The Pollution Attack in P2P Live Video Streaming: Measurement Results and Defenses

Prithula Dhungel, Xiaojun Hei, Keith W. Ross, and Nitesh Saxena
Department of Computer and Information Science
Polytechnic University, Brooklyn, NY, USA 11201
pdhung01@utopia.poly.edu, {heixj,ross,nsaxena}@poly.edu

ABSTRACT

P2P mesh-pull live video streaming applications – such as CoolStreaming, PPLive, and PPStream – have become popular in the recent years. In this paper, we examine the stream pollution attack, for which the attacker mixes polluted chunks into the P2P distribution, degrading the quality of the rendered media at the receivers. Polluted chunks received by an unsuspecting peer not only effect that single peer, but since the peer also forwards chunks to other peers, and those peers in turn forward chunks to more peers, the polluted content can potentially spread through much of the P2P network. The contribution of this paper is twofold. First, by way of experimenting and measuring a popular P2P live video streaming system, we show that the pollution attack can be devastating. Second, we evaluate the applicability of four possible defenses to the pollution attack: blacklisting, traffic encryption, hash verification, and chunk signing. Among these, we conclude that the chunk signing solutions are most suitable.

1. INTRODUCTION

P2P live video streaming leverages the upload bandwidth capacity of peers for the distribution of video/audio content. Unlike traditional client-server based systems, peers forward content to other peers in the network. Various techniques have been used for implementing P2P live video streaming systems. The most popular P2P live video streaming applications today, such as PPLive [1] and PPStream, use the data-driven mesh-pull technique [6]. The idea is similar to that used in BitTorrent file sharing systems. Each video stream is divided, at the source of the stream, into chunks. A peer makes partnerships with a subset of other peers in the network watching the same video stream. Each participating peer periodically sends to its neighbors "buffer maps," which indicate which chunks it has available for sharing. In order to watch a particular stream, a peer actively requests chunks from its partners based on the buffer maps of the partners. Meanwhile, it also forwards requested chunks to its neighbors.

The distributed P2P architecture of such systems makes them prone to various security threats. One potentially devastating threat is stream pollution. In this attack, the attacker mixes into the stream bogus chunks, which degrade the quality of the rendered media at the receivers.

A similar type of attack has already been deployed on a large scale in P2P file sharing systems [9, 11]. In file sharing, the attacker corrupts the targeted content (for example, with white noise or with warnings about copyright violations), rendering the content unusable, and then makes this polluted content available for sharing from one or more peers. Unable to distinguish polluted files from unpolluted files, unsuspecting users download the polluted files into their own file sharing folders, from which other users may then download the polluted files. In this manner, polluted files spread through the file sharing system.

In a P2P live video streaming system also, a polluter can introduce corrupted chunks. Specifically, an attacker can join an ongoing video channel and establish partnerships with other peers watching the channel. The attacker can then advertise to its partners that it has a large number of chunks for the ongoing video stream. When the neighbors request advertised chunks, the attacker sends bogus polluted chunks instead of legitimate chunks. Each receiver integrates into its playback stream the polluted chunks it receives from the attacker along with other chunks it receives from its other neighbors. The polluted chunks degrade the quality of the rendered video at the receiver.

Importantly, polluted chunks received by an unsuspecting peer not only effect that single peer, but since the peer also forwards chunks to other peers, and those peers in turn forward chunks to more peers, and so on, the polluted content can potentially spread through much of the P2P network. If the amount of polluted data is significant, users might eventually get frustrated and entirely stop using the system.

Polluters are expected to have different motivations, depending on the video content. If a content source distributes non-authorized copyrighted content, the owner of the copyrighted content may hire a "pollution company" to pollute the ongoing video stream, similar to what has been observed in file sharing. If two channels are competing with each other, one channel may attempt to pollute the stream of the other channel. If an individual disagrees with a channel's political message, that individual may be motivated to pollute the channel's video stream. And there can always be amateur hackers who attempt to disrupt channels just for fun. For P2P live video streaming, we anticipate a variety of motivations that go well beyond copyright issues.

The paper [4] provides a scheme to prevent jamming attacks caused due to the introduction of corrupted blocks in architectures that use network coding. There has also been some recent work on security aspects of streaming systems [3, 5, 14], however, none on pollution attacks in P2P live video streaming, to the best of our knowledge. The contribution of this paper is twofold. First, by way

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
P2P-TV'07, August 31, 2007, Kyoto, Japan.
Copyright 2007 ACM 978-1-59593-789-6/07/0008 ...$5.00.

of experimenting and measuring a popular P2P live video streaming system, we show that the pollution attack can be devastating. In our experiment, before launching the attack in Brooklyn, a particular channel had about 3300+ viewers before the attack; during the attack the number of viewers dropped to about 500 quickly, indicating that video quality became unacceptable for a large majority of peers. We also observed that for a peer in Hong Kong, located geographically far from the attacking peer, a large fraction of its downloaded and uploaded chunks were polluted. The second contribution of this paper is a survey of four different classes of defenses to the pollution attack: blacklisting, traffic encryption, hash verification, and chunk signing. Among these, we conclude that chunk signing should be an effective defense.

2. POLLUTION EXPERIMENT

In this section, we present the results of a pollution experiment that we conducted on a popular P2P live video streaming system called PPLive. We demonstrate the feasibility of launching a pollution attack and analyze the severity of the attack. For the experiment, we instrumented our own customized PPLive client that aggressively advertises video chunks, and in response to requests for advertised chunks, sends polluted chunks. We demonstrate that even a single malicious peer, equipped with a high bandwidth network access, is able to inject a large number of polluted chunks. The experimental results also verify that PPLive peers naively forward polluted chunks to one or more other peers. If these peers also have high bandwidth network access, the pollution in a streaming network propagates to a high level quickly. To the best of our knowledge, none of the P2P live video streaming systems available in the market employs any kind of defense against such an attack. Therefore, our pollution results for PPLive can most likely be duplicated for other live streaming systems (such as PPStream and CoolStreaming).

2.1 Experiment Setup

Figure 1 depicts our pollution experiment. For this experiment, we selected a popular channel with a chunk size of 7220 bytes and a playback bit rate 342 kbps as the target channel. We monitored two normal PPLive peers, capturing their incoming and outgoing traffic. These two peers, labelled as "Brooklyn peer," and "Hong Kong peer" are equipped with Ethernet network access. The Brooklyn peer is located in the Ethernet domain at Polytechnic University, New York. The Hong Kong peer is located in Hong Kong. The instrumented polluter is located in the same Ethernet domain as the Brooklyn peer. This polluter implements the PPLive protocol for joining a channel and exchanging buffer maps and video chunks with one target peer. While doing the pollution experiments, we also ran our PPLive crawler, which tracks the number of peers active in the channel [6].

2.2 Experiment Results

In our experiments, the polluter only sends polluted chunks to the Brooklyn peer. First, the polluter establishes peer partnership with the Brooklyn peer; it then advertises that it has a large number of video chunks for the channel. As a result, the Brooklyn peer starts to request chunks from the polluter. Since the polluter has a high upload rate, the Brooklyn peer finds that it can download video chunks from the polluter with a very high network throughput. We found that after an initial transient stage, the Brooklyn peer downloads almost all video chunks from the polluter. It also uploads these polluted chunks to other peers in the network.

The pollution propagation amplified the pollution level significantly in the network and severely impacted the service of the

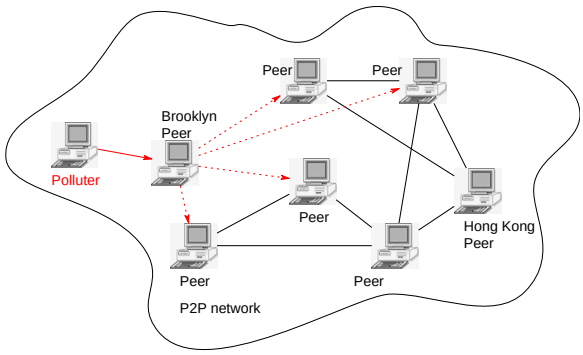


Figure 1: PPLive pollution experiment setup

channel. This can be observed from the sharp decrement in the number of peers for the channel after the polluter started at time $t = 34$ minutes, as shown in Figure 2. Figure 2 also shows that for the same channel, when the system was pollution free on some other day, during the same time period of the day, the peer number remained quite steady.

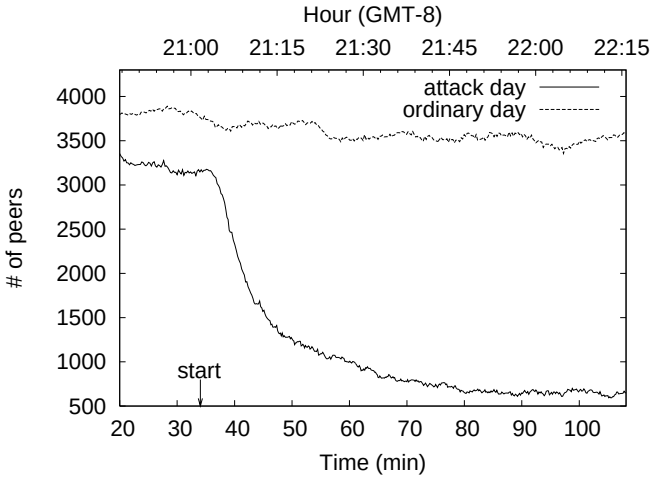
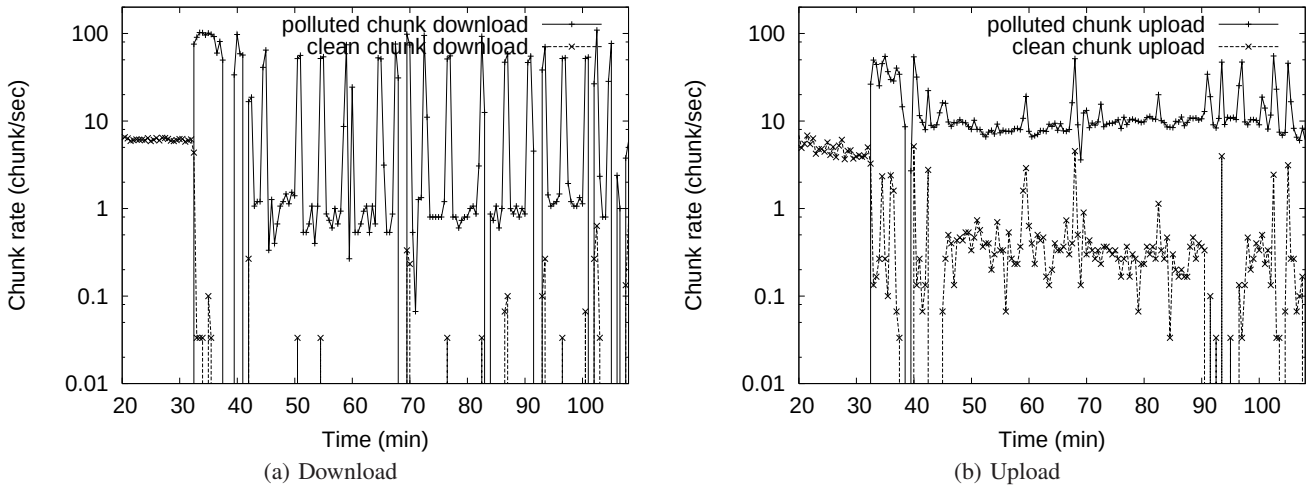


Figure 2: Number of peers viewing channel over experiment periods

Each polluted chunk that the polluter uploads to the Brooklyn peer has the same binary content that we prepared before the experiment. Therefore, using byte-by-byte comparison with the originally prepared binary content, we can distinguish polluted chunks from other clean chunks in the traces for both Brooklyn and Hong Kong peers. In the remaining part of this section, we describe results in terms of the numbers of polluted chunks observed.

In Figure 3, we plot the chunk download rate and upload rate of the Brooklyn peer, which is the initial pollution target. These video chunks are divided into polluted chunks and normal chunks. Before launching the pollution attack at $t = 34$ minutes, the chunk download rate is 5.92 chunk/sec, matching the video playback bit rate $5.92 \times 7220 \times 8 = 342$ kbps. In addition, all the downloaded chunks are clean chunks. After the pollution is launched, the Brooklyn peer receives most of the video chunks from the polluter. Only sporadically, the Brooklyn peer downloads clean chunks from other peers. It also starts to upload these polluted chunks to multiple other PPLive peers.

The Hong Kong peer is far away from and is not in the peer lists



(a) Download (b) Upload

Figure 3: Clean and polluted chunks to/from Brooklyn peer

of the polluter and the Brooklyn peer; however, we find that the polluted chunks propagate quickly and impact it significantly. In Figure 4, we plot the chunk download rate and upload rate of the Hong Kong peer. Similar to the Brooklyn peer, it sustains a steady download bit rate before pollution starts; however, after the pollution starts at around $t = 34$ minutes, it starts to download a significant amount of polluted chunks. Unlike the Brooklyn peer, which receives polluted chunks from the polluter directly, the Hong Kong peer still manages to receive observable portion of clean chunks. It also uploads polluted chunks to other peers, acting as a pollution redistributor.

To examine the pollution redistribution done by a node with high-bandwidth network access, we plot the number of partners with which the Hong Kong peer exchanges video chunks, over each 30-second period. Figure 5(a) depicts the number of neighbors that provided at least one polluted chunk, and the number that provided at least one clean chunk. As shown in Figure 5(a), before the pollution attack, the Hong Kong peer downloads clean video chunks from around 30 peers. After the pollution attack starts at $t = 34$ minutes, it is affected quickly in that it downloads polluted chunks from around 20 peers. Nevertheless, it still downloads some clean chunks since it is only polluted indirectly by the polluter. This high polluted-peer/clean-peer ratio indicates that the pollution level of the system has reached at a significant level. The pollution amplification is more clearly demonstrated in the upload of the polluted chunks by the Hong Kong peer. As shown in Figure 5(b), after the attack is launched, it uploads polluted chunks to around 30 peers; however, it only uploads clear chunks to less than 10 peers. In summary, the Hong Kong peer (which is a high bandwidth peer that is not being manipulated by us) provides more damage than contribution to the streaming system.

3. POLLUTION DEFENSE MECHANISMS

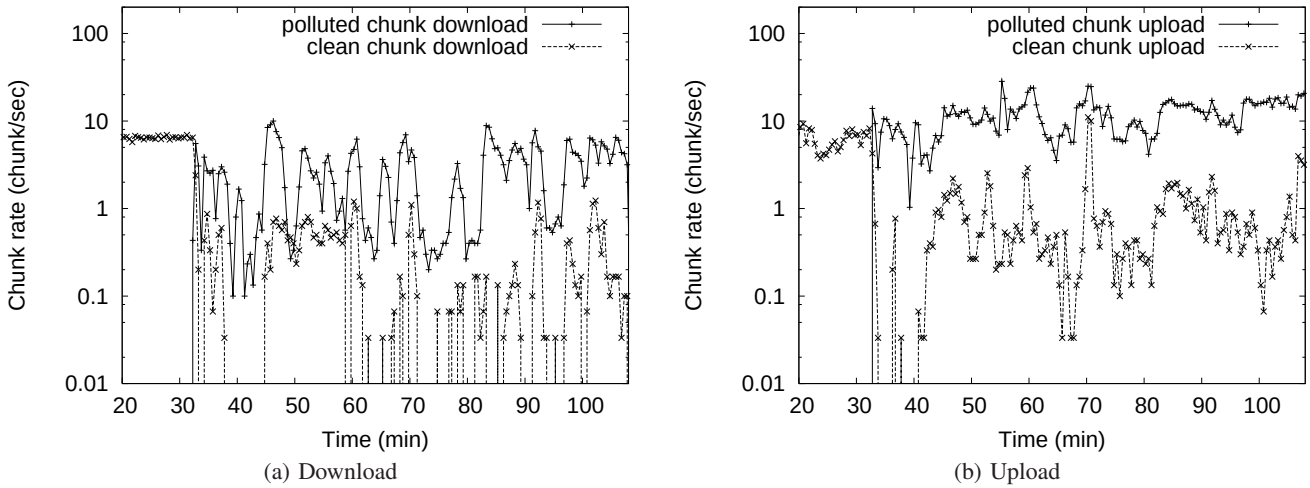
3.1 Blacklisting

In the blacklisting approach, we attempt to determine - in a centralized or decentralized manner - the peers that originate and relay pollution. All such peers are put onto a blacklist. Peers neither send chunks to nor receive chunks from peers on the blacklist.

This general approach was taken in [10] for P2P file sharing systems. For P2P file sharing, it was observed that the attackers often advertise inordinate numbers of files. Peers can thus individually count advertisements from IP prefixes and assign reputations to the prefixes accordingly. As part of a distributed reputation system [7], the peers then share the reputations with each other and update the reputations of the IP prefixes. Finally, each peer creates a blacklist based on the reputations it has of the other peers. In a similar manner, for P2P streaming, we can blacklist peers that advertise an unusually large number of chunks, as these peers are apparently trying to attract downloaders. However, an attacker could easily circumvent such a defense by being less aggressive in its chunk advertisements. Furthermore, peers that relay pollution are not likely to exhibit unusual advertising behavior.

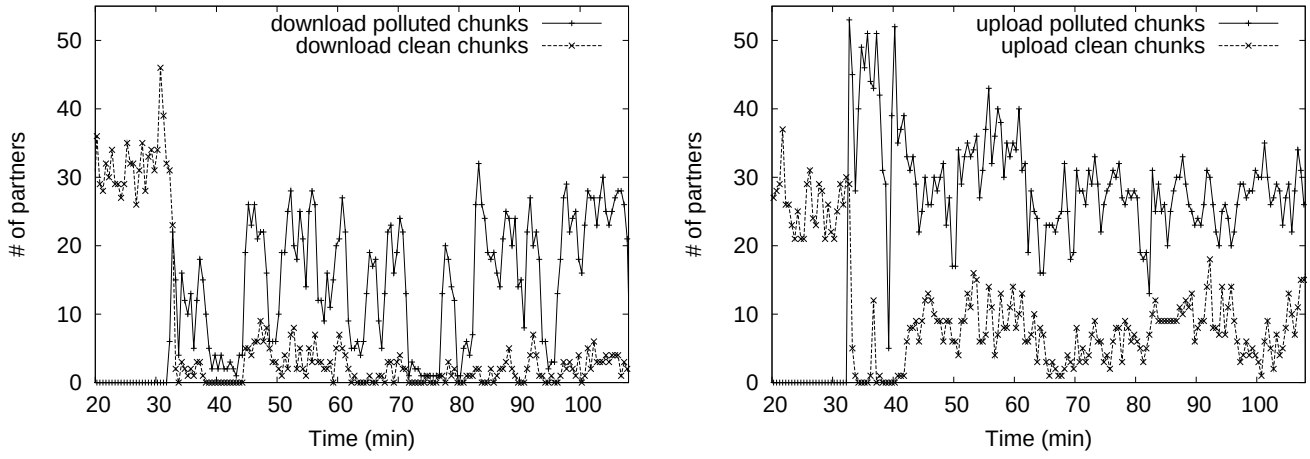
An alternative approach is for each peer to attempt to determine whether a chunk is polluted. If a chunk is determined polluted, then the peer that sent the chunk can be assigned a low reputation value. Again, the reputations can be shared and a distributed blacklist can be created. The critical step in this approach is accurately determining whether a chunk is polluted or not. In P2P live video streaming, a receiver typically obtains chunks from more than one peer. Therefore, by comparing characteristics of the received chunks, one might be able to distinguish between the fake and the legitimate copies. Video and audio processing techniques can possibly be used to detect polluted chunks. However, an attacker should be able to circumvent such an approach by creating chunks that resemble neighboring chunks (in the stream of chunks) but nevertheless significantly diminish the quality of the rendered video. For example, the attacker could insert duplicate chunks into the stream.

For file sharing systems, a large fraction of pollution can be reduced if users are careful enough not to forward polluted content into the network. The same is true for P2P live streaming. This requires the user to observe and manually indicate to the P2P streaming client the presence of pollution (if any) in the stream being played. All nodes that are sending data to the client at this time could then be put on the list of candidate polluters/relayers. These lists can form the basis of a distributed reputation system, from which blacklists can be created. This approach requires active involvement from all users. Moreover, for any P2P live distribution, some users may be absent from their stations even though their sta-



(a) Download (b) Upload

Figure 4: Clean and polluted chunks to/from Hong Kong peer



(a) Download (b) Upload

Figure 5: Numbers of polluted and clean partners of Hong Kong peer

tions are actively participating in the distribution. In summary, it is unlikely that any of these reputation/blacklisting approaches will be able to consistently stop the pollution attack.

3.2 Traffic Encryption

One reason the current P2P video streaming systems are prone to pollution attack is that all of these systems have their (control as well as data) messages transmitted in clear text. To inject pollution into a stream, the attacker needs to send the correct messages to the other peers with the correct header and data format. This requires the attacker to first sniff some traffic specific to the streaming system and analyze the traffic to understand the protocol sequence and message formats. If all the messages a system uses were encrypted, it would be difficult for the attacker to determine the message structure in distributed application. This would prevent the attacker from inserting crafted messages into system, such as message containing polluted data. This idea is not completely new. Skype is an example of a widely deployed P2P system that uses encryption techniques to obfuscate its application specific traffic. Of course, this idea works only if the system under consideration is not open source.

To achieve traffic encryption, any pair of communicating peers need to establish a shared key with each other. Public-key based key exchange protocols, such as Diffie-Hellman, can be used for this purpose. However, in a *dynamic* P2P live video streaming environment, where a peer is typically connected with a number of other peers, such a continuous key generation might not be feasible, especially on devices such as PDAs.

The main disadvantage of using traffic encryption as a means to preventing pollution, however, is that it works well to protect the privacy of the application protocol and message formats until the system is subjected to a reverse engineering of the source. For example, although the Kazaa/FastTrack protocol was proprietary and encrypted, it was nevertheless reverse engineered (see [8]). If the reverse engineering process is thorough enough, considerable fraction of the system protocol and messages can be revealed, thus facilitating the polluter to inject pollution into one or more streams.

3.3 Hash Verification

In BitTorrent, before a peer begins to download a file, it obtains a torrent file which provides the hashes of all the chunks of the file. When a peer receives chunks from other peers, it compares the hashes of the chunks received with the corresponding hashes in the torrent file to verify their integrity.

We now consider applying the same general technique for P2P live video streaming. The simplest approach for this would be for each receiver to get the hash of each chunk form the source itself. As in BitTorrent, this would allow each peer to verify the integrity of each chunk before forwarding it to other peers. However, the load on the source will be very high for a large number of receivers. The load on the source can be reduced by distributing the hashes of the chunks through the P2P system itself. But this allows an attacker to easily replace an original chunk from the source with a fake chunk and replace the corresponding valid hash with a hash for the fake chunk. When an unsuspecting peer receives the fake chunk, it verifies the fake chunk with the fake hash, thus being fooled into believing the integrity of the chunk. In summary, hash version, as done in BitTorrent, is not a viable solution for P2P live streaming.

3.4 Chunk Signing

In this section, we survey three techniques involving chunk signing and evaluate their applicability for detecting pollution in P2P live video streaming systems, based on computational, bandwidth, and delay overhead (note that delay is an important factor in the context of a live video). Table 1 summarizes the overheads for each of the techniques. In each technique, the so-called "authentication information" needs to be transmitted to the receivers along with the chunks. This authentication information can either be provided by the source (in which case the load on the source might be high) or could be distributed through the P2P system itself, in the form of a separate stream or be piggybacked onto chunks.

3.4.1 "Sign-All" Approach

In the "Sign-All" approach, each chunk is individually signed by the source, the signature (which is the authentication information) is appended with the chunk and delivered to the receivers. The receiver receives each chunk and its corresponding signature one by one, verifies its integrity and plays back (and forwards) only if the chunk is valid, otherwise rejects the chunk as being polluted.

This approach is fast in terms of playback, as it has a delay corresponding to the processing of only 1 packet at the source and the receiving and processing of only 1 packet at the receiver . However, it incurs high computation overhead. For a stream consisting of m chunks, the source needs to compute and the receiver needs to verify m signatures.

For channels with high bit rates, the number of chunks generated per second can be very high. This means that the number of times per second the signature and verification operations to be performed can be equally high, leading to high computational requirements at the source and the receivers. Thus, we conclude that the "Sign-All" approach is computationally very expensive, especially for devices such as PDAs and smart phones.

3.4.2 Signature Amortization Approaches

For reducing the computational overhead incurred in "Sign-All" scheme above, the "Signature Amortization" approaches of [15], originally designed for IP multicast, can be used. In these approaches, the chunks are divided into blocks such that only one signature operation per block is required. However, each chunk can be individually verified. This is achieved, however, at the cost of a slightly higher bandwidth overhead than the "Sign-All" approach. Two different approaches that provide signature amortization that have been discussed in [15] can be used. We summarize these next.

Star Chaining. In this approach, the source computes the hash of the concatenation of the hashes of all chunks in the block, and signs it to produce the block signature. The authentication information consists of the block signature, chunk position in the block, and the hashes of all other chunks in the block. On receipt of a chunk in the block and the corresponding authentication information, the receiver first creates the hash of the concatenation of the hashes of all the chunks and verifies the signature against this hash.

For a block of n chunks, the source needs to perform $n + 1$ hash operations and one signature operation. The receiver has to perform a total of $n + 1$ hash operations and 1 signature verification. The scheme incurs a delay overhead equivalent to the processing of n chunks at the source and receiving and processing of 1 chunk at the receivers. The total bandwidth overhead is equivalent to the size of $n(n - 1)$ hashes and n signatures.

Consider an example of a channel with a stream generation bit rate of 372 kbps at the source. If the chunk size is 4000 bytes, the number of chunks generated per second at the source is approximately 12. Using the star chaining approach, by grouping 32 chunks in a block, the source needs to perform 33 hash operations and only one block signature operation, about every 3 seconds. Referring to the results of [2], on PDAs of moderate capabilities, for a message of 2KB, each hash operation takes a fraction of a millisecond, and a signature operation takes about 80 milliseconds. Hence, the time taken for the source to perform the hashing and signing operations for a single block is less than 100 milliseconds. This indicates that the star chaining approach is computationally feasible even for devices with lower computational capabilities working as video sources. Furthermore, since signature verification is much faster than the generation, the computational overhead at the receiver is even lower. The total bandwidth overhead is equivalent to around 20KB (around 16%), when using 128-bit MD5 hashing and 1024-bit RSA signing.

Merkle-Tree Chaining. This approach based on Merkle-Tree [13] requires building an authentication tree at the source for each block. The leaf nodes correspond to the hashes of the chunks in the block. Other nodes are constructed as hashes of their children. The signature on the root node becomes the block signature. The authentication information for each chunk is the block signature, chunk position in the block, and the siblings of each node on the path from the leaf node corresponding to the chunk to the root in the authentication tree. On receiving a chunk in the block and the corresponding authentication information, the receiver first creates the hash of the root node and then verifies the block signature against this hash.

For a block of n chunks, the source needs to perform $2n - 1$ hash operations and one signature operation. The receiver, on the other hand, has to perform $2n - 1$ hash operations and 1 signature verification to authenticate a block. As in the star approach, the delay overhead to authenticate a block in this scheme is equivalent to the processing of n chunks at the source and receiving and processing of 1 chunk at the receiver. The total bandwidth overhead, on the other hand, is equivalent to the size of $n \log_2 n$ hashes and n signatures.

Using the example described above, for a block of 32 chunks, the source and receiver need to perform 63 hash operations each and 1 signature and 1 verification operation, respectively, about every 3 seconds. This is almost twice the cost incurred in the star chaining approach, however, is still computationally feasible for devices like

Table 1: Computational overhead, bandwidth overhead, and delay for various chunk signing approaches for a block containing n chunks. $|h|$ is size of hash output (bytes), $|s|$ is size of signature (bytes)

Approach	Computational Overhead		Bandwidth Overhead	Delay	
	Source	Receiver		Source	Receiver
Sign All	n signatures	n verifications	$n\|s\|$	1	1
Star Chaining	$(n+1)$ hashes & 1 signature	$(n+1)$ hashes & 1 verification	$n(\|h\|(n-1)+\|s\|)$	n	1
Merkle Tree Chaining	$(2n-1)$ hashes & 1 signature	$(2n-1)$ hashes & 1 verification	$n(\|h\|log_2 n+\|s\|)$	n	1
Sign and Correct	n hashes & 1 signature & 1 RS encoding	βn hashes & constant number of verifications & 1 RS decoding	$n\|h\|/\rho$	n	βn

PDAs. The bandwidth overhead is around 5% when using 128-bit MD5 hashing and 1024-bit RSA signing. This implies that the Merkle-Tree chaining approach is much more efficient than the star chaining approach in terms of bandwidth.

3.4.3 "Sign-and-Correct" Approach

We now summarize a solution [12], which we call "Sign-and-Correct" approach. Refer to [12] for details. The source first hashes each chunk of the given block separately and signs the concatenation of all these hashes. The hashes and the signature together (which is the authentication information) are then error corrected using the Reed-Solomon (RS) error correcting code with the rate $\rho = \frac{\alpha^2}{(1+\epsilon)\beta}$, where α denotes the survival rate (which means if a block of n chunks is sent, the receiver obtains at least αn valid chunks); β denotes the flood rate (which means if a block of n chunks is sent, an attacker can not flood more than βn chunks); ϵ denotes the tolerance of the RS decoder. The source then sends out each chunk c_i along with the error corrected information s_i. On receipt of m chunks ($\alpha n \leq m \leq \beta n$), the receiver can reconstruct the authentication information using only αn valid chunks. Using the authentication information, the validity of all chunks in the block can be determined.

For a block consisting of n chunks, the source needs to perform n hashes and 1 signature operation (in addition to RS encoding), and the receiver needs to perform a constant number of signature verification operations and n hashes in the best case and βn hashes in the worst case (in addition to RS decoding). Since the RS encoding and decoding involve expensive multiplication and addition operations over large fields, the computation overhead in this approach is higher than that in Merkle-Tree chaining.

The total bandwidth overhead is approximately equivalent to the size of n/ρ hashes. This implies the bandwidth overhead for this approach is less than that for Merkle-Tree chaining when $n(h \log_2 n + s) > nh/\rho$ (where h is hash size and s is signature length). For the MD5-RSA combination, it is when $\log_2 n + 8 > 1/\rho$. Clearly, for large values of n, the overhead for Sign-and-Correct approach is much lower than the overhead incurred in Merkle-Tree chaining. Using the same example as before, for a block of 32 chunks and for $\alpha = 0.5, \beta = 1.5$ and $\epsilon = 0.1$, the bandwidth overhead comes out to be around 3380 bytes (around 3%).

This solution incurs a delay corresponding to the processing of n chunks at the source and receiving and processing of αn chunks, in the best case, and βn in the worst case, at the receiver. In comparison to Merkle-Tree chaining, the delay at the receiver here is slightly higher, however, it can be acceptable based on the type of application.

4. CONCLUSION

In this paper, we considered the pollution attack for P2P live video streaming systems. The contributions made by the paper are twofold. First, we showed that this attack is potentially devastating for the streaming network provided that the attacker has access to a high bandwidth connection. Second, we evaluated the applicability of four different classes of solutions against these attacks, namely, blacklisting, traffic encryption, hash verification, and chunk signing. Among these, we conclude that the chunk signing solutions – Merkle-Tree chaining and Sign-and-Correct – are most suitable. Of the two, the former is more efficient in terms of computational overhead and the delay at the receiver, whereas the latter is more efficient in terms of bandwidth. Based on the type of applications, and computational and bandwidth requirements therein, either of the solutions could be used. In our future work, we would like to implement these solutions and evaluate their effectiveness towards controlling pollution in P2P live video streaming.

5. REFERENCES

[1] PPLive. `http://www.pplive.com`.

[2] P. G. Argyroudis, R. Verma, H. Tewari, and D. O'Mahony. Performance analysis of cryptographic protocols on handheld devices. In *NCA'04*, 2004.

[3] W. Conner, K. Nahrstedt, and I. Gupta. Preventing DoS attacks in peer-to-peer media streaming systems. In *MMCN*, 2006.

[4] C. Gkantsidis and P. Rodriguez. Cooperative security for network coding file distribution. In *IEEE INFOCOM*, 2006.

[5] M. Haridasan and R. V. Renesse. Defense against intrusion in a live streaming multicast system. In *P2P'06*, 2006.

[6] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. A measurement study of a large-scale P2P IPTV system. *IEEE Transactions on Multimedia*, Oct. 2007. to appear.

[7] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *WWW'03*, 2003.

[8] J. Liang, R. Kumar, and K. W. Ross. The FastTrack overlay: A measurement study. *Computer Networks*, 2006.

[9] J. Liang, R. Kumar, Y. Xi, and K. W. Ross. Pollution in P2P file sharing systems. In *IEEE INFOCOM*, 2005.

[10] J. Liang, N. Naoumov, and K. W. Ross. Efficient blacklisting and pollution-level estimation in P2P file-sharing systems. In *AINTEC*, 2005.

[11] J. Liang, N. Naoumov, and K. W. Ross. The index poisoning attack in P2P file-sharing systems. In *IEEE INFOCOM*, 2006.

[12] A. Lysyanskaya, R. Tamassia, and N. Triandopoulos. Multicast authentication in fully adversarial networks. In *IEEE Symposium on Security and Privacy*, 2004.

[13] R. C. Merkle. A digital signature based on a conventional encryption function. In *Crypto'87*, 1987.

[14] W. Wang, Y. Xiong, Q. Zhang, and S. Jamin. Ripple-Stream: Safeguarding P2P streaming against DoS attacks. In *ICME*, 2006.

[15] C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. *IEEE/ACM Trans. Netw.*, 1999.