Diagnosis of Content Pollution in P2P Live Streaming Networks

Roverli P. Ziwich[1], Emanuel A. Schimidt[1], Elias P. Duarte Jr.[1], Ingrid Jansch-Pôrto[2]
[1]Federal University of Paraná (UFPR)
Dept. Informatics, P.O.Box 19018, 81531-990, Curitiba, PR, Brazil
[2]Federal University of Rio Grande do Sul (UFRGS)
Informatics Inst., P.O.Box 15064, 91501-970, Porto Alegre, RS, Brazil
Email: {roverli,emanuel,elias}@inf.ufpr.br, ingrid@inf.ufrgs.br

Abstract

Content pollution is one of the challenges for massively deploying live streaming P2P networks in the Internet. As the peers themselves are responsible to retransmit data, there is no trivial solution to this problem. This work presents a new strategy to detect content pollution that employs comparison-based diagnosis to identify modifications on the data stream. A peer compares randomly selected chunks received from its neighbors. Based on the comparison results, peers that transmitted polluted content are identified. The proposed solution was implemented using Fireflies, a scalable and intrusion-tolerant overlay network. Experimental results show that the strategy represents a feasible solution to detect content pollution and causes a low overhead in terms of network bandwidth.

1 Introduction

Peer-to-peer (P2P) live streaming video transmissions are becoming increasing popular in the Internet [18]. Several systems that implement video live streaming in P2P networks have appeared in last years, for instance PPLive[1], SopCast[2], and PPStream[3]. In contrast with the traditional client-server solutions, the peers in a peer-to-peer live streaming network not only consume but also share the stream content among themselves. In this way, using a P2P network to transmit live streaming sessions decreases the demand on the source, and allows thousands, or even millions of users to join the network and follow the transmission at the same time.

In a P2P live streaming network the content is initially transmitted by a *source* server. The transmitted data is divided in *chunks*, which are shared between the peers themselves. On the other hand, these systems present several challenges. One of which is to deal with the *churn*, i.e. the fact that peers join and leave the network continuously, without decreasing the quality of the transmission. Another important challenge is to deal with malicious peers, that can harm the system in several ways, including by polluting content as described below.

A content pollution attack consists of the unauthorized modification of the original data by one or more malicious peers. The modifications can be of different types, including changes of the original data, creation of new data, or even data destruction or omission [12]. A characteristic that also contributes to the content pollution problem in P2P live streaming networks is the restriction on the maximum time data is allowed to reach a peer. This is relevant because the detection of the polluted data and the creation of new requests to replace those chunks can increase delays and also cause jumps (or gaps) in the transmission [29].

Some of the solutions that handle the content pollution problem in P2P live streaming networks assume that all peers in the network previously know – or receive during the transmission – the hash value [28] of all chunks. In this way, all peers in the P2P network are able to identify any change in any chunk. A peer can generate a new hash of the received chunk and verify if it matches the previously received hash value. However, this strategy has a drawback, which is the hash distribution. A malicious peer can transmit a set of fake hash values within modified chunks, deceiving other peers in the network.

Another approach for dealing with pollution detection is the usage of digital signatures, basically consisting of hash values encrypted with a private key [13]. In this strategy the chunk's digital signatures are generated by the source server and they are transmitted with the chunks. Even if a malicious peer transmits a modified chunk to its neighbors, the malicious peer will not be able to forge the corresponding signature. On the other hand, all peers will need to check

[1]PPLive - *http://www.pplive.com/en*
[2]SopCast - *http://www.sopcast.com*
[3]PPStream - *http://www.ppstream.com*

978-0-7695-4962-0/13 $26.00 © 2013 IEEE
DOI 10.1109/LADC.2013.13

the digital signatures, which is a computationally expensive procedure.

This work presents an alternative solution to diagnose content pollution in live streaming P2P networks. The proposed solution uses comparison-based diagnosis [9] to detect unauthorized modifications in the transmitted content. Each system peer executes comparisons over randomly selected chunks obtained from all of its neighbors. Based on the comparison results, a classification of the peers in sets is performed allowing the detection of content pollution.

The proposed solution was implemented using Fireflies – a scalable and intrusion tolerant overlay network protocol [14, 13]. Fireflies uses the *pull-based* strategy to transmit data, and the mesh topology is employed. The proposed strategy was implemented using the same Fireflies simulator described in [13]. Thousands of experiments were performed in order to evaluate the overhead of the comparisons executed. Results show that this overhead is low enough, and that the application of comparison-based diagnosis to detect content pollution in P2P networks is feasible.

The rest of the paper is organized as follows. Section II gives a brief introduction to P2P live streaming, comparison-based diagnosis and to the Fireflies protocol. Section III presents the proposed strategy to diagnose content pollution in P2P live streaming networks. In section IV experimental results are presented. Section V presents a description of related work and is followed by the concluding remarks in section VI.

2 Preliminary Definitions

This section starts with a brief introduction to live streaming in P2P networks; next the Fireflies protocol – a scalable and intrusion-tolerant overlay network – is described; finally, an overview of comparison-based diagnosis is presented, that is the basis on which the proposed pollution detection strategy was devised.

2.1 Live Streaming in P2P Networks

Live streaming in P2P networks starts at a *source* server that have the content that is disseminated. The content is divided in small pieces, called *chunks*. The source server is responsible to insert the chunks in the network. The chunks are then shared by *peers*, that both consume and retransmit content to other peers. Two topologies are usually employed to transmit live streaming on P2P networks [19]: the *tree* topology and the *mesh* topology, described below.

In the *tree* topology, the peers form a tree, of which the source server is the root. The main advantage of using the *tree* topology is that, after the tree is built, the data transmission decisions are simple: a peer receives data from its parent and forwards data to its children [6]. In a fault-less system this topology presents a low delay between the transmission of the data by the source server and the arrival of the data at all system peers. However the tree topology presents three important disadvantages. (1) If data is lost at a peer located close to the root, it will affect its whole subtree. (2) This topology has low resilience under churn: if a peer leaves the system and it is not a leaf, then its whole subtree will stop receiving data until the tree is rebuilt [19]. (3) The average upload rate is lower than in other topologies. This happens because all leaf peers only receive data and do not participate by actively transmitting data.

The *mesh* topology is not structured, i.e. it is not based on a predefined network structure [24]. In this topology, when a peer wants to join the transmission, it simply connects itself to a list of peers and starts to share information. The main problem of mesh networks is related to the way peers exchange data: to receive a particular chunk, a peer needs to first issue a request and to keep a list of the data available at all its neighbors. Thus a mesh-based transmission requires more bandwidth compared to the tree-based alternative, and this may lead to higher delays in the data propagation throughout the system.

Three strategies are usually employed by the peers to exchange data [19]: *push-based*, *pull-based* and *push-pull-based*. The *push-based* strategy is mainly used in tree topologies. Data is transmitted by some peers to other peers without having the need for previous requests. Although in a strictly push-based system this strategy avoids the overhead of requests, there is no way to request any data, even if the transmission had previously failed. Another disadvantage of this strategy is that, if multiple data senders are present in the network, an specific peer can receive duplicated data, that represents a waste in terms of bandwidth usage.

In the *pull-based* strategy, specific data is sent by peers that have received a request for that data. If a particular chunk has not been received due to some fault, then the peer can reissue the request and try to obtain the lost data. On the other hand, peers need to keep data availability information. The *push-pull-based* strategy combines both approaches: data can be transmitted without specific requests, but it is also possible to request a particular piece of data [19]. This strategy can be implemented as pull-based, also allowing the requests to be issued (push-based) if the pull-based strategy missed some chunk [11].

2.2 The Fireflies Protocol

Fireflies is a protocol that builds a scalable and intrusion-tolerant overlay network [14]. All network peers execute the Fireflies protocol using a pull-based strategy to transmit data, and the peers are organized in a mesh topology. Besides the peers, the system also assumes a source server,

which generates the chunks. The source server is considered to be a reliable unit that never fails.

The chunks are sent from the source server to a limited number of system peers. The peers then share the chunks between themselves, in order to have all peers receive all chunks disseminated by the source. All peers receive a sequential identifier and they are organized on multiple rings [14]. The number of rings is configurable and each ring contains all peers. The rings dictate how peers communicate among themselves i.e. the protocol determines which peers are the neighbors of which other peers. Considering all rings, it is possible that a particular peer has some specific neighbor in more than a ring: every peer always has at least 2 and at most $(2 * \lambda)$ neighbors, where λ is the number of rings configured in Fireflies.

As an example, Figure 1 shows a system with 9 peers organized in 3 rings. Note that the neighbors of peer 1 are the peers 2, 3, 4, 7 and 9. The figure also shows that peer 1 has peer 3 as a neighbor in two different rings. In Fireflies the source server receives identifier 0, does not participate in the ring configurations and its neighbors are randomly chosen. The amount of source neighbors is a configurable parameter in the protocol.

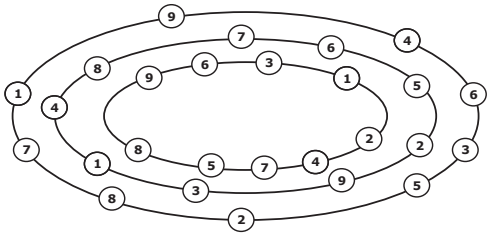

Figure 1. Fireflies example: 9 peers organized in 3 rings.

The Fireflies protocol configures in each peer an *availability window* that is a list that indicates which are the chunks every peer has available to send to its neighbors. The protocol also configures in every peer an *interest window* which keeps a list of the chunks that the peer needs to receive. When a peer receives a chunk, it notifies all its neighbors about the availability of that chunk. Based on these notifications all system peers are able to maintain a list of which chunks are available at their neighbors. If a peer p knows that one of its neighbors n has an available chunk c, and this chunk is in the interest window of peer p, this peer requests chunk c to peer n. When peer n receives the request from p, peer n verifies if chunk c is still in its availability window. In this case, chunk c is sent to peer p; otherwise, the request is simply ignored. This is exactly same procedure that happens with all chunks generated by the source server: when the source generates a new chunk, it notifies all its neighbors about the availability of that chunk, and then the diffusion of the chunk starts.

2.3 Comparison-Based Diagnosis

Comparison-based diagnosis [9] allows the state of the units of a given system to be identified as either *faulty* or *faulty-free*. Diagnosis is based on the comparison of task outputs produced by the execution of a given task by pairs of units. The complete system diagnosis is obtained by processing the results of all comparisons. The set of all comparison results is called the *syndrome* of the system. The MM comparison-based model was proposed by Maeng and Malek [21] to diagnose multiprocessor systems composed of homogeneous units. The system is represented by a graph $G = (V, E)$, where V is the set of units and E is the set of communication links. In this model the output of the same task executed by a pair of units are compared. An unit k is a comparator – or tester – of other two units i and j if and only if $(k, i) \in E$ and $(k, j) \in E$, and also if $k \neq i$ and $k \neq j$. If the comparison of units i and j by unit k indicates a match and also if the comparator unit k is fault-free, then units i, j are also faulty-free. If the comparison indicates a mismatch, at least one of the units i, j, or k is faulty.

In the MM model, the outputs of a task returned by two faulty units is assumed to be always different. If the comparator k is faulty, all the comparison results are not reliable, i.e. the comparator may lie. After the comparison of the task outputs, the comparator sends the comparison result to a central observer. This central observer is a reliable unit that performs the complete system diagnosis. Maeng and Malek also presented a special case of the MM model, called MM*, in which every unit compares all of its neighbors to which they are connected.

In [31] a generalized comparison-based diagnosis model is presented in which the units themselves are able to perform the system diagnosis, instead of a central observer. And most important, one of the main differences of this model to others is that the comparison result of tasks executed by pairs of faulty units may result in a match. In this work we employ this diagnosis model, but also assume that every unit compares results from all its neighbors, such as in the MM* model.

3 The Proposed Approach: Diagnosis of P2P Content Pollution

This section presents the proposed solution to diagnose content pollution in P2P live streaming networks. The proposed approach uses comparison-based diagnosis to detect pollution and is based on the Fireflies overlay network. Besides the source server and the peers – which are already

components of the Fireflies architecture itself – the proposed solution employs two components: the *comparator module* and the *tracker*, described next.

The *comparator module* is a component executed by every peer, and is integrated to the Fireflies protocol itself, having access to both the received chunks and the availability window of the peer. This module is responsible to execute comparisons of predetermined chunks. Comparisons are executed on a particular chunk which is obtained from all the peers' neighbors. The neighbors are then classified in sets according to the comparison results, and the classification is sent to the tracker. The *tracker* is a reliable unit that is accessible to all system peers. The tracker is responsible to unify the classifications received from the comparators in a system wide way, thus performing the diagnosis of the system, i.e. determining if there is content pollution in the network, and which peers have received polluted data.

The comparator module executes at every peer i and periodically requests chunks with identifier cid (*chunk identifier*) from all neighbors. The comparator request is a regular Fireflies request, sent through a Fireflies connection. Even if the peer that receives the request is malicious, it cannot determine that this request was issued by a comparator module, and thus it does not handle this request in a different way.

The identifier (cid) of the chunks to be compared is randomly selected by the tracker, that notifies the comparator modules. As soon as those chunks become available at the neighbors of peer i, the comparator module requests the chunks. After all chunks are received, the comparator module compares and classifies the neighbors in sets $U_{i,cid}$. Set $U_{i,cid}$ keeps for each version of the chunk cid received from the neighbors, the set of identifiers of the peers that sent that content, i.e.

$$U_{i,cid} = \{(chunk_a, \{peer_i, peer_j, ...\}), (chunk_b, \{peer_k, peer_l, ...\}), ...\}.$$

One specific subset keeps the list of peers that have not send replies. As soon as set $U_{i,cid}$ is complete, i.e. with information about all neighbors of peer i, it is sent to the tracker.

An optimization was made for the purpose of reducing the length of the message peers send to the tracker: set $U_{i,cid}$ keeps the hash value of the chunk instead of the whole chunk itself. In this way,

$$U_{i,cid} = \{(hash\_chunk_a, \{peer_i, peer_j, ...\}), (hash\_chunk_b, \{peer_k, peer_l, ...\}), ...\}.$$

The comparator module at a given peer is assumed to be able to correctly classify the peer's neighbors and to exchange messages reliably with the tracker. To implement this assumption, it is possible to use an approach with TSL/SSL in which asymmetric cryptography is employed in the beginning of each session and then a secret key is established for communication between the tracker and the comparator.

Figure 2 shows an example to illustrate the proposed strategy. In the example, chunk 13 is one of the chunks that the tracker randomly chose to be compared. Peers 4 and 6 have requested chunk 13 from all their neighbors. The figure shows the transmission of chunk 13 for all neighbors of peers 4 and 6. The directed edges represent the transmission of chunks that were requested. The other undirected edges represent the communication links between peers or between a peer and the source server.

The example considers that the original chunk 13 has hash value equal to AA, and that a polluted version of this same chunk improperly modified by peer 5 has hash value AB. Based on this considerations, the sets $U_{i,cid}$ after the classification performed by peers 4 and 6 are shown in Table 1. Peers 4 and 6 put themselves in the sets $U_{i,cid}$, being inserted in the group corresponding to the content of the chunk they have.

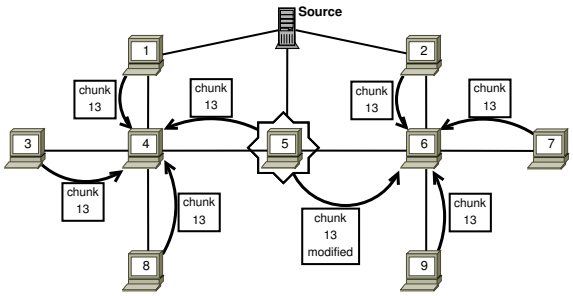

Figure 2. Transmission of chunk 13 to peers 4 and 6; peer 5 is malicious.

After the tracker receives set $U_{i,cid}$ from every peer i, it classifies all peers using another set T_{cid}. Although set T_{cid} has the same format of set $U_{i,cid}$, in T_{cid} a given peer may appear in more than one subset. An example of this situation is shown in Figure 2, in which peer 5 has sent chunks with different hash values to its neighbors: peers 4 and 6. In this example the tracker will classify peer 5 in two different subsets of the corresponding set T_{cid}: in the subsets indexed by hash values AA and AB.

As the source is a reliable unit that does not send polluted chunks, the source will always be in only one subset of set T_{cid}. To perform the diagnosis, the tracker considers as faulty the peers that have chunks classified in different subsets of T_{cid} as the subset to which source server belongs. Figure 3 illustrates the transmission to tracker of sets $U_{i,13}$ by peers 4 and 6.

Table 2 shows set T_{13} with the final classification performed by the tracker for chunk 13. The set was obtained with the union of sets $U_{4,13}$ and $U_{6,13}$. This set T_{13} is still

Peer	*chunk*	*Set* $U_{i,cid}$
4	13	$U_{4,13} = \{(AA, \{1, 3, 4, 5, 8\})\}$
6	13	$U_{6,13} = \{(AA, \{2, 6, 7, 9\}), (AB, \{5\})\}$

Table 1. Sets $U_{4,13}$ and $U_{6,13}$ generated respectively by peers 4 and 6.

chunk	*Set* T_{cid}
13	$T_{13} = \{(AA, \{source, 1, 2, 3, 4, 5, 6, 7, 8, 9\}), (AB, \{5\})\}$

Table 2. Set T_{13} generated by the tracker based on received sets $U_{4,13}$ and $U_{6,13}$.

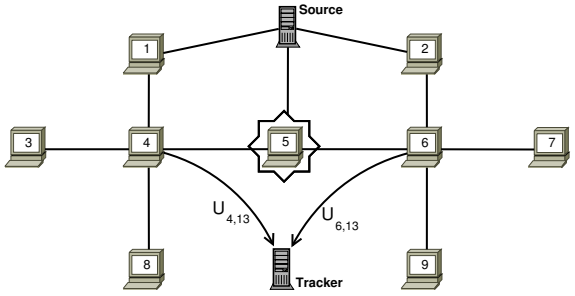

Figure 3. Peers 4 and 6 send their sets $U_{i,13}$ sets to the tracker.

```
Algorithm: ComparatorModule
 1: begin
 2:    list_of_cids ← get from tracker the list of
 3:                   chunks that needs to be compared;
 4:    whenever a neighbor n makes available a new chunk cid do
 5:       if cid ∈ list_of_cids then
 6:          if timer_cid is not initialized then
 7:             initialize timer_cid;
 8:          end if
 9:          request and get chunk cid from n;
10:          update U_{i,cid};
11:       end if
12:    end whenever
13:
14:    whenever ((U_{i,cid} has data about all its neighbors) or
15:              (timer_cid > response_time_limit)) do
16:       if timer_cid > response_time_limit then
17:          include neighbor peers that have not
18:                answered in a specific set of U_{i,cid};
19:       end if
20:       send U_{i,cid} to tracker;
21:       list_of_cids ← get from tracker the list of
22:                      chunks that needs to be compared;
23:    end whenever
24: end
```

Figure 4. The proposed comparator-module algorithm that is executed by all peers.

partially complete, because the tracker remains waiting for the remaining $U_{i,13}$ sets from the other peers.

As the comparator module executes continuously, the tracker may be still receiving information about a given chunk cid_a from some peers while other peers are already sending information about another chunk cid_b. For this reason, different sets are kept for different chunks, e.g. T_{cid_a} and T_{cid_b}.

Figure 4 presents the algorithm executed by the comparator module at peer i. Initially (line 2) the comparator obtains the list of chunks that needs to be compared. This information is obtained from the tracker. The comparator module then waits for its neighbors to have the chunk available. The algorithm executes from line 4 whenever any neighbor n has a new chunk available. If the cid of the new available chunk is in the list of chunks that need to be compared, peer i checks if the local timer associated to that cid was already initialized (line 6). This timer is used as a *timeout* limit to peer i perform all comparisons related to chunk cid. The timer is started at the first time the chunk becomes available at a neighbor (line 7). In line 9, a request for chunk cid is sent to peer n. After this, set $U_{i,cid}$ is updated in order to include peer n, which is classified based on the comparison of the content of the received chunk.

From line 14 the algorithm checks if set $U_{i,cid}$ is complete, i.e. has information from all neighbors of peer i, and if the time limit to get information about that chunk was reached. In both cases set $U_{i,cid}$ is sent to the tracker (line 20). Peers that have not sent the chunk are classified accordingly (line 17). The response timeout interval ($response\_time\_limit$) is based on the size of the peers' availability window that also considers the frequency in which new chunks are generated by the source server.

The tracker continually receives sets $U_{i,cid}$ for a given chunk cid, as shown in line 2 of algorithm $Diagnosis$ presented in Figure 5. Whenever the tracker receives a set $U_{i,cid}$, it classifies the peers in the subsets of that $U_{i,cid}$ in a corresponding T_{cid} set (lines 4–10). After completing the classification of all peers or if the time limit for the peers to send their respective sets $U_{i,cid}$ has been reached (line 14), the tracker outputs the system diagnosis (line 20). If peer i does not send set $U_{i,cid}$ within the response time limit, this peer i is classified into a specific subset of the set T_{cid} (lines 16 and 17). As it processes set T_{cid}, the tracker classifies as faulty all peers not in the same subset as the source.

```
Algorithm: Diagnosis
1: begin
2:    whenever receives a U_{i,cid} set do
3:       for every subset u of U_{i,cid}
4:          if hash_chunk_u ∈ T_{cid} then
5:             insert peers from hash_chunk_u in the
6:                          corresponding set of T_{cid};
7:          else
8:             create new subset in T_{cid} with the peers of subset
9:                          u = (hash_chunk_u, {list of peers});
10:         end if
11:      end for
12:   end whenever
13:
14:   whenever ((T_{cid} has data about all its neighbors) or
15:             (timer_cid > response_time_limit)) do
16:      if timer_cid > response_time_limit then
17:         include peers that have not answered in
18:                          a specific set of T_{cid};
19:      end if
20:      print the diagnosis related to the comparisons
21:                          of chunk cid based on T_{cid};
22:   end whenever
23: end
```

Figure 5. Diagnosis algorithm executed by the tracker.

4 Experimental Results

The proposed strategy was implemented using the Fireflies simulator described in [13]. Several experiments were executed for network sizes of 200, 500 and 1000 peers. Each experiment lasted 200 seconds and the source server generated 30 chunks/second. The chunk size was 10 KB. Both the availability window and the interest window of all peers were configured for 3000 chunks. Moreover, Fireflies was set up to organize the peers in three rings, so that each peer had at least two and at most six neighbors.

The main purpose of the experiments were to (a) check that peers that received polluted data were properly diagnosed; (b) compute the overhead added by the proposed solution in terms of the amount of additional chunks transmitted in the network; (c) check the effect of churn; (d) check the effectiveness and overhead of the proposed strategy for different network sizes (in terms of the number of peers) and for different monitoring frequencies, and (e) evaluate the tracker scalability.

The main parameters that were varied were:

(1) the total number of peers: 200, 500 and 1000 peers;

(2) the number of malicious peers: 0%, 5%, 10%, 15%, 20% and 25% of the amount of peers;

(3) the monitoring frequency: 1 and 15 seconds; this is the frequency in which the tracker randomly chooses a chunk to be monitored (that is also the frequency in which the tracker produces a new diagnosis report of system pollution;

(4) the behavior a malicious peer presents, two types were possible: (a) modify the chunk content with a probability of 100%, and (b) modify the chunk content with a probability of 50%;

(5) churn could be turned on and off: for experiments with churn, either 50% or 100% of the number of peers had a probability to leave and join the system. Peers joined the system following a normal distribution with average of either 50% or 100% of the number of peers and standard deviation of 20. Peers left the network following a Poisson distribution with average of either 50% or 100% of the number of peers.

The experiments were executed a total of 20,000 times. Results are summarized and presented in the graphs of Figures 6–13. The lines in the graphs represent the averages, while vertical interval lines show the 95% confidence interval.

Figure 6 shows the number of chunks sent over the network by the Fireflies protocol without the proposed solution on systems with 500 peers. This figure show both results of the experiments performed with churn (50% of peers) and without churn. It is possible to note that the average number of chunks sent by Fireflies was always between 1.9 and 2.8 million chunks. In all figures, curves identified with "always" in the graph refer to experiments in which the malicious peers alter every chunk content. On the other hand, the curves identified with "random" refer to the experiments in which malicious peers alter content randomly, with a probability of 50%.

Figure 7 shows the average number of chunks requested by the comparators for systems with 500 peers. The comparator module was configured with a monitoring interval of 15 seconds. It is possible to note that the average number of chunks requested by both faithful and malicious peers is in the range of 22,000 and 33,000. Thus in comparison with the number of chunks sent by the Fireflies protocol alone (Figure 6), the proposed solution generated an overhead of about 1.2%. Note that this small overhead was observed with a monitoring interval of 15 seconds; depending on the bandwidth available on the network, this frequency can be increased or reduced.

Figure 8 shows the average number of peers that received polluted data (also for systems with 500 peers). In this figure it is possible to note that even with only 5% of malicious peers, the total number of peers that received polluted chunks was up to 86 of peers. If 25% of the peers are malicious, the total number of peers that have received polluted data reached 295 (59%) of all peers.

Figure 9 shows the number of peers that had received polluted data but under varying churn rates. The experiments were also executed for systems with 500 peers. It is possible to note that the average number of polluted peers

with the higher churn rate of 100% was actually lower than that with the 50% rate. This happened for with a higher churn rate, more malicious peers were removed from system.

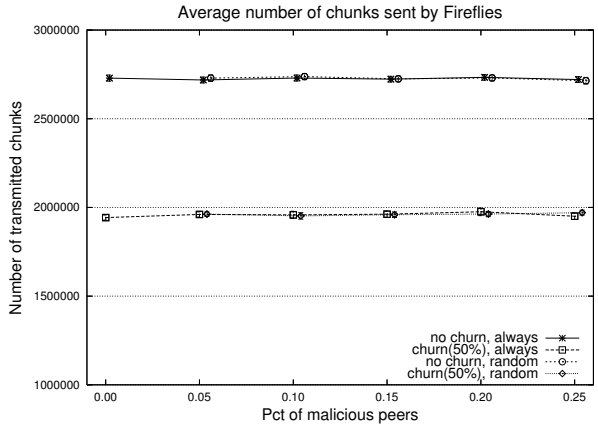

Figure 6. Number of chunks sent by Fireflies.

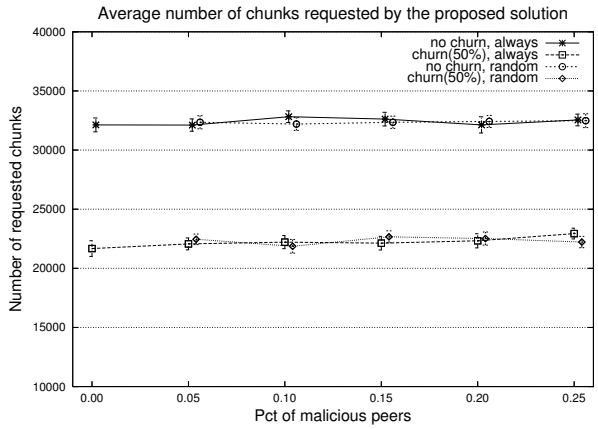

Figure 7. Number of chunks requested by comparator modules.

Figure 10 shows the average percentage of peers that received polluted chunks and were correctly diagnosed by the proposed strategy (again for systems with 500 peers and monitoring interval of 15 seconds). The figure shows that in all experiments the proposed solution correctly identified between 95% and 97% of the peers that have received polluted content. Content pollution was not diagnosed only in cases in which peers did not receive chunks in time – i.e. these cases are due to the nature of the P2P network itself, and is a consequence of the peers interest and availability windows employed.

The next Figure 11 shows the number of chunks em-

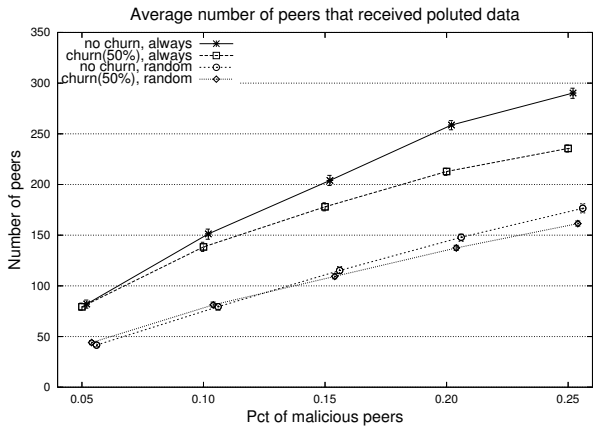

Figure 8. Peers that received polluted data.

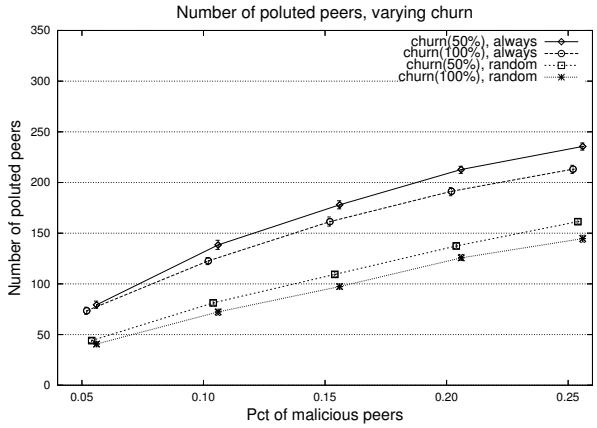

Figure 9. Peers that received polluted data varying churn.

ployed by the proposed strategy, but now varying the amount of peers in the experiments: 200, 500 and 1000 peers. The figure summarises both the experiments executed with no malicious peers and those with 25% of malicious peers. The monitoring interval was 15 seconds. It is possible to note that the overhead generated by the proposed strategy increases linearly with the network size.

Figure 12 shows the average number of chunks requested by the comparator module, but now varying the monitoring interval: 1 second and 15 seconds. The network size was 500 peers. Note that the y axis is in logarithmic scale. Results confirm that increasing the monitoring interval also increases the overhead linearly.

Finally, Figure 13 shows tracker's average bandwidth usage in kbits per second, for networks with 500 and 1000 peers (also for a monitoring interval of 15 seconds). The amount of bandwidth used is below 500 kbits per second most of the time, reaching 2.8 mbps peak once after 80 sec-

onds of transmission. This can be considered a low amount of bandwidth, and shows that the proposed tracker is scalable.

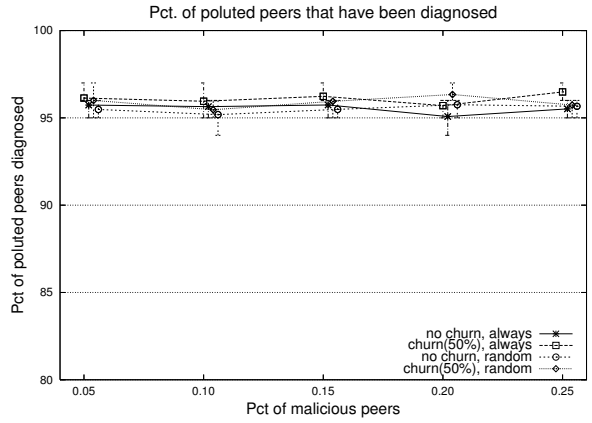

Figure 10. Percentage of polluted peers that have been correctly diagnosed.

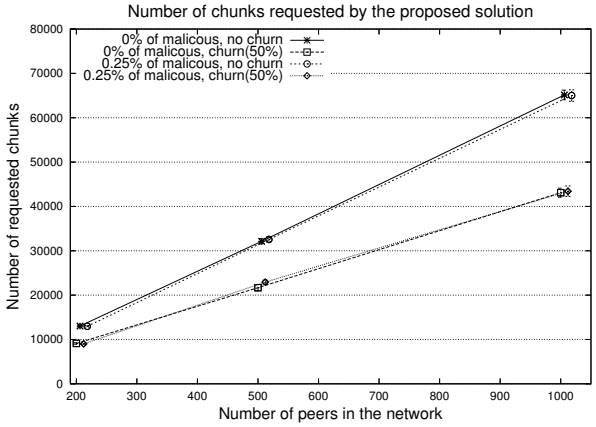

Figure 11. Number of chunks requested by comparator modules.

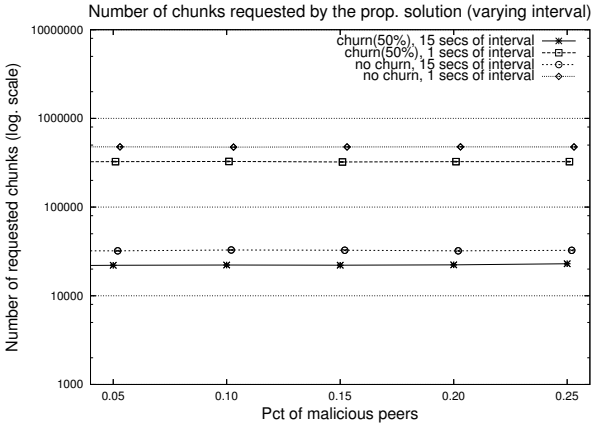

Figure 12. Number of chunks requested by comparator modules, varying the monitoring interval.

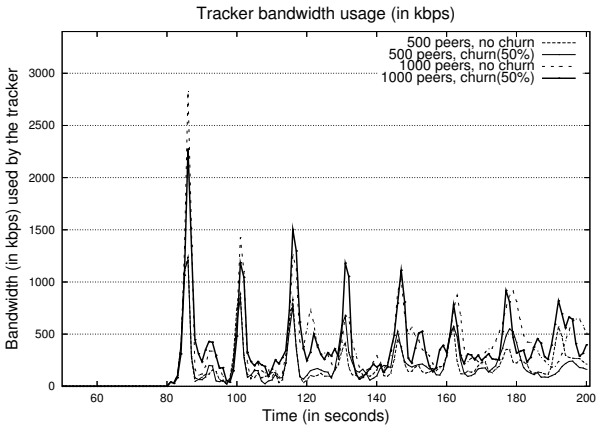

Figure 13. Tracker bandwidth usage.

5 Related Work

Several strategies have been published on dealing with pollution – or *poisoning* [4] – in P2P networks. Some of the most relevant are briefly described below.

Black lists [17] identify the polluters by keeping ranges of IP addresses that include the addresses of peers that are known to have disseminated polluted content. Those ranges are designed to include the minimum possible number of non-polluters. When applied to live streaming transmissions, this technique can be expensive [12]. Furthermore a vulnerability can be explored in which a malicious peer may assume a new address that is not in the black list.

A basic technique, employed by BitTorrent [7], is to allow peers to obtain the hash-based signatures [28] of the chunks, so that when a chunk is received the peer can check its integrity. In live streaming systems the problem is to be able to have all peers receive all signatures of all hashes in advance.

Another similar approach is to have the source server generate a digital signature for every chunk [13]. An advantage of this strategy is that the signature can be transmitted in the chunk itself, as it is signed with the source's private signature. A drawback is that this approach is expensive, as it involves public-key cryptography, and this may be prohibitive given the requirements of live streaming. *Linear Digests* are a generalization of this technique [13], in which the hash values of several chunks are grouped in one digital

signature generated by the source. When comparing *Linear Digests* to the solution proposed on this work, it is possible to see that [13] is a technique that uses more computation than communication, which is the opposite of the solution being proposed.

Some other tools [16, 15] apply cryptography to the whole chunk using a predefined shared secret key. In [15] the authors also propose a secure distributed key management scheme in which the source server periodically recomputes and retransmits an updated shared key to a limited number of peers.

In [3] the authors present another solution that uses groups of peers to maintain the integrity of content transmitted by the source. The server publishes the content information to a group of peers. Any requesting peer can verify the integrity of the requested content from the group of content maintainers.

The authors in [25] present *Credence*, a P2P system for file sharing based on ranking. In this system, a peer can classify another peer as honest, that can consequently access the shared content. In [1] the authors also present other two solutions based on ranking which are applied to live streaming.

Other alternative strategies avoid the cost of authentication. In a Merkle-tree [28] the source server computes the hash value of n consecutive chunks. These hashes are used as the leaves of a Merkle tree and the intermediate nodes are identified by the hash values of its sons in the tree. The hash values of all nodes in this tree structure are combined to perform the authentication of each chunk.

In [8] the authors evaluate four of the strategies mentioned above: black lists, cryptography, hash verification and digital signatures. The authors conclude that the use of Merkle trees is one of the most efficient in terms of computational overhead. More recently, the authors of [18] have evaluated the impact of pollution attacks, and they show that the impact of an attack is not directly related to the network size, but is strongly depend on the network stability and on the bandwidth available by the malicious peers and by the source.

A network coding [26, 10] strategy called MIS (Malicious node Identification Scheme) is presented in [27] to identify and limit content pollution in P2P live streaming systems. The source splits the transmitted content in segments; each segment is then divided in blocks and each block is subdivided in codewords – that convert every segment in a matrix of elements of the Galois Field (GF). The coded blocks – that are created based on the GF matrix combining a coefficient vector to the original blocks – are the information transmitted by the source server to the peers. The peers receive the coded blocks, which are decoded to reconstruct the original segments.

In [20] the authors present a strategy to hide the identity of source servers in P2P-VoD (Video-on-Demand) networks. This is important as with that information a bad intentioned entity can direct attacks such as DDoS to those servers in order to harm the live streaming session. In [2, 23] a characterization of traffic generated by SopCast is presented. One of the conclusions is that a malicious peer was able to compromise 50% of the network peers and 30% of the download bandwidth.

In [30] the authors evaluate two of the most common attacks against P2P file-sharing networks: index poisoning and content pollution. Their analysis show that three factors have an impact on content distribution: the persistence of the original files, the false positive rate, and the initial situation of the P2P network. In [22] a method is presented to quantify content pollution in the KAD network by analysing file names and their corresponding content. A large number of files was considered and results show that 2/3 of the content is polluted.

The authors of [12] presented a survey on security and privacy aspects of P2P live streaming networks. They discuss aspects that include access control, identity management strategies and mechanisms for incentives and punishments. They show that the tree-based topology is not only vulnerable to pollution attacks but is also vulnerable to protocol failures.

Recently in [5] the authors present an evaluation of content authentication mechanisms in P2P live streaming networks. They compare the overhead and the security of several techniques and show that for high resolution streaming, the mechanisms with acceptable overheads are not resilient under pollution attacks.

6 Conclusion

This work introduced a new strategy to diagnose content pollution in P2P live streaming networks. The proposed solution employs comparison-based diagnosis to identify unauthorized changes in the transmitted content. Each peer executes comparisons of randomly selected chunks received from all its neighbors. Diagnosis is accomplished by processing results of all comparisons, i.e. from the comparison results it is possible to identify the peers that have received polluted data. The proposed solution was implemented using the Fireflies protocol. A large number of simulation experiments were performed. Results show that the overhead is only about 1.2% in terms of extra bandwidth required, and that the proposed strategy represents a feasible solution to detect content pollution in live streaming systems. Future work includes implementing the diagnosis strategy for a real Internet-based streaming service, evaluating the comparison approach for other types of overlay networks, and also extending the solution to work with several trackers in order to improve its robustness and scalability.

References

[1] A. Borges, J. Almeida, and S. Campos. Fighting Pollution in P2P Live Streaming Systems. *IEEE Intl. Conf. on Multimedia and Expo (ICME'08)*, pages 481–484, 2008.

[2] A. Borges, P. Gomes, J. Nacif, R. Mantini, J. M. Almeida, and S. Campos. Characterizing SopCast Client Behavior. *Computer Communications*, 35(8):1004–1016, May 2012.

[3] R. Chen, E. K. Lua, J. Crowcroft, W. Guo, L. Tang, and Z. Chen. Securing Peer-to-Peer Content Sharing Service from Poisoning Attacks. *Proc. of the 8th IEEE Intl. Conf. on Peer-to-Peer Computing (P2P'08)*, pages 22–29, 2008.

[4] N. Christin, A. S. Weigend, and J. Chuang. Content Availability, Pollution and Poisoning in File Sharing Peer-to-Peer Networks. *Proc. of the 6th ACM Conf. on Electronic Commerce (EC'05)*, pages 68–77, 2005.

[5] R. V. Coelho, J. T. Pastro, R. S. Antunes, M. P. Barcellos, I. Jansch-Porto, and L. P. Gaspary. Challenging the Feasibility of Authentication Mechanisms for P2P Live Streaming. *Proc. of the 6th Latin America Networking Conference (LANC'2011)*, pages 55–63, 2011.

[6] H. Deshpande, M. Bawa, and H. Garcia-Molina. Streaming Live Media over a Peer-to-Peer Network. *Technical Report, Stanford InfoLab*, (2001-30), 2001.

[7] P. Dhungel, X. Hei, K. W. Ross, and N. Saxena. The Pollution Attack in P2P Live Video Streaming: Measurement Results and Defenses. *Proc. of the Workshop on Peer-to-peer Streaming and IP-TV (P2P-TV'07)*, pages 323–328, 2007.

[8] P. Dhungel, X. Hei, K. W. Ross, and N. Saxena. Pollution in P2P Live Video Streaming. *Intl. Journal of Computer Networks and Communications (IJCNC'09)*, 1(2), 2009.

[9] E. P. Duarte Jr., R. P. Ziwich, and L. C. P. Albini. A Survey of Comparison-Based System-Level Diagnosis. *ACM Computing Surveys (CSUR)*, 43(3):22:1–22:56, 2011.

[10] C. Feng and B. Li. On Large-Scale Peer-to-Peer Streaming Systems with Network Coding. *ACM Multimedia (MM'2009)*, 2009.

[11] V. Fodor and G. Dan. Resilience in Live Peer-to-peer Streaming. *IEEE Communications Magazine*, 45(6), 2007.

[12] G. Gheorghe, R. L. Cigno, and A. Montresor. Security and Privacy Issues in P2P Streaming Systems: A Survey. *Peer-to-Peer Networking and Applications*, 4(2):75–91, 2010.

[13] M. Haridasan and R. van Renesse. SecureStream: An Intrusion-Tolerant Protocol for Live-Streaming Dissemination. *Computer Communications*, 31(3):185–192, Feb. 2008.

[14] H. Johansen, A. Allavena, and R. van Renesse. Fireflies: Scalable Support for Intrusion-Tolerant Network Overlays. *Proc. of the 1st ACM EuroSys.*, C-25, 2006.

[15] J.-S. Li, C.-J. Hsieh, and Y.-K. Wang. Distributed Key Management Scheme for Peer-to-Peer Live Streaming Services. *Intl. Journal of Communication Systems*, 2012.

[16] J. Liang, R. Kumar, and K. W. Ross. The FastTrack Overlay: A Measurement Study. *Computer Networks*, 2006.

[17] J. Liang, N. Naoumov, and K. W. Ross. Efficient Blacklisting and Pollution-Level Eestimation in P2P File-Sharing Systems. *Asian Internet Engineering Conference*, pages 173–175, 2005.

[18] E. Lin, D. M. N. de Castro, M. Wang, and J. Aycock. SPoIM: A close Look at Pollution Attacks in P2P Live Streaming. *Proc. of the 18th Intl. Workshop on Quality of Service (IWQoS'10)*, pages 1–9, 2010.

[19] T. Loocher, R. Meier, S. Schmid, and R. Wattenhofer. Push-to-Pull Peer-to-Peer Live Streaming. *21st Intl. Symp. on Distributed Computing (DISC'07)*, pages 388–402, 2007.

[20] M. Lu, P. P. C. Lee, and J. C. S. Lui. Identity Attack and Anonymity Protection for P2P-VoD Systems. *Proc. of the ACM/IEEE Intl. Workshop on Quality of Service (IWQoS'2011)*, 2011.

[21] J. Maeng and M. Malek. A Comparison Connection Assignment for Self-Diagnosis of Multiprocessor Systems. *Proc. of the 11th IEEE Fault-Tolerant Computing Symp.*, pages 173–175, 1981.

[22] G. Montassier, T. Cholez, G. Doyen, R. Khatoun, I. Chrisment, and O. Festor. Content Pollution Quantification in Large P2P Networks : A Measurement Study on KAD. *IEEE Intl. Conf. on Peer-to-Peer Computing (P2P'2011)*, pages 30–33, 2011.

[23] J. Oliveira, A. Borges, and S. Campos. Content Pollution on P2P Live Streaming Systems. *Proc. of the 15th Brazilian Symposium on Multimedia and the Web (WebMedia'09)*, 2009.

[24] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, A. E. Mohr, and E. E. Mohr. Chainsaw: Eliminating Trees from Overlay Multicast. *Proc. of the 4th Intl. Workshop on Peer-To-Peer Systems (IPTPS'05)*, pages 127–140, 2005.

[25] K. Walsh and E. G. Sirer. Experience with an Object Reputation System for Peer-to-Peer Filesharing. *Proc. of the 3rd USENIX Symp. on Networked Systems Design and Implementation (NSDI'06)*, 3, 2006.

[26] M. Wang and B. Li. Lava: A Reality Check of Network Coding in Peerto-Peer Live Streaming. *Proc. of the 26th IEEE Intl. Conf. on Computer Communications (INFOCOM'2007)*, 2007.

[27] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana. MIS: Malicious Nodes Identification Scheme in Network-Coding-Based Peer-to-Peer Streaming. *Proc. of the 29th IEEE Intl. Conf. on Computer Communications (INFOCOM'2010)*, pages 1–5, 2010.

[28] C. K. Wong and S. S. Lam. Digital Signatures for Flows and Multicasts. *IEEE/ACM Trans. on Networking*, 7(4):502–513, 1999.

[29] 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. *Proc. of the 37th Intl. Conf. on Parallel Processing (ICPP'08)*, pages 652–659, 2008.

[30] P. Zhang and B. E. Helvik. Modeling and Analysis of P2P Content Distribution under Coordinated Attack Strategies. *IEEE Consumer Communications and Networking Conference (CCNC'2011)*, pages 131–135, 2011.

[31] R. P. Ziwich, E. P. Duarte Jr., and L. C. P. Albini. Distributed Integrity Checking for System with Replicated Data. *Proc. of the 11th IEEE Intl. Conf. on Parallel and Distributed Systems*, pages 363–369, 2005.