IET Communications

Research Article

Distributed mitigation of content pollution in peer-to-peer video streaming networks

ISSN 1751-8628
Received on 18th June 2019
Revised 27th November 2019
Accepted on 23rd March 2020
E-First on 15th May 2020
doi: 10.1049/iet-com.2019.0627
www.ietdl.org

Roverli P. Ziwich[1], *Elias P. Duarte Jr*[1] ✉, *Glaucio P. Silveira*[2]
[1]*Federal University of Parana (UFPR), Department of Informatics, P.O.Box 19018, Postal Code 81531-980 – Curitiba, PR, Brazil*
[2]*Federal University of Parana (UFPR), Electronic Computing Center, P.O.Box 19037, Postal Code 81531-980 – Curitiba, PR, Brazil*
✉ *E-mail: elias@inf.ufpr.br*

Abstract: Video streaming has become increasingly popular in the Internet. Frequently, video transmissions are based on peer-to-peer networks, in which peers running on end-user hosts transmit data among themselves. An important security vulnerability of this strategy is that content can be easily altered by malicious users. Thus, it becomes essential to diagnose and fight content pollution in these systems. In this work, the authors present a novel strategy that relies on comparison-based diagnosis to mitigate content pollution in live video streaming peer-to-peer networks. This strategy is fully distributed and effectively combats the dissemination of content pollution. In the strategy, peers independently identify and avoid polluters. The solution works on top of the scalable overlay network Fireflies. Experimental results are presented showing the effectiveness and the low overhead of the solution. In particular, the strategy was able to significantly reduce content pollution propagation in diverse network configurations.

1 Introduction

Video streaming has become one of the most popular Internet applications, to the point that it corresponds to the majority of the Internet traffic [1]. If on one hand users expect to get high-quality image and sound, on the other hand this is one of the most demanding applications from the point of view of infrastructure providers, consuming high bandwidth and requiring a well-behaved network in terms of delay and jitter. Placing video sources closer to users is a way to deal with those issues. Peer-to-peer (P2P) networks have been shown to be able to sustain millions of registered users – e.g. [2–5]. Content delivery networks (CDNs) are prevalent today [6], including those that take advantage of peers on the edge to bring content closer to users [7].

In P2P video streaming networks, a *source* server generates the stream, which consists of a sequence of *chunks*, which are distributed the peers themselves. In contrast to the traditional client-server paradigm, live streaming network peers not only reproduce the video, but also obtain the video chunks from each other. Thus, the demand on the source servers decreases. These systems present several challenges, of which one of the most important is *churn*, which corresponds to the dynamic nature of the network with peers joining and leaving the system continuously. As the peers themselves transmit data, malicious peers represent a significant threat and a formidable challenge to identify and mitigate. A malicious peer can harm the system in different ways, one of which is content pollution [8–11], which we describe next.

Content pollution is defined as a malicious fault by which the original video content is modified by a peer that does not have the authorisation to do that. Different types of modifications are possible, such as modifications of the original data, the insertion of new data, and also omission and data destruction [12–15]. If the P2P system does not deploy any strategy to fight pollution attacks, the transmission can be seriously harmed even if the number of malicious peers is low [16–18].

Among the multiple strategies that have been proposed to handle content pollution in P2P networks, some assume chunk integrity is guaranteed with the use of hashes [19]. Although this is an effective way to handle message corruption on a communication channel, in live streaming transmissions it is a challenge for the peers to obtain the hashes of forthcoming chunks. Furthermore a peer can maliciously modify chunks and transmit correct hashes for polluted chunks, deceiving other peers.

The use of black lists containing ranges of IP addresses [20] has also been investigated to control content pollution in the context of P2P live streaming. The main concern with this approach is related to the difficulty of building comprehensive lists and also dealing with the fact that malicious peers can assume new addresses not black listed. In [21] the authors designate a group of peers to maintain the integrity of content transmitted by the source; in this scheme some selected peers can confirm the integrity of transmitted chunks. The traditional file-sharing reputation and ranking strategies have been also applied to live streaming [22, 23] challenges include the collusion of malicious peers and the delay to propagate conclusions. Still other approaches to deal with content pollution in P2P networks apply cryptography to the whole transmission, at a very high computational cost [21, 24].

In this work we present a novel strategy to mitigate content pollution propagation in live streaming P2P networks. The proposed strategy is fully distributed and leverages comparison-based diagnosis [25] to detect unauthorised modifications of the transmitted content. Through comparison-based diagnosis, each peer independently identifies and stops requesting chunks from those neighbours identified as polluters.

Comparison-based diagnosis [25] was originally proposed to identify the states of the units of a distributed system as either *faulty* or *faulty-free*. This is part of a broader field called system-level diagnosis, with several important results in terms not only of system monitoring strategies but also on their complexity and theoretical limits. System-level diagnosis is based on tests that are executed among system units. A test intrinsically depends on the specific technology of the system, for it allows the identification of faulty units in the sense that their behaviour deviates from the corresponding specification. Note that this is exactly our definition of malicious fault above.

The first system-level diagnosis models employed tests executed on a single unit, and then evolved in several directions including comparison-based diagnosis. In comparison-based diagnosis a single task is sent to be executed by two different units, the outcomes that result are then compared. Intuitively, it is easy to see that if the task outcomes are different, then there is a problem. Note that the comparator itself may be faulty, and in this case it may lie and its comparison results are not reliable. Diagnosis is

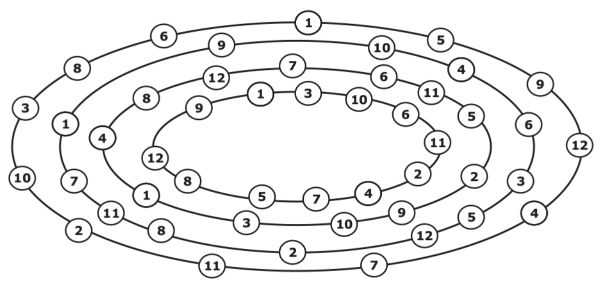

Fig. 1 *An example Fireflies topology with 12 peers organised in 4 rings*

accomplished by processing the set of all comparisons executed across the whole system, which is called the system *syndrome*.

In this work we present a strategy to combat pollution in P2P live streaming networks that relies on comparison-based diagnosis. Peers are first identified as polluters, so that neighbours can stop obtaining and disseminating chunks received from them. Peers form a Fireflies scalable overlay network [26, 27] which presents a mesh topology [28] on which peers employ a pull-based strategy to exchage video chunks. Experiments are presented that show the overhead of executing comparisons. Results confirm that the overhead is low and the proposed strategy is effective in mitigating pollution. In particular, the solution was able to nearly eliminate the propagation of pollution in many transmissions.

Next in this work, in Section 2 we briefly introduce live streaming in P2P networks including Fireflies, and also comparison-based diagnosis. Section 3 presents the distributed solution to combat content pollution propagation. Section 4 presents the implementation and experimental results. Then Section 5 presents related work and finally the concluding remarks are in the last section.

2 Preliminary definitions

In live streaming P2P networks the source is a server that generates content which is divided in *chunks*. The source server is responsible to send the chunks to a limited number of *peers* which continue the dissemination. Peers both consume and exchange the chunks among themselves.

The most popular P2P live streaming systems employ the *mesh* topology with the pull-based strategy to transmit data in which peers make explicit chunk requests [28]. In order to receive a particular chunk, a peer needs to keep a list of the data available at all its neighbours, and then sends requests for specific chunks.

The next two subsections give brief description of the Fireflies network overlay protocol and comparison-based diagnosis.

2.1 The fireflies overlay

Fireflies [26] is a scalable intrusion-tolerant P2P overlay network which organises peers in mesh for content distribution. A pull-based strategy is used for peers to obtain data chunks from their neighbours in the topology. Peers are organised in a mesh topology. Fireflies assumes a source which is a server that generates the content which is transmitted in the network. This server is assumed to be permanently correct and thus never becomes faulty.

The source generates and sends the content divided chunks to some peers. Those peers then retransmit the chunks to their neighbours and so on. The purpose is that all peers eventually receive all chunks. Each peer is assigned an identifier, peers are on a virtual topology that consists of several rings [26]. Note that all peers are in all rings. The number of rings is is specified as a parameter, λ. The peers communicate with neighbours on the diverse rings. In this way each peer is a neighbour of at the minimum only two other peers (if by chance the same peers are neighbours in all rings) and at most $(2 * \lambda)$ neighbours (in this case the neighbours in different rings are all different).

A Fireflies example topology is shown in Fig. 1. The virtual topology in this case consists of twelve peers placed across four rings. Node 1 has as neighbours peers 3, 4, 5, 6, 7 and 9. A repetition of neighbours occurs as nodes 3 and 9 are neighbours of peer 1 in two rings. The source is the node with identifier 0, note that the source does not take part of any ring. The number of peers that receive chunks from the source is a parameter, and the those peers are selected randomly.

Fireflies employs a pull-based strategy for peers to exchange chunks. At any instant of time, each peer keeps the list of chunks it has available to send to its neighbours in the *availability window*. Analogously, each peer also keeps an *interest window* in which the needed chunk list is maintained, i.e. the list of chunks that the peer still has to obtain. After a chunk is received by a specific peer, it sends a notification to all its neighbours so that they become aware that the chunk is available. In this way each peer can keep a list of all chunks that are available at all its neighbours. Thus if chunk c is available at a peer j that is a neighbour of peer i which has c in its interest window, then i makes a request for c and sends this request to j. As the request is received by neighbour j it first checks whether the chunk is in its local availability window, and in case it is the chunk is sent back to the requesting peer. If chunk c is not in the availability window of peer j then the request from neighbour i is ignored. The same procedure that is adopted by the source, so that after a fresh chunk is generated, the server sends a notification of the availability of that chunk to its neighbours, after that the neighbours will make requests and the chunk will propagate across the network.

2.2 Comparison-based diagnosis

System-level diagnosis is a strategy to identify the state of the units of a given system as *faulty* or *faulty-free* [25, 29]. The system units execute tests among themselves, according to a test assignment. A test consists of a procedure sent by the tester and received and executed by the tested unit, which sends back the outcomes to the tester. Based on the outcomes, or the total lack of response, the tester classifies the tested unit as fault-free or not. The result of all tests executed is called the system syndrome, and diagnosis completes by processing the syndrome to extract information on which nodes have become faulty. Comparison-based diagnosis is a system-level diagnosis strategy which, instead of tests, relies on the comparisons of task outputs produced by the execution of the same task by pairs of units.

Several different comparison-based diagnosis models have been proposed in the past decades. Maeng and Malek in [30] introduced the so-called MM model in which graph $G = (V, E)$ is employed to represent the system. The set of units, which correspond to processing nodes, is represented by the set of vertices V. The set of edges E represents the links, which correspond to either physical or logical connections between the processing units, often simply representing the fact that a unit can communicate directly with another unit without having to go through an intermediary. In traditional comparison-based diagnosis a single task is send to a pair of units, that receive and execute the task, and send back the outputs to be compared by the comparator or tester. In the MM model unit k can only test a pair of 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 k is fault-free, whenever the comparison of the outcomes produced by i and j results in a match, then it is possible to conclude that both tested units are fault-free. On the other hand, whenever the comparison results in a mismatch, the conclusion is that at least one of the units is faulty: i, j and/or k.

It is important to highlight that the MM model assumes that if one or both of the units that execute the task are faulty, then they always produce different outputs for the task they execute. Furthermore if the tester is faulty then the comparison is unreliable, in the sense that the results can have been manipulated so that they may not correspond to reality. A central observer is also assumed that receives the results of the comparisons executed by all testers. The central observer completes the diagnosis after having gathered the system syndrome, i.e. the whole set of comparisons executed throughout the system. The MM* model was derived from the MM model and assumes that each and every unit is a tester of all its neighbours.

The generalised comparison-based diagnosis model [31] eliminates the central observer by allowing the units themselves to receive and process the syndrome, thus completing the diagnosis.

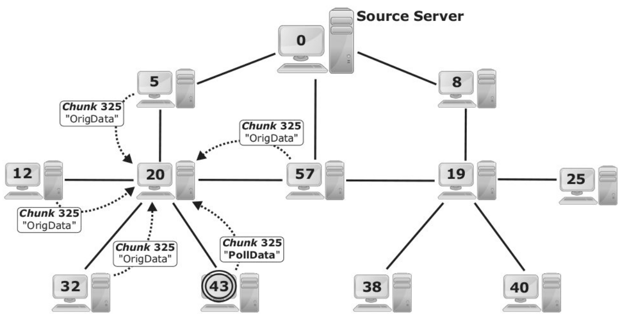

Fig. 2 *Peer 20 receives requested chunk 325 from its neighbours; peer 43 is a polluter*

Furthermore, this model assumes that two faulty units can return exactly the same outcome for a given task, i.e. the comparison of two faulty units can be a match.

The present work presents a solution that employs tests that consist of comparing chunks received from pairs of peers. The work assumes the generalised comparison-based diagnosis model introduced in [31]. If there is no pollution, the comparison results in a match. However, as in P2P live streaming systems two peers may have the same polluted version of a given chunk, it is important to allow a pair of faulty units to produce and return the same output.

3 The proposed strategy to mitigate pollution in P2P live streaming networks

This section presents the strategy that allows peers to combat content pollution propagation in live streaming overlay networks. The solution relies on comparison-based diagnosis to detect pollution and polluters and is implemented on the Fireflies protocol. It is important to highlight that the proposed solution is fully distributed, i.e. each peer independently performs the necessary actions to stop requesting chunks from neighbours considered to be polluters.

The proposed solution employs a *comparator module*, besides the source server and the peers. This is a component that executes integrated to the Fireflies code at every peer. The comparator module has access to the chunks received by the peer as well as to its availability window. The comparator compares particular chunks, which are selected randomly, considering a configurable *monitoring interval*. This interval indicates the maximum time in which every comparator module has to choose and compare a random chunk from all its neighbours.

Each peer i performs the procedure described next. As soon as peer i receives a chunk with identifier cid from neighbour j, peer i checks whether this was one of the chunks randomly chosen to be compared. In case it is, the comparator module executing at peer i requests chunk cid from all its neighbours as soon as they notify the availability of that chunk. It is important to highlight that the additional requests performed by the comparator modules are regular Fireflies requests, sent by the peers themselves and using the Fireflies system itself. Note that neighbours can be polluters thus the request for a chunk to compare must be exactly equal to the request of a regular chunk to play, otherwise the malicious peers can identify that a chunk is being requested for comparison and can easily deceive the requester by sending the correct chunk. In other words, a malicious peer that receives a request sent by a comparator module will not handle that request in a different way.

As soon as peer i requests and receives the responses of the requested chunk cid from each of its neighbours, peer i compares the received chunks in pairs, and according to the comparison results classifies its neighbours in the $U_{i,\mathrm{cid}}$ set. Set $U_{i,\mathrm{cid}}$ has the following format:

$$U_{i,\mathrm{cid}} = \{(\mathrm{chunk}_a, \{\mathrm{peer}_j, \mathrm{peer}_k, \dots\}), \quad (\mathrm{chunk}_b, \{\mathrm{peer}_m, \mathrm{peer}_n, \dots\}), \dots\}.$$

As each peer has a time limit to send a reply to every chunk request, if a given neighbour j does not reply within that expected time limit, peer i classifies peer j in a specific subset of $U_{i,cid}$ which contains the identifiers of all peers that have not replied. It is also important to highlight that the maximum time that a given peer waits for a reply is the same amount of time configured for the interest windows, i.e. it is the same amount of time that any peer normally waits for any reply in the system. Furthermore the solution assumes that the majority of the neighbours of a peer are not polluters.

As soon as each peer i completes the $U_{i,cid}$ set, i.e. when the set contains all neighbours of peer i, the following procedure is executed, in order to maintain a so called *list of blocked peers*. Peer i checks whether there is a subset of $U_{i,cid}$ with size greater than $N(i)/2$, where $N(i)$ is the number of neighbours of peer i. In case all subsets are smaller than $N(i)/2$, the proposed solution does not take any action and does not modify the list of blocked peers. Otherwise peer i resets the list of blocked peers to include all peers *not* in the largest subset. Note that neighbours that are kept in a *list of blocked peers* include both malicious peers that generate pollution and also victims peers that receive and propagate polluted content.

Whenever the list of blocked peers is not empty, peer i ignores every received chunk and availability notifications from all peers in the list. Note that this list is constantly updated as soon as the comparator module updates a new $U_{i,cid}$ set. As this list of blocked peers is periodically updated, the proposed solution allows the rehabilitation of previously blocked peers. In other words, if a blocked peer changes its behaviour, it can be eventually removed from the list of blocked peers. In particular, victims that just receive and propagate polluted content also execute the algorithm, so they eventually will stop obtaining and propagating polluted chunks.

In synthesis, the procedure is as follows (the algorithm in pseudo-code also is presented below):

- A peer obtains a chunk from a neighbour.
- Some of the chunks are randomly selected to be compared.
- In case a chunk is to be compared, the peer requests it from all neighbours.
- The peer builds set $U_{i,\mathrm{cid}}$ with the chunk versions received.
- The valid chunk is the one received from a majority of neighbours, the other neighbours are classified as blocked peers;
- As a new chunk is selected to be compared, both set $U_{i,cid}$ and the set of blocked peers are reset as empty.

As an example, Fig. 2 shows the requests issued by a peer running the proposed solution. In this example peer 20 chooses the chunk with identifier 325 to be compared. The neighbours of peer 20 in this example are peers 5, 12, 32, 43 and 57. As soon as these neighbours notify peer 20 about the availability of chunk 325, the comparator module of peer 20 will send the request for that chunk to its neighbours. In this figure the arrows represent the replies with chunk 325 from the neighbours of peer 20; the undirected edges represent the communication links between the peers and the source server. Moreover, in this example the original content of chunk 325 is shown as 'OrigData' and a polluted version of the same chunk is shown as 'PollData'. This example also considers that only peer 43 is malicious and that in this moment peers 5, 12, 32 and 57 have a non-polluted copy of chunk 325.

As soon as the comparator module of peer 20 receives information about chunk 325 from all its neighbours, peer 20 will perform the comparison of those chunks, in pairs, and will classify all its neighbours in the $U_{20,325}$ set. For this example, the U set generated is $U_{20,325} = \{\,(\mathrm{OrigData}, \{5, 12, 32, 57\}), (\mathrm{PollData}, \{43\})\,\}$. As soon as set $U_{20,325}$ is complete, peer 20 is able to identify that there exists a subset with more than half of its neighbours, as subset $(\mathrm{OrigData}, \{5, 12, 32, 57\})$ has $4 > (5/2)$ peers, which is the majority of peers. From this moment, peer 20 stops requesting chunks from peer 43.

Fig. 3 presents the algorithm of the comparator module which is responsible to issue requests, execute comparisons and finally update the list of blocked peers.

```
Algorithm: $ComparatorModule$ /* executed at peer $i$ */
1: begin
2:   $list\_of\_cids \leftarrow$ randomly choose chunks to be compared;
3:   Thread 1: whenever a neighbor $j$ notifies the availability of a
4:             new chunk with identifier $cid$ do
5:     if $cid \in list\_of\_cids$ then
6:       make a request to obtain the chunk with $cid$ from $j$;
7:       update $U_{i,cid}$;
8:     end if
9:   end whenever
10:
11:  Thread 2: whenever (peer $i$ completes set $U_{i,cid}$) or
12:    (timeout to obtain chunk $cid$ from its neighbors arrived) do
13:    if timeout expired then
14:      include all neighbors that have not answered
15:                                in a specific set of $U_{i,cid}$;
16:    end if
17:    if ($U_{i,cid}$ has a subset with more than $N(i)/2$
18:                                           neighbors) then
19:      $list\_of\_blocked\_peers \leftarrow \emptyset$; /* cleanup the list */
20:      $list\_of\_blocked\_peers \leftarrow$ {peers from $U_{i,cid}$ except
21:                                 peers from largest subset};
22:    end if
23:
24:    $list\_of\_cids \leftarrow$ choose new chunks to compare;
25:  end whenever
26: end
```

Fig. 3 *The comparator algorithm*

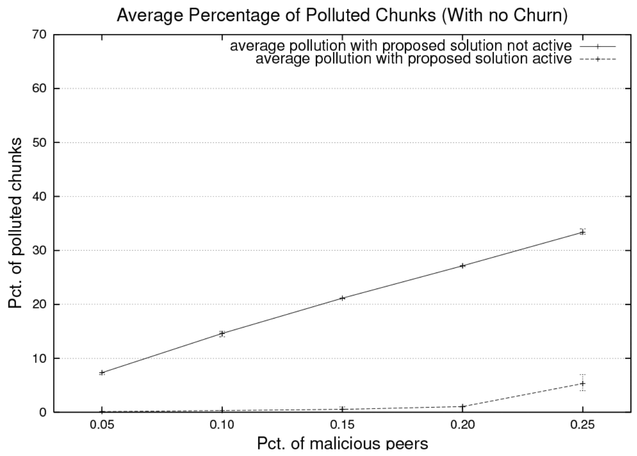

Fig. 4 *Average percentage of polluted chunks; experiments with no churn*

Initially (in line 2) the comparator selects the chunks to compare. The identifiers of these chunks are selected randomly using the following procedure: *next_cid_to_compare* ← *last_cid_chose* + *mod*(*random*, (*monitoring_interval* ∗ *mcast_rate*)), where *random* represents an aleatory number, *monitoring_interval* is the maximum time (in seconds) configured as the monitoring interval, and *mcast_rate* is the number of chunks generated by the source at each second. In other words, the next chunk to be compared is a chunk generated by the source in one of the next *monitoring_interval* seconds after the instant the source selected last chunck *last_cid_chose*.

As an example, if *monitoring_interval* = 15 and considering *mcast_rate* = 30 and *last_cid_chose* = 5674, the value of the *next_cid_to_compare* will be a random number between 5674 and (5674 + (15 ∗ 30)). As soon as the identifiers of the next chunks to be compared are chosen, they are inserted in the *list_of_cids*.

The comparator module then waits for new chunk availability notifications from its neighbours. Whenever any neighbour *j* notifies the availability of a new chunk cid (line 3), peer *i* executes lines 5–8: if cid is the identifier of a chunk to be compared, peer *i* requests the chunk from neighbour *j* (line 6). As soon as peer *i* receives chunk cid from peer *j* set $U_{i,cid}$ is updated (line 7) and peer *j* is classified accordingly.

Lines 11–25 are executed whenever set $U_{i,cid}$ is complete, i.e. whenever peer *i* has already received replies from all its neighbours or the time limit to obtain the replies has elapsed. In this last case, the peers that have not sent replies are classified in a specific subset (line 14). Line 17 then verifies if the corresponding $U_{i,cid}$ set has a subset with more than $N(i)/2$ peers that have returned chunk cid. In this case, the comparator module updates the *list_of_blocked_peers* (lines 19–21).

Finally, each peer must perform one additional and simple test: every time node *i* receives a notification from neighbour *j* of the availability of a new chunk, peer *i* must verify if peer *j* is in its *list_of_blocked_peers*. If it is, peer *i* simply ignores that notification. Otherwise peer *i* executes the regular steps according to the Fireflies protocol.

4 Experimental results

In this section we present an empirical evaluation of the proposed solution to mitigate pollution in P2P live streaming networks. As mentioned earlier, the strategy works on top of a Fireflies virtual topology and it was implemented with the Fireflies simulator presented in [27]. In terms of hardware, we employed an AMD Phenom 9500 quad-core x64 processor with 4GB of RAM. The operating system was the Linux 64-bit kernel version 2.6.18-238.el5.

Each experiment consisted of a live streaming session of 180 s. The source server generated 30 chunks/s. In this way, we are simulating aggressive subsecond chunks so that the results can also be applied to different systems based on a variety of codecs and bitrates, including those that employ larger chunk sizes. Fireflies organised the neighbourhood of each peer using 15 rings for networks with 200 peers. Both the availability and interest windows were configured to keep 3000 chunks. Moreover, in all experiments the monitoring interval was set to 15 s, i.e. at most at every 15 s the comparator modules of each peer randomly chose a chunk to be compared.

The objective of the empirical evaluation is to (i) estimate the impact of content pollution in transmissions for different percentages of malicious peers and (ii) compute the cost of the proposed solution in terms of the extra bandwidth required, i.e. the percentage of additional chunks transmitted in the network. The main parameters varied in the simulations were the following:

(i) The number of polluters computed as a percentage of the total number of peers: 0, 5, 10, 15, 20 and 25%.

(ii) Experiments were executed with and without churn. For transmissions with churn, during the transmission 100 peers joined and 100 peers left the network. New peers joined the system according to a normal distribution with an average of 100 and standard deviation of 20. Peers left the network following a Poisson distribution with an average equal to 100. It is also important to highlight that the churn was configured to not remove malicious peers from the network.

(iii) The comparator module also can be turned on and off in order to measure the effect of the pollution on systems with and without the proposed solution.

Each experiment was repeated 1000 times. Results are shown in Figs. 4–11, and show averages and also the 95% confidence interval.

The average number of polluted chunks disseminated during a transmission is shown in Fig. 4. Curves are shown for the system with and without our proposed strategy. It is possible to note that in experiments with no churn and in which 20% of peers in the networks were configured as polluters, the proposed solution was able to reduce the average percentage of polluted chunks in the transmissions from 27.1 to 1%. For the experiments in which 25% of the peers were polluters, the average percentage of polluted chunks was reduced from 33.3 to 5.3%.

Next, Fig. 5 shows the average percentage of polluted chunks measured in the experiments with churn. The percentage of polluted chunks was reduced from 45.7 to 7% in average, for experiments with 20% of malicious peers, and was reduced from 52.7 to 16% in experiments in which 1/4 of the peers were malicious.

The next two figures – Figs. 6 and 7 – show the percentage of polluted chunks transmitted per second in the experiments without the proposed solution: Fig. 6 summarises experiments with no churn and Fig. 7 show results for experiments with churn. Based on

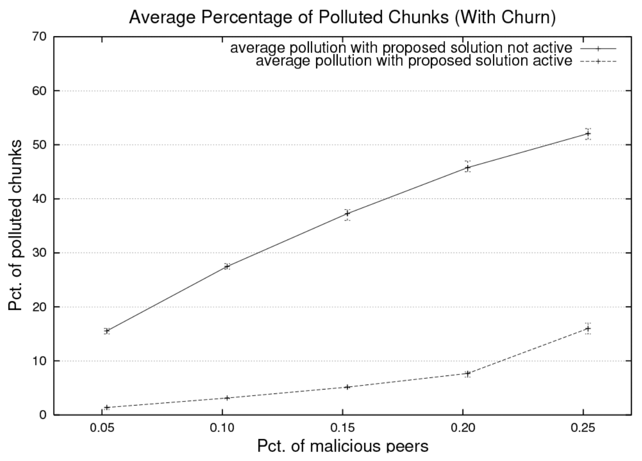

Fig. 5 *Average percentage of polluted chunks; experiments with churn*

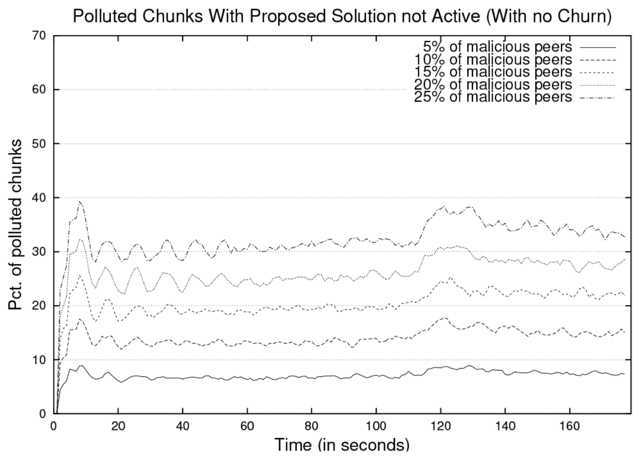

Fig. 6 *Polluted chunks transmitted per second, without the proposed solution; experiments with no churn*

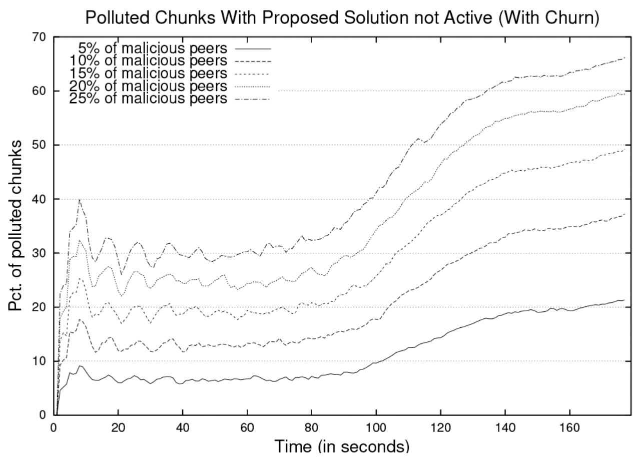

Fig. 7 *Polluted chunks transmitted per second, without the proposed solution; experiments with churn*

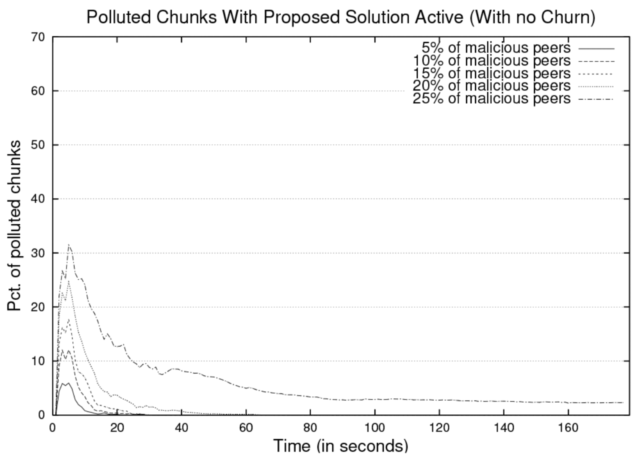

Fig. 8 *Polluted chunks transmitted per second, with proposed solution active; experiments with no churn*

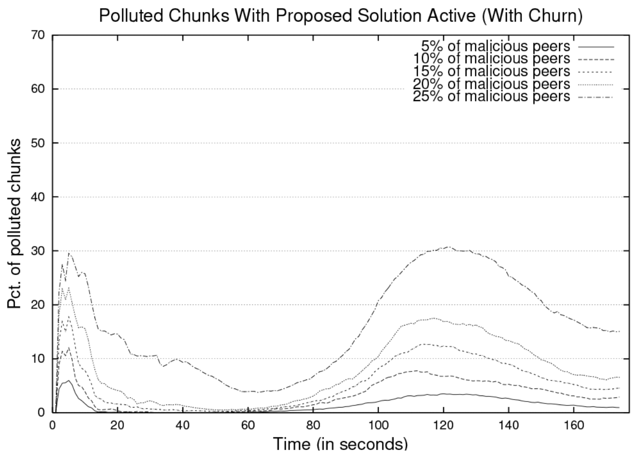

Fig. 9 *Polluted chunks transmitted per second, with proposed solution active; experiments with churn*

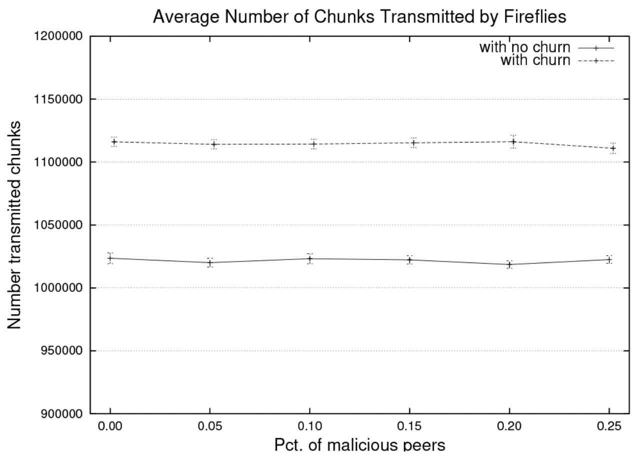

Fig. 10 *Chunks regularly transmitted by Fireflies*

Fig. 6 it is possible to see that in transmissions with no churn the number of polluted chunks corresponds to a different percentage for each different number of malicious peers, but in general each curve presents a small dispersion. On the other hand, Fig. 7 shows that as the simulation reached half of the transmission time – the moment with the highest churn rates – the percentage of polluted chunks starts to increase, reaching values close to 70% of polluted chunks in transmissions with 25% of malicious peers. It is important to remember that churn was configured to not remove malicious peers from the network.

The next two figures – Figs. 8 and 9 – also show the percentage of polluted chunks per second, but now with the proposed solution active: Fig. 8 shows results for the experiments with no churn and Fig. 9 shows results for experiments experiments with churn. Based on Fig. 8 it is possible to see that in transmissions with the proposed solution active and with no churn, the solution nearly eliminated the pollution after 40 s of transmissions for simulations with 20% of peers configured as polluters. In the presence of 25% of malicious peers the percentage of polluted chunks also dropped to about 2% after 80/,s of transmission. The reason that in this latter case the pollution was not completely eliminated is that the majority of the neighbours of some peers were polluters. Note that as the percentage of malicious peers increases, the probability that the majority of the neighbours of a given peer are polluters also increases.

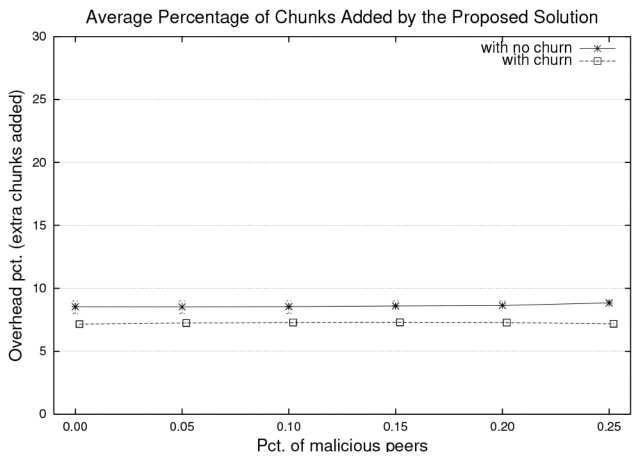

Fig. 11 *Overhead in terms of the extra chunks generated*

Fig. 8 also shows that in general the percentage of polluted chunks was higher during the first seconds of the transmissions. After peers started completing their first comparisons they stopped requesting chunks from those peers identified as polluters. Moreover, if the number of polluted chunks transmitted in the first seconds were not considered, the average percentage of pollution during the whole transmission – previously shown in Fig. 4 – would be even lower.

Fig. 9 shows the number of polluted chunks transmitted per second for experiments with the proposed solution active and also with churn. During the time interval when the highest number of peers joined the system (between time instants 100 and 120) the percentage of polluted chunks transmitted increased. The pollution increased because each new peer that joins the network does not have any knowledge about which of its neighbours are polluters, i.e. its list of blocked peers starts empty. On the other hand, as these peers perform their first comparisons, the percentage of polluted chunks starts to drop again. In this sense, the experiments with no churn – shown in Fig. 8 – illustrate the general behaviour of a new peer that joins the overlay: as they start learning which neighbours are sending polluted chunks they avoid those neighbours, reducing the percentage of polluted chunks in the system.

In terms of the performance impact of the comparator, note that each comparison is actually a simple string comparison, which is linear and can be executed in the order of a few microseconds on current processors. Thus the time to obtain the chunks from the neighbours, typically in the order of milliseconds, is the dominant component, but the comparator module remains idle during this time. Furthermore it also remains idle from the time a comparison is completed until the next chunk is selected for comparison. In this way, the frequency in which the comparisons are executed can be adjusted to meet performance requirements. Actually, the major overhead imposed by the comparator module is in terms of bandwidth, i.e. the extra chunks obtained from the neighbours for comparison. Next we evaluate this overhead.

The average number of chunks generated by all nodes in the Fireflies topology is shown in Fig. 10. These are the results of simulations executed both with and without churn, and without our solution for pollution mitigation running. The average number of chunks sent by Fireflies was about 1 and 1.15 million chunks. Now, Fig. 11 shows the percentage of extra chunks generated by our solution. It is possible to see that the overhead of the proposed solution is about 7 to 8% of extra chunks. As a final note, these results employed a 15 s monitoring interval, which can be increased or decreased, with a lower or higher impact on bandwidth consumption respectively.

5 Related work

In [32], Gheorghe, Cigno and Montresor present a survey of the non-trivial and multiple security and privacy issues of P2P live streaming networks, including problems and solutions for access control, identity management, and incentive/punishment mechanisms. The authors highlight that tree topologies are particularly vulnerable to pollution attacks.

Recently proposed strategies to deal with content pollution in P2P video streaming networks include employing band codes [11, 12]. The authors first highlight how hard it is to identify polluters, as besides malicious peers a potentially large number of other peers involuntarily relay the polluted chunks. Band codes are a family of rateless network codes that employ adjustable window sizes, so that the probability that honest nodes relay polluted packets is reduced. The identification of the malicious nodes in the network is described as follows. Each peer counts the number of times that it has obtained polluted chunks from each neighbour. Peers exchange the counters and so that each peer can compute for each neighbour a score of ‘honesty’, that estimates the probability that the neighbour does not retransmit polluted chunks. This probability is modelled as a function of the band code window size. The authors show the accuracy and effectiveness of the strategy for building distributed black lists of polluters. In an earlier version [13] the authors employed a reference node to collect chunks and do the classification using the following criteria: earlier chunks are less likely to be polluted, and preferring chunks that are retransmitted less times helps cleaning pollution.

Feng and Li present in [33] another strategy based on network coding that allows the identification and prevents the propagation of content pollution in P2P live streaming networks. The content is sent in segments by the source, which are further broken into blocks and those in codewords. In this way each segment is converted to a matrix of elements of the Galois Field (GF). The blocks are coded using the GF matrix combining a coefficient vector to the original blocks, and these coded blocks are received by the peers from the source. Coded blocks must be later decoded to reconstruct and the reproduce the original content.

One of the earliest and most popular solutions to prevent pollution is to keep polluters on Black Lists [20]. Actually instead of individual addresses, ranges of IP addresses are stored that include the addresses of polluters – actually of peers that have disseminated polluted content, which can be malicious or not. The idea is to reduce mistakes by minimising addresses of non-polluters. Black Lists can be expensive to keep [32] and present several challenges, perhaps the most obvious is the fact that malicious peers can keep changing to new addresses that have not been stored in the lists.

BitTorrent [17] uses a simple centralised technique: a peer obtains hash-based signatures [34] for all chunks in advance. In this way when a chunk is received, the peer can check its integrity. This technique is efficient to handle corruptions in the transmission channels. Although this strategy works well for stored data it cannot be used for live streaming, as it is impossible to generate all hashes for all the data in advance, for the content is generated during the transmission itself.

Another similar yet even more extreme technique is to distribute chunks digitally signed by the source [27]. Besides also being unfeasible for live streaming, the overhead of adding signatures to chunks as they are generated also represents a significative performance impact. Merkle-trees [34] have also been used to enforce the integrity of streaming systems with hashing. A Merkle-tree codifies hashes computed for sequences of chunks of a sequence. The hashes are tree leaves, the value associated with an internal tree node depends of its sons hashes. The tree guarantes the integrity of chunks in the context of the chunk sequence. Merkle trees have been compared [35] to several other strategies such as using hashes, cryptography, signatures and Black Lists: they present the best performance in terms of computational overhead.

Other strategies to secure the streaming against unauthorised modifications is to encrypt the complete chunks using cheaper symmetric cryptography methods [21, 24]. The unavoidable problem in this case is to share the predefined secret key with all end-users – excluding the malicious ones. A strategy to all the source to periodically recompute and share a new secret key with a small number of trusted peers is presented in [21].

In [10] the authors focus on content validation. Besides evaluating the cost to do the validation, they describe techniques to reduce spreading polluted content, and which are resistant to collusion. Earlier, in [36] the authors presented another solution based on peer groups responsible to guarantee chunk integrity. In this way the source sends chunks only to group peers, other peers have to check chunk integrity by accessing the group.

In [37] an evaluation of the authentication of content spread in P2P high resolution live streaming systems is presented. The conclusion is that the strategies that presented the best performance were not effective under pollution attacks. In [38] an evaluation of pollution attacks reaches the conclusion that the size of the network has less influence on the impact of an attack then network stability and the bandwidth available for both the source and malicious peers.

The authors of [39, 40] present solutions based on reputation and ranking applied to P2P systems devoted to file sharing. A peer is able to assess whether other peers are honest, and can obtain content from those peers. The authors of [23] emphasise that reputation systems can suffer with (i) the collusion of malicious peers, (ii) with false positives and (iii) with the delay to propagate conclusions. Nevertheless reputation-based defense mechanisms are considered to be some of the most effective and practical solutions to deal with malicious peers. Another strategy based on ranking is proposed in [22]. These solutions employ reputation mechanisms based on the experience of each peer, which peers communicate among themselves. In [15] the authors present a framework to simulate different variations of the reputation strategies, from centralised to distributed approaches. Results lead to the conclusion that the distributed strategy is the best given the dynamic nature of peer-to-peer networks.

In [41] the authors propose a scheme to detect content pollution by implementing retransmission requests of the polluted data in live streaming transmissions. To detect the pollution, a confidence management strategy is proposed. One disadvantage of this solution is that the number of retransmissions can be high and the solution is also highly depended on the delay for propagating reputation information.

In [9] a strategy is proposed based on an overlay to monitor and isolate content pollution in video-on-demand (VoD) P2P networks. Earlier, in [42] the authors presented a strategy to hide source identity in these networks. The purpose is to avoid DDoS attacks directed at the sources, which can completely destroy live streaming sessions. In [43] a thorough evaluation of the SopCast P2P video streaming system is presented that shows that a single polluter can contaminate the contents of 50% of peers and consume up to 30% of the available bandwidth.

Our previous work [44] describes a non-distributed strategy proposed to perform the identification of polluted peers in live streaming networks.

In [14] the authors propose a novel attack, which although not presented specifically in the context of pollution can certainly be extended in that direction. The attacker places malicious peers in prominent positions, so that the impact they cause on the network is devastating. By dropping chunks that should be forwarded (or injecting pollution on chunks) the malicious peers can be very effective and at the same time they avoid suspicions. A detection mechanism that identifies the attack and removes potential malicious peers from their disruptive positions is also proposed. In another work, the same authors consider the outgoing eclipse attack which blocks the outgoing connections of honest super peers. A mitigation strategy is proposed which is shown to be effective by simulation.

Other recent work includes an analytical model of pollution attacks that allows the estimation pollution in the context of live streaming in P2P networks [8]. The model takes into account user behaviour. The accuracy was empirically evaluated with real attack traces. Yet another model is presented in [45], where the authors investigate whether it is possible to combat pollution by simply using trusted peers. They show that the number of trusted peers required is at least the number of malicious peers. They present preliminary results on a game theory formulation where the malicious peer's strategy is improved to detect trusted peers, which in turn improve their defense strategy.

Perhaps the most related alternative approach to video streaming using P2P technologies are CDNs [6]. There have been multiple efforts to combine CDN and P2P technologies on hybrid architectures [7]. The idea of peer-assisted CDNs is to employ peers to reduce the load on content delivery servers. Nevertheless, the challenges are multiple, including the inherent unreliability of P2P networks, the lack of incentives for peer participation, as well as copyright issues. A reputation-based strategy to prevent the dissemination of polluted content in peer-assisted CDNs was proposed in [46]. The strategy, which is called multi-level mechanism, employs temporary peer blocking of peers with low reputation.

Table 1 presents a consolidation of the related works described above.

The first column shows references for the related works; a flag at the 'Mitigate?' column indicates wether the work combats content pollution propagation in the network (instead of proposing a model or a monitoring strategy); and then the last column presents a summary of each corresponding work.

A Comparison of Solutions for Pollution Mitigation

Next we compare our solution with the others that also mitigate polluted content.

Black Lists [16] can be effective if they are always up-to-date, however this is nearly impossible for the Internet as a whole, as new malicious peers can appear at any time and old malicious peers can adopt new addresses. An attack started by a malicious peer not black listed will not be detected. In a way black lists are similar our strategy, as both avoid receiving chunks from polluters. However, our strategy does not rely on any previous classification of peers, neither on IP address ranges, but only refrains from obtaining chunks from peers that disseminate content that result in comparisons that do not match. Moreover the classification is adaptively updated.

Another solution is to employ hashes computed by the source server [19] that can be based on Merkle trees [43] for *every* chunk. As malicious peers can replace the content and hash for any given chunk, the objective is not to handle content pollution, but to guarantee transmission integrity. Furthermore, the hashes have to be processed by all peers for all chunks. In our strategy there is no need to modify the chunks.

Other existing solutions employ cryptography, so that every peer can check every chunk [26, 27]. These solutions present an overhead in terms of CPU utilisation, while the overhead of our solution is in terms of network bandwidth. Nevertheless in those solutions every chunk has to be decrypted by all peers, while in our solution just a configurable percentage of chunks are selected for comparisons and generate extra overhead.

In [5] a limited group of peers is responsible for the verification of chunk integrity, and they are assumed to be reliable. The major difference is that our solution does not rely on any central peer or group of peers to check anything: it is fully distributed in the sense that each peer independently takes decisions regarding pollution.

Some solutions are based on coding theory, including band codes [13–15]. In these solutions peers encode and decode the chunks. The solutions only give a probability that a given peer is a polluter. This is different from our strategy in the sense that we simply compare chunks as they are retransmitted by neighbours and avoid polluters. Another work also based on coding [42] requires chunks to be modified with an extra field that has to be processed by all peers for all chunks. In our strategy there is no need to modify the chunks, furthermore in our case only a percentage of chunks is selected for comparisons.

Finally yet another strategy [40] employs a reputation mechanism with which each peer ranks the other peers. If the reputation for a particular peer is below a certain threshold, then that peer is avoided. However in order to compute the reputations, the strategy relies on any existing method to detect polluted content once it is received, which could be for example our solution. In this way, it is not possible to directly compare this reputation with our solution, they can be seen as complementary.

Table 1 Summary of related work

Work	Mitigate?	Summary
[8]	–	presents a model and estimates content pollution propagation
[11–13]	✓	employs band codes to construct black lists of polluters
[47]	✓	network coding is employed to allow the identification of polluters and limit content pollution
[32]	✓	proposes the application of black lists to live streaming
[17, 27]	✓	transmits hash-based signatures of all chunks; the challenge is to receive hashes in advance for live streaming
[21, 24]	✓	applies cryptography to every whole chunk and employs a distributed key management scheme
[36]	✓	employs peer groups to ensure chunk integrity; the server publishes content information in that group; peers access the group to verify chunks integrity
[34]	✓	Merkle-trees are employed that use hashes to guarantee the integrity of streams of chunks
[35]	–	presents an evaluation of black lists, cryptography, hash-based verification and digital signatures
[38]	–	presents an evaluation of the impact of pollution attacks; shows that the impact depends on the stability of the network, and on the bandwidth available at both malicious peers and the source
[37]	–	evaluates content authentication mechanisms to live streaming
[39, 40]	✓	employs reputation and ranking to file sharing P2P systems
[22]	✓	employs reputation and ranking to live streaming; the reputation mechanisms are based on peer experience
[23]	–	shows that reputation-based approaches can suffer with the collusion of malicious peers, with false positives, and present a high delay to propagate conclusions
[41]	–	a confidence management strategy is proposed based on retransmissions of the polluted data; depending on the situation the number of retransmissions can be high
[42]	–	in order to prevent DDoS attacks on streming sources proposes a strategy to hide source identity; used in the context of P2P VoD
[43]	–	presents a through evaluation of SopCast reaches the conclusion that a single attacker can harm up to 50% of peers and consume up to 30% of the available bandwidth
[44]	–	a centralised (non-distributed) solution is proposed to detect polluters in live streaming networks employing comparasion-based system-level diagnosis
[this work]	✓	presents a distributed strategy that employs comparison-based diagnosis to combat pollution in live streaming; each peer independently identifies and stops requesting chunks from its polluter neighbours

6 Conclusion

In this paper we proposed a new fully distributed strategy to combat content pollution in P2P live video streaming networks. The core of the solution is to employ comparison-based diagnosis to detect unauthorised changes of the transmitted video chunks. Each peer executes comparisons of randomly selected chunks received from all its neighbours. The proposed strategy is fully distributed in the sense that the peers themselves independently stop requesting chunks from those neighbours identified as polluters. Peers employ the Fireflies protocol as the underlying topology used to communicate video chunks. The strategy was implemented and a comprehensive empirical evaluation is presented, showing the effectiveness for deadling with polluters, and efficiency, in particular the overhead in terms of the extra chunks transmitted is shown to be about 8%. The results show that the strategy was able to significantly reduce the pollution in the network, in many cases it virtually eliminated all the pollution.

Future work includes the investigation of how to implement the proposed strategy on a hybrid P2P video streaming network that also employs CDN. Besides Fireflies, evaluating the solution running on top of other overlay networks and different P2P live streaming structures is also relevant future work. Furthermore, different neighbour selection strategies and omission attacks can also be investigated. Moreover the proposed strategy can also be extended to consider alternative ways to treat the comparison sets of a given peer, including the case in which there is no subset containing the majority of neighbours. We also plan to investigate dynamic and self-adaptive monitoring intervals which could be increased or decreased in runtime based on the system load.

7 References

[1] Dernbach, S., Taft, N., Kurose, J., *et al.*: 'Cache content-selection policies for streaming video services'. IEEE Intl Conf on Computer Communications, San Francisco, CA, USA, 2016

[2] Coolstreaming: 'CoolStreaming - Live Tv Streaming Platform'. Available at http://wwwcoolstreamingus, accessed in June 2019

[3] PPS.tv: 'PPS.tv (PPStream)'. Available at http://ppstv, accessed in June 2019

[4] PPLive: 'PPLive - The most popular net TV in the world'. Available at http://wwwpplivecom/en, accessed in June 2019

[5] SopCast: 'SopCast - Free P2P internet TV'. Available at http://wwwsopcastcom, accessed in June 2019

[6] Sahoo, J., Salahuddin, M.A., Glitho, R., *et al.*: 'A survey on replica server placement algorithms for content delivery networks', *IEEE Commun. Surv. Tutor.*, 2017, **19**, (2), pp. 1002–1026

[7] Anjum, N., Karamshuk, D., Shikr-Bahaei, M., *et al.*: 'Survey on peer-assisted content delivery networks', *Comput. Netw.*, 2017, **116**, pp. 79–95

[8] Wang, H., Chen, X., Wang, W., *et al.*: 'Content pollution propagation in the overlay network of peer-to-peer live streaming systems: modelling and analysis', *IET Commun.*, 2018, **12**, (17), pp. 2119–2131

[9] Gkortsilas, I., Deltouzos, K.: 'Detecting and isolating pollution attacks in peer-to-peer voD systems'. European Conf on Networks and Communications, Athens, Greece, 2016

[10] Hawa, M., Al-Zubi, R., Darabkh, K.A., *et al.*: 'Adaptive approach to restraining content pollution in peer-to-peer networks', *Inf. Syst. Front.*, 2017, **19**, (6), pp. 1373–1390

[11] Fiandrotti, A., Gaeta, R., Grangetto, M.: 'Securing network coding architectures against pollution attacks with band codes', *IEEE Trans. Inf. Forensics Sec.*, 2019, **14**, (3), pp. 730–742

[12] Fiandrotti, A., Gaeta, R., Grangetto, M.: 'Characterization of band codes for pollution-resilient peer-to-peer video streaming', *IEEE Trans. Multimedia*, 2016, **18**, (6), pp. 1138–1148

[13] Fiandrotti, A., Gaeta, R., Grangetto, M.: 'Simple countermeasures to mitigate the effect of pollution attack in network coding-based peer-to-peer live streaming', *IEEE Trans. Multimedia*, 2015, **17**, (4), pp. 562–573

[14] Ismail, H., Roos, S., Suri, N.: 'A detection mechanism for internal attacks on pull-based P2P streaming systems'. IEEE Int. Symp. on A World of Wireless, Mobile and Multimedia Networks, Chania, Greece, 2018

[15] Tauhiduzzaman, M., Wang, M.: 'Fighting pollution attacks in P2P streaming', *The Int. J. Comput. Telecommun. Netw.*, 2015, **79**, (C), pp. 39–52

[16] Borges, A., Almeida, J., Campos, S.: 'Fighting pollution in P2P live streaming systems'. IEEE Int. Conf. on Multimedia and Expo, Hanover, Germany, 2008, pp. 481–484

[17] Dhungel, P., Hei, X., Ross, K.W., *et al.*: 'The pollution attack in P2P live video streaming: measurement results and defenses'. Proc Workshop on Peer-to-peer Streaming and IP-TV, Kyoto, Japan, 2007, pp. 323–328

[18] Haizhou, W., Xingshu, C., Wenxian, W.: 'A measurement study of polluting a large-scale P2P IPTV system', *China Commun.*, 2011, **8**, (2), pp. 95–102

[19] BitTorrent: 'BitTorrent'. Available at http://wwwbittorrentcom, accessed in June 2019

[20] Liang, J., Naoumov, N., Ross, K.W.: 'Efficient blacklisting and pollution-level estimation in P2P file-sharing systems'. Asian Internet Engineering Conf., Bangkok, Thailand, 2005

[21] Li, J.S., Hsieh, C.J., Wang, Y.K.: 'Distributed key management scheme for peer-to-peer live streaming services', *Int. J. Commun. Syst.*, 2012, **26**, (10),

[22] Vieira, A.B., Almeida, R.B., Almeida, J.M., *et al.*: 'SimplyRep: a simple and effective reputation system to fight pollution in P2P live streaming', *Comput. Netw.*, 2013, **57**, (4), pp. 1019–1036

[23] So, J., Reeves, D.: 'AntiLiar: defending against cheating attacks in mesh based streaming'. Proc. IEEE 12th Int. Conf. on Peer-to-Peer Computing, Tarragona, Spain, 2012

[24] Liang, J., Kumar, R., Ross, K.W.: 'The fastTrack overlay: a measurement study', *Comput. Netw.*, 2006, **50**, (6), pp. 842–858

[25] Duarte, Jr.E.P., Ziwich, R.P., Albini, L.C.P.: 'A survey of comparison-based system-level diagnosis', *ACM Comput. Surv.*, 2011, **43**, (3), pp. 22:1–22:56

[26] Johansen, H., Allavena, A., van Renesse, R.: 'Fireflies: scalable support for intrusion-tolerant network overlays'. Proc First ACM SIGOPS/EuroSys European Conf. on Computer Systems, Leuven, Belgium, 2006

[27] Haridasan, M., van Renesse, R.: 'SecureStream: an intrusion-tolerant protocol for live-streaming dissemination', *Comput. Commun.*, 2008, **31**, (3), pp. 563–575

[28] Hei, X., Liu, Y., Ross, K.W.: 'IPTV over P2P streaming networks: the mesh-pull approach', *IEEE Commun. Mag.*, 2008, **46**, (2), pp. 86–92

[29] Ziwich, R.P., Duarte, Jr. E.P.: 'A nearly optimal comparison-based diagnosis algorithm for systems of arbitrary topology', *IEEE Trans. Parallel Distrib. Syst.*, 2016, **27**, (11), pp. 3131–3143

[30] Maeng, J., Malek, M.: 'A comparison connection assignment for self-diagnosis of multiprocessor systems'. Proc. 11th IEEE Fault-Tolerant Computing Symp, Portland, OR, USA, 1981

[31] Ziwich, R.P., Duarte, Jr.E.P., Albini, L.C.P.: 'Distributed integrity checking for system with replicated data'. Proc. 11th IEEE Int. Conf. on Parallel and Distributed Systems, Fukuoka, Japan, 2005

[32] Gheorghe, G., Cigno, R.L., Montresor, A.: 'Security and privacy issues in P2P streaming systems: a survey', *Peer-to-Peer Netw. Appl.*, 2011, **4**, (2), pp. 75–91

[33] Feng, C., Li, B.: 'On large-scale peer-to-peer streaming systems with network coding'. Proc. 16th ACM Int. Conf. on Multimedia, Vancouver, Canada, 2008

[34] Wong, C.K., Lam, S.S.: 'Digital signatures for flows and multicasts', *IEEE/ACM Trans. Netw.*, 1999, **7**, (4), pp. 502–513

[35] Dhungel, P., Hei, X., Ross, K.W., *et al.*: 'Pollution in P2P live video streaming', *Int. J. Comput. Netw. Commun.*, 2009, **1**, (2), pp. 99–110

[36] Chen, R., Lua, E.K., Crowcroft, J., *et al.*: 'Securing peer-to-peer content sharing service from poisoning attacks'. Proc. of the Eighth IEEE Int. Conf. on Peer-to-Peer Computing, Aachen, Germany, 2008

[37] Coelho, R.V., Pastro, J.T., Antunes, R.S., *et al.*: 'Challenging the feasibility of authentication mechanisms for P2P live streaming'. Proc. Sixth Latin America Networking Conf., Quito, Ecuador, 2011, pp. 55–63

[38] Lin, E., de Castro, D.M.N., Wang, M., *et al.*: 'SPoIM: a close look at pollution attacks in P2P live streaming'. Proc. 18th Int. Workshop on Quality of Service, San Jose, CA, USA, 2010, pp. 1–9

[39] Maheswari, B.U., Sudarshan, T.S.B.: 'Reputation based mesh-tree-Mesh cluster hybrid architecture for P2P live streaming'. Proc. Third Int. Conf. on Devices, Circuits and Systems, Coimbatore, India, 2016

[40] Yu, X., Fujita, S.: 'Whitewash-aware reputation management in peer-to-peer file sharing system'. Proc. World Congress in Computer Science, Computer Engineering, and Applied Computing, Las Vegas, NV, USA, 2012

[41] Hu, B., Zhao, H.: 'Joint pollution detection and attacker identification in peer-to-peer live streaming'. Proc. IEEE Intl Conf on Acoustics Speech and Signal Processing, Dallas, TX, USA, 2010

[42] Lu, M., Lee, P.P.C., Lui, J.C.S.: 'Identity attack and anonymity protection for P2P-VoD systems'. Proc. ACM/IEEE Int. Workshop on Quality of Service, San Jose, CA, USA, 2011, pp. 1–9

[43] Borges, A., Gomes, P., Nacif, J., *et al.*: 'Characterizing sopCast client behavior', *Comput. Commun.*, 2012, **35**, (8), pp. 1004–1016

[44] Ziwich, R.P., Schimidt, E.A., Duarte, Jr. E.P., *et al.*: 'Diagnosis of content pollution in P2P live streaming networks'. Proc. Sixth Latin-American Symp on Dependable Computing, Rio de Janeiro, Brazil, 2013

[45] Medina-Lopez, C., Shakirov, I., Casado, L.G., *et al.*: 'On pollution attacks in fully connected P2P networks using trusted peers'. Int. Conf. on Intelligent Systems Design and Applications, Porto, Portugal, 2016

[46] Kooshkaki, H., Akbari, B., Ghaffari Sheshjavani, A.: 'A multi-level reputation-based pollution attacks detection and prevention in P2P streaming'. Int. Symp. on Telecommunications, Tehran, Iran, 2016

[47] Wang, Q., Vu, L., Nahrstedt, K., *et al.*: 'MIS: malicious nodes identification scheme in network-coding-Based peer-to-peer streaming'. Proc. 29th IEEE Int. Conf. on Computer Communications, San Diego, CA, USA, 2010