Enforcing Fairness in a Live-Streaming System*

Maya Haridasan[a], Ingrid Jansch-Porto[b] and Robbert van Renesse[a]

[a]Dept. of Computer Science, Cornell University
Ithaca, New York
[b]Institute of Informatics, Federal University of Rio Grande do Sul
Porto Alegre, Brazil
maya@cs.cornell.edu, ingrid@inf.ufrgs.br, rvr@cs.cornell.edu

ABSTRACT

We describe a practical auditing approach designed to encourage fairness in peer-to-peer streaming. Auditing is employed to ensure that correct nodes are able to receive streams even in the presence of nodes that do not upload enough data (opportunistic nodes), and scales well when compared to previous solutions that rely on tit-for-tat style of data exchange. Auditing involves two roles: local and global. Untrusted local auditors run on all nodes in the system, and are responsible for collecting and maintaining accountable information regarding data sent and received by each node. Meanwhile, one or more trusted global auditors periodically sample the state of participating nodes, estimate whether the streaming quality is satisfactory, and decide whether any actions are required. We demonstrate through simulation that our approach can successfully detect and react to the presence of opportunistic nodes in streaming sessions. Furthermore, it incurs low network and computational overheads, which remain fixed as the system scales.

1. INTRODUCTION

Video and audio streaming account for a large percentage of content accessed over the web. One popular style of streaming on the web is *on demand*, in which users access pre-stored content at will. Another style requires streams to be generated and disseminated in real-time. This may be the case with important social, political, or sporting events. An important property of *live-streaming* is that data is not available in advance, being generated just before transmission at the sender. Furthermore, interested users ideally want to receive the stream without much delay from its original transmission.

Several practical live-streaming systems now allow large numbers of interested users to receive streamed data in near real time, without requiring extensive amounts of resources. These systems are based on the peer-to-peer (P2P) paradigm, where nodes interested in receiving data also help disseminate it to each other, alleviating the bottleneck at the source. Initial protocols were based on building a tree-based overlay of nodes through which data would be pushed.[1–3]

More recent systems, such as Chainsaw and Coolstreaming, have shown that the use of a mesh of connected nodes and a pull-based data dissemination approach can provide similar results with better resilience to failures and churn (nodes joining and leaving the system).[4–7] In Chainsaw, for example, nodes notify each other of receipt of data packets, and request packets from their neighbors based on the received notifications. Practical systems based on pull-based streaming now exist in China, where they are used to disseminate television channels to thousands of users.[8]

Even though the P2P paradigm allows systems to scale with the number of users, it also leaves them vulnerable to opportunistic behavior. Opportunistic nodes attempt to receive a stream without uploading their fair share of data, reducing the overall upload capacity of the system. Despite the damage that they may cause, not much work has been done in studying mechanisms to avoid their presence in live-streaming systems. The goal of this

*The authors were supported by AFRL award FA8750-06-2-0060 (CASTOR), NSF award 0424422 (TRUST), AFOSR award FA9550-06-1-0244 (AF-TRUST), DHS award 2006-CS-001-000001 (I3P), and Intel Corporation. The views and conclusions herein are those of the authors.

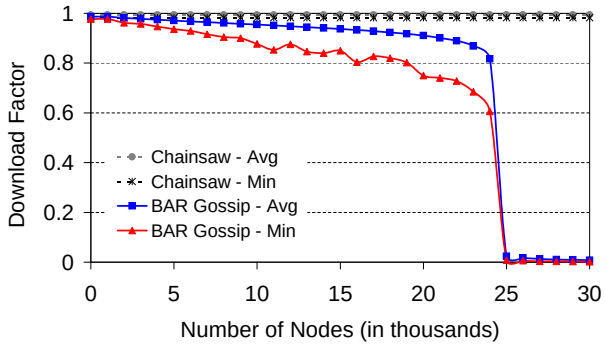

Figure 1. Minimum and average download rates across all nodes when using the BAR Gossip and Chainsaw protocols.

paper is to propose and evaluate a mechanism that can defend against this problem, whithout incurring large overheads.

The approach that most closely relates to our work is the BAR Gossip protocol,[9] which employs a *tit-for-tat* approach for encouraging nodes to contribute: a node only sends as much data to another node as it receives back. It provides an elegant solution shown to tolerate both opportunistic behavior and other malicious attacks. However, reliance on tit-for-tat does present a few undesirable requirements. To be efficient, the data source should ensure that packets are evenly spread across the system by sending data to a fixed proportion of nodes, and by sending different packets to different nodes. Furthermore, it requires the source and all nodes to have full membership knowledge. These restrictions affect scalability when the data source has bounded upload bandwidth.

To illustrate this problem, we fixed the upload capacity of a data source at 5 Mbps and simulated BAR Gossip when streaming 500 Kbps with increasing numbers of receivers, varied between one and thirty thousand nodes. We compare its scalability against the Chainsaw protocol,[4] for which we fixed the source's upload bandwidth to 2 Mbps. In Figure 1, we present the average and minimum download rates (as ratios of the stream rate) of both protocols when the number of nodes is increased. As observed, BAR Gossip is not able to sustain its performance without scaling the upload capacity of the source proportionally with the size of the system. Meanwhile, Chainsaw is able to scale well even with a fixed lower upload bandwidth at the source, but cannot handle the presence of opportunistic nodes.

We propose to use auditing to encourage data-sharing in live-streaming systems like Chainsaw. Our auditing approach establishes a minimum threshold for the amount of data sent by any node in the system, and removes nodes that upload less data than the threshold. Instead of relying on a tit-for-tat mechanism, we focus on encouraging nodes to respect the established protocol. Nodes are forced to provide accountable information regarding packets sent to and received from neighbors, and the auditing system is responsible for detecting and removing misbehaving nodes.

Notice that identifying the misbehaving nodes is not a trivial task, since there is no fixed minimum amount of data that nodes should contribute to the system. If we assume a model where misbehaving nodes simply did not upload any data, detecting them would be an easier task. However, once we assume that misbehaving nodes may adjust their contribution level based on the policy used by an auditing system, a more elaborate approach is required. This paper presents and evaluates an auditing model based on sampling the system and using the sampled information to build a global view of how the system is currently behaving. Based on it, auditors employ strategies to identify the misbehaving nodes that should be punished.

The paper is organized as follows. In section 2, we state the exact problem that we aim to solve and the assumptions considered in this work. In section 3, we review the pull-based streaming protocol employed in our system, followed by a description of our novel auditing approach in section 4. In section 5, we evaluate the proposed approach. We then discuss the costs of auditing, and briefly describe how to extend our model for heterogeneous systems, in section 6. Finally, we present related work in section 7, and conclude in section 8.

2. PROBLEM STATEMENT

Our approach focuses on a target streaming system consisting of one data *source* (assumed non-compromised), which disseminates data at a fixed rate to a dynamic set of receivers. The source has limited upload bandwidth, and hence can only send data directly to a small subset of interested receivers. Participating nodes are consequently required to forward packets to their neighbors, helping disseminate all packets across the system. The streamed data should be received by all nodes within a fixed latency from the source's original transmission, even in the presence of opportunistic nodes.

For simplicity, we first assume a system in which all nodes, except the source, have similar upload and download bandwidths; in Subsection 6.2, we briefly discuss how to extend our model to work in heterogeneous scenarios.

We assume that malicious nodes exhibit Byzantine behavior, while correct nodes follow the protocol as defined, requesting data as needed and sending data as requested from them. Altrustic nodes are a subgroup of correct nodes that are willing to upload more data than required from them. Finally, we employ the term *opportunistic* to refer to a subgroup of Byzantine nodes that attempt to give less data than they would if they behaved as correct nodes, with the intention of obtaining as much data as possible at least feasible cost. These may employ a simple strategy, such as refuse to contribute any upload resources, or a more elaborate strategy that allows them to cheat without being easily detected.

Notice that our model diverges from the one used in BAR Gossip,[9] in which nodes are classified as *Byzantine*, *Altruistic*, or *Rational*. In that model, rational nodes attempt to maximize their utility while still following the defined protocol. Our model is actually less lenient: nodes employing strategies to maximize their utility are classified as Byzantine, so that we can build a practical punishment-based system in which any node not contributing its fair share of data may be expelled from the system.

Throughout the paper we use the terms *upload factor* and *download factor* to refer to the ratio between an upload or download rate and the original stream rate. For example, given a stream rate of 500 Kbps, a download rate of 400 Kbps corresponds to a download factor of 0.8.

3. STREAMING SYSTEM MODEL

Our auditing approach is used over the Chainsaw protocol.[4] All nodes participating in the system are organized into a fully connected mesh overlay, where each node has the same number of neighbors. The source is randomly connected to a small subset of the nodes.

The streaming process starts at the source, which breaks the data stream into packets and sends notifications to its neighbors as soon as it has packets to disseminate. These notifications are small messages used only to inform neighbors of the availability of new packets. Based on the received notifications, each node requests missing packets, and the source satisfies as many requests as allowed by its upload capacity. Unlike BAR Gossip, with Chainsaw the upload capacity of the source does not need to increase with the size of the system; even an upload capacity of twice the stream rate is sufficient to ensure that the system performs and scales well.

As nodes receive packets, they mimic the role of the source, sending notifications to their own neighbors in the mesh, allowing packets to be propagated through the system. This pull-based approach to acquisition of packets (notify-request-send data) provides some resilience to failure or malicious behavior, since a participant will have multiple possible sources for each packet. The mesh overlay defines a predetermined set of neighbors for each peer, which also makes it hard for malicious peers to round up on individual peers since attackers lack a deterministic means of acquiring control of all of its neighbors. All nodes with exception of the source have a fixed upper limit on their upload contribution (e.g. 1.2 times the stream rate), defined by the protocol. Of course, this upper limit is not respected by opportunistic nodes, who attempt to reduce it with the goal of uploading less data.

On the course of a streaming session, each node stores packets and forwards them to other peers only while the packet is within its *availability window*, usually spanning a few seconds. Each node also maintains an *interest window*, which represents the set of packets in which the peer is currently interested. Nodes choose packets to request from each of its neighbors, respecting a maximum limit l on the number of outstanding requests to each

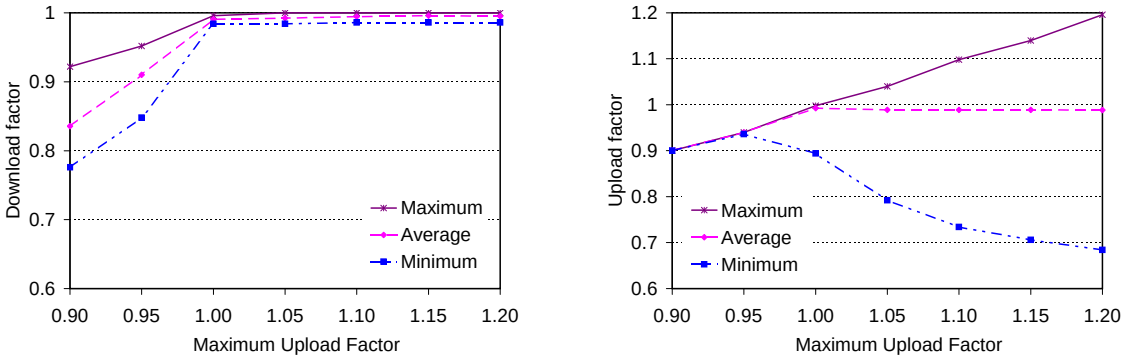

Figure 2. Download and upload factors of nodes in an ideal system where all nodes behave correctly.

neighbor. This limit not only improves the general flow of packets, but also makes it harder for malicious peers to overrequest packets from their neighbors: peers maintain a queue of non-satisfied requests from its neighbors, keeping only the l most recent ones.

3.1 Expected Behavior

Our first goal is to explore the typical signature of the system, since an understanding of the behavior of pull-based dissemination in the absence of opportunistic nodes will turn out to be important when we set out to introduce auditing. We conducted experiments using an event-based simulator, which is described in more detail in section 5.

In Figure 2, we evaluate the performance of 1000 nodes during an ideal execution of Chainsaw, where all the nodes behave correctly. We fixed the upload factor of the source at 4.0 (2 Mbps), and the stream rate to 500 Kbps. We varied the maximum upload factor of nodes to see how it affected both the download and upload factors of nodes across the system. The maximum upload factor is a fixed parameter which defines the maximum rate at which a node will upload data to all its neighbors. For fairness in nodes' bandwidth consumption, we would like all nodes to upload data at a factor as close as possible to 1.0. We varied the maximum upload factor of nodes from 0.9 to 1.2.

The left graph shows the minimum, average and maximum download factors across the nodes when the maximum upload factor of nodes is increased. As observed, by increasing the maximum upload factor of nodes, we increase the global upload capacity of the system, leading to a better flow of packets. However, the discrepancy among the upload factors of individual nodes also increases, as seen in the graph to the right. When the maximum upload factor is increased, some nodes participate more actively in dissemination while others end up contributing less, even though all of them are behaving correctly. This is an important consideration: when we introduce auditing, we do not want to punish nodes that are willing to contribute but cannot do so because of factors such as their physical positioning in the system. In all our future experiments we set the maximum upload factor to 1.1.

3.2 Effect of Opportunistic Behavior

Our next goal was to understand the expected behavior of correct nodes under different scenarios where opportunistic nodes compromise the system. We therefore studied how the download and contribution rates of correct nodes are affected under these conditions. Opportunistic nodes may contribute with some data in an attempt to disguise their opportunistic behavior. Therefore, we considered different rates of contribution for opportunistic nodes: 0 (pure freeloaders), 100, 200, 300 and 400 Kbps.

Figure 3 presents the average and minimum download factors among all correct nodes under different configurations. The stream rate was fixed at 500 Kbps, and all correct nodes had a maximum upload factor of 1.1 (550 Kbps). We ran experiments with 1000 nodes and increasing percentages of opportunistic nodes in the system (from 0 to 90%). On the x-axis, we vary the percentage of opportunistic nodes. As expected, we can observe that the download factors of correct nodes decreases since the aggregated upload capacity in the system becomes

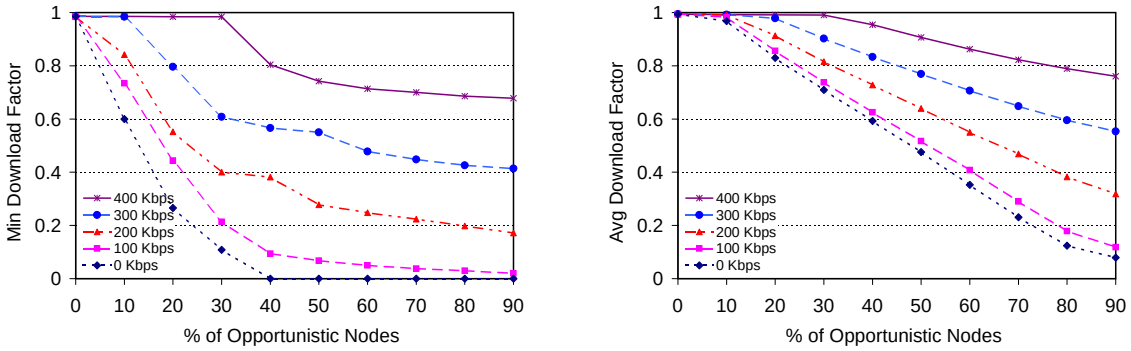

Figure 3. Minimum and average download factors across all correct nodes when opportunistic nodes are present. Each curve corresponds to a different contribution rate used by opportunistic nodes.

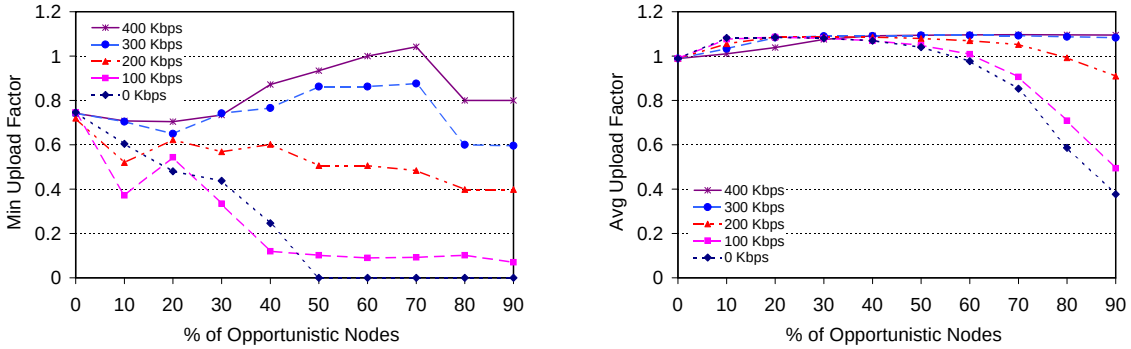

Figure 4. Minimum and average upload factors across all correct nodes when opportunistic nodes are present. Each curve corresponds to a different contribution rate used by opportunistic nodes.

insufficient to provide all nodes with all data. Nonetheless, the extent of the impact may be surprising: with just 10% opportunistic nodes, performance drops by as much as 40%.

Figure 4 presents the average and minimum upload factors among all correct nodes. Once again, on the x-axis we vary the percentage of opportunistic nodes, and on the y-axis we present the upload factors of nodes, which can vary up to 1.1. It is interesting to note that the average upload factor among correct nodes initially increases, and then starts falling when the percentage of opportunistic nodes increases significantly. This behavior can be explained by the fact that, initially, correct nodes start contributing more to compensate for the lack of data provided by a small percentage of opportunistic nodes; however, once the effect of opportunistic nodes becomes significant, the system collapses and correct nodes are not able to keep contributing.

Another important point to note is that the minimum upload factor does not follow a clearly defined pattern, making it hard to estimate the minimum contribution of correct nodes under compromised scenarios. Therefore, by applying thresholds to punish opportunistic nodes, correct nodes may also be unfairly penalized.

4. AUDITING PROTOCOL

Our idea for auditing the described live-streaming system against opportunistic behavior is motivated by the graphs presented in the previous section: we propose to employ auditing to ensure that all nodes in the system contribute more than a particular specified threshold. In Figure 5, we illustrate the potential benefit from using auditing in a system where 70% of the nodes are correct and 30% are opportunistic. The latter do not upload any data. During the first 100 seconds, no punishment was applied in an attempt to simulate a system with no auditing. At time t = 100s, auditing is enabled and opportunistic nodes start to be expelled from the system for low contribution. For this experiment, the minimum upload factor for nodes to stay in the system was set to 0.5.

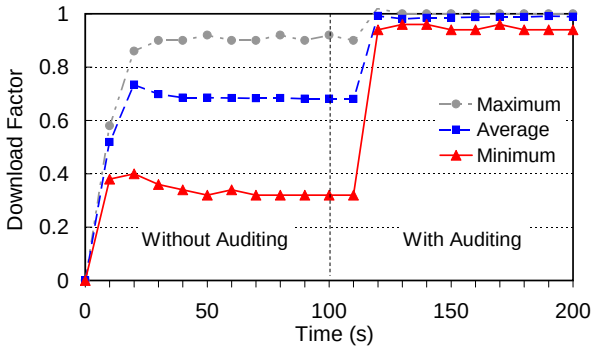

Figure 5. Download factor of correct nodes during a 200 second streaming session with 30% opportunistic nodes. Auditing is enabled in the last 100 seconds.

We present the minimum, average and maximum download factors across correct nodes varying along 200 seconds. As observed in this particular example, auditing has the potential to improve the quality of streamed sessions significantly, and at low cost. One important concern is that if the specified threshold is too high, more opportunistic nodes may be caught, but correct nodes may also be unfairly punished. In this experiment, no correct nodes were mistakenly expelled from the system.

4.1 Auditing components

We now give some additional details of the auditing architecture, focusing upon two aspects: (1) collecting accountable information about the download and upload factors of individual nodes in the system; and (2) establishing and applying the best threshold at any given time during execution. We employ two types of components to perform these two roles: local and global auditors. Local auditors are executed on the nodes participating in the system, and therefore cannot be trusted; if a node is malicious, it might report false data. Global auditors are trusted components that run on dedicated external nodes. There can be just one or a few global auditors. We describe their roles and interactions in detail below.

4.1.1 Local Auditors

Each node n runs a local auditor, which interacts with other local auditors and has two main roles:

Publish n's data exchange history: n's local auditor periodically compiles and distributes the history of packets exchanged by n. To acomplish this, every δ seconds, it queries the local streaming application running on n for the set of packets it sent and received using the streaming protocol in the most recent time interval (Figure 6). The local auditor signs and publishes the collected history to an assigned subset of its neighboring nodes, from whom other auditors may obtain it. This level of indirection is used to prevent nodes from masking their real upload and download factors by presenting different information to different auditors.

Audit n's neighbors' histories: n's local auditor periodically audits the published histories of the nodes with whom n exchanges packets. For instance, if node n exchanges packets with nodes p, q and r in the live-streaming protocol, n's local auditor compares these three nodes' histories with n's own history. This involves ensuring that: (1) the amount of data sent by these nodes satisfies the defined minimum threshold for the system; and (2) the set of packets they claim to have sent to and received from node n corresponds to the set of packets n claims to have respectively received from and sent to them. If the first check comparison fails, the local auditor issues an *accusation* against the node to a global auditor. In the second case, the local auditor is not able to prove the neighbor's misbehavior; instead, it instructs its local streaming application to not further exchange packets with the misbehaving neighbor. More complex types of checks may also be performed to address other types of Byzantine behavior.

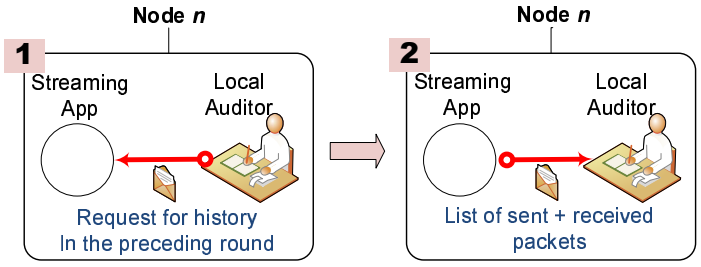

Figure 6. Local Auditing

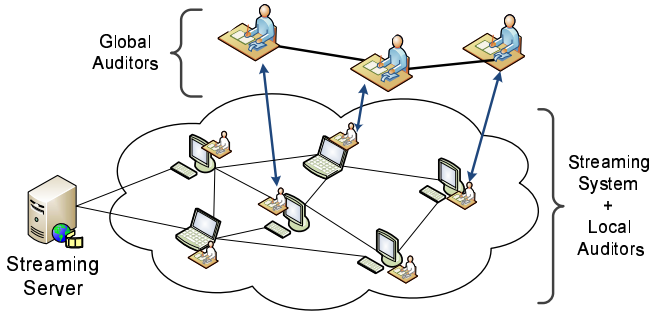

Figure 7. Global Auditing

There are two ways in which a node could pretend to be sending more or receiving less data than it actually does. It could send different histories to each neighbor, always lying about its interactions with other neighbors. For example, n could send a history to p pretending to send more data to q than it actually did, while it sends a different history to q where it pretends to send more data to p than it actually did. n's goal would be to send less data while not being caught by any of its neighbors. The process of publishing a node's history to a predefined set of neighbors ensures that the node cannot send conflicting histories to different neighbors undetected, therefore avoiding this problem.

A node could also lie about the set of packets sent to or received from a particular neighbor p. In this case, p will be able to identify that the node has lied and will therefore stop exchanging packets with n. Given that an opportunistic node's goal is to maximize its utility, it should have no interest in losing data exchange partners. Therefore, opportunistic nodes have no incentive to publish incorrect histories.

Summary: Local auditing ensures that correct information is available regarding the set of data sent and received by any node, and allows nodes to monitor each other's contribution rates.

4.1.2 Global Auditors

Global auditors are trusted components with global membership knowledge, who interact with one another and with the local auditors. As shown in Figure 7, global auditors execute on nodes external to the system. Their main roles are:

Define the minimum upload threshold: Global auditors periodically sample the state of the system by querying local auditors. They then cooperate to analyze the collected samples, and on this basis compute the minimum upload contribution threshold. Different strategies may be employed for choosing the best possible threshold, given different scenarios. Once thresholds are varied, they are gossiped to all local auditors, who then enforce the determined threshold.

Expurge nodes from the system: Global auditors are also responsible for verifying accusations issued by local auditors against particular nodes, and after validating the accusation, expurging misbehaving nodes from the system. Validation involves verifying that the accused node's history indeed indicates that the

node is sending less data than the current threshold. Expurging a node involves informing the nodes' immediate neighbors of its status and forcing the removal of the node from the overlay mesh.

The number of global auditors may vary according to different parameters, such as the size of the system. The use of more global auditors distributes the load of sampling and improves efficiency in reacting to accusations against nodes. Global auditors are also perfect candidates to perform membership tasks such as acting as entry points to the P2P system, since they are required to have full membership knowledge of the system for performing their auditing roles.

Summary: Global auditing monitors the global health of the system to identify the best value for the minimum upload threshold at any time during a streaming session, and makes final decisions regarding punishment of nodes.

4.2 Adaptive Threshold Strategies

Choosing an upload threshold requires care: a low threshold may not be sufficient to identify opportunistic nodes, while high thresholds may incorrectly punish correct nodes. We considered different strategies for the choice of the minimum contribution t hreshold used for identifying misbehaving nodes.

The simplest strategy sets a fixed threshold (e.g., $t = 0.5$), independent of the current state of the system. In this case, any node contributing at a rate of less than 50% of the stream rate would be removed. One downside of using a fixed threshold is that opportunistic nodes that learn the threshold can simply contribute at the lowest possible upload factor, thus avoiding detection. From the graphs in section 3, it is clear that such a stretagy may disrupt the streaming session. Meanwhile, choosing a high threshold is not a practical option, since correct nodes would get unfairly punished.

To avoid this problem, we have explored adaptive strategies. One simple strategy starts with a minimum threshold (e.g., $t = 0.5$), increasing it only if the system is compromised. Global auditors sample the system to identify the average download factor, and if this factor is lower than 0.98, increase the threshold. Once the download factor reaches a satisfactory level again, the threshold may be reduced back to its initial value. This *stepwise* approach allows the system to catch opportunistic nodes in case their presence starts affecting the performance of the system, while avoiding incorrect accusations of correct nodes.

We also considered a second adaptive strategy (*percentile-based*) for computing the threshold based on periodically sampled download and upload factors. The average download factors once again are used for detecting whether the threshold should be varied or not. In this strategy, our initial threshold is set to null, and the threshold is chosen from sampled upload factors. After each sampling, if the system seems to be in a compromised state, the collected upload factors are ordered and the value dividing the lowest 10 percent is used as the new threshold. This approach relies on efficiently sampling the system, and on fact that if the system's performance is not satisfactory, then at least 10 percent of the nodes are opportunistic.

5. EVALUATION

In this section, we evaluate the performance of our proposed auditing strategy over the original streaming protocol. We built an event-driven simulator and used it to simulate streaming sessions on networks with 1000 nodes and an average of 50ms inter-node latency. The target streaming rate in the experiments was fixed to 500 Kb/second, and all our experiments were repeated 10 times. Confidence intervals were small, and for simplicity are omitted from the graphs.

In all experiments, the source of the stream has an upload capacity of four times the stream rate (2 Mbps) and is connected to 20 arbitrarily selected nodes. Other nodes have enough download capacity to receive the stream, and upload factor of 1.1. We defined an availability window of 10 seconds and an interest window of 8 seconds. To evaluate the quality of each auditing strategy, we evaluate the average download factors of correct nodes during a 100 second time interval after auditing is first applied to the system. For the sample-based techniques, we considered that global auditors collected information from 100 nodes between each interval of 20 seconds. Notice that the sample size does not increase with the size of the system, which is a positive aspect of the auditing approach. In subsection 6.1 we discuss the costs involved in collecting these samples.

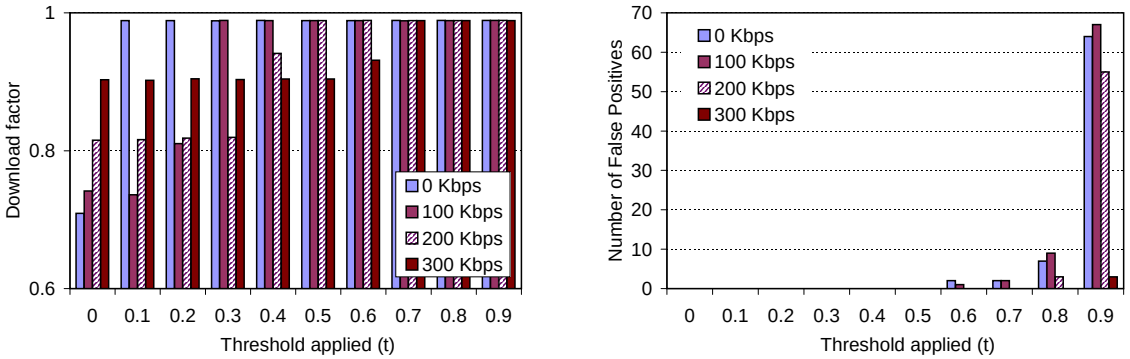

Figure 8. Quality of streaming when applying the fixed threshold strategy. Threshold is varied from 0 to 0.9 (x-axis), and the contribution rate of opportunistic nodes is varied from 0 to 400 Kbps. The first graph (left) presents the average download factors across all correct nodes. The second graph (right) presents the number of correct nodes incorrectly punished (false positives).

In Figure 8, we consider the use of fixed thresholds. We studied the effects of using different values for t, starting from 0 (no auditing) and increasing it until 0.9 (90% of the stream rate), and present a detailed set of results on applying different thresholds to different scenarios. In each scenario, the ratio of opportunistic nodes is fixed to 30%, but their contribution factor (profile) is varied among 0, 100, 200, and 300 Kbps. All other 70% nodes follow the protocol, with a maximum contribution rate set to 550 Kbps (upload factor = 1.1). We present the average download rates (left) and the number of correct nodes mistakenly removed from the system, termed false positives (right), for each of these configurations. The threshold applied is presented on the x-axis. In the left graph, as the threshold increases, higher download averages are observed, since more opportunistic nodes are detected and punished. However, the number of nodes incorrectly accused also increases with higher thresholds, as observed in the right graph.

Scenarios where opportunistic nodes contribute at higher rates (300 Kbps) are less disruptive to the system, but they also require higher thresholds to be applied. Different thresholds yield best results under different scenarios, but overall, from the results presented in Figure 8, we concluded that the best fixed threshold is $t = 0.6$, providing the best compromise in terms of performance and false positives across all scenarios.

In Figure 9, we compare all three strategies proposed in subsection 4.2 against each other and against a configuration with no auditing, under different scenarios. We set $t = 0.6$ for the fixed threshold strategy and as the initial threshold in the *stepwise adaptive* strategy. We summarize the three strategies in Table 1. We simulated sessions where 30% of the nodes were opportunistic and with varying ratios of contribution. In the x-axis, the contribution rate of opportunistic nodes is varied from 0 to 450 Kbps. All other nodes are correct, contributing at a maximum rate of 550 Kbps. We present both the average and the minimum download factors across all correct nodes in the system. As the contribution rate of opportunistic nodes increases, the download factors are expected to increase, which is clear from the curves presented.

Strategy	**Description**
No auditing	Fixed $t = 0.0$
Fixed threshold	Fixed $t = 0.6$
Stepwise adaptive	Minimum $t = 0.6$. If avg sampled download factor < 0.98, increase t by 0.1. Decrease t back to 0.6 when avg download is satisfactory again.
Percentile-based adaptive	Minimum $t = 0.0$. If avg sampled download factor < 0.98, t is chosen based on sampled upload factors ($t >$ lower 10% sampled values).

Table 1. Strategies used for defining the minimum upload threshold t

Figure 9 shows that all strategies yield significantly better results compared to an approach with no auditing. While both adaptive strategies yield excellent download rates to correct nodes, the fixed threshold strategy's performance is not as good when opportunistic nodes are contributing with 300 or slightly more Kbps (near 0.6

contribution factor). At those rates opportunistic nodes are harmful to the system, yet the fixed threshold of 0.6 is not able to detect them.

Finally, in Figure 10, we consider a scenario where opportunistic nodes contribute with different rates. We varied the percentage of opportunistic nodes in the system from 0 to 90%, and evenly assigned them different contribution rates. The graphs present the average and minimum download rates for these scenarios. Once again, no auditing performs significantly worse than any of the proposed strategies. Here, the stepwise adaptive approach yields the best results when large percentages of opportunistic nodes are present in the system. It is also simpler than the percentile-based approach, since it is based only on samples of the download rates of nodes. In both sets of experiments, the number of false positives was practically null under all three strategies considered (at most one in some cases).

6. DISCUSSION

6.1 Auditing Costs

The overheads imposed by auditing are an important consideration, which we address in this subsection. Most of the work of auditing is performed by local auditors, which are executed on the user nodes. The overhead is constant, independent of the size of the system, and is not significant, since nodes only exchange a small amount of accounting data at pre-defined intervals of time (for example, 10 seconds). If we consider a packet rate of 50 packets/s, in 10 seconds the maximum number of packets received and sent by each node is 1000. For each packet sent or received, the history needs to indicate which neighbor sent or received the packet. By using 4 bits to identify each neighbor, the history's size adds up to 4000 bits, or 500 bytes. This is not significant compared to the amount of regular data exchanged in a streaming session.

We also analyzed the costs of the global auditors. Since they are dedicated and external to the system, the overhead imposed by them is of higher concern. Global auditors' main tasks consist of sampling the system to collect download and upload rates of nodes, and of occasionally disseminating updates to the threshold value, through gossip. The sample size remains fixed independent of the size of the population. We ran simulations to estimate the worst-case standard deviation of the download rates across all nodes. Accordingly, we estimate that a sample size of 300 nodes is sufficient to provide 95% confidence, independent of the population size. For smaller systems, such as the ones simulated in this work, even a smaller number of samples was found to be sufficient to yield satisfactory results. Therefore, centralized costs are fixed, and provide a clear advantage for using auditing against tit-for-tat approaches in large-scale systems.

6.2 Heterogenous Systems

So far we considered the use of auditing to enforce node contribution in systems where all nodes are assumed to have homogeneous bandwidth resources, enough to upload and download at a rate close to the stream rate. Pull-based streaming may be extended to heterogenous systems by organizing nodes into multiple groups, according

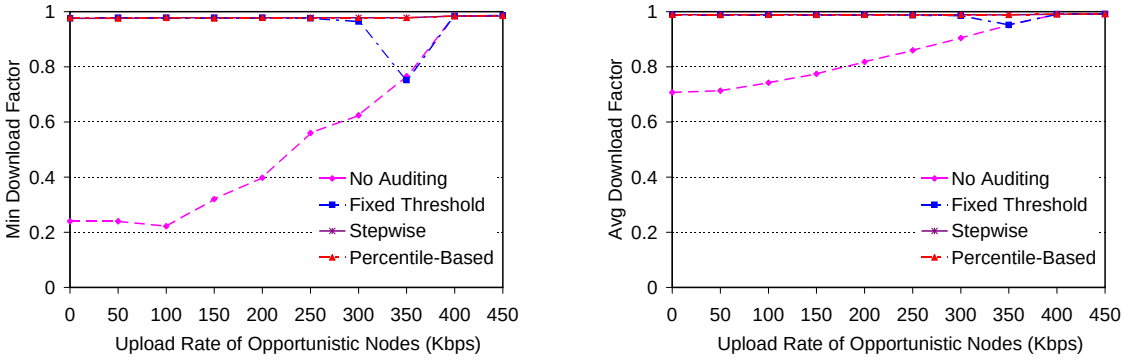

Figure 9. Minimum and average download factors across all correct nodes when using different strategies for choosing the threshold. The upload contribution rate of opportunistic nodes is varied in the x-axis, and the number of opportunistic nodes is fixed at 30%.

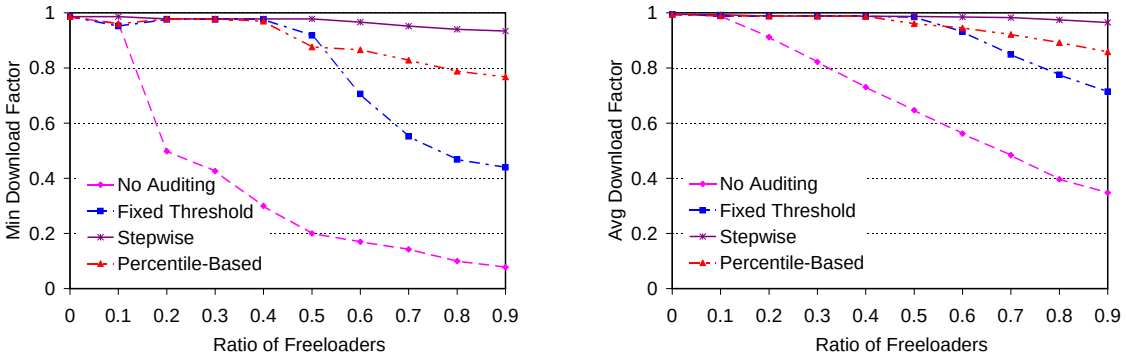

Figure 10. Minimum and average download factors across all correct nodes when using different strategies for choosing the threshold. Each session has mixed set of opportunistic nodes (contributing at different rates) and percentage of opportunistic nodes is varied on the x-axis.

to their upload bandwidths: nodes able to upload at a rate higher than the stream rate are placed in higher-lever groups, which are closer to the source. The source sends data to the highest level group only, who uses the basic protocol to disseminate data among each other. Nodes in lower levels may receive data at smaller rates, after some filtering is applied, and higher-level nodes may be used to act as sources to the lower-level nodes, alleviating the burden at the source.

Auditing can be used to avoid the presence of opportunistic and lower bandwidth nodes in the higher-level groups. It can ensure that the hierarchy of nodes is obeyed by all nodes, while allowing the system to leverage additional resources from privileged altruistic nodes to forward data to lower level groups. We intend to explore this further in future work.

7. RELATED WORK

Several P2P live-streaming protocols have been previously proposed. The first generation of systems (Overcast,[2] Narada[1]) relied on approaches based on pushing data through a single dissemination tree. Later approaches focused on improving fairness among peers and resilience to churn by breaking data into multiple substreams and sending them along disjoing paths (SplitStream,[3] Bullet[10]).

More recent systems like CoolStreaming[5] and Chainsaw[4] use a pull-based style of data dissemination. Coolstreaming breaks the data into packets, and peers organized into a mesh request packets from their neighbors using a scheduling algorithm. As we saw earlier, Chainsaw uses a simpler policy for requesting packets, randomly fetching them while respecting a maximum limit on the number of outstanding requests to each neighbor. Chainsaw presents smaller delays for the receipt of packets compared to the Coolstreaming protocol. In a more recent work,[11] mesh-based approaches are shown to present better performance over tree-based approaches.

Previous papers have considered a variety of possible mechanisms to encourage node contribution. Oversight[12] is a framework proposed to enforce download rate limitations on P2P media streaming systems. The protocol relies on a set of trusted nodes that store information on the data downloaded by each node receiving data. Nodes only send an object after consulting the trusted nodes to verify if the nodes requesting the stream are not overrequesting data. It is targeted to systems where nodes upload full media objects from each other, and not for live-streaming systems where all nodes are interested in receiving the exact same data in close to real time.

Ngan et al.[13] consider fairness issues in the context of tree-based peer-to-peer streaming protocols. The authors present mechanisms that rank peers according to their level of cooperation with the system. One of their techniques involves the reconstruction of trees as a way of punishing opportunistic nodes. Most of their mechanisms require peers to keep track of their parents' and children's behavior.

Pai et al. studied the effect of different types of incentives on the Chainsaw protocol.[14] After exploring tit-for-tat and some variations, the authors propose an algorithm that sets up local markets at every node, where neighbors compete for the node's upload capacity. Nodes favor neighbors who contribute more. Experiments

were limited, with nodes classified as fast or slow nodes. The results indicate that the proposed algorithm improves the performance of the system when the total upload capacity is not enough to supply all the nodes. Pulse[15] is another live-streaming system where nodes choose their neighbors based on their history of interaction. Nodes are placed in the system according to their current trading performances, encouraging nodes to contribute more and therefore be closer to the source.

BAR Gossip[9] is a more recent live-streaming approach that tolerates the existence of opportunistic and malicious nodes. Time is divided into rounds, in which each peer communicates with another peer selected using a pseudo-random function. In each round, peers exchange their current history containing the identifiers of all the current data they hold, as basis for the next exchanges. Nodes also perform a phase of *optimistic push*, forwarding useful updates to pseudo-randomly picked peers with no guarantee of useful return.

8. CONCLUSION

We propose and evaluate a scalable auditing-based technique for enforcing fairness in a live-streaming system. Our approach employs local auditors that execute on all nodes in a streaming session. They are responsible for collecting auditable information about other neighbors' data exchanges, and for verifying that neighbors upload more data than a specified threshold. This threshold is defined by dedicated global auditors, which periodically sample the state of the system to determine if the overall download rate is compromised by the presence of opportunistic nodes. Global auditing determines the minimum threshold for uploads, and works with local auditing to punish nodes that do not upload enough data. We study the efficiency of our auditing approach through simulation, and show that it is able to maintain the throughput of the streaming system even in the presence of a large number of opportunistic nodes.

REFERENCES

1. Y.-H. Chu, S. G. Rao, and H. Zhang, "A Case for End System Multicast," in *Proc. of ACM Sigmetrics*, (Santa Clara, CA), 2000.
2. J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. O. Jr., "Overcast: Reliable Multicasting with an Overlay Network," in *Proc. of the 4th Symposium on Operating Systems Design and Implementation (OSDI)*, (San Diego, CA), 2000.
3. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth Content Distribution in Cooperative Environments," in *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, (Bolton Landing, NY), October 2003.
4. V. Pai, K. Kumar, K. Kamilmani, V. Sambamurthy, and A. E. Mohr, "Chainsaw: Eliminating Trees from Overlay Multicast," in *4th International Workshop on Peer-to-Peer Systems (IPTPS)*, (Ithaca, NY), February 2005.
5. X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming," in *Proc. of the 24th Conference on Computer Communications and Networking (INFOCOM)*, (Miami, FL), 2005.
6. M. Haridasan and R. van Renesse, "Defense Against Intrusion in a Live Streaming Multicast System," in *Proc. of the 6th IEEE International Conference on Peer-to-Peer Computing (P2P)*, (Cambridge, UK), September 2006.
7. N. Magharei and R. Rejaie, "PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming," in *Proc. of the 26th Conference on Computer Communications (INFOCOM)*, (Anchorage, Alaska), April 2007.
8. "PPLive Homepage. Available: http://www.pplive.com."
9. H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin, "BAR Gossip," in *Proc. of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, (Seattle, WA), 2006.
10. D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," in *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, (Bolton Landing, NY), 2003.
11. N. Magharei, R. Rejaie, and Y. Guo, "Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches," in *Proc. of the 26th Conference on Computer Communications (INFOCOM)*, (Anchorage, Alaska), April 2007.

12. W. Conner, K. Nahrstedt, and I. Gupta, "Preventing DoS Attacks in Peer-to-Peer Media Streaming Systems," in *Proc. of the 13th Annual Multimedia Computing and Networking Conference (MMCN)*, (San Jose, CA), 2006.
13. T.-W. Ngan, D. S. Wallach, and P. Druschel, "Incentives-Compatible Peer-to-Peer Multicast," in *2nd Workshop on the Economics of Peer-to-Peer Systems*, (Cambridge, Massachussetts), June 2004.
14. V. Pai and A. E. Mohr, "Improving Robustness of Peer-to-Peer Streaming with Incentives," in *Proc. of the 1st Workshop on the Economics of Networked Systems (NetEcon)*, (Ann Arbor, MI), 2006.
15. F. Pianese, J. Keller, and E. W. Biersack, "PULSE, a Flexible P2P Live Streaming System," in *Proc. of the Ninth IEEE Global Internet Workshop*, (Barcelona, Spain), 2006.