A Design for Securing Data Delivery in Mesh-Based Peer-to-Peer Streaming

Jeff Seibert, Xin Sun, Cristina Nita-Rotaru, Sanjay Rao
Purdue University
jcseiber@cs.purdue.edu, sun19@ecn.purdue.edu, crisn@cs.purdue.edu, sanjay@ecn.purdue.edu

Abstract

While mesh-based approaches have emerged as the dominant architecture for P2P streaming, the performance of these approaches under malicious participants has received little attention. In this paper, we provide a taxonomy of the implicit commitments made by nodes when peering with others. We show that when these commitments are not enforced explicitly, they can be exploited by malicious nodes to conduct attacks that degrade the data delivery service. We consider mesh-based specific P2P attacks where malicious nodes deliberately become neighbors of a large number of nodes and do not upload data to them, or malicious nodes deliberately delay packets till they are not useful anymore for the application. We focus on these attacks given the limited attention paid to them, and the significant impact they can have on overall data delivery. We present mechanisms that can enhance the resilience of mesh-based streaming against such attacks. A key part of our solution is a novel reputation scheme that combines feedback from both the control and data planes of the overlay. We evaluate our design with real-world experiments on the PlanetLab testbed and show that our design is effective. Even when there are 30% attackers, nodes receive 92% of the data with our schemes, however without our schemes they only receive 10% of the data.

1. Introduction

The vision of enabling simultaneous video broadcast as a common Internet utility in a manner that any publisher can broadcast content to any set of receivers has been driving the research agenda in the networking community for over two decades. For much of the 1990's, the research and industrial community investigated support for such applications using the IP Multicast architecture [1]. However, serious concerns regarding its scaling, support for higher level functionality, and deployment have dogged IP Multicast. The sparse deployment of IP Multicast, and the high cost of bandwidth required for server-based solutions or Content Delivery Networks (CDNs) are two main factors that have limited broadcast to only a subset of Internet content publishers. While many network service providers have enabled IPTV services that deliver quality video to their own subscribers using packet switching, there remains a need for cost-effective, ubiquitous support for Internet-wide video broadcast.

Over the last decade, there has been significant interest in the use of peer-to-peer (P2P) technologies for Internet video broadcast [2, 3, 4, 5, 6, 7]. There are two key drivers making the approach attractive. First, such technology does not require support from Internet routers and network infrastructure, and consequently is extremely cost-effective and easy to deploy. Second, in such a technology, a participant that tunes into a broadcast is not only downloading a video stream, but also uploading it to other participants watching the program. Consequently, such an approach has the potential to scale with group size, as greater demand also generates more resources.

The extensive research in the design of P2P streaming systems [2, 3, 8, 9, 10, 11, 12] has matured to the extent that we are today seeing several efforts aimed at commercializing the technology [4, 5, 13, 14, 15, 16, 17, 18, 19, 20, 21]. High user demand for these systems has been shown by their increasingly large user base [6, 7]. Not surprisingly, recent studies indicate that over 60% of Internet traffic is generated by P2P systems [22], with video accounting for more than one-third of all Internet traffic today [23, 24].

P2P streaming can be divided into two main approaches, tree-based [10, 8, 25, 26] and mesh-based [9, 11, 27, 28] architectures (see [29] for a survey). Tree-based overlays construct a tree, rooted at the source, which broadcasts the stream. Mesh-based overlays disseminate data in a less structured manner, where nodes exchange data with a subset of the nodes in the network without using any predefined route. Mesh-based approaches have received a lot of attention in recent times because they are more resilient to churn [30] and node failures, and have been shown to perform better than tree-based approaches [31, 30].

While mesh-based approaches have several attractive properties, the performance of these systems in the presence of malicious participants has received little attention. Dhungel et al. [32] show the vulnerability of such systems to attacks where malicious nodes upload polluted data to other nodes in the overlay. Similarly Haridasan et al. [33] focus on polluted data but also denial of service attacks on nodes by flooding them with requests. Several works have

focused on the problem of peers which download data from other nodes but do not in turn upload data [34, 35, 36, 37], however these works focus on selfish rather than malicious node behavior.

In this paper, we systematically analyze the vulnerabilities of the components of mesh-based streaming overlays. We focus on an important and broad class of attacks where malicious nodes deliberately become neighbors of a very large number of nodes in the system and do not upload data to them. We also focus on attacks that are particular to streaming systems such as when malicious nodes artificially delay the uploading of data, so while nodes still receive the data, because of real-time deadlines they are less likely to have opportunities to forward that data to others. We focus on these attacks given they have received limited attention, they can have significant disruption on data delivery, and they are applicable to many mesh-based systems. For instance, our evaluation with a state-of-the-art mesh-based streaming system shows that when the attacks are conducted with just 10% of nodes in the system being malicious, the average data rate received across all nodes is only 45% of the source rate when nodes upload no data and 47% when nodes delay some data.

We wish to emphasize that our focus in this paper is on mesh-based approaches for live video streaming, rather than file-download systems like BitTorrent [38]. While some of the attacks we consider may also be relevant to file-download systems, the impact on application performance is far more serious for streaming applications given that they are associated with stringent real-time deadlines. Consequently, the solutions must also be tailored to the unique demands of streaming applications.

Our contributions are:

- We provide a taxonomy of the implicit commitments made by nodes when peering with others. We show that when these commitments are not enforced explicitly, they can be exploited by malicious nodes to conduct attacks that degrade the data delivery service. To our knowledge, this is the first effort at taxonomizing attacks on mesh-based streaming protocols.

- We present a novel reputation scheme that combines feedback from the data plane (based on data received from the nodes) and the control plane (based on who a node has as neighbors) to increase the robustness of the mesh-based streaming overlay to the identified attacks. Through detailed security analysis, we show that our scheme is resistant to attacks commonly associated with reputation schemes such as self-promotion and slandering [39]. In particular, we show that our scheme ensures that a malicious node must contribute a minimum amount of data in a timely fashion to acquire a certain reputation. In addition, we show that a benign node that contributes data is assured a certain minimum reputation and cannot be slandered.

- We further augment the system, with a more comprehensive approach that also addresses potential vulnerabilities in the bootstrap mechanism, and with the source of the broadcast. We present a set of simple mechanisms to achieve this goal. Specifically, we present a scheme that prevents malicious nodes from influencing the membership bootstrap service and a source protection scheme that disallows malicious nodes to be overly connected to it.

- We evaluate our design using experiments on the PlanetLab testbed. Our results show that our schemes are extremely effective in ensuring good performance under attacks. With the local-reputation scheme, with 10% of the nodes being malicious, the average data-rate received across nodes from the source increases from 45% to 65%. Augmenting the solution with source and bootstrap protection mechanisms results in nodes receiving 95% of the source-rate on average. Our schemes also work well when attackers use advanced techniques such as data delaying. In fact, even with 30% of the nodes being malicious, more than 85% of the peers receive over 90% of the data. Overall, our results show the feasibility of augmenting mesh-based P2P streaming schemes to be resistant to attacks that target data delivery.

The rest of the paper is organized as follows. In Section 2 we survey related work and in Section 3 describe the mesh model we consider for this work. We describe attacks against data delivery in meshes in Section 4 and present our design to mitigate such attacks in Section 5. In Section 6 we provide an analysis for the security of our design. We explain the methodology and results of our experiments in Section 7 and 8, respectively. Finally, we summarize and conclude in Section 9.

2. Related Work

Much recent work has gone on in improving the efficiency and performance of P2P streaming systems. Lui et al [40] present algorithms that find near-optimal streaming rates when nodes can only support a bounded number of children. Picconi et al [41] demonstrate that P2P live streaming systems can incorporate locality-awareness and thus be ISP-friendly. Several works [42, 43] have also focused on utilizing network coding for improving download speeds and reducing the scarcity of data. We note that works such as these are orthogonal to ours and can be incorporated with our design.

However, the security challenges in designing mesh-based streaming protocols has received little attention. Recent work [44, 45] has surveyed security issues in P2P streaming, but cite a lack of solutions in this area. The only prior work we are aware of focuses on attacks where malicious nodes pollute data sent to other nodes [32, 46] or malicious nodes overload others with requests [33]. In

contrast, our focus is on data availability and prevention of neighbor selection attacks.

Attacks on data availability have been considered in the context of tree-based multicast [47]. The proposed solution takes advantage of the tree structure, knowing that if a child did not receive a message then an ancestor can be traced back to that is at fault for dropping it. Meshes do not have parent-child relationships but rather nodes get data from many neighbors, so this approach cannot be applied to them. Attacks against measurement-based neighbor selection were studied in the context of tree-based streaming [48]. The proposed solution uses outlier detection to identify malicious nodes that report wrong measurement results. This approach only works with systems that employ such measurement-based adaptation.

Dealing with selfish and Byzantine behavior using game theoretic principles has been investigated in several previous works [37, 49]. Most similar to our work is Flightpath [37], a P2P streaming system that is designed to give selfish peers incentives to obey protocols and can tolerate Byzantine behavior. Unlike their work, we do not assume synchronized clocks or synchronous communication channels.

Several previous works have dealt solely with selfish users in P2P streaming. Contracts [35] develops incentives that rewards nodes by giving them higher quality playback based on how effective a node's contributions to the entire system are. Substream trading [34] applies BitTorrent's tit-for-tat mechanism to a streaming context to encourage uploading, in this context nodes commit to sending each other parts of the video stream for a period of time. Pulse [36] also applies a tit-for-tat mechanism to live P2P streaming, but also combines it with incentives for altruistic behavior.

Several schemes have been proposed to mitigate neighbor selection attacks (referred to as eclipse attacks) in the context of distributed hash tables (DHTs) [50, 51]. The solutions are DHT-specific and do not apply to streaming protocols. A key aspect that distinguishes streaming protocols is the potential for feedback from the data-plane. In particular, it is possible to infer malicious behavior based on lack of data received from a neighbor. Our solutions leverage this observation resulting in significantly simpler designs.

Reputation systems have been a subject of wide interest, especially for P2P file-sharing systems. File-sharing reputation systems generally fall into two categories of purpose, incentivizing users to share files [52, 53], or thwarting file pollution [54]. Piatek et al. [53] show the feasibility of using one-hop reputations to incentivize interactions between users in BitTorrent. They take advantage of the fact that there are some users who are in many BitTorrent overlays and thus can be used as intermediaries, keeping track of long-term reputation values for others and facilitating data exchanges. While our work also uses local reputations, we differ in that our goal is mitigating malicious adversaries and not creating incentives. Also, as users usually only watch one video stream at a time, this precludes them from being in many overlays at once, making it impossible for some users to be intermediaries. Thus, streaming presents new challenges for reputation systems and has unique features that create opportunities, such as the continual downloading of data and stringent data deadlines, that we take advantage of.

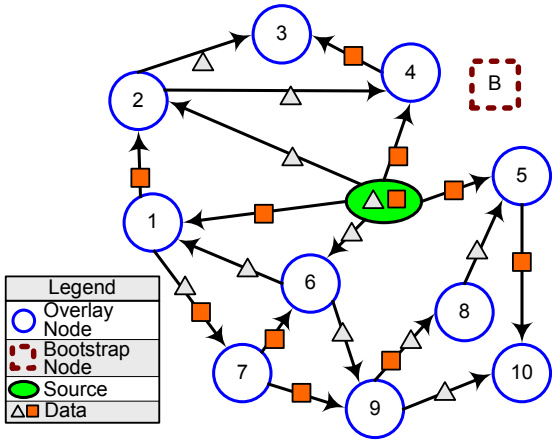


Figure 1: Example of a unidirectional mesh-based streaming overlay in which the source sends two different data chunks as denoted by the gray triangle and orange square. Each node has an in-neighbor set and an out-neighbor set. For example, for node 6, the in-neighbor set consists of node 7 and the source,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

3. Mesh-Based Peer-to-Peer Streaming

We consider a unidirectional mesh-based P2P overlay consisting of a bootstrap node, a source node and peer nodes. As seen in Figure 1, the mesh allows peers to download a stream generated by the source, while the bootstrap maintains a list of alive peers used to assist peers to join the network. We consider a unidirectional mesh since it is more general than a bidirectional mesh. Also, unidirectional meshes have been shown to perform better than bidirectional meshes [55].

Every peer node maintains two sets of nodes, in-neighbors and out-neighbors. The in-neighbors represent the nodes that the peer node is receiving data from. The size of the in-neighbors is a system parameter. The out-neighbors represent the nodes that the peer node is sending data to. Each node decides independently the number of out-neighbors to support which will be proportional to its bandwidth. The source has no in-neighbors, only an out-neighbors set, whose size is usually larger than the size of an out-neighbor set of a peer node.

At join time, a peer node j first contacts the bootstrap node to receive a set of candidate nodes to serve as its neighbors in the overlay. Node j then contacts each candidate node and requests to become one of its out-neighbors. If a candidate node c accepts the request, then in turn, j

will add c to its in-neighbor set. Each node pro-actively looks for several out-neighbors to connect to as well.

After it joins the overlay, a node discovers other peers by occasionally contacting its neighbors to learn about their own neighbors. This gossip protocol allows a node to update its in-neighbor set when neighbors leave or crash. A node also registers with the bootstrap node occasionally to allow the bootstrap node to have an up-to-date list of alive nodes. We will refer to these protocols as the *control plane* of the overlay.

The source node splits the stream into data chunks of a fixed size, each uniquely identified by a sequence number. To receive a chunk a node will send a request to an in-neighbor with that chunk's sequence number. If the requested node does not respond before a deadline then the requesting peer will consider that request lost. Each peer node maintains a buffer that it is trying to fill with data chunks. The buffer corresponds to a playback deadline, such that if a block of the stream is not received before that deadline, the data is considered lost and thus the quality of the playback stream is diminished. We will refer to this protocol as the *data plane* of the overlay.

This model is general enough to capture the characteristics of several previously proposed and deployed mesh-based systems [11, 27, 28, 4, 13].

4. Attacks Against Data Delivery

We state the assumptions we make about the attacker and provide a taxonomy of attacks against mesh-based P2P streaming systems.

4.1. Attack Model

We assume that a fraction f of peers are compromised and can behave arbitrarily. The percentage f is the largest fraction of nodes that the system is willing to tolerate as malicious. Their main goal is preventing the overlay from delivering data to each peer in a timely fashion. An attacker can disrupt the data delivery *directly* by attacking the data plane, or *indirectly* by attacking first the control plane to gain control over the data delivery path and then disrupting the data delivery.

We assume a defense against Sybil attacks [56] is in place, such as binding IP addresses to certificates or one that leverages social networks [57]. We also assume that data integrity is ensured and data is protected from pollution [32, 6]. We assume that the source and the bootstrap node are trusted and always behave correctly.

4.2. Attacks on the Data Plane

When two nodes A and B accept each other as out-neighbor, and in-neighbor, respectively, they assume several implicit commitments from each other:

• **Data delivery commitment:** A commits to B that it is going to deliver a certain amount of data to B.

• **Data download commitment:** B commits to A that it is going to download a certain amount of data from A.

• **Data upload commitment:** B is going to upload to the overlay what it downloaded from A.

• **Source upload commitment:** If B is connected to the source, then it will upload the data downloaded from the source to others in the overlay. This is similar to the data upload commitment, however we list it separately given that the source is a special entity where all the data originates.

• **Data delay commitment:** A will upload the data requested by B as soon as possible and not arbitrarily delay it.

• **Data integrity commitment:** A commits to B that it is not going to upload to B meaningless data.

However, in many mesh systems, not all of these commitments are explicitly enforced by the system. As a result, malicious nodes can exploit them to attack the data plane. We identify the following attacks (summarized in Table 1).

Table 1: Attacks against data and control planes

Data plane	Data dropping
	Data delaying
	Neighbor exhaustion
	Source
	Free-riding
	Pollution
Control plane	Bootstrap list pollution
	Neighbor selection

• **Data dropping attacks:** If the data delivery commitment is not met, a malicious node can accept benign nodes as its out-neighbors, but not deliver data to them. The attacks are effective because each data chunk has a strict deadline. A node only has time to make a few downloading attempts for a chunk, and will miss it once the deadline is passed.

• **Data delaying attacks:** If the data delay commitment is not met, a malicious node can send data to its out-neighbors yet delay the sending of it. Delaying data makes the attacker seem less malicious since it is actually delivering data before the playback deadline. However, the data is less useful to the recipient since there will be fewer opportunities to upload the data to others.

• **Neighbor exhaustion attacks:** If the data download commitment is not met, a malicious node can become out-neighbors of benign nodes, but not download data from them. As many meshes limit the number of out-neighbors to ensure that nodes can honor the bandwidth requirements, by being included in the out-neighbors a malicious node exhausts the slots in that set thus denying access to other benign nodes.

• **Source attack:** If the source upload commitment is not met, malicious nodes do not forward data given to it by the source. Thus, if a particular chunk is only received by malicious nodes it will not be available to any benign node. To amplify this attack, malicious nodes can also become out-neighbors of benign nodes connected to the source and

similarly not forward data given to them.

• **Free-riding attacks:** If the data upload commitment is not met, malicious nodes could also download data but not upload them to other peers, and basically obtain free service without contributing to the system.

• **Data pollution attacks:** If the data integrity commitment is not met, malicious nodes can upload meaningless data, thus polluting the information in the overlay.

4.3. Attacks on the Control Plane

The above data plane attacks are more effective when they impact many nodes in the overlay. A malicious node can increase the impact of its attack by first attacking the control plane. The control plane provides nodes with two mechanisms to discover peers. The first consists of the list of alive peers provided by the bootstrap node when a node joins the overlay. The second consists of exchanging membership information between the node and known peers. The bootstrap list is up-to-date if peers periodically register with the bootstrap node to inform it that they are alive. Assuming the bootstrap node is trusted, the control plane achieves its goals if the following commitments are met:

• **Registration with the bootstrap node commitment:** A peer commits that it will register occasionally with the bootstrap node, at a rate specified by the protocol.

• **Referral list commitment:** A node commits to provide a neighbors list that does not purposely contain malicious nodes and is not biased towards some nodes.

We identify the following attacks that have an impact on neighbor selection:

• **Bootstrap list pollution attacks:** If the registration with the bootstrap node commitment is not met, malicious nodes can register fast and often with the bootstrap node filling the bootstrap node's list of alive peers. Thus, although the bootstrap node is trusted, the list that it will provide to the joining peers will be polluted with malicious nodes. Note that malicious nodes can also register infrequently or not at all, but in this case they will not impact the list of the bootstrap node.

• **Neighbor selection attacks:** If the referral list commitment is not met an attacker can collude with other malicious nodes and when contacted about its own neighbors, refers only other malicious nodes. This attack is epidemic in nature since soon benign nodes will also be referring the malicious nodes they know to other benign nodes.

4.4. Our Focus

We focus on the attacks that we believe can be the most effective strategy for an attacker to disrupt the data delivery, and allow him to inflict maximum damage on the system with minimal resources. The most effective strategy for a malicious node is to (i) become neighbors of as many nodes as possible, (ii) deliver as little data as possible and (iii) data that is delivered should be as useless as possible. Hence we focus on control plane attacks (i.e. bootstrap list pollution and neighbor selection) that seek to increase the connectivity of malicious nodes and also on several data plane attacks (i.e. dropping, delaying, neighbor exhaustion, and source) as they can create considerable damage in the network.

We note that many of these attacks are specific to streaming, as file-distribution systems do not have real-time deadlines of data, nor need to download at a particular streaming rate, and often have centralized membership protocols (e.g. BitTorrent).

We do not consider attacks such as free-riding or data pollution as they relate to selfish behavior and data integrity but not attacks on data delivery. Furthermore, several solutions to free-riding have been proposed in previous work [38, 34, 35]. Also, to prevent data pollution, Dhungel et al. [32] have shown that a suitable means to accomplishing this is the source digitally signing hashes of the chunks. We note that solutions to these attacks can be used to complement our work.

5. A Design For Securing Data Delivery

In this section we describe our design for securing the data delivery for a P2P mesh-based streaming overlay. We first outline the design goals, then describe the details of our design.

5.1. Design Goals and Overview

Our focus is on ensuring that the P2P system achieves its intended goal which is continuous data delivery, even when under attack. However, achieving the same level of service in the presence of insider attacks as in the benign case is not always possible. As a result, our specific goals are:

(G1) Limit the impact of the attack: We seek to raise the bar for the attacker and bound the amount of damage per attacker. The damage created is directly proportional with the number of attackers and the amount of data dropped or delayed by the attacker nodes. Our goal is to limit the degree of connectivity in the mesh that malicious nodes can obtain. We integrate mechanisms that use control plane feedback to mitigate bootstrap list pollution, source, and neighbor selection attacks and mechanisms that use data plane feedback to detect data dropping, delaying and neighbor exhaustion attacks.

(G2) Limit the overhead of the defense mechanisms: Because malicious behavior is not a priori known, some of the components of our design are proactive, thus they must be enabled regardless of the presence of attacks. One specific concern is the overhead of the defense mechanisms. Our goal is that when no attack takes place, the system performance with the defense mechanisms enabled is the same as if those defense mechanisms were not used.

To achieve the goals identified above we design several proactive and reactive protocols. Our schemes use local observations to help nodes identify malicious peers and

build a robust neighbor set. We also design schemes tailored for the source and bootstrap nodes given their critical roles.

Peer protection: To limit the impact of attacks and the overhead of the defense solution, we use decentralized mechanisms deployed at each individual peer that allow it to make local decisions about accepting, rejecting, or excluding other peers from its set of neighbors. Each node individually derives reputation scores for the other peers it is aware of in the overlay. The use of reputation is a natural choice in a distributed system with malicious participants. Since many existing reputation systems require additional overlays or have high computational or bandwidth overhead [52], we design schemes that are tailored to streaming overlays. The novelty of our scheme lies in combining feedback from the data plane and control plane to build reputations for each peer.

Source protection: As the source is a producer but not a consumer of data, the protection mechanisms used for peers are not applicable to the protection of the source. We use mechanisms that limit the impact of source attacks by allowing nodes to notify the source if certain data was not received.

Bootstrap protection: The bootstrap node plays a critical role in the control plane. Attacks against the control plane can be amplified if the bootstrap is not a reliable and unbiased source of information on who is currently in the overlay. Our scheme discourages nodes from registering at a fast rate and thus limits the percentage of malicious nodes in the bootstrap list.

Below we describe in details each of these protection mechanisms. First, we describe in Section 5.2 the details on a local reputation mechanisms that protects against data dropping and data delaying attacks. Then, we describe the source and bootstrap node protection mechanisms, in Sections 5.3 and 5.4, respectively.

5.2. Protecting Peers through Local Reputation

We propose a mechanism that allows peers to select as neighbors the nodes that provide the best performance while being resilient to data dropping and neighbor selection attacks. We also show how to extend this mechanism to protect against data delaying attacks. A node uses locally observed data and control plane information to compute scores for each of its neighbors. The lower the score, the higher the chance that a node is malicious. Nodes that have a score lower than a threshold T_d are evicted from the in-neighbors set. The local reputations are also sent across one hop to neighbors, so that they can avoid accepting malicious nodes as in-neighbors. The score consists of two components:

• *Data score:* This score is a *positive reputation* (it rewards good behavior) and it is calculated based on how much data a node has received from a particular neighbor. The goal of the data score is to capture regular performance degradation and data dropping attacks. Nodes who do not deliver sufficient data will have a lower data score. Nodes with a data score below a threshold T_s are considered to be *suspicious*. This approach forces malicious neighbors to deliver a certain amount of data. Note that for a node to be evicted from the neighbors set, his total score has to be smaller than T_d ($T_d < T_s$).

• *Graph connectivity score:* This score is a *negative reputation* (it penalizes bad behavior) and it is calculated based on how connected a node is to other nodes. The goal of the graph connectivity score is to target neighbor selection attacks. This score is relevant only for suspicious nodes because if the nodes deliver enough data (i.e. corresponding to a data score above T_s) they do not disturb the overlay. A high graph connectivity score indicates that a node is potentially conducting a neighbor selection attack. This score is used because if the data score is neither high nor low, it may not be obvious if a node is malicious.

Below we provide details about the data and graph connectivity score computation, about the way they are combined into a reputation score, and about how the reputation score is used to make decisions on what nodes to allow as neighbors. Algorithm 1 also describes this computation, specifically how a node i calculates the *Local Reputation* for node j.

Data score computation. Every node i calculates a data score for every in-neighbor j as follows:

$$L_{ij}(t) = min\left(1, \frac{G_{ij}(t)}{E(t)}\right) \quad (1)$$

where $G_{ij}(t)$ is the number of chunks received by node i from j before deadline D_r in a time period. D_r is the amount of time the requesting peer will wait before considering that the request was dropped. If a request for a chunk is honored after the D_r deadline has passed, it is not included in $G_{ij}(t)$. $E(t)$ is the expected number of chunks to be received by a node in a time period. Typically, the expected value is the same for all nodes and if it is received from all in-neighbors the full streaming rate will be received. We take the minimum of $\frac{G_{ij}(t)}{E(t)}$ and 1 so that if a node performs better than expected the end result will still be between 0 and 1. The more data j delivers to i, the bigger the $L_{ij}(t)$. If $L_{ij}(t)$ is less than a threshold T_s, then i marks j as *suspicious*.

A score for a node's out-neighbors is calculated by replacing $G_{ij}(t)$ with the number of requests fulfilled for that node in a time period. Such a score allows nodes to mitigate neighbor exhaustion attacks.

Graph connectivity score computation. Every node also calculates a graph connectivity score for each of its neighbors that were marked suspicious. This score relies on the observation that a malicious node conducting a neighbor selection attack will be an in-neighbor for many honest nodes. In particular, if node i has as in-neighbors node k and j, and node j is malicious, then it is likely that j is an in-neighbor for node k as well. Furthermore, the more in-neighbors of i that j is connected to, the more

likely it is that j is conducting an attack. We propose the following graph connectivity equation for each node i to calculate the likelihood of each of its in-neighbor j being malicious:

$$C_{ij}(t) = \frac{K_{ij}(t)}{N_i(t)} \quad (2)$$

where $N_i(t)$ is the total number of non-suspicious neighbors of i (i.e. a non-suspicious neighbor is one whose data score L is greater than T_s), and $K_{ij}(t)$ is the number of these non-suspicious neighbors for whom j is also an in-neighbor. Intuitively, the equation calculates a score equal to the percentage of non-suspicious neighbors that a neighbor j is currently an in-neighbor for. The score will be high if a neighbor is in many neighbor sets, indicating that it is malicious. We consider only non-suspicious nodes so that in the case a malicious node wants to falsely advertise other nodes in its in-neighbor set, it has to first perform some work for the system.

Reputation score computation. Every node combines the data and graph connectivity score as follows:

$$R'_{ij}(t) = \begin{cases} L_{ij}(t) - \alpha * C_{ij}(t) & \textit{if } j \textit{ is suspicious} \\ L_{ij}(t) & \textit{otherwise} \end{cases} \quad (3)$$

If a node had a low data score and was marked as suspicious, then we take into account the graph connectivity score as it is a negative score and will further reveal if the node is misbehaving. Specifically, we subtract from the data score the graph connectivity score and we weight the latter with a parameter α. However, if the node was not marked as suspicious, we do not take into account the graph connectivity score. This choice was made based on the observation that if the nodes deliver enough data, it does not matter how connected they are as they do not disturb the honest nodes.

Incorporating history. Every node takes into account the history of its neighbors by calculating for each neighbor the following equation:

$$R_{ij}(t) = \lambda * R'_{ij}(t) + (1 - \lambda) * R_{ij}(t-1) \quad (4)$$

where λ is a value less than 1. We take into account history to accommodate transient network conditions, such as congestion. This gives nodes the opportunity to recover and not be disconnected due to non-persistent problems. All nodes start with a reputation equal to T_s.

Reputation based neighbor selection. A node uses reputation scores to decide when to drop or add neighbors. To decide if he keeps a node j as a neighbor, node i compares the reputation score R_{ij} for node j with a threshold T_d. If j's score becomes less than T_d, then i will drop j from its neighbor set and will not allow j to be in either its in-neighbor or out-neighbor sets from then on. We identify j by its IP address to avoid trivial Sybil attacks.

A node also uses the reputation score to determine if a node is non-malicious when deciding to add a neighbor. Consider the case when a node s refers a neighbor k to node i, s will also send the reputation score of k. To decide if he adds k as a neighbor, i computes $R_{is} * R_{sk}$. Node i will then add node k as a neighbor if the resulting number is greater than the suspicion threshold, T_s.

Protecting against data delaying. As long as there is no enforcement of data delivery, an attacker's best strategy is to drop all data. However, once the above protection mechanisms are introduced, an attacker's best strategy is to send as little data as possible, but delay everything it sends so that the node receives it just before the D_r deadline. The delaying of data is advantageous to attackers in multiple ways. Attackers can still get credit for sending data, yet it is less likely that the benign node will have as many opportunities to pass that data on to others. Delaying data also creates a temporary scarcity of data chunks, and as few benign nodes will have that data, malicious nodes will be able to fill that request for many nodes.

As data delaying unnecessarily increases the amount of delay in receiving chunks, nodes can measure the delay and then penalize the offenders. To do this we introduce the inverse relative stretch (IRS_u) metric which node i will calculate for each chunk u received from node j. We define IRS_u as the ratio between the delay from the source to node i and the delay from when the source generates a chunk u to when node i actually receives it. An IRS_u of 1 would indicate node i received the chunk with no extra delay whatsoever while less than 1 indicates that there was some extra delay. To incorporate this value into the data score, we recalculate G_{ij} (as referenced in Equation 1) as follows:

$$G_{ij}(t) = \sum_u min(l * IRS_u, 1) \quad (5)$$

During one time period, node i evaluates the IRS_u of every chunk u received from node j and calculates a summation based on these values. Specifically, the summation of G_{ij} is calculated by adding the minimum of 1 and l times IRS_u for every chunk received. We multiply the IRS_u by some parameter l as some stretch is normal for any application-layer multicast and l lets us determine how much stretch we are willing to tolerate. We then take the minimum of that value and 1 to normalize it and ensure that we are adding at most 1 for every chunk received. We would expect then that a benign node that does not add extra delay to a chunk would receive a score equal to the number of chunks it sent. However, any malicious node adding extra delay would receive a lower score.

Overhead. The *Local Reputation* scheme adds minimal overhead to the system. This is due to it only involving simple calculations, the most expensive being multiplications, and it scales linearly with the number of neighbors a node has, which typically runs between 15 and 60, depending on how much bandwidth a node has. Furthermore, the network overhead is negligable as we use messages that are already being sent to transport the extra data about reputation scores.

Algorithm 1: *Local Reputation* computed by node i for node j.

```
//This algorithm is run every t seconds
//R_ij is initialized to be T_s when node j becomes a neighbor
G_ij = 0;
E = source_streaming_rate / num_neighbors;
//compute data score
foreach chunk u sent by j do
    G_ij += 1;
    // if protecting against data-delaying do instead:
    // G_ij += min(l * IRS_u, 1);
end
L_ij = min(1, G_ij/E);
if L_ij < T_s then
    //node is suspicious, consider graph connectivity score
    N_i = 0;
    K_ij = 0;
    foreach in-neighbor k of i do
        if L_ik > T_s then
            // in-neighbor k is not suspicious
            N_i += 1;
            if j is in-neighbor of k then
                K_ij += 1;
            end
        end
    end
    C_ij = N_i/K_ij;
    //reputation considers both data and
    // graph connectivity scores
    R'_ij = L_ij - α * C_ij;
else
    //reputation only considers data score
    R'_ij = L_ij;
end
//take into account history of reputation
R_ij = λ * R'_ij + (1 - λ) * R_ij;
if R_ij < T_d then
    //node j is below drop threshold
    disconnect j
end
```

5.3. Source Protection with Health Monitoring

The source is a critical component of the overlay. As will be shown in Section 8 attacks against the source can significantly degrade the performance of the system. While *Local Reputation* is effective for peers, it can be intuitively seen that such a mechanism is ill-suited for the source. This is for two reasons. First, the source does not request data from its neighbors, it only gives data, so it cannot judge a node based on data received. Second, malicious nodes will prefer to receive data from the source rather than from peers so this also will not lead to the source suspecting them. As a result the source can not differentiate between a benign and malicious node.

We first observe that in some P2P live streaming systems today, there is extensive gathering of statistics from peers [6]. This allows for further refinement of protocols and code so that the quality of the experience can continue to improve. One very important metric to collect is the amount of data that peers miss from the stream. This gives a way to measure the overall health of the system. We then use this monitoring information to protect the source.

Specifically we propose that the source keeps track of who it sends which data chunks to. Then if peers miss some data chunks, they can report the specific ones missed to the source. One would expect that if many nodes miss a chunk, it is due to malicious nodes not forwarding data received from the source. Therefore, once the source has received complaints from a percentage of nodes greater than f it can then disconnect the nodes it sent those data chunks to. The percentage f must be the largest fraction of nodes less than 50% that the system can tolerate as malicious. Also, as this scheme might create an implosion of messages at the source, nodes can collect them and send them in batches.

Overhead. In practice, the network overhead is small as we batch complaints into a single message and only send them periodically. Furthermore, the source will only receive messages when nodes are not receiving the data chunks, which should be abnormal behavior. To further reduce the overhead, nodes could piggy-back the complaints on messages that are already being sent to the bootstrap.

5.4. Rate-limiting Bootstrap

Our solution for protecting the bootstrap relies on the observation that nodes that register with the bootstrap node many times in a short period are most likely malicious. Thus to detect and discourage this behavior, if nodes register faster than once every w seconds, they will not be put into the bootstrap list and then will not be propagated by the bootstrap node. To detect misbehavior the bootstrap keeps track of all registrations that have occurred in the past w seconds. From the registration information it will make a list of k nodes that have only registered once.

The w parameter decides how often nodes can register, so the larger it is the more resilient the bootstrap will be against attacks. However, if it is too large it will prevent good nodes from legitimately re-registering. The k parameter allows the bootstrap to decide how it will pick nodes from the recent time window and in what quantity. There are different strategies to fill the bootstrap list, for our design though, we simply choose the k most recently registered nodes to ensure the freshness of the list.

To only do rate-limiting and nothing else might bring about scenarios where there are still very few honest nodes in the bootstrap list. This could be due to very few nodes joining the overlay for a period of time. To ensure that the bootstrap list still can not be filled with malicious nodes, we have each node randomly register once every w to $2w$ seconds.

Overhead. The *Rate-limiting Bootstrap* scheme introduces no new overhead into the system, as periodically registering with the bootstrap is normal behavior. Additionally, the rate of at which registrations occur, and thus the amount of overhead, can be adjusted by setting w to the desired rate.

6. Security Analysis

In this section, we analyze how robust the *Local Reputation* scheme is in defending against common classes of attacks. Recall that the final reputation score is derived by combining the data score, which is a positive score, and the graph connectivity score, which is a negative score. The node uses the final reputation score to decide who should remain as neighbors and who to admit as neighbors. Possible attacks that can be conducted on these reputation calculations and uses include [39]:

Self-promoting: Malicious nodes falsely inflate their own reputation. This attack is only effective in positive feedback based systems.

Slandering: Malicious nodes attack the reputation of other nodes by reporting untrue information about them. This attack is only effective in negative feedback based systems.

Orchestrated: Colluding nodes combine several strategies to game the system.

Whitewashing: Malicious nodes take advantage of a system vulnerability to restore a damaged reputation. One possible way to do this is by assuming new identities.

6.1. Attacks on Data Score Calculation

The reputation system is designed so that a node cannot get a high data score and thus a high reputation without doing useful work. Therefore, the data score cannot be influenced by slandering or self-promoting attacks, as the only way to change it is for a node to deliver more data. We present the following lemma which quantifies the amount of useful work done by a node given a particular data score, which can be derived from Equation 1.

Lemma 1: *For a node j to obtain a data score of L_{ij} at a neighboring node i, j must deliver data to i at a minimum rate of $E * L_{ij}$, where E is the expected amount of data a node should deliver to a neighbor in a time window (Section 5.2).*

This lemma guarantees that benign nodes will receive good performance even when surrounded by a significant number of malicious neighbors, for example, when under an orchestrated attack. This is because each malicious neighbor is forced to deliver a minimum amount of data in order not to be dropped. More specifically, if we assume a node with a fraction f of its neighbors is malicious, and assume benign neighbors always deliver the expected amount of data, then the node will receive at least $(1-f)+T_d f = 1-f(1-T_d)$ of the streaming rate (T_d is the drop threshold). For example, with $T_d = 0.5$ and $f = 0.3$, the node will receive at least 85% of the stream rate.

Furthermore, Lemma 1 imposes a high bandwidth cost on malicious nodes who seek to be a neighbor of a large number of nodes. To highlight this, consider a streaming system with 150K nodes [6], and that a malicious node desires to maintain a reputation score of T_d at every node. According to Lemma 1, with a streaming rate of 1Mbps, a neighbor-set size of 15, and assuming a T_d value of 0.5, the node must deliver data at a minimum total rate of 5Gbps.

We note that though Equation 5 modifies how the data score is calculated to protect against data delaying, this simply raises the bar for attackers, forcing them to send even more data if they wish to delay the data they are sending. Hence, even when data delaying protection is in place malicious attackers still must send at minimum $E * L_{ij}$. We present the following lemma which quantifies how much more data a node must send if it delays it.

Lemma 2: *For a node j to obtain a data score of L_{ij} at a neighboring node i when delaying data, j must deliver data to i at a minimum rate of $\frac{E*L_{ij}}{l*IRS}$, where IRS is the inverse relative stretch and l is a system parameter.*

Lemma 2 shows that attackers must increasingly send more data the longer they delay it. For example, if node i and the source have a delay of 100 ms and node j delays data so that it arrives after 600 ms, the IRS will be 1/6. Assuming l is set to 3, then node j must send data at a rate of $2 * E * L_{ij}$ to get a score of L_{ij}, in this case doubling the amount of data it would normally have to send.

6.2. Attacks on Graph Connectivity Score Calculation

When a node calculates its neighbors' graph connectivity score, it takes into account neighbor set information provided by all of its non-suspicious neighbors. This scheme is subject to slandering attacks where a malicious neighbor can provide fake neighbor set information. Slandering can be seen from two different perspectives, the ability of a node to slander others and the resistance a node has from slandering attempts. We first present the following lemma that shows the limitations a node has in its ability to slander others, which can be derived from Equation 2.

Lemma 3: *A node j can only influence the graph connectivity scores of the neighbors of node i if j has a data score of T_s with i.*

Lemma 3 shows that malicious nodes must themselves do a substantial amount of work to remain non-suspicious, which means having a data score above T_s. According to Lemma 1, this means they must deliver data to i at a minimum rate of $E * T_s$. Given that $T_s > T_d$, this imposes an even greater bandwidth constraint on attackers that want to slander others over attackers that want to simply not be dropped.

We next present a lemma that demonstrates that a node can resist slandering attacks from others. The key insight behind the lemma is that if a node transmits data at a high enough rate, its final reputation as computed by the neighbor depends on the data score alone, and is not impacted by the graph connectivity score.

Lemma 4: *Any node that delivers data to a neighbor at a rate greater than $E * T_s$ is assured of a reputation greater than T_s with the neighbor.*

We expect that most benign nodes will be cooperative and deliver data at rates close to the expected rate, which

is well above $E * T_s$. Therefore, benign nodes will not be subject to slandering attacks as their graph connectivity score will not even be considered.

6.3. Other Attacks

We discuss other attacks on the schemes, and why our approach is resilient to them:

Whitewashing attacks: In these attacks, malicious nodes who received a bad reputation may choose to re-join the network with a different identity. We believe this attack is not a concern because of the following reasons. First, the reputation is initialized to T_s, and all new nodes will be marked suspicious initially. Therefore, a new node cannot refer other nodes or report connectivity information about other nodes until it has done work and improved its reputation. Further, the newly added node will be quickly dropped unless it transmits data at a sufficient rate. Second, in our model, nodes are identified by their IP address. To cause damage, a malicious node cannot acquire a new identity by simply spoofing an IP address, but must be able to receive packets targeted to the IP address. By our attacker model in Section 4.1, we assume only a fraction f of the total number of IP addresses are controlled by malicious nodes.

Attacks on reputation-based neighbor selection: A node adds new neighbors by taking referrals from existing non-suspicious neighbors. This process is subject to attacks where a malicious neighbor (m) could refer other malicious nodes to a benign node (i). However, to conduct this attack, the malicious neighbor m must be considered non-suspicious, and hence must deliver data at a minimum rate of $T_s * E$, where T_s is the suspicion threshold. Further, each newly inserted malicious node referred by m must also do a substantial amount of work to obtain a minimum data score of T_d (the drop threshold), or it will be dropped quickly by node i.

7. Experimental Methodology

In this section, we describe how we evaluate and compare our protection schemes with several alternative schemes. We implemented the unidirectional mesh described in Section 3 in a mesh streaming codebase [9]. We also implemented all of our own protection schemes plus some alternatives which we will describe next, which are summarized in Table 2.

7.1. Schemes Considered

No Protection (NP): This is our baseline scheme which has no protection for any of the system components.

Local Reputation (LR): This peer level scheme, as described in Section 5.2 builds up a reputation from information gathered from the control and data planes. With this reputation scheme in place, nodes are able to decide if a node is malicious and thus can better select who they should accept as neighbors.

LR Data Delaying (LR-DD): This is the extended Local Reputation scheme that adds protection against data delaying attacks.

Health Monitoring (HM): This source protection scheme, described in detail in Section 5.3, uses information gathered from peers to decide who should stay as neighbors of the source. If a percentage of nodes declare that a certain data chunk was missed by them, the source will drop the nodes that it originally sent those chunks to.

Rate-limiting Bootstrap (RB): This is a bootstrap protection scheme, described in detail in Section 5.4, keeps track of how often nodes register and penalizes the ones who register fast. The bootstrap will only refer nodes who register at a rate less frequently than the rate it specifies.

Least Performing Peer (LP): This is a peer level scheme, similar to the one used in CoolStreaming [11], that drops the in-neighbor that is currently contributing the least amount of data. We chose this alternative to LR because of its simplicity and to show that while simple schemes such as this prove to be effective in a setting where all nodes are benign, more robust methods are needed when malicious nodes are present.

Drop Periodically (DP): This is a source protection scheme that induces churn [58] on the source. We note that as time progresses and benign nodes churn in and out of the system, malicious nodes can continue to stay and eventually eclipse the source as its neighbors. To address this problem we allow a single node to stay as an out-neighbor for only a certain amount of time and then disconnect it. To further stagger the disconnection times of nodes, we only allow one node to be disconnected in a time period.

Periodic Register (PR): This is a bootstrap protection scheme that requires all peers to re-register every r time. We chose this scheme as an alternative to RB since it also requires re-registration of nodes, but does not do any rate-limiting. Thus, it demonstrates that more robust methods are needed when malicious nodes target the bootstrap service.

7.2. Attacks Considered

To show the effectivness of our schemes, we also implemented the most effective attacks available for an attacker to disrupt the data delivery.

- **Data dropping attack:** A malicious node will advertise the chunks of data that it has, but will not fulfill any requests made for those chunks unless otherwise stated.
- **Data delaying attack:** A malicious node will advertise and fulfill a small amount of chunk requests to avoid being dropped from another node's neighbor set. However, the malicious node will delay the sending of the chunk until very close to the chunk's deadline.
- **Source attack:** A malicious node will not forward data given to it by the source. As the source has limited bandwidth, if only malicious nodes receive a particular chunk, then that chunk will be effectively lost and no benign nodes will receive it.

Table 2: Mechanisms for each component of system

Peers	Source	Bootstrap
Least Performing Peer (LP)	Drop Periodically (DP)	Periodic Register (PR)
Local Reputation (LR)	Health Monitoring (HM)	Rate-limiting (RB)
LR Data Delaying (LR-DD)		

• **Bootstrap list pollution attack:** A malicious node will register often with the bootstrap, to ensure that it is always in its list of peers and to increase its chances of being referred to benign nodes.
• **Neighbor selection attack:** When a benign node contacts a malicious node to discover more peers to connect to, the malicious node will bias its referrals to include only other malicious nodes. This results in benign nodes being neighbors with many malicious nodes.

7.3. Experiment Configuration

Table 3: Notation

D_r	Deadline at which a peer considers a request for data dropped
T_s	The suspicion threshold
T_d	The drop threshold
α	When calculating $R'_{ij}(t)$ gives a weight to $C_{ij}(t)$
λ	When calculating $R_{ij}(t)$ gives a weight to the previous value of $R_{ij}(t-1)$ and the current value of $R'_{ij}(t)$
l	When using IRS to calculate $G_{ij}(t)$, defines how much stretch is to be tolerated

The experiments were run on the PlanetLab overlay testbed. The source was located on a host at our lab. We set D_r (see Table 3 for a list of parameters and their definitions) to be 1 second. We determined this value experimentally as we observed that in a non-malicious scenario 96% of nodes receive 99% of chunks within 1 second. Each node is configured to obtain up to 15 in-neighbors and the maximum number of out-neighbors is proportional to its bandwidth. The source will obtain 30 out-neighbors.

We used overlay deployments of 300 nodes. Each experiment lasted for 10 minutes. For each experiment we varied the percentage of malicious nodes from 0 to 30% and fixed the source's streaming rate at 1 Mbps. Each experiment was run for 10 times and the results were averaged. Standard deviations are plotted where appropriate. The malicious nodes joined at the beginning of the experiment and stayed for the entire duration. Benign nodes both join at the beginning of the experiment and also during the experiment. We modeled the join times by using a Poisson process and the participation time by a Pareto distribution. The mean of the Poisson process was 3 and the Pareto distribution is used with a shape parameter of 1.42, giving a mean participation time of 300 seconds and we also assume a minimum participation time of 90 seconds. The parameters have been used previously by Bharambe *et al.* [59] and were motivated by traces of real multicast systems [3] and Mbone sessions [60].

Choosing Parameters: For *Local Reputation* we by reason set its parameters to appropriate values and validated them experimentally. We set T_s to be 0.7 to tolerate transient network conditions. We note that T_s can be set by the user, to the minimum quality threshold that he is willing to tolerate. We set α to be 0.5 since we consider data plane feedback to be more useful than control plane feedback. We also conduct a sensitivity study of α in Section 8. For nodes to evict malicious nodes that are both suspicious and highly connected, the equation $T_d \geq T_s - \alpha$ must hold. Therefore we set T_d to be 0.2. We set λ to be 0.4 to give a greater weight to the history of the reputation but also be able to change quickly if nodes consistently behave badly. We set the time period for the recalculation of the scores to be every 3 seconds. For Local Reputation with Data Delaying protection (*LR-DD*) we experimentally set l to be 3. Therefore, the delay of the chunk must be 3 times greater than the delay to the source before a node is penalized.

7.4. Performance Metrics

We evaluate the effectiveness of the attacks and solutions with the following metrics.
Goodput Ratio: This represents the percentage of useful data a node received while in the overlay, averaged across all nodes. We use it to measure the effects of churn on the quality of the goodput. The higher the goodput ratio, the higher the quality of the stream received.
Corruption Factor: This represents the percentage of nodes in the neighbor set that are malicious. We use it to measure the level of control an adversary has on the neighbor set of a particular node. The higher the corruption factor, the more adversarial neighbors a node has.

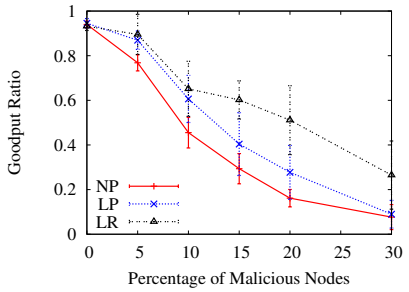
8. Experimental Evaluation

In this section we experimentally show that the schemes we proposed in Section 5 are able to effectively mitigate attackers.

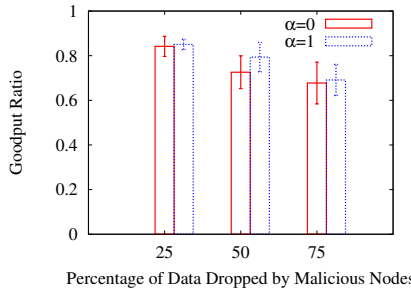
8.1. Robust Neighbor Selection

To give motivation to our *Local Reputation (LR)*, we first compare it to *Least Performing Peer (LP)* and *No Protection (NP)*. Malicious nodes perform data dropping, source and neighbor selection attacks. As can be seen in Figure 2(a) both *NP* and *LP* perform worse than *LR*. This difference becomes more pronounced as the percentage of malicious nodes increases. *NP* is ineffective simply because nodes never change who their neighbors are, regardless of their poor performance.

LP is not as effective as *LR* since a node never drops all of the malicious nodes from its neighbor set. Further investigation shows that for a node running *LP* the number of malicious nodes in its in-neighbor set decreases as

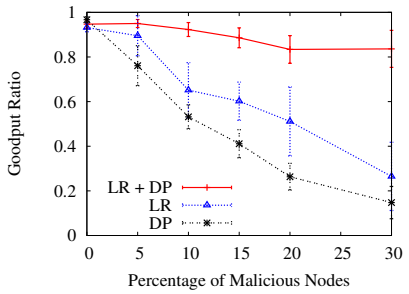


(a) *No Protection*, *Least Performing Peer*, and *Local Reputation* schemes running at the peers with *No Protection* at the source.

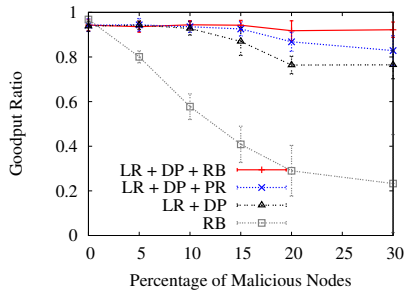


(b) Goodput ratio when 10% of participants are malicious and they drop varying amounts of the stream. *Local Reputation* is protecting the peers and α is varied.

Figure 2: Importance of peer protection.



(a) Peers running the *Local Reputation* scheme with *Drop Periodically*, *Pretrusted Peers*, and *No Protection* at the source. For comparison we also plot the *Drop Periodically* only scheme.



(b) *Rate-limiting Bootstrap*, *Periodic Register*, and *No Protection* are combined with *Local Reputation* and *Drop Periodically*. *Rate-limiting Bootstrap* with *No Protection* is also shown.

Figure 3: Importance of source and bootstrap protection.

some of the malicious nodes will be dropped. However, there are still malicious nodes present in the in-neighbor set because *LP* does not prevent the node from reconnecting multiple times to the same malicious nodes. When the node is running *LR*, it does not reconnect anymore to malicious nodes since malicious behavior is captured in the reputation score for those nodes.

Importance of considering graph connectivity: We examine the contribution of the graph connectivity score on *LR* and identify regimes in which its use is beneficial. We compare the case when the reputation score computation is based only on the data feedback (i.e. $\alpha = 0$) to the case when both data and graph connectivity are considered (i.e. $\alpha = 1$).

As we can see in Figure 2(b) when attackers drop 25% or 75% of the data they were expected to deliver, the performance does not change no matter the value of α. For the case of 25% dropping, recall that a node i will only calculate the graph connectivity score for a neighbor j if it marks j as suspicious (i.e. $L_{ij} < T_s$). When j drops 25% of the data it will not be marked as suspicious since we use a T_s value of 0.7, thus the graph connectivity score will not be considered. In the case of 75% dropping, enough data is dropped that the neighbor will be perceived as malicious by its data score alone. Hence graph connectivity is most useful in regimes where the amount of data dropped by a malicious node is large enough to be marked as suspicious, but not large enough to be interpreted as malicious by their data scores alone. This is the case for 50% dropping. In Figure 2(b), when attackers drop 50% of the data, *LR* combining the two scores performs better than *LR* using only data score. The information from the control plane about the existence of a neighbor selection attacks helps effectively identify malicious nodes.

We varied α even more to find values that give better performance but we found that a value of 1 is sufficient across all percentages of attackers.

8.2. Source Protection

While *LR* performs much better than other schemes, the goodput ratio achieved is still far from satisfactory. We believe this is because *LR* does not protect the streaming source, as we explained in Section 5. Further investigation into the source's performance confirms our hypothesis.

While peers using *LR* expel all malicious nodes from their in-neighbor set, the source's out-neighbor set is almost full of malicious nodes. This illustrates the importance of having additional mechanisms to protect the source.

We next evaluate mechanisms that can be used to protect the source. When using *Drop Periodically (DP)* the source will drop a node after it has been a neighbor for 1 minute. In these experiments malicious nodes again perform data dropping, source and neighbor selection attacks. Figure 3(a) shows the results. The goodput ratio is significantly raised for *DP* combined with *LR* (i.e. the curve titled *LR+DP*). This is because *DP* effectively reduces the corruption factor at the source to a value that is very close to the percentage of malicious nodes in the overlay at all times, for all settings.

Figure 3(a) also shows that *DP* alone is not sufficient. This is not surprising, because *DP* protects only the source, not the peers. This again highlights that solutions must be employed at both the source and peers to achieve satisfactory performance.

8.3. Rate-limiting Bootstrap

We now consider when malicious nodes also conduct a bootstrap list pollution attack, along with the data dropping, source, and neighbor selection attacks. We evaluate the effectiveness of *Rate-limiting Bootstrap (RB)* in mitigating such attacks and compare it with *Periodic Register (PR)*. Two parameters influence the performance of *RB*: the time period in which a node may register only once to be considered as non-malicious (w) and the size of the short list maintained by the bootstrap (k).

Selection of w: Taking into consideration the trade-offs described in Section 5.4, we set w conservatively at 300 seconds. This value is much smaller than the typical session length in P2P streaming systems, which is usually in the order of tens of minutes [6, 61, 3]. For *Periodic Register (PR)*, we use a w value of 120 seconds to show that even when sacrificing overhead for a more up-to-date list and thus better security, rate-limiting schemes are still preferred.

Selection of k: We experimentally determine the value of k. We fixed the system solution to be *LR+DP+RB* and varied the value of k. The malicious nodes are aware of the solution and only register every w seconds in order not to get themselves excluded from the bootstrap's short list. The malicious nodes all register at the same time. Note that if they space out their registrations, the impact on the bootstrap list would be diluted. In Figure 4(a) there are four sets of bars each with a different k value, and two bars in each set corresponding to the maximum and average corruption factor at the source. The figure shows that as k increases, the maximum corruption factor decreases, but the average corruption factor increases. This is because the smaller the k, the easier it is for the attacker to flood the bootstrap's short list in a burst, thus achieving a high corruption factor at the source. However, each node will only remain on the list until k more nodes have joined. Thus, the larger the k, the longer a malicious node will stay on the list, resulting in a higher average corruption factor. From the figure we conclude that setting k at 50 is a good trade-off between having a large corruption factor all the time and having a large spike in the corruption factor every w seconds. In the rest of our experiments we set k to be 50.

Figure 3(b) shows the evaluation results. *RB* combined with solutions for source and peers (i.e. the curve titled *LR+DP+RB*) performs the best and mitigates the attack across all malicious percentages. *PR* combined with other solutions (i.e. *LR+DP+PR*) works equally well for small percentage of attackers (up to 15%). For higher percentages of attacker nodes, *PR* effectiveness decreases because the scheme simply puts both benign and malicious nodes on equal footing. Thus, while the bootstrap's list of nodes is very close to being up-to-date, it does not punish attackers. On the other hand the *RB* solution is more effective for exactly this reason, if nodes register too fast they are not made known to nodes who request a list of peers. We also note that *PR* incurs a large overhead at the bootstrap node as it requires all nodes to re-register with the bootstrap node often. Lastly, both schemes perform significantly better than *NP*, highlighting the importance of having additional mechanisms to protect the bootstrap node.

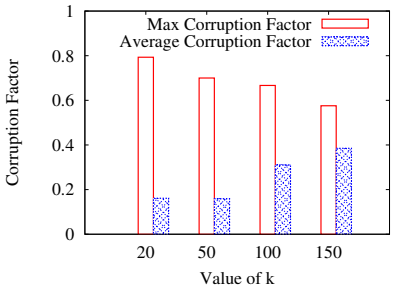
To gain more insight into these results, we also plot in Figure 4(b) the corruption factor at the source for each solution. Recall that *DP* at the source requires that the source only obtains neighbors from the bootstrap node, thus the degree of pollution at the bootstrap node directly affects the corruption factor at the source. The figure shows that the corruption factor is significantly lower with the *LR+DP+RB* than other solutions, further confirming its effectiveness.

8.4. Data Delaying

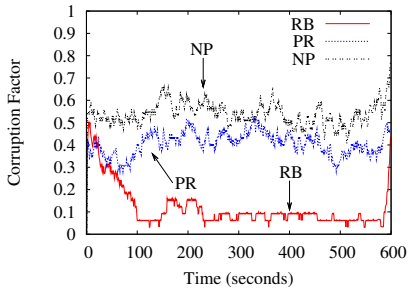
As evidenced by our analysis and experiments, attackers will be thwarted as long as they continue to drop data. To prolong the amount of time they can stay as neighbors and thus do more damage, attackers will necessarily have to actually give data to others. However, attackers are motivated to make sure the data that is given out is as useless as possible to benign nodes. Attackers can achieve this by delaying the sending of data to the last possible moment.

For the attack to succeed even though data is still being given away the attacker will need to make sure it has a good strategy for only giving out data that will become very common and not data that will remain rare. Attackers obviously do not know the future, but can assume that if no other benign nodes have the data then they should not pass it on to others, but if some other benign nodes do have the data, they can upload it to others.

To show how effective delaying is, we now run experiments where malicious nodes conduct data delaying instead of data dropping attacks. Malicious nodes also con-



(a) Corruption factor at the source on a single experiment when 20% of nodes are malicious and they are aware that *Rate-limiting Bootstrap* is running and do not register fast. We vary k to find the best value.



(b) Corruption factor at the source on a single experiment when 20% of nodes are malicious. We compare the effects on the source of the three different bootstrap node schemes.

Figure 4: Evaluating the corruption factor in different scenarios.

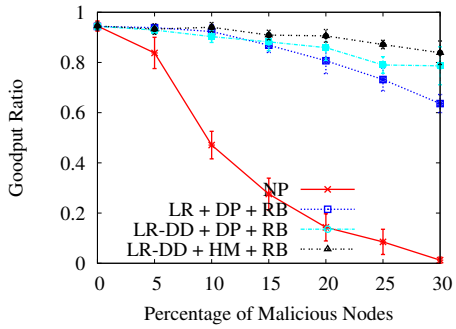


Figure 5: Peers running the local reputation scheme while attackers conduct data delaying attacks.

duct source, neighbor selection and bootstrap list pollution attacks. We deploy the *LR-DD* and *HM* protection schemes and show how well they mitigate attacks. For *HM* we set the fraction f of nodes that the source must get complaints for before it disconnects a node to be 30%.

We present the results in Figure 5, which demonstrate that these attacks are effective in increasing damage done. For example, when peers and source are just protected with *LR* and *DP*, (i.e. the curve entitled *LR+DP+RB*) nodes increasingly have worse performance as the fraction of malicious nodes increases, culminating in only a .68 goodput ratio when there are 30% malicious attackers. As *LR-DD* and *HM* protection schemes are added performance increases, effectively mitigating the attack. *LR-DD* proves to be effective as most benign nodes receive data fairly quickly after the source sends it out, thus malicious nodes delaying data are promptly removed from in-neighbor sets. *HM* also outperforms *DP* as it is able to actually identify malicious nodes who do not forward data to others and disconnect them, rather than simply keeping the fraction of malicious nodes low.

9. Conclusion

In this paper, we present one of the first efforts aimed at systematically analyzing and addressing the vulnerabilities of mesh-based P2P streaming systems to malicious insider attacks. We consider both direct attacks on the data plane, as well as attacks on the control plane which could in turn lead to further disruption of data delivery. These include data dropping and neighbor selection attacks, as well as data delaying, which is a novel attack on P2P streaming. We present a design for securing data delivery, of which a key component is a reputation scheme that helps nodes identify malicious peers and build a robust neighbor set. Through detailed security analysis, we show that our scheme is resistant to a variety of attacks commonly associated with reputation schemes such as self-promotion, slandering, and white-washing [39].

We present an extensive evaluation of our design through experiments on PlanetLab. Our results show that (i) without our solution, the data delivery can be seriously disrupted by attacks exploiting the vulnerabilities we identified. For example, 15% malicious nodes caused the average goodput ratio to decrease to less than 30%. (ii) Our solution is effective in mitigating the attacks; it achieves an average goodput ratio of more than 90% even when there are 30% malicious nodes conducting data dropping attacks and over 83% average goodput ratio when there are 30% malicious nodes conducting data delaying attacks. (iii) While each of the mechanisms we introduce can individually benefit the system, the solution is most effective when all the mechanisms are combined.

References

[1] S. Deering, D. Cheriton, Multicast routing in datagram internetworks and extended lans, ACM Transaction on Computer Systems 8 (1990) 85–110.

[2] Y. Chu, S. G. Rao, H. Zhang, A case for end system multicast, in: ACM SIGMETRICS, 2000.

[3] Y.-H. Chu, A. Ganjam, T. S. E. Ng, S. Rao, K. Sripanidkulchai, J. Zhan, H. Zhang, Early experience with an internet broadcast system based on overlay multicast, in: USENIX, 2004.

[4] PPLive, http://www.pplive.com (2010).
[5] PPStream, http://www.ppstream.com (2010).
[6] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, C. Huang, Challenges, design and analysis of a large-scale p2p-vod system, in: SIGCOMM, 2008.
[7] X. Hei, C. Liang, J. Liang, Y. Liu, K. W. Ross, A measurement study of a large-scale p2p iptv system, IEEE Trans. on Multimedia 9 (2007) 1672 – 1687.
[8] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, A. Singh, Splitstream: High-bandwidth multicast in cooperative environments, in: SOSP, 2003.
[9] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, A. Mohr, Chainsaw: Eliminating trees from overlay multicast, in: IPTPS, 2005.
[10] V. Venkataraman, K. Yoshida, P. Francis, Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast, in: ICNP, 2006.
[11] X. Zhang, J. Liu, B. Li, T.-S. P. Yum, CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming, in: IEEE INFOCOM, 2005.
[12] D. Kostic, A. Rodriguez, J. Albrecht, , A. Vahdat, Bullet: High bandwidth data dissemination using an overlay mesh, in: SOSP, 2003.
[13] UUSee, http://www.uusee.com (2010).
[14] SOPCast, http://www.sopcast.com/ (2010).
[15] Pdbox, http://www.pdbox.co.kr (2010).
[16] QQLive, http://live.qq.com (2010).
[17] StreamerOne, http://www.streamerone.it/ (2010).
[18] TVUnetworks, http://www.tvunetworks.com (2010).
[19] VGO, http://vgo.21cn.com (2010).
[20] Zattoo, http://zattoo.com (2010).
[21] Veetle, http://www.veetle.com (2010).
[22] K. Cho, K. Fukuda, H. Esaki, A. Kato, Observing slow crustal movement in residential user traffic, in: CONEXT, 2008.
[23] M. Meeker, D. Joseph, The state of the internet, part 3, in: Web 2.0, 2006.
[24] H. Schulze, K. Moschalski, Ipoque internet study 2008/2009, http://www.ipoque.com (2009).
[25] V. N. Padmanabhan, H. J. Wang, P. A. Chou, Resilient peer-to-peer streaming, in: ICNP, 2003.
[26] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, J. O. Jr., Overcast: Reliable multicasting with an overlay network, in: OSDI, 2000.
[27] D. Ren, Y.-T. Li, S.-H. Chan, On reducing mesh delay for peer-to-peer live streaming, in: INFOCOM, 2008.
[28] F. Wang, J. Liu, Y. Xiong, Stable peers: Existence, importance, and application in peer-to-peer live video streaming, in: INFOCOM, 2008.
[29] J. Liu, S. G. Rao, B. Li, H. Zhang, Opportunities and challenges of peer-to-peer internet video broadcast, in: Proceedings of the IEEE, Special Issue on Recent Advances in Distributed Multimedia Communications, 2007.
[30] J. Seibert, D. Zage, S. Fahmy, C. Nita-Rotaru, Experimental comparison of peer-to-peer streaming overlays: An application perspective, in: IEEE LCN, 2008.
[31] N. Magharei, R. Rejaie, PRIME: Peer-to-peer receiver driven mesh-based streaming, in: IEEE INFOCOM, 2007.
[32] P. Dhungel, X. Hei, K. Ross, N. Saxena, The pollution attack in p2p live video streaming: Measurement results and defenses, in: ACM SIGCOMM P2P-TV Workshop, 2007.
[33] M. Haridasan, R. van Renesse, Defense against intrusion in a live streaming multicast system, in: P2P, 2006.
[34] Z. Liu, Y. Shen, K. W. Ross, S. S. Panwar, Y. Wang, Substream trading: Towards an open p2p live streaming system, in: ICNP, 2008.
[35] M. Piatek, A. Krishnamurthy, A. Venkataramani, R. Yang, D. Zhang, A. Jaffe, Contracts: Practical contribution incentives for p2p live streaming, in: NSDI, 2010.
[36] F. Pianese, S. Member, D. Perino, J. Keller, E. W. Biersack, Pulse: an adaptive, incentive-based, unstructured p2p live streaming system, in: IEEE Transactions on Multimedia, 2007.
[37] H. C. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, M. Dahlin, Flightpath: Obedience vs choice in cooperative services, in: OSDI, 2008.
[38] B. Cohen, Incentives build robustness in BitTorrent, in: P2P Economics, 2003.
[39] K. Hoffman, D. Zage, C. Nita-Rotaru, A survey of attack and defense techniques for reputation systems, ACM Computing Surveys.
[40] S. Liu, M. Chen, S. Sengupta, M. Chiang, J. Li, P. A. Chou, P2p streaming capacity under node degree bound, 2010.
[41] F. Picconi, L. Massoulié, ISP-Friend or Foe? Making P2P Live Streaming ISP-aware, in: ICDCS, 2009.
[42] Z. Liu, C. Wu, B. Li, S. Zhao, Uusee: Large-scale operational on-demand streaming with random network coding, in: INFOCOM, 2010.
[43] A. T. Nguyen, B. Li, F. Eliassen, Chameleon: Adaptive peer-to-peer streaming with network coding, in: INFOCOM, 2010.
[44] J. Seedorf, Security issues for p2p-based voice- and video-streaming applications, in: J. Camenisch, D. Kesdogan (Eds.), iNetSec 2009 Open Research Problems in Network Security, Vol. 309 of IFIP Advances in Information and Communication Technology, Springer Boston, 2009, pp. 95–110.
[45] G. Gheorghe, R. Lo Cigno, A. Montresor, Security and privacy issues in p2p streaming systems: A survey, Peer-to-Peer Networking and Applications (2010) 1–17.
[46] Q. Wang, K. N. Long Vu, H. Khurana, Identifying malicious nodes in network-coding-based peer-to-peer streaming networks, in: INFOCOM, 2010.
[47] L. Xie, S. Zhu, Message dropping attacks in overlay networks: Attack detection and attacker identification, ACM Trans. Inf. Syst. Secur. 11 (3) (2008) 1–30.
[48] A. Walters, D. Zage, C. Nita-Rotaru, Mitigating attacks against measurement-based adaptation mechanisms in unstructured multicast overlay networks, in: ICNP, 2006.
[49] I. Keidar, R. Melamed, A. Orda, Equicast: Scalable multicast with selfish users, in: PODC, 2006.
[50] M. Castro, P. Drushel, A. Ganesh, A. Rowstron, D. Wallach, Secure routing for structured peer-to-peer overlay networks, in: OSDI, 2002.
[51] A. Singh, T.-W. J. Ngan, P. Druschel, D. S. Wallach, Eclipse attacks on overlay networks: Threats and defenses, in: IEEE INFOCOM, 2006.
[52] S. Kamvar, M. Schlosser, H. Garcia-Molina, The EigenTrust Algorithm for Reputation Management in P2P Networks, in: Proceedings of WWW2003, ACM, 2003.
[53] M. Piatek, T. Isdal, A. Krishnamurthy, T. Anderson, One hop reputations for peer to peer file sharing workloads, in: NSDI, 2008.
[54] K. Walsh, E. G. Sirer, Experience with an object reputation system for peer-to-peer filesharing, in: NSDI, 2006.
[55] B. Biskupski, R. Cunningham, J. Dowling, R. Meier, High-bandwidth mesh-based overlay multicast in heterogeneous environments, in: AAA-IDEA, 2006.
[56] J. Douceur, The Sybil Attack, in: IPTPS, 2002.
[57] H. Yu, P. B. Gibbons, M. Kaminsky, F. Xiao, Sybillimit: A near-optimal social network defense against sybil attacks, in: Symposium on Security and Privacy, 2008.
[58] T. Condie, V. Kacholia, S. Sankararaman, J. M. Hellerstein, P. Maniatis, Induced churn as shelter from routingtable poisoning, in: NDSS, 2006.
[59] A. Bharambe, S. Rao, V. Padmanabhan, S. Seshan, H. Zhang, The impact of heterogeneous bandwidth constraints on DHT-based multicast protocols, in: IPTPS, 2005.
[60] K. Almeroth, M. Ammar, Characterization of mbone session dynamics: Developing and applying a measurement tool, Tech. Rep. GIT-CC-95-22, Georgia Institute of Technology (1995).
[61] B. Li, S. Xie, G. Y. Keung, J. Liu, I. Stoica, H. Zhang, X. Zhang, An empirical study of the coolstreaming+ system, IEEE Journal on Selected Areas in Communications 25 (9) (2007) 1627–1639.