POLLUTION-RESISTANT PEER-TO-PEER LIVE STREAMING USING TRUST MANAGEMENT

Bo Hu and H. Vicky Zhao

ECE Dept., University of Alberta, Edmonton, AB T6G 2V4 Canada

ABSTRACT

In the emerging peer-to-peer (P2P) live streaming, users cooperate with each other to support efficient delivery of video over networks in live streaming applications. Pollution attack is an effective attack against P2P live streaming, where attackers upload bogus multimedia data to their peers. The polluted data can spread over the entire network, and cause severe quality degradation of the videos. To resist pollution attacks in P2P live streaming, this paper proposes a trust management system that identifies attackers and excludes them from further sharing of multimedia data. We investigate possible attacks against the trust management system and analyze the attack resistance of the proposed system. Our simulation results show that the proposed trust management system can efficiently detect attackers and stimulate user cooperation even under attacks. It helps users receive more clean data and improves the performance of P2P live streaming.

Index Terms— Peer-to-peer live streaming, trust management, pollution attacks

1. INTRODUCTION

We witness the emergence of large-scale multimedia social networks in the past decade, where millions of users share and exchange digital media content. Peer-to-peer (P2P) live streaming is one of the popular multimedia social networks on the Internet, and we have seen many successful deployments, for example, PPLive, Coolstreaming, etc. In P2P live streaming, users watch live TV programs over networks simultaneously [1, 2]. P2P live streaming systems rely on voluntary contributions of resources from individual users to achieve high system scalability and robustness. Cooperation also enables users to access extra resources from others and thus benefits each individual user.

The distributed nature of P2P live streaming systems makes them vulnerable to attacks, for example, the pollution attack where attackers (polluters) upload bogus data to their peers. Unable to differentiate between the polluted and the clean data, users may also unintentionally forward polluted data to other users. A recent study showed that without proper defense mechanisms, polluted data may spread over the entire network, degrade the quality of the rendered videos, and significantly damage the system [3]. Also, such pollution attacks may cause distrust among users and prohibit user cooperation. To provide reliable service and further proliferate P2P live streaming, it is of ample importance to design pollution-resistant P2P live streaming and to ensure the quality of service.

Pollution resistance in P2P live streaming is challenging due to the stringent delay constraint, that is, a clean packet must be received before its playback time to improve the quality of the rendered video. Furthermore, attackers may use the hand wash attack, where they temporarily leave the system after being detected and come back later with new user IDs [4]. To resist pollution attacks, in [3], a chunk signing algorithm was proposed, where digital signatures of the video were transmitted together with the compressed bit stream to help verify the authenticity of the video. The work in [5] proposed a theoretical framework to analyze the number of infected users. In [6], blacklisting and reputation system were used to detect attackers, while their work did not consider the hand wash attack.

This paper proposes a trust management system for P2P live streaming, which can identify polluters and exclude them from further uploading and downloading of data. To resist the On-Off attack against trust management, we use the adaptive time window-based algorithm to periodically update the trust values [7]. To resist the flooding attack with hand wash, we address the challenging stringent delay constraint in P2P live streaming and the powerful hand wash attack, and propose an early decoding method to detect the polluted data chunks as early as possible. Our results show that this early detection of polluted chunks can help speed up the identification of polluters, significantly reduce the trust management system's false alarm rate, stimulate user cooperation even under attacks, and improve the quality of the videos rendered at the users' side.

The rest of the paper is organized as follows. Section 2 introduces the mesh-pull P2P live streaming systems and the pollution attacks. Section 3 introduces the trust management system for P2P live streaming. In Section 4, we analyze the performance of the trust management system under the On-Off attack and the flooding attack with hand wash. Conclusions are drawn in Section 5.

2. MESH-PULL P2P LIVE STREAMING

2.1. Mesh-pull P2P Live Streaming

In mesh-pull P2P live streaming systems, multimedia content is divided into small data chunks of M bits per chunk, all of which are available at the streaming server [1]. Each user maintains a buffer to store received data chunks that have not been decoded. Each user keeps a buffer map to record the indices of the received chunks, and users periodically exchange buffer map information with each other.

In P2P live streaming, time is divided into rounds of equal length (τ seconds per round). At the beginning of each round, user i selects one missing chunk k, and in our work, data chunks with tighter playback deadlines are given higher priorities. Then, based on the exchanged buffer map information, user i selects either the streaming server or another user j whose buffer has chunk k, and sends a request. The streaming server answers all data chunk requests in a round robin fashion. When user j receives a request from user i, he/she can either reject the request or upload chunk k to user i. Following the work in [2], we assume that each user can accept at most one request per round, and chunks that do not arrive at the receivers within one round will be dropped. When user i first joins

The authors can be reached at bhu2@ualberta.ca and vzhao@ece.ualberta.ca.

the network, he/she buffers enough contiguous data chunks before launching the video player and rendering the video. Then, user i periodically moves the first B chunks in the buffer with the earliest playback time to the video player for decoding and playing. The above process is repeated for all rounds by all users. In this paper, the credit line mechanism in [2] is adopted to avoid free riding and to stimulate user cooperation in P2P live streaming.

2.2. The Pollution Attack

In the pollution attack, an attacker (polluter) enters P2P live streaming with a valid user ID, establishes partnership with other users and *intentionally* uploads bogus chunks to his/her peers [3]. Such polluted data chunks render the content useless and degrade the quality of the video. Note that unsuspicious users may *unintentionally* forward these polluted chunks to others. Thus, the polluted content can quickly spread over the entire P2P network and cause serious damages to the system [3]. Polluters aim to send as many polluted chunks as possible and prevent other users from getting clean chunks.

To generate a polluted data chunk, a simple solution is to randomly generate M bits. Such randomly generated polluted chunks can be easily detected by a quick syntax check and verification of its format compliance with the video compression standard. A polluter can also generate a format-compliant polluted data chunk with incorrect or meaningless content. Such a polluted chunk can pass the syntax verification and can only be detected when it is fully decoded (after IDCT and motion compensation). In this paper, we assume that a randomly generated polluted chunk can be detected immediately after its arrival, and we focus on the more challenging format-compliant pollution attack.

This paper considers a simple scenario where polluters attack the system individually and independently. We plan to investigate in the future the more challenging scenario where a group of attackers collude together to attack the system more effectively.

3. BUILDING TRUST IN P2P LIVE STREAMING

In our trust management system, users assign trust values to each other. $T_{i(j)} \in [0,1]$ measures the trust that user i has on user j to upload a clean chunk, and a higher value of $T_{i(j)}$ indicates that user i has a higher confidence that user j will upload clean chunks. Note that a polluted chunk can only be detected when it is decoded. Thus, in our trust management system, for every received data chunk, each user keeps a record of its source. When user i moves the first B chunks from his/her buffer to the video player for decoding, user i detects polluted chunks in the newly decoded B chunks, and updates his/her trust values $T_{i(j)}$.

3.1. Trust Definition

Following the work in [6], $T_{i(j)}$ contains two components, the direct trust value $DV_{i(j)}$, which reflects the previous direct contact experience between user i and j, and the indirect reference value $IDV_{i(j)}$, which is the reference value about user j that user i receives from other users. At time t, among all the decoded chunks that user i received from user j, let $Nc_{i,j}(t)$ and $Np_{i,j}(t)$ be the total number of clean copies and that of the pollute copies, respectively. Then,

$$DV_{i(j)}(t) = \frac{c_{i(j)}(t)}{c_{i(j)}(t) + p_{i(j)}(t)}, \tag{1}$$

where $c_{i(j)}(t) = f(Nc_{i,j}(t))$ and $p_{i(j)}(t) = g(Np_{i,j}(t))$ are functions of $Nc_{i,j}(t)$ and $Np_{i,j}(t)$, respectively. A simple example is to let $c_{i(j)}(t) = Nc_{i,j}(t)$ and $p_{i(j)}(t) = Np_{i,j}(t)$, and in this case, $DV_{i(j)}(t)$ is the percentage of clean chunks that user j has uploaded to user i by time t. For an unknown user j, i sets $DV_{i(j)}(t) = 0.5$.

The indirect reference value $IDV_{i(j)}$ captures the network (that is, the communities) opinion on user j [6]. Periodically, each user i sends requests to a set of randomly selected peers to collect their past direct experiences with j, and computes

$$IDV_{i(j)}(t) = \frac{\sum_{k \in U_{i(j)}(t)} DV_{i(k)}(t) RV_{k(j)}(t)}{\sum_{k \in U_{i(j)}(t)} DV_{i(k)}(t)}, \tag{2}$$

where $U_{i(j)}(t)$ is the set of peers that respond to user i's request at time t asking for their opinions about user j [6]. $RV_{k(j)}(t)$ is user k's reference about user j, and in this paper, we let $RV_{k(j)}(t) = DV_{k(j)}(t)$ [8]. In (2), we normalize user k's reference $RV_{k(j)}(t)$ with $DV_{i(k)}(t)$, user i's local reputation regarding user k, to resist user defamation [8]. Here, the assumptions are that untrusted peers are more likely to submit false feedbacks to hide their own malicious behavior, and trustworthy peers are more likely to be honest on the feedbacks [8]. In this work, each user randomly selects 20 peers and asks them for their reference values.

Given $DV_{i(j)}(t)$ and $IDV_{i(j)}(t)$ as defined in the above,

$$T_{i(j)}(t) = \beta DV_{i(j)}(t) + (1-\beta) IDV_{i(j)}(t), \tag{3}$$

where $0 \le \beta \le 1$ is a parameter to adjust the weight between the direct trust value and the indirect reference value. User i updates $T_{i(j)}(t)$ when he/she decodes new data chunks and when he/she receives updated reference values from others.

3.2. Performance Criteria

Given the above trust definition, at time t, user i trusts user j if $T_{i(j)}(t) \ge th$ where th is a predetermined threshold. Users will only cooperate with their trusted peers to exchange data chunks. If $T_{i(j)}(t) < th$, user i considers user j as a malicious polluter, and will not send requests or upload chunks to user j. In this paper, we select $th = 0.5$ so that users can cooperate with each other at the beginning of the live streaming program.

Let U_m be the set containing all *malicious polluters*. For all other users who do not intentionally send polluted chunks, we call them *selfish users*, and U_s is the set including all selfish users. To measure the performance of the trust management system in identifying polluters, we use the following criteria:

$$\begin{aligned} P_d(t) &= \frac{\sum_{i \in U_s, j \in U_m} I\left[T_{i(j)}(t) < th\right]}{|U_s| \cdot |U_m|} \\ \text{and } P_f(t) &= \frac{\sum_{i,j \in U_s} I\left[T_{i(j)}(t) < th\right]}{|U_s|^2}, \end{aligned} \tag{4}$$

where $I[.]$ is the indicator function and $|A|$ returns the size of the set A. P_d is the rate that a selfish user correctly detects a malicious polluter, and P_f is the rate that a selfish user falsely accuse another selfish user as a polluter. A larger P_d and a smaller P_f indicate that the trust management system is more effective in identifying polluter without falsely accusing innocents.

At time t, when a selfish user j forwards the first B data chunks in his/her buffer to the video player for decoding, let $N_j(t)$ be the number of clean chunks among those B chunks that are to be forwarded. We use $P_c(t) \triangleq \sum_{j \in U_s} (N_j(t)/B) / |U_s|$ to measure the effectiveness of the trust management system in reducing polluted chunks. A larger P_c indicates that selfish users receive more clean chunks and thus decode videos of higher quality.

4. ATTACK-RESISTANT TRUST MANAGEMENT

Given the above trust management system, polluters try all means to hide their malicious behavior while sending out as many polluted chunks as possible. Attack resistance is a crucial requirement for trust management in P2P live streaming. In this paper, we consider two different types of attacks, the On-Off attack and the flooding attack with hand wash, and investigate the performance of the trust management systems under these attacks.

4.1. The On-Off Attack

With the On-Off attack, a polluter uploads polluted and clean chunks alternatively, and pretends to be a selfish user who "unintentionally" sends polluted chunks from time to time. We define R_{offon} as the off-on ratio, which is the average number of clean chunks that a polluter uploads per polluted chunk that he/she uploads. A larger R_{offon} indicates that the polluter sends more clean chunks before he/she sends another polluted copy. The On-Off attack exploits time-domain inconsistency to attack the trust management system, and helps the attacker avoid being detected while causing damages to the live streaming system [7]. Thus, it poses new challenges to distinguish malicious polluters who deploy the On-Off attack from selfish users who unintentionally upload polluted chunks.

To address this dynamic personality of peers, following the work in [7, 8], we use an adaptive time window-based algorithm and introduce an adaptive forgetting factor $0 < f < 1$. The basic idea is as follows: the trust value cannot be quickly increased by uploading only a small number of clean chunks, and it will drop immediately if a user starts uploading polluted chunks.

In our trust model, similar to [7], the forgetting factor f is defined as

$$f = \begin{cases} 1-\epsilon & \text{if } DV_{i(j)}(t) \leq 0.5, \\ \epsilon & \text{if } DV_{i(j)}(t) > 0.5, \end{cases} \tag{5}$$

where $0 < \epsilon < 0.5$. We let $\epsilon = 0.1$ in this paper. When a selfish user i updates the trust values at time t, based on his/her last updated trust values at time $t-1$, user i first calculates

$$\begin{aligned} c_{i,j}(t) &= c_{i,j}(t-1)f + Nc_{i,j}(t) - Nc_{i,j}(t-1), \\ p_{i,j}(t) &= p_{i,j}(t-1)f + Np_{i,j}(t) - Np_{i,j}(t-1), \end{aligned} \tag{6}$$

and then user i calculates $DV_{i(j)}(t)$ and $T_{i(j)}(t)$ using (1) and (3), respectively. The adaptive forgetting factor makes the trust management system remember bad behavior for a longer time than good behavior, and a user needs to continuously upload many clean chunks to recover his/her trust value.

Figure 1 shows our simulation results on the On-Off attack. In our simulations, there are a total of 144 users. 36 of them use DSL connections with upload bandwidth of 1Mbps, and the rest are cable users whose upload bandwidth are 300Kbps. There are 10 polluters who send polluted data chunks and attack the trust management system independently, and the rest are selfish users. The streaming server's upload bandwidth is 6Mbps. Each user's buffer can store 30 seconds' video frames. Each round lasts $\tau = 1/9$ second, and $B = 4$ data chunks are forwarded to the video player for decoding per second. We use the "foreman" video sequence with frame rate 30 frames per second and encode it at bit rate 64Kbps. Each group of picture (GOP) has a duration of 1 second, and each GOP is divided into 4 data chunks of equal length.

Figure 1 plots P_d and P_f when polluters use different R_{offon}. From Figure 1, $P_f < 1\%$ after the initial buffering stage and the

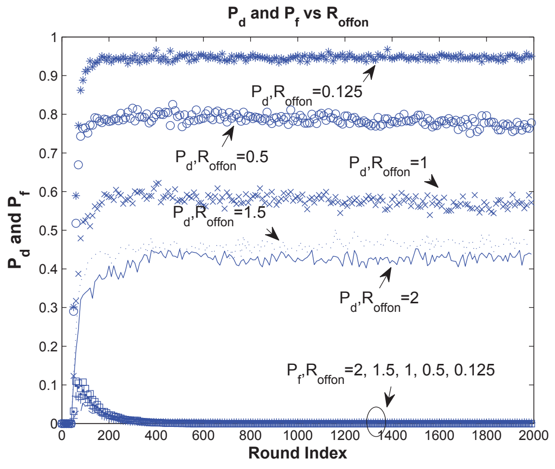

Fig. 1. Performance of the trust management system under On-Off attacks. $\beta = 0.25$, $th = 0.5$, and $\epsilon = 0.1$.

false alarm rate is relatively unaffected by the off-on ratio. In addition, when a polluter chooses a higher R_{offon} and uploads more polluted data chunks, he/she has a larger chance of being detected. Thus, to remain undetected, a polluter has to upload much more clean chunks than polluted ones, and the adaptive window-based trust management system can effectively resist the On-Off attack.

4.2. The Flooding Attack with Hand Wash

In the flooding attack, a polluter claims that he/she has all the chunks, thus attracting a lot of requests from other users. He/she then sends polluted chunks whenever possible. The flooding attack utilizes the fact that in the current P2P live streaming system, a polluted data chunk cannot be detected until its playback time, and there is a time difference between the arrival and the detection of a polluted chunk. The polluter uses this lag to flood selfish users' buffers with as many polluted chunks as possible. In addition, since the selfish user is unaware of the existence of a polluted chunk, he/she may forward it to other users, which makes the polluted chunk spread over the whole network. To make it even worse, the unintentional forwarding of polluted chunks reduces the selfish user's trust value and increases the false alarm rate of the trust management system. Thus, selfish users will not cooperate with each other, which significantly damages the performance of P2P live streaming systems. The flooding attack is especially effective if used together with the hand wash attack.

Figure 2 and Figure 3 demonstrate the effectiveness of the flooding attack with hand wash. The simulation setup is the same as in Figure 1, and a polluter reenters the system with a new ID every 150 rounds. From Figure 2, whenever the polluters apply the hand wash attack, the trust management system takes a few dozen rounds to detect the polluters again (with $P_d \geq 0.9$), which is long enough for polluters to cause significant damages. Also, the flooding attack with hand wash gives a false alarm rate of 0.9, which prevents selfish users from cooperating with each other and causes the extremely low $P_c \leq 5\%$ in Figure 3. Consequently, it is critical to design trust management systems that can resist the flooding attack with hand wash in P2P live streaming.

We propose *early decoding* that decodes data chunks and detects polluted ones as early as possible. If a polluted data chunk is not detected until its playback time, the selfish user has no chance to ask for a clean copy. Early detection of a polluted chunk gives the selfish user a second chance to request a clean copy, and thus im-

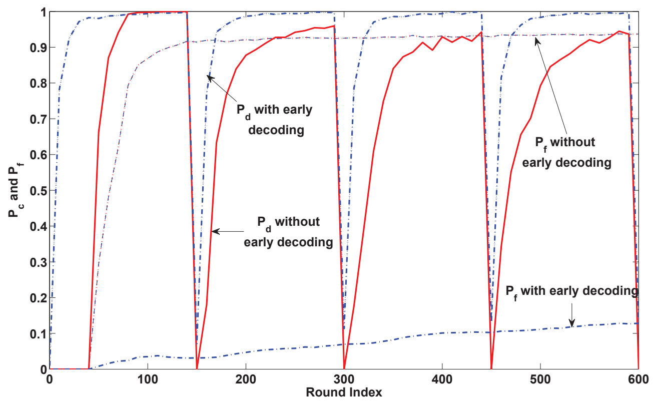

Fig. 2. P_d and P_f under the flooding attack with hand wash.

proves the quality of the rendered video. Furthermore, once a chunk is detected as polluted, the selfish user deletes it from his/her buffer, which prevents the propagation of the polluted chunk. Also, early decoding will enable early detection of polluters and prevent them from upload more polluted data. Finally, early decoding reduces the number of polluted chunks that are unintentionally forwarded by selfish users, and helps reduce the false alarm rate of the trust management system. Thus, it stimulates user cooperation and improves the system performance.

Note that current video compression standards use motion estimation to remove temporal/spatial redundancy, which introduces data dependency in the compressed video bit stream. To address this issue, we explore the independence among different GOPs, and divide each user's buffer into blocks of equal length, each with B chunks that belong to one GOP. With the simulation setup in Figure 1, each GOP includes one second's frames and $B = 4$ data chunks. When a user determines which chunk to request, he/she first decides which block to request and gives higher priorities to blocks with earlier playback time. Once block k is selected, the user will request the *first* missing chunk in block k. Such a GOP-based chunk request algorithm allows users to decode a data chunk and determine if it is polluted right after its arrival.

An issue with early decoding is that additional storage space is required to save the decoded frames that have not reached their playback time. Such huge storage space may not always be available. To address this issue, we assume that each user has enough space to store τ_s seconds' video frames in uncompressed format, and it is updated periodically in a sliding window fashion. We apply early decoding only to the first τ_s blocks in the buffer. With a larger τ_s, more data chunks can be decoded immediately after their arrivals and, therefore, more polluted chunks can be detected earlier. Thus, a larger τ_s is preferred to improve the robustness of the system against the flooding attack with hand wash.

Figure 2 and Figure 3 show the effectiveness of our proposed chunk request and early decoding algorithm with $\tau_s = 6$ seconds. From Figure 2, early decoding helps detect malicious polluters earlier and reduces the false alarm rate from 0.9 to $P_f \leq 0.16$. It stimulates user cooperation and P_c is increased above 0.9 as shown in Figure 3. Even under the flooding attack with hand wash, early decoding helps a selfish user receive more than 90% of the data chunks in the video, all of which are clean, and it improves the robustness of the trust management system in P2P live streaming.

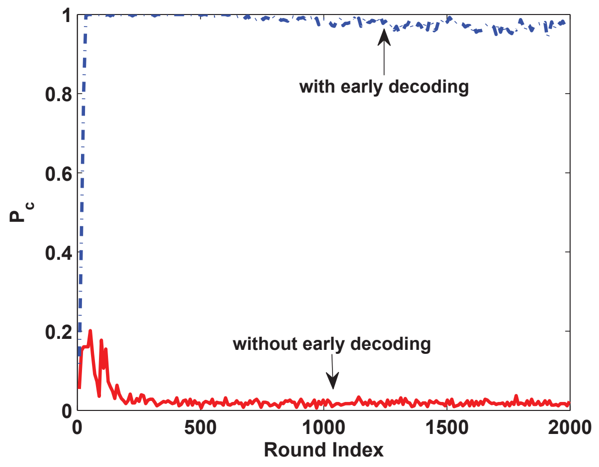

Fig. 3. P_c under the flooding attack with hand wash.

5. CONCLUSIONS

In this paper, we proposed a trust management system to resist pollution attacks in peer-to-peer live streaming, and analyzed its attack resistance. To resist the On-Off attack, we used a window-based trust updating algorithm with an adaptive forgetting factor, which forces polluters to upload many clean chunks if they wish to remain undetected. To resist the flooding attack with hand wash, we proposed to use early decoding and the GOP-based chunk request algorithm to detect polluted chunks as early as possible. Our simulation results show that the proposed trust management system can effectively identify polluters, stimulate user cooperation even under these attacks, and significantly improve the quality of the rendered videos at the user's side.

6. REFERENCES

[1] Z. Liu, Y. Shen, S. Panwar, K. Ross, and Y. Wang, "Using layered video to provide incentives in P2P live streaming," *ACM SigComm Workshop on P2P Streaming and IP-TV*, pp. 311–316, Aug. 2007.

[2] W. Lin, H. V. Zhao, and K. J. R. Liu, "Incentive cooperation strategies for Peer-to-Peer live streaming social networks," *to appear, IEEE Trans. on Multimedia*, April 2009.

[3] P. Dhungel, X. Hei, K. W. Ross, and N.Saxena, "The pollution attack in P2P live video streaming: measurement results and defenses," *ACM SigComm Workshop on P2P Streaming and IP-TV*, pp. 323–328, Aug. 2007.

[4] M. Feldman, C. Papadimitriou, J. Chuang, and I Stoica, "Free-riding and whitewashing in Peer-to-Peer systems," *IEEE Journal. sel. areas in comm.*, vol. 24, no. 5, pp. 1010–1019, May 2006.

[5] S. Yang, H. Jin, B. Li, X. Liao, H. Yao, and X. Tu, "The content pollution in peer-to-peer live streaming systems: Analysis and implications," *International Conference on Parallel Processing*, Sept. 2008.

[6] A. Borges, J. Almeida, and S. Campos, "Fighting pollution in p2p live streaming systems," *IEEE International Conference on Multimedia and Expo*, June 2008.

[7] Y. Sun, Z. Han, and K. J. R. Liu, "Defense of trust management vulnerabilities in distributed network," *IEEE Comm. Magazine*, vol. 46, no. 2, pp. 112–119, Feb. 2008.

[8] X. Li and L. Liu, "PeerTrust: Supporting reputation-based trust for Peer-to-Peer electronic communities," *IEEE Trans. on Knowledge and Data Engineering*, vol. 16, no. 7, pp. 843–857, July 2004.