Defense against Intrusion in a Live Streaming Multicast System*

Maya Haridasan
Department of Computer Science
Cornell University, Ithaca, NY
maya@cs.cornell.edu

Robbert van Renesse
Department of Computer Science
Cornell University, Ithaca, NY
rvr@cs.cornell.edu

Abstract

Application-level multicast systems are vulnerable to attacks that impede nodes from receiving desired data. Live streaming protocols are especially susceptible to packet loss induced by malicious behavior. We describe SecureStream, an application-level live streaming system built using a pull-based architecture that results in improved tolerance of malicious behavior. SecureStream is implemented as a layer running over Fireflies, an intrusion-tolerant membership protocol. Our paper describes the SecureStream system and offers simulation and experimental results confirming its resilience to attack.

1. Introduction

Scalable content distribution is a broadly useful tool and has recently received considerable attention. Several application-level multicast (ALM) protocols have been proposed in support of content distribution, and previous work has shown that ALM can achieve efficiencies comparable to IP multicast [3, 5, 2, 6, 17, 9, 14].

One style of content distribution maps the problem to file sharing, whereby peers download data and may assist one-another in obtaining copies. Our work focuses on a second approach, in which data must be delivered with minimal latency, ideally through live streaming. We explore this problem in a peer-to-peer context, because such systems allow data to be streamed without requiring the source to have a high upload capacity. Such systems have become quite popular, and in places like China, are now widely used for broadcasting television channels to increasing numbers of users [17].

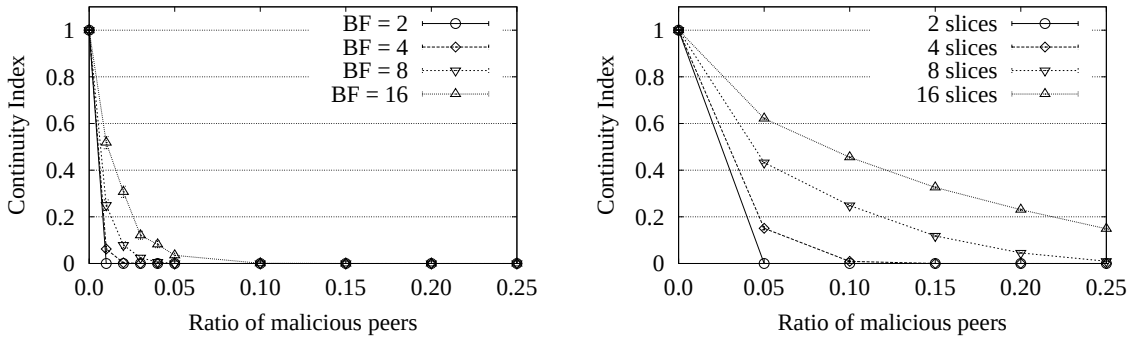
The main difficulties with live streaming emerge from its sensitivity to delay. Data must be received within a fixed delay after the original source produced the data, whereas the kinds of reliability and ordering properties afforded by protocols such as TCP are not required. The simultaneous emphasis on a non-traditional property and weakening of more familiar ones adds up to a challenging new problem. Significant progress has been made on solving the mixture of delivery requirements, but to date, there has been little attention to guaranteeing the desired behavior in the presence of attack.

Malicious behavior has long been a serious concern in systems deployed over the Internet. Centralized systems are particularly because they have easily targeted points of failure. In contrast, peer-to-peer systems offer potentially improved tolerance to attacks due to their redundant nature: multiple peers collaborate to carry out important tasks, offering some options if individual peers come under attack. Nonetheless, few systems take full advantage of these possibilities. Most existing peer-to-peer systems are partially centralized or employ a non-uniform overlay structure, and are therefore still vulnerable to attacks. Moreover, attackers may gain control over internal nodes in the system enabling omission attacks in which compromised peers malfunction, for example by failing to forward packets to other peers. This type of attack is hard to repel, since attackers cannot easily be identified, and the impact on overall performance can be severe. This is the problem tackled in our work.

To illustrate the issue, we looked into the effects of malicious attacks when using a single dissemination tree with varying branching factors and when using the popular SplitStream system [2] with varying numbers of slices. As a measure of resilience, we compute the *continuity index* of a streaming session, which is the ratio of packets received by a peer within acceptable time. Through simulation we computed the minimum continuity index across participants for sessions with a thousand homogeneous nodes and varying ratios of malicious peers not forwarding packets. In Figure 1, each point presents the median with 95 percentile intervals across one thousand runs. In the case of single trees, malicious behavior can prevent healthy nodes from receiving data even with as few as 5% malicious members. SplitStream has better resilience, but attacks are still very visible to participating nodes.

Our work introduces several techniques that reduce the opportunity for an attacker to compromise the quality of a streaming session, without incurring a high computational or network overhead. To repel forgery attacks, we employ

*Our effort is supported by the NSF CyberTrust program, the NSF TRUST STC, the AFRL, and the AFOSR MURI program.



(a) Single Tree (b) SplitStream

Figure 1. Expected continuity index across all correct members under omission attacks.

an efficient packet authentication technique based on computing and distributing verification digests. To prevent attacks on the overlay structure (the membership protocol on top of which multicast systems operate), SecureStream is built upon *Fireflies*, a scalable one-hop Byzantine membership protocol [13].

Fireflies is a probabilistic protocol, in which members are presented with a reasonably current view of which members are live or not. To achieve tolerance to denial-of-service attacks, SecureStream uses a pull-based packet dissemination approach, similar to the one used by the CoolStreaming [17] and Chainsaw protocols [9]. This approach is attractive because it offers participants a choice among multiple candidate packet sources. Because participants are not dependent on any particular peer and can immediately react to failures or attacks, attacks are less damaging.

Our paper makes the following contributions. First, to our knowledge, ours is the first exploration of end-system attacks in the context of live streaming peer-to-peer protocols. Second, we leverage previous work and present a comparison of different authentication protocols for signing and verifying packets efficiently in the context of application-level multicast. Finally, we thoroughly evaluate the effects of internal malicious peers on pull-based protocols. The issue is more serious than has previously been recognized.

2 System Model

The system consists of one source, assumed non-compromised, disseminating data at a fixed rate to a set of receivers with limited buffering capacity. The desired behavior is that the streamed data be received within a fixed latency relative to the source's original transmission.

Multicast systems exhibit a wide range of security properties: secrecy, authenticity, data integrity, anonymity, non-repudiation, access control and service availability. Not all applications require secrecy and anonymity of data, hence we do not provide these properties. On the other hand, we believe authenticity, non-repudiation and access-control are essential. We assume that the original data is non-compromised, and therefore implicitly achieve data integrity through authenticity. Our primary focus is on guaranteed availability, namely mechanisms that prevent nodes from being isolated or severely harmed during a streaming session. We expect the system to repel external attacks and tolerate a limited fraction of internal Byzantine nodes, and to degrade gracefully as the fraction of Byzantine nodes increases.

SecureStream is an application-level streaming system, and only attacks to the end system hosts are addressed in this work. Attacks on the underlying network infrastructure and low-level *denial-of-service* attacks are thus beyond the scope of this work.

3 Malicious behavior model

Malicious attacks on a streaming multicast system come in many flavors. Attacks may originate outside the system or be internal, and attackers may compromise nodes and then work in cooperation with these faulty internal peers.

The simplest form of internal attacks are those in which a single node is compromised. The extent of harm that results depends on many factors, such as the multicast protocol being used and the location of the malicious node in the overlay. These effects can be localized and minimized if the protocol in use has no single points of failure. On the other hand, vulnerable systems like those based on a single dissemination tree can be crippled if a node high in the tree is compromised.

Collusion attacks pose much more complex problems; in these, an attacker compromises a set of nodes and exploits them to perform a coordinated attack to the system, and may orchestrate the attack to confound whatever defensive mechanisms are built into the dissemination infrastructure.

For the work presented here, we make several assumptions about compromised members. They do not have sufficient computational power to break cryptographic building blocks, and cannot forge public key certificates or signatures of correct or stopped members. We also assume a bounded probability that a live member is Byzantine, which probabilistically limits the fraction of total nodes that may

collude in an attack. A classification of the types of attacks that we address in our system is presented below.

Membership attacks: The system may be attacked by compromising the underlying overlay or membership protocol on which it runs. For example, systems that run on top of ring-based overlays are vulnerable to eclipse attacks [11], in which an attacker controls a large fraction of the neighbors of correct nodes, preventing correct overlay operation. Malicious nodes may also mimic flaky but correct members, or accuse other correct members of being down.

Forgery: In this category we include all attacks that involve fabrication and tampering of data being streamed in the system. Given time, these attacks can be easily avoided by use of a public key infrastructure. However, in the context of streaming the cost of signatures can become prohibitively high, forcing us to consider other kinds of data authentication protocols.

Denial-of-service (DoS) Attacks: Attacks in which malicious nodes overload peers with requests for packets or large amounts of duplicate packets, or other attacks that might compromise their ability to contribute to the streaming session.

Omission Attacks: Given our emphasis on low-latency data delivery, send-omission is an especially serious type of attack. By not forwarding all or part of the packets, a malicious node may disrupt overall system's availability.

4 Overview of the System

Our system employs a set of techniques to achieve resilience to the attacks previously mentioned. We use an intrusion-tolerant membership protocol to tolerate attacks to the membership layer. We also employ an efficient technique for avoiding forgery of packets by malicious peers. Finally, by using a pull-based streaming protocol and imposing a structure to define what peers are allowed to communicate with one-another, we can avoid high-level DoS attacks and tolerate omission attacks. In this section we describe these main components in further detail.

4.1 *Fireflies* Membership Protocol

Peers in SecureStream use the membership knowledge provided by Fireflies to track the status of other peers. *Fireflies* is composed of three subprotocols: a pinging protocol is used to detect failures of nodes with an accuracy independent of message loss; an intrusion-tolerant gossip protocol is used for dissemination of information between correct members with probabilistic time bound Δ; and a membership protocol uses accusations and rebuttals to implement the membership information that *Fireflies* provides. These components are briefly described below.

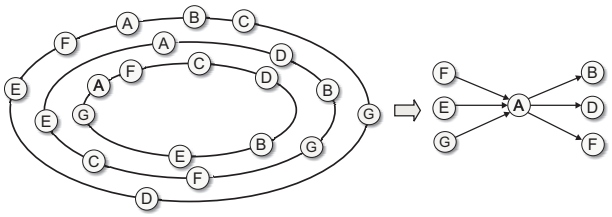


Figure 2. In Fireflies multiple rings are used to define which peers monitor each other: A monitors B, D and F, and is monitored by E, F and G.

Members are organized into rings, and their position on each ring depends on their identifier. These rings determine which nodes monitor, and are allowed to accuse, which other nodes (Figure 2). On each ring, each member m_i monitors the lowest ranked successor m_j that it believes to be live, and if it detects a failed node, it issues an $accusation$ for that node.

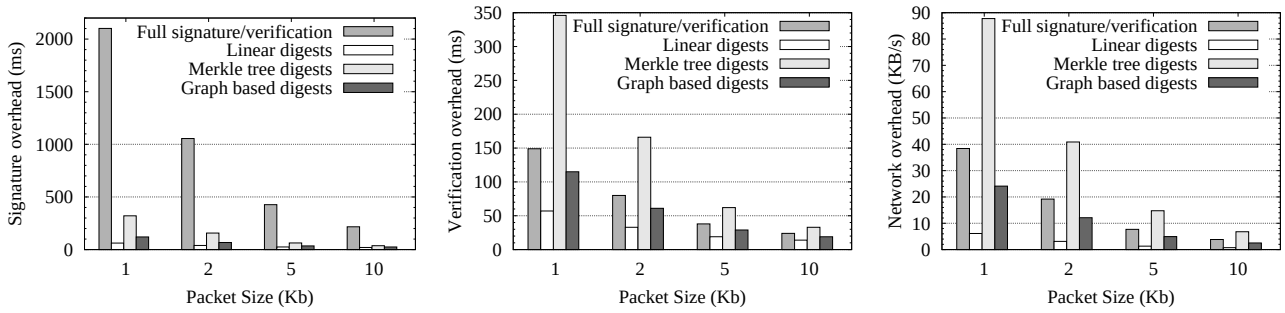
When an accusation for a member m_i is received by a member m_j, m_j waits a time period of length 2Δ, and then removes m_i from its view if the accusation is valid. This time period is established so that an accused member may issue a new $note$ (a $rebuttal$) to an $accusation$ against itself. In order to avoid malicious nodes from abusively accusing its correct neighbors in the rings, nodes may invalidate up to t rings, implying that accusations issued by its neighbors on those rings will not be accepted as valid by any correct member. All notes and accusations are signed, and a certification authority is responsible for issuing private/public key pairs and public key certificates.

The dissemination of information such as accusations and rebuttals is performed using a robust gossip protocol. Each member periodically picks a random member from its view to exchange state information. The multiple ring structure induces a gossip mesh resilient to malicious attacks.

4.2 Efficient Packet Verification

The standard point-to-point approach of appending a message authentication code (MAC) computed using a shared key does not map to multicast sessions, while signing every packet using an asymmetric cryptographic protocol induces high overhead. SecureStream employs an efficient authentication scheme to avoid forgery of data.

To avoid signing and verifying every packet, we group the hashes of n packets into a special message, and have it signed by the source (we call this approach *linear digests*). The signed message needs to be sent to the receivers prior to the dissemination of data that it corresponds to. This implies a buffering of content on the source prior to the dissemination of data. The advantage is that this approach incurs the minimal network overhead of one hash per packet, while amortizing the cost of a single signature/verification operation over n packets. Subsection 4.3 describes how



(a) Signature overhead (b) Verification overhead (c) Network overhead

Figure 3. Overheads per second when transmitting 300 Kb/s using varying packet sizes.

we guarantee that the digest is received prior to the data to which it corresponds.

Other approaches have been proposed to address the high costs of authenticating packets in a flow [16, 4, 7, 12, 10]. Wong and Lam[16] propose that the source compute the hashes of a limited number n of consecutive packets in the stream, and use them as leaves in a Merkle Tree where each internal node consists of the hash of its children. Each packet is verifiable upon receipt, since it is appended with the signed root node and the hashes of all needed interior nodes in the path from the root to itself in the Merkle Tree.

In graph-based authentication [7, 12], the source only signs one packet, and the following packets in the stream are linked to it through hash chains that allow them to be verifiable. To tolerate packet loss, a graph is used instead of a single chain. Packets are represented by vertices in the graph, and a directed edge between nodes that represent packets P_i and P_j indicates that packet P_j contains the hash of packet P_i. A packet corresponding to a node can be authenticated if there is a path of already verified packets between the node and the source node of the graph.

The computational costs at the source and at the receiver and network overheads for different authentication approaches when streaming 300 Kb/s are presented in Figure 3. We compared 4 techniques: signing and verifying every packet, linear digests, Merkle tree digests and a simple scheme of graph-based authentication. The code used for the evaluation was written in Python and executed on a Linux-based Pentium III 850 Mhz with 256 Mb RAM. Linear digests yield the lowest computational costs. Although the Merkle Tree approach is appealing due to its immediate verifiability, its network overhead is the highest, since one signature and a few hashes need to be appended to each packet in the flow.

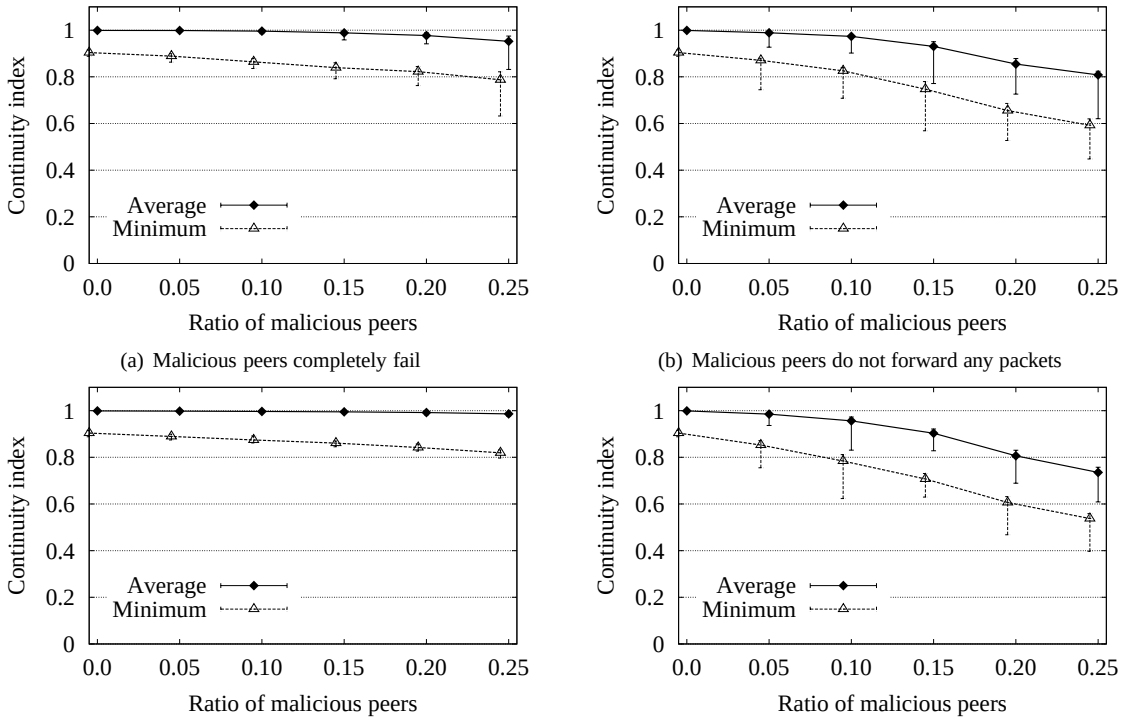
When the packet sizes are large, which reduces the rate of packets per second, the computational costs of the three latter techniques are not significantly different. We therefore used linear digests since it minimizes network overhead and is the simplest technique. Our experience indicated that when using pull based streaming, keeping the rate of packets per second larger or equal to 30 yields good results, and reducing it further affects the quality of the streaming.

4.3 Pull-Based Content Streaming

We employ a pull-based approach to disseminate packets, following ideas used in the Chainsaw protocol [9]. The same rings used in *Fireflies* are used to determine a fixed set of neighbors with which each peer can exchange packets. This imposed mesh structure and the use of authenticated channels between neighbors allows the system to avoid high-level DoS attacks. Initially, the source sends notifications to its neighbors as soon as it has available packets to disseminate. These notifications are small messages used only to inform neighbors of availability of packets. Each neighbor requests missing packets according to some pre-specified policy, to avoid overloading the source. As peers receive packets, they propagate notifications to their neighbors, and so packets get disseminated along the mesh. This pull-based approach to acquisition of packets yields a highly resilient multicast, since failure or misbehavior of one neighbor does not impede a peer from fetching packets from other neighbors. The predetermined set of neighbors for each peer also makes it hard for malicious peers to attack individual peers, since attackers lack a deterministic means of acquiring control of all of its neighbors.

Each member stores packets and forwards them to other peers while the packet is within its *availability window*. It also maintains an *interest window*, smaller than the availability window, which represents the set of packets in which the peer is currently interested. Different policies can be employed by peers about what packets to pick from each of its neighbors, and the choice of the appropriate policy is crucial to achieving best overall performance. Random selection of neighbors is usually a good candidate, leading to fair load balancing.

There is a predefined limit l on the number of outstanding requests to any neighbor. This policy not only improves the flow of packets in the abscence of malicious behavior, but also makes it harder for malicious peers to overrequest packets from their neighbors. Peers maintain a queue of non-satisfied requests for packets, and if more than l requests by the same neighbor are present in the queue at any time, only the l most recent ones are maintained.



(a) Malicious peers completely fail

(b) Malicious peers do not forward any packets

(c) Malicious peers overrequest packets

(d) Malicious peers overrequest and do not forward packets

Figure 4. Resilience under different types of Byzantine behavior and varying ratios of attackers

The protocol is simple and yet highly resilient to failures and attacks. The overhead incurred by notifications is not significant if large packets are used, and the protocol avoids receipt of duplicate packets. Since it is completely decentralized, the protocol does not present any single points of failure, another important consideration when building an intrusion-tolerant streaming protocol.

We make use of the request packets to request the authentication digest packets presented in Section 4.2. The receipt of digest data permits the verification of those packets to which the digest refers. Whenever a peer requests a packet, it checks whether it has already received the corresponding digest, and if not, it sets up a flag in the request message. This flag indicates to the selected data source that it should append the digest to the packet prior to sending the data, guaranteeing that a node can always verify the integrity of a packet at the time of its receipt.

5 Evaluation

We evaluated the performance and resilience of pull-based streaming both through simulation and experiments on the Emulab testbed [15]. We built an event-driven simulator and simulated 200 node networks with 50ms internode latency. It would be possible to simulate and present results for networks with larger numbers of nodes, but a set of experiments on increasing numbers of nodes revealed that the behavior remains the same for networks as large as 5000 nodes. We opted for a smaller size but repeated each experiment 100 times to obtain better confidence in our results.

The target streaming rate in the experiments was fixed to 300 Kb/s, and packets of 10 Kb were used. Higher streaming rates yield similar results as long as the packet size is accordingly increased to maintain a rate of 30 packets/s. Each streaming session lasted for 200 seconds. In the basic setting, the seed's upload capacity was fixed to twice the streaming rate while other peers had a fixed maximum upload capacity of 1.2 times the streaming rate. These values are used as our baseline since they are the lowest upload rates at the seed and non-seed nodes respectively that lead to good throughput when the system is not under attack.

For each streaming session we computed the average and minimum download and upload rates across all correct members. We repeated each experiment 100 times, and we present the median and 95 percentile intervals across these repetitions.

We considered four types of malicious behavior. In the first type of attack malicious peers act as failed, neither requesting nor satisfying requests. In attack 2 they request packets but do not forward any packets. In attack 3 they overrequest packets from their neighbors, requesting as many distinct packets as possible from every neighbor. Finally, in attack 4 they overrequest packets and do not forward packets. The fourth type of attack is the most disruptive type and therefore the most likely, while the other three are considered mainly for comparison purposes.

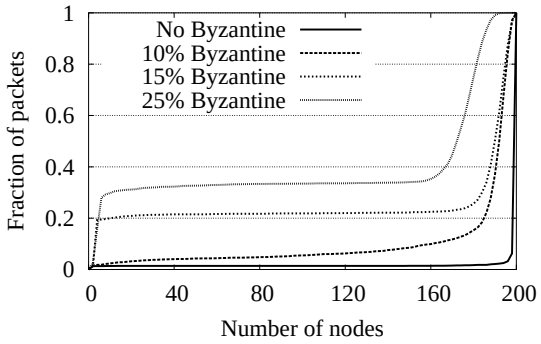


Figure 5. CDF: Fraction of packets received by given number of nodes

Figure 4 presents results for the basic setting under each of the attack types. We are interested in minimizing the overall damage to the streaming session. Damage is quantified by the impact on average download rates to healthy nodes, and the minimum download rate for any single healthy node.

As would be expected, the results show that peer failure does not significantly affect the download rates since peers can still request packets from other correct neighbors (Figure 4(a)). Since malicious peers do not request packets in this mode, they do not disrupt the total overall upload capacity. Even though upload rates are limited, overrequesting attacks are also not significantly disruptive, due to the random policy used by peers when satisfying neighbors' requests for packets and the upper limit on the number of outstanding requests by any neighbor (Figure 4(c)).

Figures 4(b) and 4(d) show that attacks in which peers consume packets from their neighbors, but do not forward packets, inflict the most harm. There are two main reasons for this vulnerability. First, since peers upload at a maximum rate of 1.2 times the streaming rate, the overall upload capacity of the system gets compromised from peers consuming and not contributing to the system. Second, malicious nodes neighboring the seed might impede some packets from ever being received by any other peer other than itself.

The latter effect causes the 95 percentile interval bars to be wide: there is a lot of variation depending on the number of compromised peers near the seed. To make this point clear, in Figure 5 we show the percentage of packets received by increasing numbers of peers during sample streaming sessions with varying ratios of Byzantine peers. The metric to focus on here is the fraction of packets only received by one peer, which is an indicator of malicious nodes neighboring the seed. Packets received only by malicious peers at the first hop will never be disseminated in the system. To confirm this hypothesis, we executed the same set of experiments and restricted the malicious attackers to being located at least 2 hops away from the seed. The obtained medians were very close to the medians obtained in the previous experiments. The main difference was that the percentile intervals were significantly reduced when the seed had no immediate malicious neighbor, which is an important result since the intervals are significant in the original experiments with attacks 3 and 4.

To improve the resilience, we can vary parameters to improve the overall upload capacity of the system, or to avoid situations in which malicious peers can isolate certain packets. First, we considered the upload capacity of the members. In Figure 6 we varied the value from 1.0 to 2.0 times the streaming rate and verified the improvements to resilience under attack 4. This graph presents the average and minimum download rates when the system has 25% of Byzantine members. The results show that the higher the upload capacity at non-seed peers the more resilient the system becomes. From Figure 7, which presents the minimum, average and maximum upload rates of members, we can see that as a consequence of increasing the upload capacity of peers the system becomes more unfair, with an increased difference between the maximum upload rate and minimum upload rate across peers. For the next few experiments we fixed the upload capacity of non-seed members to 1.4 times the streaming rate.

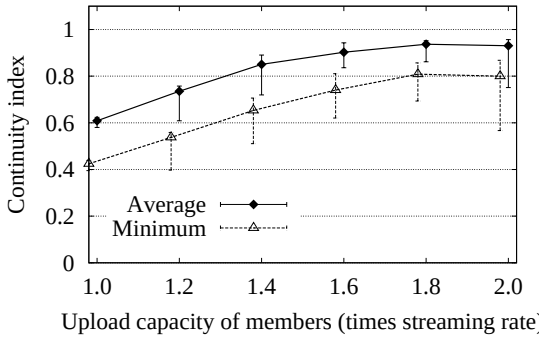


Figure 6. Continuity index across nodes when maximum upload capacity is varied

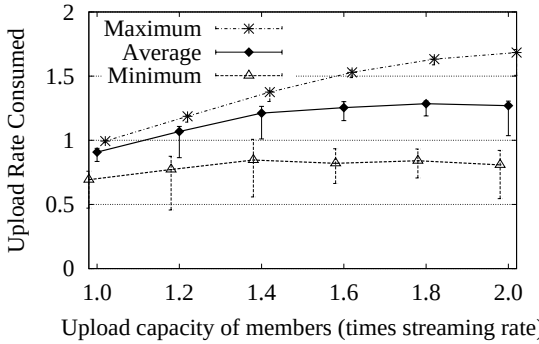


Figure 7. Upload rate across nodes when maximum upload capacity is varied

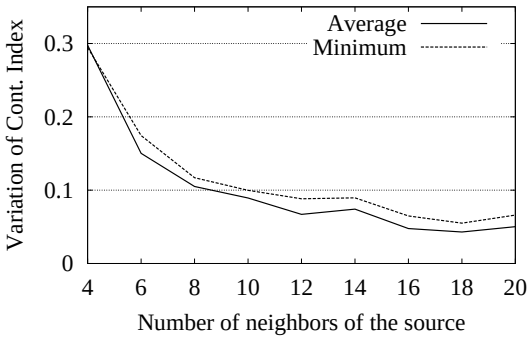


Figure 8. Sensitivity to source's neighbors

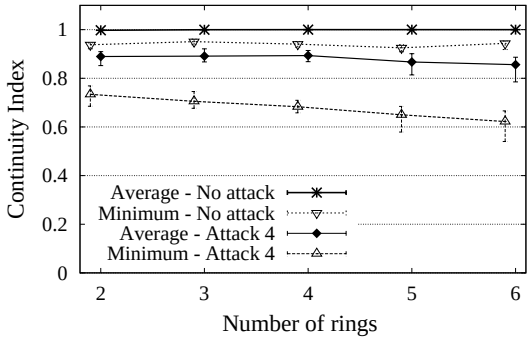


Figure 9. Sensitivity to number of neighbors

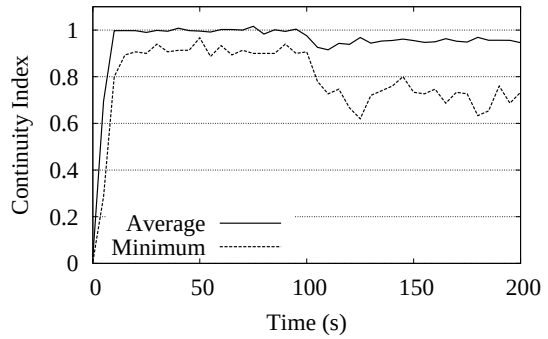


Figure 10. Sample session on Emulab testbed

To improve the packet loss ratio at the first hop from the seed, we varied the upload capacity of the seed from 1.0 all the way to 6.0 times the streaming rate. Our results indicated that this naive approach to increasing the upload rate at the seed does not significantly affect the resilience of the system. We also observed that the number of neighbors of the seed is a more significant parameter than the upload capacity of the seed. We fixed the ratio of malicious nodes at 25%, the upload rate at non-seed nodes to 1.4 times the streaming rate and at the seed to 4.0 times the streaming rate, and varied the seed's number of neighbors from 4 to 20. The median slightly improves as the number of neighbors is increased, but more important, the percentile intervals are significantly reduced. In Figure 8 we present the absolute sizes of the 95 percentile intervals varying with the number of neighbors of the seed. The results show that a larger number of neighbors at the seed is desirable. This happens because with a higher number of neighbors the percentage of malicious neighbors of the seed tends to be closer to 25% across runs, and therefore there is less variation in the ratio of packets that are contained at the first hop from the seed.

Finally, to study the influence of the number of neighbors for each non-seed peer in the system, we evaluated the resilience with a varying number of rings used to define neighbors. The upload capacities at the seed and non-seed members were fixed to 4.0 and 1.4 times the streaming rate, respectively, and the seed had 16 neighbors. In Figure 9 we present the performance of the system using between 4 and 12 neighbors per node, both under no attacks and under attacks of type 4. The results surprisingly show that the use of larger numbers of neighbors does not improve resilience of the system, and even reduces when the system is under attack. Even though larger numbers of neighbors would lead to better connectivity between correct members, it also presents malicious members with more potential to overrequest packets and unbalance the system.

We also ran experiments on a 200 node LAN on the Emulab testbed to validate our simulation results. Given the large amount of resources required for these experiments, we restricted our experiments to a limited number of repetitions, with the sole purpose of validating the tendencies observed in the simulations. Our experiments indicated the same tradeoffs previously observed through simulation. We therefore present only a sample streaming session in which 25% of the nodes are malicious. During the first 100 seconds all nodes act correctly, after which the malicious nodes start overrequesting and not forwarding packets. We fixed the upload capacity of the seed and non-seed members to 4.0 and 1.4 times the streaming rate respectively, and the number of neighbors of the seed and non-seed members to 12 and 8 respectively. Figure 10 presents the minimum and average download rates throughout the session.

6 Related Work

Recent work on peer-to-peer streaming systems has focused on improving fairness among peers and resilience to churn, and have not addressed behavior in the presence of malicious peers. Splitstream [2] breaks the data into stripes and disseminates each stripe through a different dissemination tree. Ideally, each peer is an internal node in only one these trees, and therefore the system as a whole is fair. Figure 1(b) presents SplitStream's resilience to omission attacks. Bullet [6] is another protocol which attempts to improve fairness by breaking the stream into packets and sending them to peers through different dissemination paths. Packets are pushed down a tree to certain peers and then exchanged between peers through random connections. The pull-based style of streaming used in our system was previously used in CoolStreaming [17] and Chainsaw [9].

Drum[1] targets DoS attacks on gossip-based multicast protocols, eliminating vulnerabilities to such attacks. The main idea in Drum is to have half of the links of each peer be picked by the peer itself, and half be picked by other peers. That way, even if only malicious peers connect to a peer, the peer can still get correct data from the peers that it picks. The authors showed that the approach works well for multicast protocols which do not have time delays, but have not studied its performance for multicast systems where a high throughput of packets is desired and the upload capacities are limited.

There has been significant work regarding authentication of packets for multicast [16, 4, 7, 12, 10]. Most of this work was originally developed for the domain of IP multicast but can be directly transferred into the domain of application-level multicast. We presented the most relevant ones in Section 4.2 and compared their performances in the context of peer-to-peer streaming.

Omission attacks are often characterized as rational behavior and there has been a lot of work regarding incentives for peer-to-peer systems. Most work on incentives has focused in file sharing systems, which present significantly different properties, and cannot be directly transferred to streaming protocols. Ngan et al. [8] consider fairness issues in the context of tree-based peer-to-peer streaming protocols. The authors present mechanisms that can distinguish peers according to their level of cooperation to the system. One of their techniques involves the reconstruction of trees as a way of punishing freeloading nodes. Most of their mechanisms require peers to keep track of their parents' and children's behavior. Only rational behavior is considered, and the exploration of freeloading as a malicious attack is not addressed.

7. Conclusions

We have presented the design and evaluation of SecureStream, an intrusion-tolerant peer-to-peer live streaming system. We described and evaluated a set of techniques that allows SecureStream to resist against forgery, DoS, membership and omission attacks. We showed through simulation and emulation the effect of pull-based streaming on resilience to attack, and we explored how the variation of specific parameters affect the resilience of the system. Our results indicate that SecureStream tolerates a limited percentage of malicious nodes, gracefully degrading in the presence of increasing ratios of attackers.

References

[1] G. Badishi, I. Keidar, and A. Sasson. Exposing and Eliminating Vulnerabilities to Denial of Service Attacks in Secure Gossip-Based Multicast. In *International Conference on Dependable Systems and Networks*, Philadelphia, PA, 2004. IEEE Computer Society.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, 2003.

[3] Y.-H. Chu, S. G. Rao, and H. Zhang. A Case for End System Multicast. In *Proceedings of ACM Sigmetrics*, Santa Clara, CA, 2000.

[4] R. Gennaro and P. Rohatgi. How to sign digital streams. In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, London, UK, 1997.

[5] 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 Fourth Symposium on Operating Systems Design and Implementation"*, San Diego, CA, 2000.

[6] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, 2003.

[7] S. Miner and J. Staddon. Graph-Based Authentication of Digital Streams. In *SP '01: Proceedings of the 2001 IEEE Symposium on Security and Privacy*, page 232, Washington, DC, 2001.

[8] T.-W. J. Ngan, D. S. Wallach, and P. Druschel. Incentives-Compatible Peer-to-Peer Multicast. In *Second Workshop on the Economics of Peer-to-Peer Computing*, 2004.

[9] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating Trees from Overlay Multicast. In *Proceedings of the 4th International Workshop on Peer-to-Peer Systems*, Ithaca, NY, 2005.

[10] A. Perrig, R. Canetti, D. Tygar, and D. Song. The TESLA Broadcast Authentication Protocol. *Cryptobytes*, 5(2):2–13, 2002.

[11] A. Singh, M. Castro, A. Rowstron, and P. Druschel. Defending against Eclipse Attacks on Overlay Networks. In *Proceedings of the 11th ACM SIGOPS European Workshop*, Leuven, Belgium, 2004.

[12] D. Song, J. D. Tygar, and D. Zuckerman. Expander Graphs for Digital Stream Authentication and Robust Overlay Networks. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, Washington, DC, 2002.

[13] R. van Renesse, H. Johansen, and A. Allavena. Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays. In *Proceedings of the First ACM EuroSys*, Leuven, Belgium, 2006.

[14] V. Venkataraman, P. Francis, and J. Calandrino. Chunkyspread: Multitree Unstructured Peer to Peer Multicast. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems*, Santa Barbara, CA, 2006.

[15] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, Boston, MA, 2002.

[16] Wong and Lam. Digital Signatures for Flows and Multicasts. *IEEETNWKG: IEEE/ACM Transactions on Networking IEEE Communications Society*, 7, 1999.

[17] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. CoolStreaming/DONet: A Data-Driven Overlay Network for Efficient Live Media Streaming. In *Proceedings of the 2005 Conference on Computer Communications*, Miami, FL, 2005.