh-based overlays perform better than tree-based overlays.

The proliferation of P2P streaming applications on the public Internet and the extrusion of multicast functionality to end-systems that are more likely to be compromised than core routers [1], raises questions about how P2P streaming can be deployed in a secure and robust manner. P2P streaming applications have several characteristics that differentiate them from other P2P applications and make them an attractive target for attack. First, since these applications mimic television, users expect to see and hear programs quickly and without choppiness. Recent work showed that P2P streaming users are impatient [58], implying that when choppiness occurs, users will change the channel or even stop using the system. Second, P2P streaming applications have a real-time nature requiring audio and video packets to meet deadlines in order to be considered useful. Finally, these applications are bandwidth intensive, resulting in significant traffic being received by a user. An attacker can exploit these characteristics to conduct attacks that may result in targeted censorship of the content being broadcast, making particular channels unusable, or making the broadcast unavailable in particular locations. With the commercial potential of such applications, one may

expect unfair business competitors to utilize such attacks to decrease the client base of their competitors. Indeed, P2P file sharing applications have been under similar attacks in the past, as files can easily be polluted on such systems [29]. Recent work has also shown that P2P streaming applications are extremely vulnerable to pollution attacks themselves [21]. Finally, such attacks may be used by malicious entities to conduct DDoS against hosts external to the P2P overlay and impact the Internet itself, as it has been recently demonstrated in [35, 20, 51] by using real-life P2P systems.

In this paper, we focus on attacks against neighbor selection in unidirectional mesh-based P2P streaming applications. We focus on the mesh-based architecture as it is widely used and on the unidirectional variant as it efficiently capitalizes on bandwidth heterogeneity existent in real-world P2P deployments due to the presence of DSL, cable modem, and Ethernet links [48, 14, 9, 52]. An attacker can exploit the neighbor selection mechanism to be chosen to provide service for many nodes in the system. For example, in many systems new nodes select the neighbors they will obtain service from by requesting the neighbor sets of already known nodes. Attackers can dominate the honest nodes' neighbor sets by only referring other malicious nodes to honest nodes. As a result, an attacker can influence the overlay construction and maintenance, controlling a significant part of the overlay and consequently the overlay traffic. This facilitates further attacks such as selective data forwarding, cheating, traffic analysis, overlay partitioning, or denial of service against target nodes not even part of the overlay. Some of these attacks such as selective data forwarding or denial of service may ultimately be noticed by the victim so they can be effectively addressed by deploying a posteriori detection mechanism. However, other attacks such as traffic analysis, do not have immediately observable results. It is thus critical to address attacks against neighbors selection as they are the root of those secondary attacks and their mitigation represents a general mechanism to prevent or mitigate a larger class of attacks.

While significant research has been done on proper design and implementation of P2P streaming systems, less work has been conducted in studying their vulnerabilities [25]. Attacks against overlay formation and exploiting neighbor selection were previously identified in the context of structured overlays such as Distributed Hash Tables (DHTs) for file sharing [12]. However, none of the previously proposed solutions are applicable to mesh-based P2P streaming applications operating in realistic heterogeneous networks. Specifically, solutions relying on built-in invariants in the overlay structure to restrict neighbor selection [12] do not work in mesh-based overlays that are unstructured in nature. Solutions assuming homogeneous networks which restrict the size of in-neighbor and out-neighbor sets [45] do not work in heterogeneous networks where nodes have different values for these sets due to different bandwidth capabilities. Induced-churn solutions [18] have a high cost that makes them prohibitive for real-time streaming. Finally, authentication mechanisms do not protect against the identified attacks as end user systems are often easily compromised [3] and attackers can quickly gain new attack vectors [4].

Our paper has the following contributions:

- We identify and evaluate the effectiveness of neighbor selection attacks on unidirectional mesh-based P2P streaming. We identify methods to amplify the attack and the conditions under which the attacks have the greatest effect. Our results indicate the number of attackers, polluting the neighbor list of the membership server, the placement strategy of the attacker, and presence of churn are all factors which amplify the effects of the attack. We demonstrate that attacking a system under churn has a crippling effect on the performance, resulting in 20% of the original performance achieved by a stable system when no attack occurs. Finally, targeting the source by attempting to place malicious nodes as *out-neighbors* of it is the most effective strategy for an attacker to attack a stable system.
- We propose a novel solution for the defense and mitigation of neighbor selection attacks. Our solution is based on the observation that the attacks change the graph properties of the overlay topology. Specifically, we use the clustering coefficient of a node [61] to limit the selection of malicious nodes as neighbors. Our solution is scalable, has low overhead, and works in heterogeneous environments.
- We demonstrate the effectiveness of our mechanism through real-world experiments on the PlanetLab

[39] Internet testbed using the Chainsaw [37] system. In addition, we investigate the scalability of our method through simulations using the OverSim [7] simulator which provides the ability to deploy overlays with sizes beyond those available on PlanetLab.

The rest of the paper is organized as follows: We provide an overview of mesh-based peer-to-peer streaming overlays in Section 2 and attacks against them in Section 3. We propose mitigation mechanisms in Section 4. We present experimental results demonstrating the impact of the attacks and the effectiveness of our solutions in Section 5. We overview related work in Section 6 and conclude our work in Section 7.

2 System Model

In this section, we present the mesh-based P2P architecture considered in this work. The model is general enough to capture the neighbor selection process of many of the mesh-based systems used today such as [40, 42, 55].

We consider a unidirectional overlay mesh consisting of a source, one or more membership servers, and peer nodes. The source (sometimes known as the seed) initiates the broadcast traffic while the membership servers are used to help a new node to join the overlay. Note that the membership server just supplies a list of possible neighbors and does not track which nodes are actually live. Figure 1 presents an example of a unidirectional mesh with one source and one membership server. Each node maintains three lists of nodes, a list of *in-neighbors*, a list of *out-neighbors*, and a list of *candidates*. The set of *in-neighbors* of a node v consists of the nodes providing service for v. The set of *out-neighbors* of a node v consists of the nodes v is providing service for. The set of *candidates* of a node v consists of nodes which v can reach in the overlay and represent a pool of potential *in-neighbors* for v. A node may have multiple *in-neighbors* and *out-neighbors*. For example, for node 6, the *in-neighbors* set consists of node 7 and the source, while the *out-neighbors* set consists of nodes 1 and 9.

Algorithm 1: In-Neighbor Selection algorithm executed by a node v to select its *in-neighbors* set I. The code is executed when v receives an *in-neighbors* set N from node w. The target and minimum size of the *in-neighbors* set are INS_THRESHOLD and MIN_INS_THRESHOLD, respectively. The *out-neighbor* set of node v is O.

Input: N, w's list of in-neighbor nodes
Output: I, v's list of in-neighbor nodes

```
1  neighbors = I ∪ O;
2  foreach n_j in N do
3      if (sizeof(I) >= INS_THRESHOLD) then
4          break;
5      end
6      if (neighbors contains n_j) then
7          continue;
8      end
9      Request n_j to be in I;
10     if (n_j granted the request) then
11         add n_j to I;
12     end
13 end
14 if (sizeof(I) < MIN_INS_THRESHOLD) then
15     Contact membership server M and request more candidates;
16 else
17     Choose a node p from neighbors that has not been asked recently and request its in-neighbor set;
18 end
```

In order to join the mesh, a new node v contacts a membership server (Node M in Figure 1), which responds with a random subset of nodes $N = \{n_1, \ldots, n_n\}$ that are currently in the system. This represents

Algorithm 2: Out-Neighbor Selection algorithm executed by a node v to decide if it can add node n to its *out-neighbors* list O. The *in-neighbors* list of v is I and the maximum number of *out-neighbors* is MAX_ONS_THRESHOLD, set based on the available bandwidth of v.

Input: A node n that wants to be an out-neighbor and O the list of out-neighbors
Output: True if node is accepted, false otherwise

```
1 if (sizeof(O) >= MAX_ONS_THRESHOLD) then
2 |   return false;
3 else
4 |   add n to O;
5 |   send list I to n;
6 |   return true;
7 end
```

the initial *candidate* set of v. As nodes can leave, crash or become saturated and not able to service other nodes, this list of potential candidates changes over time. When a node requires more *in-neighbors* but the candidate list is too small, it contacts the membership server or a random node and requests more nodes. Additionally, when a node n_j accepts the request to be an *in-neighbor* for v, it will also provide v with a list of its current *in-neighbors*.

Algorithm 1 describes the method in which a node v selects its *in-neighbors*. After obtaining an initial list N from a membership server, v contacts each node n_j in N to request node n_j to provide service to it. If a node n_j accepts to provide service for v, v will add n_j to its list of *in-neighbors*. This process continues until node v has the target number of *in-neighbors* (usually a system constant) or no other potential *in-neighbors* candidates are available. Similarly, Algorithm 2 presents the method a node n_j uses to accept a node to its *out-neighbors* list.

We assume heterogeneous links. Each node independently decides how many *out-neighbor* nodes it can support based on the available bandwidth. The source streams data in chunks to all *out-neighbor* nodes connected to it. Due to the real-time nature of the streaming overlays, all data packets not received within a predefined time limit will be discarded as they are unusable by the application.

3 Neighbor Selection Attack

3.1 Attacker Model

We consider a constrained - collusion Byzantine adversary model. Specifically, we assume given a system of size S, the system contains a bounded percentage of malicious nodes f ($0 \leq f < 1$) behaving arbitrarily. The set of malicious nodes may collude. We assume a malicious adversary has access to all data at a node as any legitimate user would (insider access), including cryptographic keys stored at a node. This access can be the result of the adversary bypassing the authentication mechanisms or compromising a node through other means. Nodes cannot be completely trusted although they are authenticated. We assume that data authentication and integrity mechanisms are deployed and we focus only on neighbor selection attacks.

3.2 Attacks Description

We consider neighbor selection attacks as a primary mechanism for an attacker to control the mesh formation and maintenance. The goal of the attack is to enable a malicious node or coalition of malicious nodes to be selected to provide service to as many honest nodes as possible. This can be achieved if malicious nodes are selected as *in-neighbors* for the victim nodes. To control the *in-neighbor* selection, an attacker attempts to infiltrate and dominate the neighbor sets of as many honest nodes as possible. Once malicious nodes join the overlay, they accept as many nodes as they can as *out-neighbors*, possibly lying about how many *out-neighbors* they can support.

Malicious nodes may dominate the neighbor set of honest nodes by polluting the membership server or by lying about their *in-neighbors* and only referring other malicious nodes to honest nodes in either a random or strategic localized fashion. We refer to these attacks as *membership server pollution*, *pandemic* and *outbreak* attacks. These attacks are epidemic in nature since honest nodes will unintentionally refer malicious nodes that are in their neighbor set to other honest nodes.

Membership server pollution attack: The membership servers are critical points for bootstrapping the mesh overlay. One of the strongest attacks against a mesh-based streaming system is to pollute the neighbors list maintained by the membership server. As almost any new node will first contact the membership server, by contaminating it, an attacker increases its chances of being selected as an *in-neighbor*. Malicious nodes can accomplish this by performing a Sybil [22] attack on the membership server, polluting its neighbor list by registering many times with different identities.

Outbreak attack: Outbreak attacks are localized attacks that can be effectively conducted by small coalitions of malicious nodes. In an outbreak attack, malicious nodes always provide the same list of *in-neighbor* nodes to honest nodes. Such an attack allows the malicious nodes to better control the target of the attack. An attacker would choose to perform an outbreak attack when it only knows a few other malicious nodes, it wishes to appear benign, or it wishes to conceal the identities of other malicious nodes.

Pandemic attack: Pandemic attacks broadly impact the system and may require a larger set of compromised nodes than outbreak attacks. In a pandemic attack, malicious nodes select a random subset of malicious nodes every time they are queried to provide their list of *in-neighbors*. An attacker may perform a pandemic attack to mask malicious behavior as it reports a random subset of peers as its neighbors.

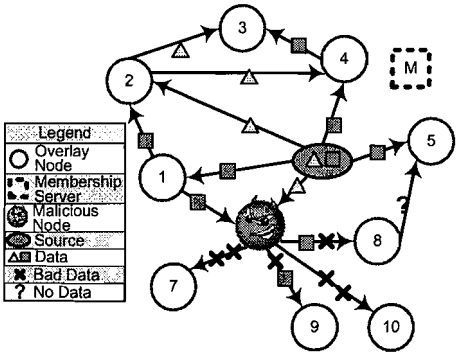
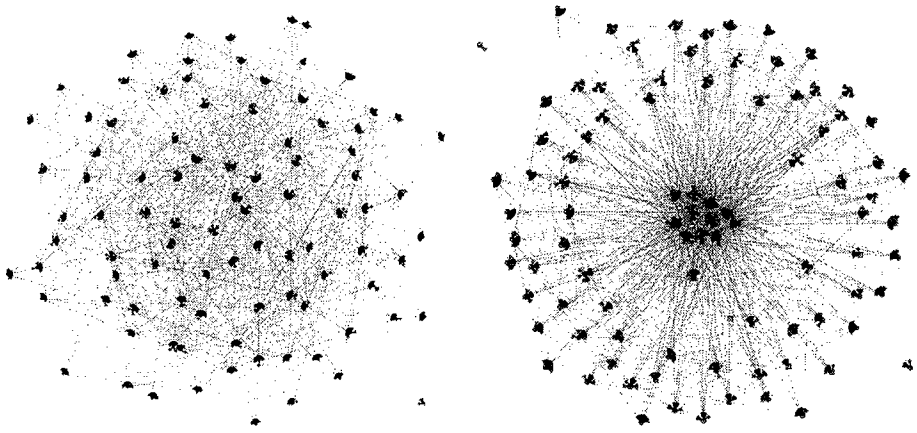


Figure 2: Example of a unidirectional mesh-based streaming overlay under attack. After conducting a neighbor selection attack, the malicious node was selected as an *in-neighbor* by nodes 7, 8, 9 and 10. If the malicious node decides to use this position to further attack the system and drop the traffic, nodes served by the malicious node will not receive the streaming data.

An attacker can distribute the attack across multiple nodes in the network or focus on particular nodes, including the source. Controlling the data flow facilitates further attacks such as selective data forwarding, cheating, traffic analysis, overlay partitioning, or denial of service against target nodes not even part of the overlay. Figure 2 depicts an attack in which the malicious node managed to be selected to the *in-neighbors* set of nodes 7, 8, 9 and 10, thus controlling the traffic to these nodes. If it decides to selectively forward data, then nodes 7, 8, 9 and 10 served by the malicious node will not receive the streaming data.

4 Mitigating Attacks

Neighbor selection attacks are conducted by compromised nodes. As a result, defense techniques against such attacks cannot rely exclusively on cryptographic mechanisms since an adversary may have complete access to all the keys stored on a compromised node. In addition, although attackers can use their control



(a) no malicious nodes (b) 20% malicious nodes

Figure 3: Connections of a Chainsaw experiment 100 seconds into an experimental run on PlanetLab. The experiment was conducted with an overlay of 100 nodes for demonstrative purposes in order to have less cluttered graphs. The same trend was noticed for larger overlay sizes. Note that in Figure 3(b) that the presence of an attacker creates a hub of malicious nodes instead of a random graph structure as seen in 3(a)

of the neighbor set to conduct further attacks such as selective data dropping, cheating, overlay partitioning, traffic analysis, our solution does not assume observable effects of these secondary attacks, since not all of them are observable by an honest node. Instead, our solution focuses only on directly observed results from the presence of malicious nodes in the neighbors set, specifically the changes created in the overlay characteristics.

One way to approach the problem is to identify invariants in the node placement in the overlay and use deviations from such invariants to detect malicious nodes and restrict their location. This method was used successfully in structured overlays such as DHTs. Unlike DHTs, mesh-based streaming systems are unstructured in nature and do not contain such invariants. Recent studies [5, 58] of mesh-based streaming systems noted that many of them use a data plane that appears to be constructed randomly. This was also confirmed by several deployments we conducted with Chainsaw, a P2P mesh-based streaming system. This random structure can be seen in Figure 3(a), which presents a 100 nodes deployment of the Chainsaw system on the PlanetLab Internet testbed.

Other possible defense techniques, such as the membership server periodically dropping nodes from its membership list, may actually have a detrimental effect on the system. The only time a node contacts the membership server is upon initially joining the network in order to receive a set of potential neighbors. If a benign node is dropped from the membership list, it will never be selected as a potential neighbor and the utility of the system will drop. Furthermore, malicious nodes may continually reintroduce themselves to the membership server. Thus, this defense mechanism can actually cause the membership list to contain a higher percentage of malicious nodes.

Our solution relies on the observation that the presence of attackers changes the properties of the graph formed by the overlay. Since malicious nodes are attempting to place themselves in as many neighbor sets as possible, this intuitively suggests that the resulting graph will be more connected and less random. Stronger attacks will result in more significant changes with some attackers being extremely connected and acting as hubs in the network. For example, Figure 3(b) shows the same 100 nodes Chainsaw deployment on PlanetLab when 20% percent of the nodes are malicious and conducting a *membership server pollution attack* combined with an *outbreak attack*. As can be seen, the malicious nodes distort the overlay graph structure and become the hub of the entire network.

Algorithm 3: Clustering Coefficient Computation algorithm used by the Modified In-Neighbor Selection algorithm. The threshold for the clustering coefficient is CC_THRESHOLD set based on the maximum size of *in-neighbors* set and the size of the overlay. The *in-neighbor* set is I and the *out-neighbor* set is O.

Input: A node v's neighbors and v's neighbors in-neighbor sets
Output: A node n_i to disconnect if the clustering coefficient is too high, null otherwise

```
1  neighbors = I ∪ O;
2  edges = 0;
3  foreach n_j in neighbors do
4  |  n_j.edge_count = 0;
5  end
   // Compute clustering coefficient
6  foreach n_j in neighbors do
7  |  foreach n_i in n_j.I do
8  |  |  if (neighbors contains n_i) then
9  |  |  |  n_i.edge_count++;
10 |  |  |  edges++;
11 |  |  end
12 |  end
13 end
14 clustering_coefficient = edges / (sizeof(neighbors)*(sizeof(neighbors) - 1));
15 if (CC_THRESHOLD < clustering_coefficient) then
   |  // Select node that contributes most to clustering coefficient to disconnect
16 |  return node n_i with greatest edge_count;
17 else
18 |  return NULL;
19 end
```

We propose to measure how connected an honest node will become when adding a malicious node as an *in-neighbor* by using the clustering coefficient introduced in [61]. The clustering coefficient is computed for a node with respect to its neighbors by dividing all the existent edges in its neighborhood graph by the total number of possible edges. Consider a unidirectional mesh-based streaming modeled as a directed graph $G = (V, E)$ where V is the set of nodes in the system and E is the set of ordered pairs of vertices denoting edges. Then, the clustering coefficient for v_i can be computed as follows:

$$C_i = \frac{|\{e_{jk}\}|}{k_i(k_i - 1)} : v_j, v_k \in V, e_{jk} \in E \tag{1}$$

where $|\{e_{jk}\}|$ denotes the cardinality of the set of edges between nodes which are neighbors of node v_i, and $k_i(k_i - 1)$ represents the total number of possible edges that could exist among the vertices within the neighborhood where k_i is the total degree of vertex v_i.

As a graph becomes more random, there exists fewer edges between neighbors of a particular node v and thus the clustering coefficient of v approaches zero. Because malicious nodes place themselves in as many neighbor sets as possible, their presence will raise the clustering coefficient due to an increase in the number of links between neighbor nodes. By monitoring the size of their clustering coefficient, honest nodes can detect malicious nodes and remove them from their *in-neighbors* set.

Algorithm 4 presents the modified *in-neighbors* selection algorithm using the clustering coefficient computation described in Algorithm 3. The original *in-neighbors* selection is modified such that a node can detect if the addition of some node on the *in-neighbors* lists raises the clustering coefficient over a certain threshold. If the clustering coefficient is too high, the node disconnects the neighbor that occurs most frequently in all of its neighbors' in-neighbor sets and blacklists it for the time being. Each node periodically

Algorithm 4: Modified In-Neighbor Selection algorithm executed by a node v to select its *in-neighbors* set I when it receives an *in-neighbors* set N from node w. The target and minimum size of the *in-neighbors* set are INS_THRESHOLD and MIN_INS_THRESHOLD, respectively. The maximum number of requests made from a single node's neighbor set is MAX_REQUESTS. The *out-neighbor* set of node v is O. The delay between algorithm iterations is TIME_OUT.

Input: N, w's list of in-neighbor nodes
Output: I, v's list of in-neighbor nodes

```
1  neighbors = I ∪ O;
2  if (neighbors contains w) then
3  |  w.past_neighbors = w.past_neighbors ∪ N;
4  |  w.in_neighbors = N;
5  end
6  while ((n_i = compute_clustering_coefficient()) != NULL) do
7  |  disconnect n_i and any n_i.past_neighbors that we are connected to;
8  |  add n_i and n_i.past_neighbors to blacklist;
9  end
10 requests = 0;
11 foreach n_j in N do
12 |  if ((sizeof(I) >= INS_THRESHOLD) || (requests >= MAX_REQUESTS)) then
13 |  |  break;
14 |  end
15 |  if ((blacklist contains n_j) || (neighbors contains n_j) || (n_j is already connected to more than one peer in
   |  neighbors)) then
16 |  |  continue;
17 |  end
18 |  requests ++;
19 |  Request n_j to be in in_neighbor set I;
20 |  if (n_j granted the request) then
21 |  |  add n_j to I;
22 |  end
23 end
24 sleep TIME_OUT; if (sizeof(I) < MIN_INS_THRESHOLD) then
25 |  Contact membership server M and request more candidates;
26 else
27 |  Choose a node p from neighbors that has not been asked recently and request its in-neighbor set;
28 end
```

probes each of its neighbors asking for up-to-date in-neighbor sets. As in the original neighbor selection algorithm (presented in Algorithm 1), every time a node adds a new node to its *in-neighbors* list, it requests the neighbor set of the new node.

Resiliency of the defense mechanism. The defense mechanism is itself resistant to malicious activity for several reasons. First, if a malicious node reports incorrect neighbor sets while being in the *in-neighbor* set of other benign nodes, it will be identified and disconnected. Secondly, a malicious node can attempt to falsely implicate other benign nodes. For example, a malicious node could mimic a benign node's neighbor set, thus raising the benign node's clustering coefficient. These types of attacks can be prevented by a simple augmentation to the defense mechanism that disconnects nodes with similar neighbor sets. Finally, malicious nodes may collude and choose a target benign node to include in all of their *in-neighbor* sets. In doing this, the malicious collective tries to force other benign nodes to disconnect from the targeted node and add a malicious neighbor instead. However, it takes time for the updated neighbor sets to propagate and there is no guarantee that benign nodes will select a malicious neighbor as a replacement, thus making the attack marginally effective.

5 Experimental Results

We demonstrate through experimental results the identified attacks and the effectiveness of our solution in the context of the Chainsaw [37] mesh-based P2P streaming system. As many of the existing P2P streaming systems are proprietary, we selected Chainsaw because its neighbor selection mechanism is very similar with the one used by other mesh-based P2P systems and we were able to obtain a copy of it. We conduct real-life experiments on the PlanetLab [39] Internet testbed. In addition, to validate the scalability of our method, we conducted simulations with overlay sizes beyond those that can be deployed on PlanetLab using the OverSim [36] P2P simulator. We selected OverSim because it offered increased scalability and flexibility. Below, we provide an overview of Chainsaw, describe our experimental methodology, and then present an evaluation of the attacks and of our solution technique.

5.1 Overview of Chainsaw

Chainsaw is a mesh-based P2P streaming system using a pull-based approach in which nodes request packets from a set of peer nodes, the *in-neighbor* set. We have modified Chainsaw so that it constructs a unidirectional mesh instead of a bidirectional mesh. This provides support for nodes with larger bandwidth capabilities to accept more nodes in their *out-neighbor* set. The node join and neighbor selection are as described in Section 2.

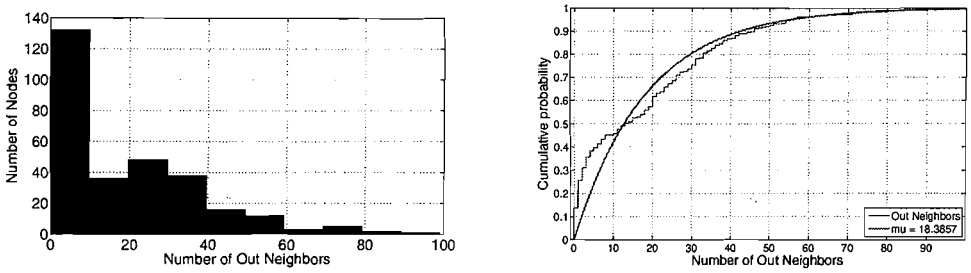
When a node receives a new packet, it notifies its neighbors about it. In addition, each node maintains information about the packets available to other peers, referred to as the *window of availability*. This window is a buffer that contains packets that have recently been received and information about which peers were notified. Packets are discarded after a predefined amount of time to prevent old data from being propagated in the overlay. Each node also maintains a list of the packets it is interested in, referred to as the *window of interest*, by tracking the notifications of available packets advertised by each of its neighbors. Based on the window of interest, a node randomly selects packets to request from all available peers.

5.2 Experimental Methodology

In this section, we present our methodology for the experiments and simulations. We discuss the attack scenarios we consider and introduce the metrics we use to evaluate the attacks and the effectiveness of our solution.

PlanetLab experimental methodology. To study the attacks under real-world conditions, we conducted our experiments on the widely-used PlanetLab [39, 16] Internet testbed. PlanetLab provides a research platform for large scale distributed experimentation of peer-to-peer systems over the Internet [47]. In order to mitigate the possible limitations of using a testbed, such as those discussed in [47], we ran several experiments at different times of the day and different days of the week. Further, we randomly selected experimental nodes for different experiments to validate the statistical significance of results and nodes were chosen to span multiple operational and administrative domains. We run each experiment ten times and average the results. Confidence intervals were omitted from the graphs due to their small range (about 30kbps), regardless of the streaming rate.

We used a maximum of 300 nodes in the experiments because this is the largest number of nodes with access bandwidth greater than 1 Mbps that could be reliably accessed. We used streaming bit rates of 400 kbps to 1 Mbps, which are representative of the bit rates used in many current video streaming applications [26]. The source was always located on a host at Purdue University. We configured the source to wait for 30 seconds before starting to send data. We consider that a packet must arrive within 5 seconds to be considered useful, according to the buffer times used in [30, 24, 2]. We configured Chainsaw such that each node uses a minimum of 15 *in-neighbors*, and assumes the request for a packet is lost after 1 second.



(a) (b)

Figure 4: *Out-neighbor* heterogeneity of a Chainsaw experiment using 300 nodes on PlanetLab, where the number of *out-neighbors* of a node is proportional to the bandwidth capabilities of that node. The average number of *out-neighbors* is $\mu = 18.38$ and the standard deviation is $\sigma = 19.4$. The distribution of the number of *out-neighbors* approximately follows an exponential curve.

Honest nodes decide the number of the *out-neighbors* they support based on their available bandwidth. The source supports 30 *out-neighbors* and pushes two copies of every packet. All Chainsaw parameters above were set based on previous reported deployments as in [37, 44].

OverSim simulations methodology. OverSim [7, 36] is an open-source overlay network simulation framework supporting several models for structured and unstructured P2P protocols and scaling to tens of thousands of nodes.

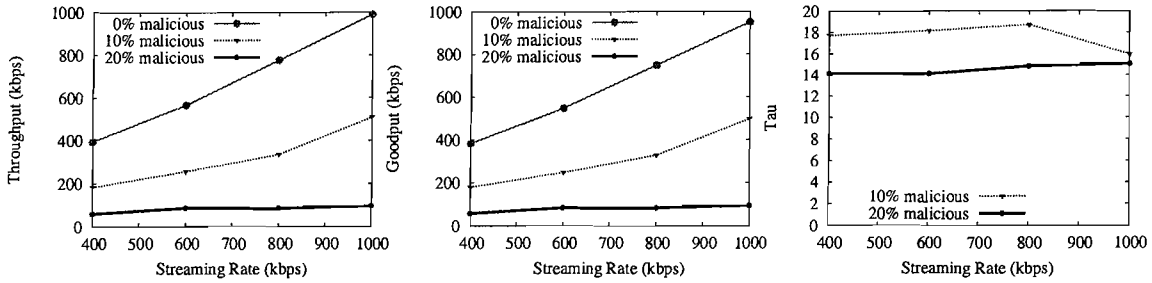
We simulate the join process described in Section 2. In each experiment, nodes join at a rate of 20 per second to allow the system to accommodate the new nodes. Once they joined the overlay, the nodes stay in the overlay until the end of the simulation. To match the Chainsaw setup, nodes attempt to maintain 15 *in-neighbors* at all times. We simulate the number of *out-neighbors* a node will accept based on results observed for Chainsaw deployments on PlanetLab. Figure 4(a) presents the *out-neighbor* heterogeneity of a Chainsaw deployment of 300 nodes on PlanetLab. As depicted in Figure 4(b), the data closely follows an exponential distribution. Based on this observation, we consider that the number of *out-neighbors* a node accepts is based on an exponential distribution with a mean of 20, matching the average number of out-neighbors observed on real deployments of Chainsaw. The duration of each simulation is 10 minutes. We run each simulation ten times and average the results. We do not simulate the actual transfer of data, as we are solely concerned with the composition of the neighbors of a node.

Attack scenarios. We conduct the neighbor selection attacks described in Section 3: *membership server pollution*, *outbreak*, and *pandemic* attacks. In order to isolate the neighbor selection attacks, malicious nodes are configured to truthfully advertise what pieces of the stream they have. However, to emphasize the amount of data controlled by an adversary as a result of the attack, malicious nodes never fulfill any requests, either by dropping the request or by sending back a message saying that the packet is now out of their window of availability. Unlike honest nodes that accept a number of *out-neighbors* based on their available bandwidth, malicious nodes are configured to accept up to 100 out-neighbors.

For the PlanetLab experiments, we consider the combination of membership pollution and outbreak attacks as this is the strongest attack that we consider. Since pandemic attacks are not possible on PlanetLab due to smaller system sizes, we investigate them in OverSim where the larger system sizes allow us to simulate the attacks.

To investigate the impact of attacker placement, we consider three scenarios:

- *General* attack: attackers attempt to get in the neighbor set of any node in the system. This is the easiest way to conduct a neighbor selection attack, but may require a significant number of nodes to generate significant damage.



(a) Throughput - General scenario (b) Goodput - General scenario (c) Tau - General Scenario

Figure 5: System performance and attack strength on PlanetLab for an overlay of 300 nodes when the malicious nodes place themselves among the *in-neighbors* of any node in the overlay (*general* scenario). Malicious nodes conduct *membership server pollution* combined with *outbreak attack.* For demonstrative purposes, malicious nodes also drop the traffic going through them.

- *Source-only* attack: attackers try to isolate the source by becoming the neighbors of the source. Intuitively, this attack allows an attacker to target the content of a particular channel or provider, while requiring less resources (compromised nodes) from the attacker.
- *Peer-only* attack: attackers try to get in the neighbor set of peers other than the source. This attack may be used when the source is not reachable by the attacker.

Metrics: We evaluate the system by using the following metrics:

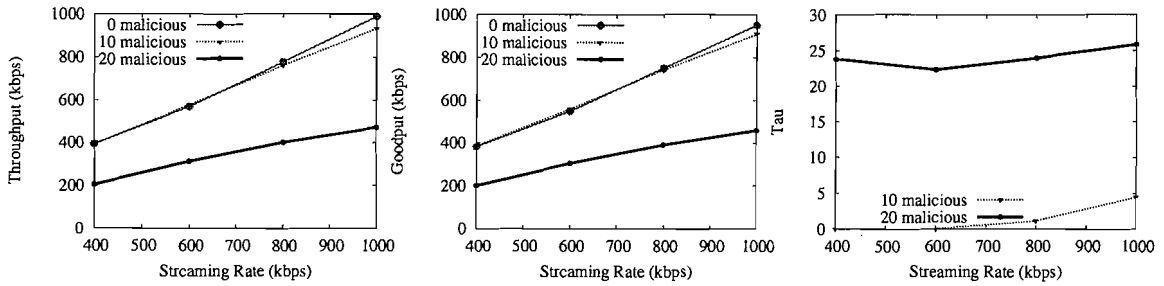
- **Goodput** is the average rate of data that was received before the deadline (5 seconds) and had not been received before. Goodput captures the useful data received by an application.
- **Throughput** is the average rate at which all application data is received. Throughput captures the total amount of data received by the users including data received after the deadline or duplicates.
- **Continuity Index**, defined by Zhang *et al.* [63], is used to measure the effect of churn. It is equal to the goodput divided by the total amount of data that could have possibly been received while a peer participated in the overlay, or $\frac{Goodput}{StreamingRate}$. Ideally, the continuity index should be 1.
- **Tau Attack Strength**, defined by Walters *et al.* [60], is used to measure the relative strength of a particular attack on the system. It is defined as:

$$\tau = \max\left(1000 \times \frac{G_{norm} - G_{adv}}{G_{norm} \times N_{adv}}, 0\right) \quad (2)$$

  where G_{norm} and G_{adv} represent the average goodput in the absence and presence of adversaries, respectively. N_{adv} is the number of adversaries. Intuitively, τ represents the amount of damage per attacker an attack inflicted on the system. The greater the performance degradation observed in the system between non-attack versus attack scenarios (the difference between G_{norm} and G_{adv}), the higher the value of τ and the more damage an attack inflicts on the system.
- **Corruption Factor**, introduced in this paper, measures the level of control an adversary has on the neighbors of a particular node. We define the corruption factor as the percentage of nodes in the neighbor set that are malicious. Ideally the corruption factor should be zero. A corruption factor of 1 indicates that all neighbors of a node are adversarial.

5.3 Attacks Effectiveness

In this section, we demonstrate the attacks using a real P2P streaming system through real-world experiments. Our results seek to determine factors that amplify the attack such as number of the attackers, attacker placement strategy, or presence of churn. We also seek to identify the strategy that provided the strongest damage per adversary's effort. Finally, we show through simulations that the attacks are effective in larger systems as well.



(a) Throughput - Source-only scenario (b) Goodput - Source-only scenario (c) Tau - Source-only scenario

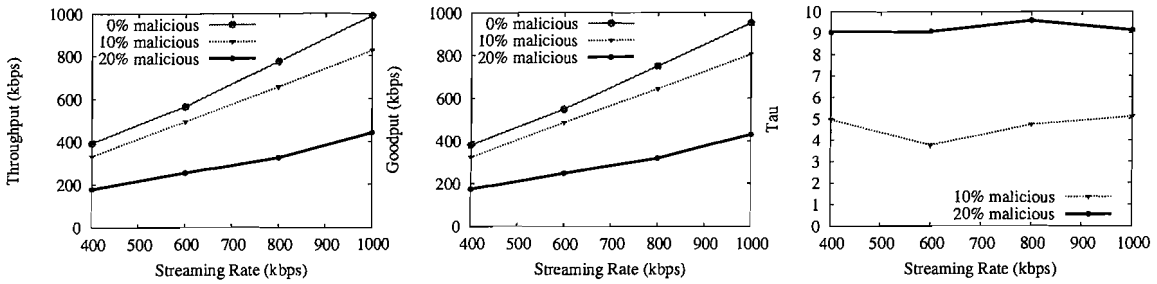
Figure 6: System performance and attack strength on PlanetLab for an overlay of 300 nodes when the malicious nodes place themselves among the *in-neighbors* of the source (*source-only* scenario). Malicious nodes conduct a *membership server pollution* combined with *outbreak* attack. For demonstrative purposes, malicious nodes also drop the traffic going through them.

Impact of number of attackers. To investigate the effect of the number of attackers, we consider the three attack scenarios, *general*, *source-only*, and *peer-only* with varying percentages of attackers. Figure 5(a) and 5(b) plot the throughput and goodput for a deployment of 300 nodes on PlanetLab under the *general* scenario attack. Even with just 10% malicious nodes (30 out of 300 nodes), the perceived quality of the system drops significantly. With 20% of the nodes being malicious, an attacker is able to effectively deny service on the overlay. In the case of the peer-only scenario (Figures 7(a) and 7(b)) the performance degradation is more visible when 20% of the nodes are malicious, suggesting that not including the source in the target nodes decreases the effectiveness of the attack. Figures 6(a) and 6(b) show the performance of the system when the only attacked node is the source. The effect of the attack is significant for different streaming rates and increases with the number of malicious nodes.

We validated that the attack results presented are not a product of system randomness. We formulate the null hypothesis $H_0 : \mu_{attack} = \mu_{normal}$, which states system goodput under non-attack and attack conditions have the same mean and distribution and implies the system behaves similarly under both conditions. Using a two-sample t-test with pooled variance [33], we disproved H_0, finding with high probability (nearly 100%) that the error results come from distributions with different means, which implies the difference observed in the results is due to the presence of malicious attackers.

Impact of attackers' placement strategy. We investigate which attack strategy is more effective in a stable system. Specifically, we are interested in answering the following question: "For the same number of attackers (or resources available to the attacker) which is the most effective strategy?" In Figure 6(a) and 6(b), the graphs depict the exact number of malicious nodes connecting to the source. The *out-neighbors* set size of the source was set to 30. Figures 7(a) and 7(b) depict the system performance when malicious nodes connect only to other peers. The figures demonstrate that the attack is effective in degrading the performance of the system in both cases. Although the absolute performance degradation appears to be similar for the curves "20 malicious" *source-only* and "20% malicious" *peer-only*, the number of attackers is different, 20 nodes in the *source-only* case and 60 nodes (20% out of 300 nodes) in the *peer-only* case. Figure 8 reveals that the attack that inflicts the greatest damage per attacker on the system is when using the *source-only* scenario. The least effective strategy is *peer-only*. We conclude that the attacker receives the "greatest" return per malicious node by targeting the source of the system and is the least efficient when only targeting peer nodes.

Sensitivity to churn. In previous cases we examined the effect of the attack when the system is stable. Previous work has shown that, in general, mesh-based streaming systems perform well under churn [44]. We seek to evaluate the effectiveness of the attack in the presence of churn. We perform a *membership server pollution* attack combined with an *outbreak* attack with a *general* strategy. We first start the overlay comprised of 80 normal nodes and 20 malicious nodes where the malicious nodes are stable and never leave.



(a) Throughput - Peer-only scenario (b) Goodput - Peer-Only scenario (c) Tau - Peer-only scenario

Figure 7: System performance and attack strength on PlanetLab for an overlay of 300 nodes when the malicious nodes place themselves among the *in-neighbors* of nodes other than the source (*peer-only* scenario). Malicious nodes conduct a *membership server pollution* combined with *outbreak* attack. For demonstrative purposes, malicious nodes also drop the traffic going through them.

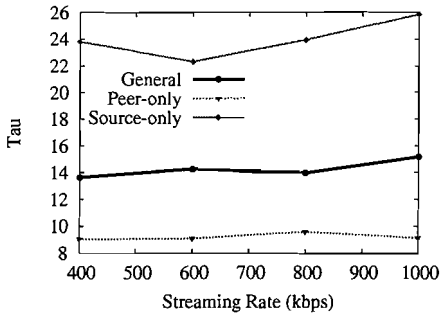


Figure 8: Attack strength for the three attack placement strategies: *general*, *source-only* and *peer-only*, on PlanetLab, for an overlay of 300 nodes when 20% of the nodes are malicious. Malicious nodes conduct a *membership server pollution* combined with *outbreak* attack. For demonstrative purposes, malicious nodes also drop the traffic going through them.

We then model node join behavior using a Poisson process and node stay time using a Pareto distribution. These choices were motivated by observations from real overlay multicast deployments [14] and Mbone sessions [6] and have been previously used by Bharambe *et al.* in [8]. For the Pareto distribution, we assume a minimum stay time of 90 seconds and an α of 1.42, which results in a mean stay time of 300 seconds. These parameters are consistent with distributions found in other live streaming applications on the Internet [49, 14]. We vary the mean of the Poisson process between 5 and 15, leading to group sizes varying from 170 to 300 nodes. For example, if the Poisson mean is set to 10, then on average, every 10 seconds a node joins the overlay. The duration of each experiment is 1000 seconds and the source streams data at 1 Mbps. Our results in Figure 9 show that the attack is amplified by the churn of the system. This is due to the fact that as neighbors leave, a node must replace that neighbor with a new node which could possibly be malicious. Since the malicious nodes are stable, over time, malicious nodes will fill up the normal node's neighbor set. Given that only 20 malicious nodes are involved in the attack (7% to 12% of the total overlay size), the effect on the system is very damaging and drastically decreases the continuity index from 0.9 to 0.2.

Scalability with system size. We investigate the scalability of the attack for larger system sizes using the OverSim simulator. We perform a *pandemic* attack with a *general* strategy. Figure 10 shows the average corruption factor over time for a system size from 300 to 3000 of nodes. Note that when the system is not under attack the corruption factor is 0. The results show the rampant nature of the attack. In less than 100 seconds, regardless of the system size, almost all nodes in the network have half or more of their neighbors

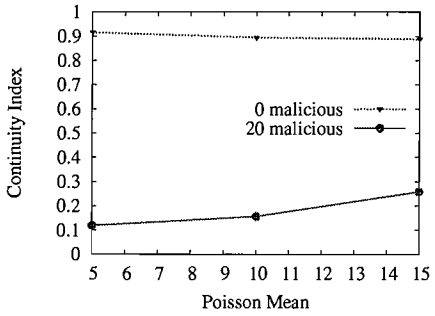
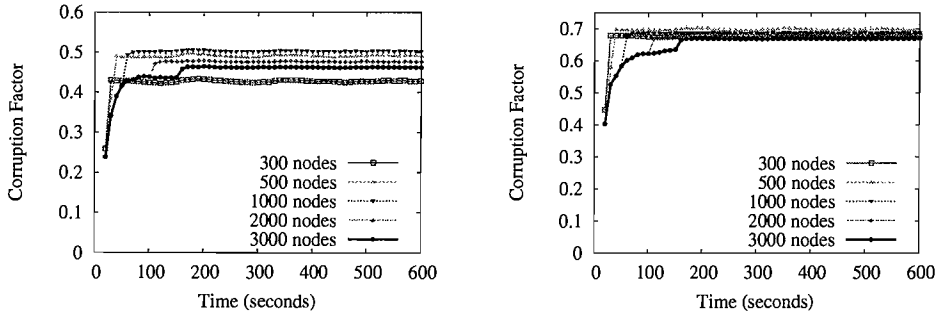


Figure 9: Continuity Index on PlanetLab with an overlay of an initial 80 honest nodes and 20 malicious nodes followed by other honest nodes joining based on a Poisson process. The total number of nodes varies between 170 and 300. Malicious nodes place themselves among the *in-neighbors* of any node in the overlay (*general* scenario), and conduct a *membership server pollution* combined with *outbreak* attack. Malicious nodes never leave the overlay. For demonstrative purposes, malicious nodes drop traffic going through them.



(a) 10% malicious nodes (b) 20% malicious nodes

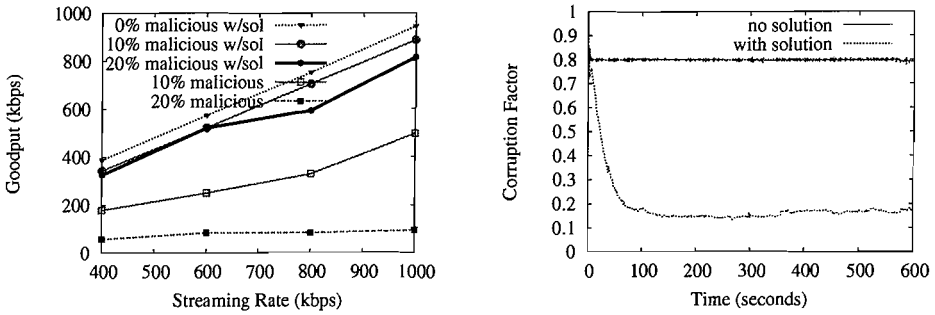
Figure 10: Corruption factor on OverSim for simulations of different overlay sizes, where malicious nodes place themselves among the *in-neighbors* of any node in the overlay (*general* scenario). Malicious nodes conduct a *pandemic* attack.

being malicious. This demonstrates the epidemic nature of this attack where just a few malicious nodes can corrupt a significant percentage of nodes in any given node's neighbor set. The results also demonstrate that the attack is effective for larger system sizes and that a small percentage of attackers can take over the neighbors of most of the nodes. For example, 20% malicious nodes present in the overlay are able to achieve a corruption factor of 0.7, indicating that on average greater than half of the neighbors of each nodes are malicious nodes.

5.4 Effectiveness of Our Solution

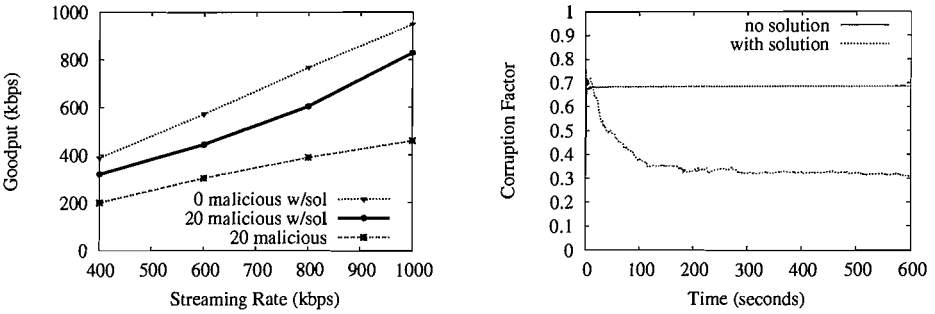
In this section, we study the effectiveness of our solution to mitigate the identified attacks. We first discuss the clustering coefficient threshold selection and then demonstrate the solution when the system is under attack. We show that our solution has little overhead and works for larger system sizes.

Threshold selection. As described in Section 4, our solution prevents malicious nodes from becoming *in-neighbors* for honest nodes based on a clustering coefficient threshold. We observe that malicious nodes will raise the clustering coefficient above that of a random graph. To accurately detect malicious nodes, we set the threshold to be the expected clustering coefficient of a random graph, which is the average node in-degree (e.g. 15) divided by the total overlay size. For example, in our PlanetLab experiments we set it to be $15/300 = .05$. We experimentally validated this threshold setup through numerous PlanetLab deployments and OverSim simulations. Further simulations where we varied the clustering coefficient based



(a) Goodput (b) Corruption Factor with 20% malicious

Figure 11: Goodput and corruption factor on PlanetLab for an overlay of 300 nodes when malicious nodes place themselves among the *in-neighbors* of any node (*general* scenario). Malicious nodes conduct a *membership server pollution* combined with an *outbreak* attack. For demonstrative purposes, malicious nodes drop traffic going through them. The system is tested without and with our solution in place.



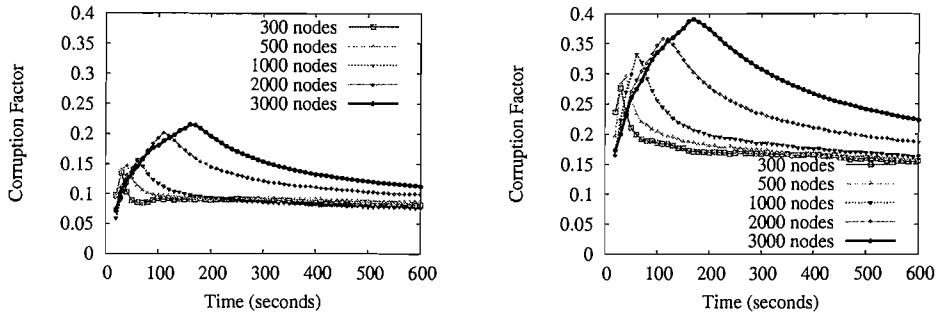
(a) Goodput (b) Corruption Factor with 20 malicious nodes

Figure 12: Goodput and corruption factor on PlanetLab for an overlay of 300 nodes when 20 malicious nodes place themselves among the *in-neighbors* of the source (*source-only* scenario). Malicious nodes conduct a *membership server pollution* combined with an *outbreak* attack. For demonstrative purposes, malicious nodes drop traffic going through them. The system is tested without and with our solution in place. Note that the corruption factor represents the corruption of the source since it is the only node under attack.

on an overestimation or underestimation of the overlay size revealed that as long as the error of the estimated size is less than 33% of the total overlay size, there is little effect on the performance of the algorithm. This implies that the algorithm is largely insensitive to the threshold. These simulation graphs were not included for lack of space.

False positives. During our experimentation and simulation, we found that false positives do not hinder the performance of the system. We believe this is true for two reasons. First, benign nodes are connected to multiple nodes in the system and if they are disconnected from a neighbor or set of neighbors, they can easily establish new connections to different neighbors. Secondly, we note that P2P streaming applications have been shown to form random graphs [5, 58]. It has been shown that this type of network structure can significantly increase the speed and extent of data propagation as long as it stays random [61]. Even if a benign node is falsely identified as malicious and disconnected (since it raised the clustering coefficient of other nodes past the desired cutoff), false positives help to maintain a random graph and may actually enhance the performance of the network.

Attack mitigation. We investigate the effectiveness of our solution on the PlanetLab testbed. Figure 11(a) presents the goodput for a deployment of 300 nodes on PlanetLab, while under a *membership server pollution* and *outbreak* attack, in the *general* scenario, without and with our solution in place. As seen in the figure, while under attack by 20% of malicious nodes, the goodput is decreased to only 100 Kbps, regardless of the stream rate. When our solution is enabled, the performance of the system is very close to



(a) 10% malicious - pandemic attack (b) 20% malicious - pandemic attack

Figure 13: Corruption factor on OverSim for a simulation for different overlay sizes, where malicious nodes place themselves among the *in-neighbors* of any node in the overlay (*general* scenario). Malicious nodes conduct a *pandemic* attack. The system is tested with our solution in place. Corresponding corruption factor for the system under attack and without our solution is presented in Figure 10.

the case when no attacker exists in the system. We attribute the attack mitigation to the fact that our method decreases the number of malicious nodes in the *in-neighbors* set. Indeed, as seen in Figure 11(b) which presents the corresponding corruption factor, our method reduces the corruption factor from 0.8 to 0.2.

We also conducted experiments to investigate the effectiveness of our solution in the *source-only* scenario since it provides a high incentive for an attacker due to its high return per malicious node. Figure 12(a) presents the case when 20 nodes of the *out-neighbors* of the source are malicious. Our solution is able to mitigate the attack and bring the performance of the system close to the case when no malicious node was present in the system. A closer examination of the corruption factor presented in Figure 12(b) shows that our method decreases the corruption factor to 0.3, value slightly higher than in the case of the *general* scenario.

We validate that the difference between the system under attack using our defense mechanisms and the system operating under non-attack conditions are statistically small, meaning our solution causes the attacks to have little effect on the system. For the various number of attackers, we test the null hypothesis $H_0 : \mu_{defense} = \mu_{normal}$. Using a two-sample t-test with pooled variance [33], we disproved H_0, finding that the actual mean of the system with defense is slightly greater than that of the system under non-attack conditions. We conclude that our technique successfully mitigates the attack.

Overhead. One of the features of our solution is that it has very little overhead. In particular, little additional communication overhead is added to the system as the nodes already exchange the set of *in-neighbors* that we use to compute the clustering coefficient. In addition, the clustering coefficient computation has very small computation overhead. In Figure 11(a), the curve labeled "0% malicious w/sol" denotes the goodput when no malicious nodes are present in the system while our detection mechanism is enabled. When comparing this curve with the one labeled "0% malicious" in Figures 5(b), which denotes the goodput of the system when no malicious nodes are in the system and our solution is not enabled, we note that the values are almost identical. This indicates that the overhead of our method is basically non-existent and our solution has no negative impact on the performance of the system when there are no adversaries in the system.

Scalability with overlay size. We find through simulations that our solution is effective and can limit the number of malicious nodes in a neighbor set even for large system sizes. In Figure 13, we plot the average corruption factor for different system sizes and different percentages of malicious nodes when malicious nodes perform a *pandemic* attack. The figures show that our solution significantly decreases the corruption factor. Even if not all the malicious nodes are completely removed, as shown in Figure 11(a) and 11(b) a small corruption factor (around 0.2) can be successfully tolerated by the system. While in all cases the system stabilizes at a low corruption factor, the convergence time increases as the overlay size increases, suggesting that additional mechanisms may be needed to speed up converge for large overlays.

6 Related Work

Attacks against overlay formation were previously identified in the context of DHTs for file sharing [12] and single-tree overlay multicast systems [60]. In addition, recent work also showed how P2P streaming systems can be used to attack the Internet [35, 20, 51]. Below, we give an overview of these works.

Exploiting the neighbor selection as a mechanism to control the overlay was first shown in the context of structured (DHT-based) homogenous overlays [12]. To defend against such attacks, referred to as eclipse attacks, Castro et al. proposed constrained routing tables [12] where a node's routing table could only be filled with other nodes that had identifiers that were limited to a certain range. Since this solution is specific to DHTs, it is unsuitable for unstructured overlays.

Singh et al. suggested anonymous auditing [45] which capitalizes on the observation that malicious nodes performing an eclipse attack will have more neighbors than the average node. Nodes audit their neighbors by asking an intermediary node to send a request for them to their neighbor for their neighbor list. If the list is larger than the average or if the originating node is not in the list, the originating node will drop that neighbor. This technique makes decisions based on the assumption that all nodes have the same number of neighbors. However, in heterogenous settings, nodes with large peer degrees are not only legitimate, but often times encouraged, such as hierarchical file sharing overlays. Hence this technique cannot be applied to heterogenous overlays.

Condie et al. proposed induced churn [18] to limit the effects of an eclipse attack on an overlay. The technique forces nodes to periodically assume new identities, wipe clean their routing tables and constantly limit the rate at which their routing tables are updated. This limits the effect of amplification of the attack by preventing malicious nodes from dominating the routing tables of normal nodes. While this technique can be applied to an unstructured overlay, is not necessarily suitable for streaming applications because of the real-time deadlines imposed on them. Churn can often degrade the performance of streaming applications to an unsatisfactory level as shown in [44].

The problem of parent selection in single-tree based multicast overlay was studied in [60]. The attacks exploit the fact that the parent selection is done based on performance measurement metrics reported by nodes in the neighbor set. The proposed solution decreases the number of malicious parents by using outlier detection and limit the impact of malicious nodes by aggregating local information to derive global reputation for each node.

Recent research has shown the feasibility of exploiting P2P systems to launch DDoS attacks impacting the external Internet environment, by causing large-scale distributed denial of service attacks on nodes not even part of the overlay system. In particular, an attacker could subvert the neighbor selection mechanism, and force a large fraction of nodes in the system to believe in the existence of, and communicate with a potentially arbitrary node in the Internet. The attacks have been shown on Overnet [35], and more recently BitTorrent [20, 51].

7 Conclusions

In this paper, we focused on attacks against neighbor selection in mesh-based P2P streaming applications deployed in heterogeneous networks. Based on the observation that attackers disturb the topology of the overlay, we proposed a technique to mitigate the impact of the attacks using the clustering coefficient. Our solution is lightweight, scalable and effectively mitigates the attack as shown through experiments on the PlanetLab Internet testbed with a P2P streaming system and simulations using the OverSim simulator. Our results show that:

- Although mesh-based streaming systems are characteristically more resilient to failures and churn, they are vulnerable to neighbor selection attacks that allow a small number of attackers to inflict significant damage to the system, irrespective of its size. Our real-world experiments demonstrate

that the attack is epidemic in nature as malicious nodes refer each other as potential service providers (i.e. *in-neighbors*).

- Several factors amplify the attack besides the number of attackers. They are: polluting the neighbors list of the membership server, the placement strategy of the attacker, and the presence of churn. Our results demonstrate that attacking the neighbors of the source is the most effective strategy for an attacker to attack a stable system. This indicates that it is critical to protect the source against malicious neighbors. Although mesh-based streaming was shown to perform well in non-adversarial networks in churn conditions, our results indicate that attacking the system in the presence of churn has a crippling effect on the performance, resulting in a decrease to just 20% of the original performance achieved by the system when no attack occurs.
- Our solution is effective in mitigating the attacks and raises the bar for the attacker without adding additional overhead in the system. In particular, our solution is very effective for stable overlays of moderate sizes (up to one thousand nodes) for up to 20% malicious nodes. For overlays of greater sizes, our solution significantly decreases the number of malicious nodes in the neighbor set, but has a higher convergence time.

References

[1] 2007 E-Crime Watch Survey - Survey Results. http://www.cert.org/archive/pdf/ecrimesurvey07.pdf.

[2] Buffer settings in windows media player. http://support.microsoft.com/kb/257535/.

[3] Sans institute system survival time. https://isc.sans.org/survivaltime.html.

[4] Shadowserver foundation. http://www.shadowserver.org/.

[5] S. Ali, A. Mathur, and H. Zhang. Measurement of commercial peer-to-peer live video streaming. In *Proc. of Workshop on Recent Advances in P2P Streaming*, August 2006.

[6] Kevin C. Almeroth and Mostafa H. Ammar. Characterization of mbone session dynamics: Developing and applying a measurement tool. Technical Report GIT-CC-95-22, Georgia Institute of Technology, 1995.

[7] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proceedings of 10th IEEE Global Internet Symposium (GI)*, pages 79–84, May 2007.

[8] A. Bharambe, S. Rao, V. Padmanabhan, S. Seshan, and H. Zhang. The impact of heterogeneous bandwidth constraints on DHT-based multicast protocols. In *IPTPS*, 2005.

[9] Michael Bishop, Sanjay G. Rao, and Kunwadee Sripanidkulchai. Considering priority in overlay multicast protocols under heterogeneous environments. In *IEEE INFOCOM*, 2006.

[10] Bartosz Biskupski, Raymond Cunningham, Jim Dowling, and René Meier. High-bandwidth mesh-based overlay multicast in heterogeneous environments. In *AAA-IDEA*, 2006.

[11] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *SOSP*, 2003.

[12] M. Castro, P. Drushel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI*, 2002.

[13] K. Cho, K. Fukuda, and H. Esaki. The impact and implications of the growth in residential user-to-user traffic. In *Proc. ACM SIGCOMM*, September 2006.

[14] Yang-Hua Chu, Aditya Ganjam, T. S. Eugene Ng, Sanjay G. Rao, Kunwadee Sripanidkulchai, Jibin Zhan, and Hui Zhang. Early experience with an internet broadcast system based on overlay multicast. In *USENIX Annual Technical Conference, General Track*, pages 155–170, 2004.

[15] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS*, 2000.

[16] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[17] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of P2P Economics*, 2003.

[18] T. Condie, V. Kacholia, S. Sankararaman, J. Hellerstein, and P. Maniatis. Induced churn as shelter from routingtable poisoning. In *NDSS*, 2006.

[19] CTV. http://www.tvoon.de/ctv.

[20] Karim El Defrawy, Minas Gjoka, and Athina Markopoulou. Bottorrent: Misusing bittorrent to launch ddos attacks. In *USENIX SRUTI*, June 2007.

[21] Prithula Dhungel, Xiaojun Hei, Keith W. Ross, and Nitesh Saxena. The pollution attack in p2p live video streaming: Measurement results and defenses. In *Sigcomm P2P-TV Workshop*, 2007.

[22] J.R. Douceur. The Sybil Attack. In *Proc. of IPTPS*, March 2002.

[23] Feidian. http://www.feidian.com.
[24] Lei Guo, Enhua Tan, Songqing Chen, Zhen Xiao, Oliver Spatscheck, and Xiaodong Zhang. Delving into internet streaming media delivery: a quality and resource utilization perspective. In *Proc. of IMC*, 2006.
[25] Maya Haridasan and Robbert van Renesse. Defense against intrusion in a live streaming multicast system. In *IEEE P2P*, 2006.
[26] Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu, and Keith W. Ross. A measurement study of a large-scale p2p iptv system. *IEEE Trans. on Multimedia*, 9(7):1672 – 1687, 2007.
[27] Joost. http://www.joost.com.
[28] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *ACM SOSP*, 2003.
[29] J. Liang, R. Kumar, Y. Xi, and K. Ross. Pollution in P2P File Sharing Systems. In *IEEE INFOCOM*, 2005.
[30] Anthony Lo, Geert Heijenk, and Ignas Niemegeers. Evaluation of mpeg-4 video streaming over umts/wcdma dedicated channels. In *WICON*, 2005.
[31] N. Magharei and R. Rejaie. PRIME: Peer-to-peer receiver driven mesh-based streaming. In *IEEE INFOCOM*, 2007.
[32] Mary Meeker and David Joseph. The state of the internet, part 3. In *Web 2.0*, 2006.
[33] D.S. Moore and G.P. McCabe. *Introduction to the practice of statistics.* WH Freeman and Company, 2003.
[34] Mysee. http://www.mysee.com.
[35] Naoum Naoumov and Keith Ross. Exploiting p2p systems for ddos attacks. In *Proc. of InfoScale*, page 47, New York, NY, USA, 2006. ACM.
[36] OverSim. http://www.oversim.org.
[37] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *IPTPS*, 2005.
[38] Pdbox. http://www.pdbox.co.kr.
[39] PlanetLab. http://www.planetlab.org.
[40] PPLive. http://www.pplive.com.
[41] PPMate. http://www.ppmate.com.
[42] PPStream. http://www.ppstream.com.
[43] QQLive. http://tv.qq.com.
[44] J. Seibert, D. Zage, S. Fahmy, and C. Nita-Rotaru. Experimental comparison of peer-to-peer streaming overlays: An application perspective. Technical Report CSDTR-07-020, Purdue University, July 2007.
[45] Atul Singh, Tsuen-Wan "Johnny" Ngan, Peter Druschel, and Dan S. Wallach. Eclipse attacks on overlay networks: Threats and defenses. In *IEEE INFOCOM*, 2006.
[46] SOPCast. http://www.sopcast.org/.
[47] N. Spring, L. Peterson, A. Bavier, and V. Pait. Using PlanetLab for network research: Myths, realities, and best practices. *ACM SIGOPS Operating Systems Review*, 40(1):17–24, 2006.
[48] Kunwadee Sripanidkulchai, Aditya Ganjam, Bruce Maggs, and Hui Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. *SIGCOMM Comput. Commun. Rev.*, 34(4):107–120, 2004.
[49] Kunwadee Sripanidkulchai, Bruce Maggs, and Hui Zhang. An analysis of live streaming workloads on the internet. In *Proc. of IMC*, 2004.
[50] StreamerOne. http://www.streamerone.com.
[51] Xin Sun, Ruben Torres, and Sanjay Rao. DDoS Attacks by Subverting Membership Management in P2P Systems. In *Workshop on Secure Network Protocols (NPSec 2007)*, October 2007.
[52] Yu-Wei Sung, Michael Bishop, and Sanjay Rao. Enabling contribution awareness in an overlay broadcasting system. In *Proc. of SIGCOMM*, pages 411–422, New York, NY, USA, 2006. ACM.
[53] TVAnts. http://www.tvants-ppstream.com.
[54] TVUnetworks. http://www.tvunetworks.com.
[55] UUSee. http://www.uusee.com.
[56] V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *IEEE ICNP*, 2006.
[57] VGO. http://vgo.21cn.com.
[58] Long Vu, Indranil Gupta, Jin Liang, and Klara Nahrstedt. Measurement and modeling of a large-scale overlay for multimedia streaming. In *Proc of QShine*, 2007.
[59] Wall Street Journal. http://online.wsj.com/article/SB112560377411829361.html.
[60] AAron Walters, David Zage, and Cristina Nita-Rotaru. Mitigating attacks against measurement-based adaptation mechanisms in unstructured multicast overlay networks. In *Proc. of ICNP*, November 2006.
[61] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):409–410, 1998.
[62] Zattoo. http://zattooblog.wordpress.com/.
[63] Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum. CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *IEEE INFOCOM*, 2005.