ARTICLE IN PRESS

Computer Networks xxx (2013) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

SimplyRep: A simple and effective reputation system to fight pollution in P2P live streaming

Alex Borges Vieira [a,*], Rafael Barra de Almeida [a], Jussara Marques de Almeida [b], Sérgio Vale Aguiar Campos [b]

[a] Computer Science Department, Universidade Federal de Juiz de Fora, Brazil
[b] Computer Science Department, Universidade Federal de Minas Gerais, Brazil

A R T I C L E   I N F O

Article history:
Received 28 April 2012
Received in revised form 30 November 2012
Accepted 1 December 2012
Available online xxxx

Keywords:
P2P
Live streaming
Attack
Reputation

A B S T R A C T

Peer-to-Peer (P2P) streaming has become a popular platform for transmitting live content. However, due to their increasing popularity, P2P live streaming systems may be the target of user opportunistic actions and malicious attacks, which may greatly reduce streaming rate or even stop it completely. In this article, we focus on a specific type of attack called content pollution, in which malicious peers tamper or forge media data, introducing fake content before uploading it to their partners in the overlay network. Specifically, we present a new decentralized reputation system, named SimplyRep, that quickly identifies and penalizes content polluters, while incurring in low overhead in terms of bandwidth consumption. We evaluate our method with both simulation and experiments in PlanetLab, comparing it against two previously proposed approaches, namely, a centralized black list and a distributed reputation system, in various scenarios. Our results indicate that SimplyRep greatly outperforms the two alternatives considered. In particular, both black list and the distributed reputation method perform poorly when polluters act jointly in a collusion attack, reaching a data retransmission overhead (triggered by polluted chunks received) of 70% and 30%, respectively, whereas the overhead experienced by SimplyRep is at most 2%. Our results also show that SimplyRep is able to quickly isolate almost all polluters under a dissimulation attack, being also somewhat robust to a whitewashing attack, although the latter remains a challenge to effective P2P streaming.

1. Introduction

P2P live streaming is becoming increasingly popular. Indeed, some important TV channels already broadcast their live content on the Internet using P2P technology. For example, CNN relied on a P2P platform to assist CDNs on the live transmission of the Barack Obama's inauguration speech, which is seen as one of the largest live video event in the history of the Internet. In fact, the live transmission exceeded 1.3 million simultaneous streams, more than half of which were delivered over a P2P network.[1]

Despite the recent success, P2P live streaming protocols are very susceptible to malicious attacks, which can hinder their adoption as an alternative architecture to traditional client server protocols. Moreover, security issues in P2P live systems are more challenging than in other P2P applications because live transmissions are more vulnerable to QoS fluctuations due to their strict delay constraints.

* Corresponding author. Tel.: +55 32 2102 3311.
E-mail addresses: alex.borges@ufjf.edu.br (A.B. Vieira), rafael.barra@ice.ufjf.edu.br (R.B. de Almeida), jussara@dcc.ufmg.br (J.M. de Almeida), scampos@dcc.ufmg.br (S.V.A. Campos).

[1] This statistic was extracted from an article entitled "CNN: Inauguration P2P Stream a Success, Despite Backslash", by Janko Roettgers, published at the GigaOM portal on February 7th 2009, and available, as of August 2012, at http://gigaom.com/video/cnn-inauguration-p2p-stream-a-success-despite-backlash/.

1389-1286/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comnet.2012.12.001

One particular malicious attack that threatens P2P streaming is content pollution, where malicious peers (polluters) tamper or forge media data, introducing fake content and/or advertisements, before uploading it to their partners. Moreover, non-malicious (i.e., legitimate) peers cannot easily distinguish between polluted and genuine content before watching it. Thus, unaware of the legitimacy of a piece of content they received, these peers end up forwarding polluted content to their own partners, acting as passive polluters. Without proper defense mechanisms, polluted data spreads quickly over the P2P network, causing significant impact on the system in terms of resource (particularly bandwidth) consumption and streaming quality [1–3].

Various strategies to fight pollution attacks have been proposed in the literature, including black listing [1,4], hash-based signatures [1,5,6,3], data encryption [1,3] and reputation systems [2,7,8]. In spite of that, most popular applications do not use any protocol or data encryption strategy [1], possibly because existing techniques may significantly increase both processing and communication overhead (and thus peer resource requirements), as well as media startup latency. Thus, most current P2P live applications remain vulnerable to pollution attacks.

In this article, we tackled the problem of fighting content pollution in P2P live streaming systems by proposing a new simple and decentralized reputation system. Our system, called SimplyRep, aims at quickly detecting peers that upload polluted content, here referred to as content polluters. Unlike previous decentralized reputation systems [2,9–15], in SimplyRep, a peer reputes its partners based solely on the rate of polluted/damaged data it received from them. To that end, SimplyRep relies on any existing method to detect polluted content once it is received [6,5,1,16–18]. A peer chooses to remove a partnership if its computed reputation score falls below a locally defined reputation threshold. Eventually, content polluters are identified and isolated from all other peers, and stop receiving the live content.

We also designed a dynamic threshold mechanism that allows previously detected polluters to rehabilitate themselves and rejoin the live transmission. The mechanism works by letting each peer independently change its local reputation threshold depending on the status of the system as perceived by it. If the peer senses the P2P system is currently free of attacks, it lowers its threshold allowing new partnerships with lowly reputed peers, thus giving them a chance to regain their reputations. If, otherwise, the peer detects that the system is currently under attack, it raises its threshold to identify and penalize polluters more quickly.

We evaluated SimplyRep, with both simulation and experiments in a real setup running on PlanetLab [19], comparing it against two previously proposed approaches to deal with malicious peers, namely, a centralized black list and a distributed reputation system called StRepS [2], in various scenarios. Recall that SimplyRep relies only on the individual experiences of the peer to compute reputations. StRepS, in contrast and like various previous reputation mechanisms [9–12], computes reputations by using the individual experiences of the local peer and of its partners (referred to as network testimony), combining them into a single reputation score. This difference makes it a good baseline to evaluate our new method. Our evaluation was performed with varying numbers of polluters as well as with and without collusion of content polluters. Moreover, we also evaluated the impact of key parameters of SimplyRep on its effectiveness, as well as its robustness to dissimulation and whitewashing attacks. In the former, content polluters dynamically change their behavior, by alternating between sending polluted content and forwarding only legitimate content. In the latter, polluters repeatedly leave and rejoin the system with new identities, aiming at loosing their prior reputations.

We highlight three main results in this work. First, to motivate the need of a method to detect content polluters, we show that simply detecting and discarding polluted content before forwarding it is not an effective defense strategy as polluters remain active flooding the system with polluted content. Moreover, peers have to request new copies of the polluted data received, generating significant data retransmission overhead and/or unacceptable delays and data loss in the transmission. Second, both black listing and StRepS do not achieve good results in case of a collusion attack from a reasonably large number of polluters, with overheads due to data retransmission of at least 90% and 30%, respectively. Specifically, we found that, by relying on the network testimony to build reputation scores, StRepS becomes very vulnerable to divergences among the individual experiences. In contrast, the new SimplyRep is able to identify and isolate polluters very fastly, even under collusion and dissimulation attacks, presenting a retransmission overhead that quickly drops to less than 2%. Finally, we show that SimplyRep is reasonably robust to the challenging whitewashing attack, presenting an overhead that, despite somewhat larger than in the other scenarios considered, is still reasonably low (8%) after an initial convergence period.

The rest of this paper is organized as follows. Section 2 presents basic concepts of P2P live streaming systems and discusses the impact of content pollution on these applications. Section 3 introduces our new decentralized reputation mechanism to fight content pollution, and also describes two previous strategies that are here compared against our method. Our evaluation methodology is discussed in Section 4, and our main results are presented in Section 5. Section 6 reviews other related studies. Finally, Section 7 concludes the paper.

2. Content pollution in P2P live streaming systems

In this section, we review the fundamental concepts and mechanisms adopted by currently popular P2P live streaming systems, on which our simulation and PlanetLab experiments are based (Section 2.1). We also report on the results of an experiment to assess the impact of a pollution attack against a currently very popular P2P live application (Section 2.2).

2.1. Live P2P streaming: basic concepts

Various currently popular P2P live streaming systems, such as SopCast, PPLive and GridMedia,[2] use a non-

[2] www.sopcast.com, www.pplive.com and www.gridmedia.com.cn, respectively.

structured mesh-based overlay network, in which peers establish partnerships among themselves to build the P2P overlay. These applications maintain a list of channels, each one transmitting live content over its own P2P network, independently from the others. For each channel, the server generates the live content, splitting the media into chunks, which are transmitted over the network for later exhibition. Peers, in turn, explicitly request media chunks from their partners and also serve their requests.

In order to join a live streaming channel, a peer p_i registers itself at a centralized bootstrap server B, which returns to p_i a list of peers that are currently active in the system and represent the potential partners of p_i. Peer p_i selects n peers from this list, and tries to establish partnerships with them. Successfully established partnerships determine the set of partners of p_i. While in the system, p_i periodically exchanges keep-alive messages with its partners. Partners that remain silent for too long are removed from the set of partners. Peer p_i may try to establish new partnerships within its list of potential partners, and may also contact the bootstrap server to obtain a new list of potential partners.

Active peers exchange media chunks only with their partners. More precisely, each peer p_i keeps a chunk map cm_i that specifies the chunks that p_i currently has in its buffer and the chunks it still needs. Peers often exchange their chunk maps with their partners, thus learning each other's chunk needs and availabilities. Note that a chunk has a window of interest for each peer, defined as the interval between the instant when the chunk is received by the peer until its playback time. Peer p_i removes a chunk from its chunk map cm_i as soon as its window of interest expires. Thus, peers only try to download and upload a given chunk during a certain period of time, up to its playback deadline.

There are two main chunk scheduling policies. In the Rarest First (RF) policy, a peer schedules chunk requests depending on chunk availability, favoring requests for chunks that are rarer in the chunk maps of its partners. In the Earliest Deadline First (EDF) policy, a peer schedules chunk requests favoring chunks that will be required sooner for playback.

Note that a malicious peer can easily alter, forge or damage live media chunks, and forward polluted content to its partners in response to their requests, thus wasting network and processing resources. Content pollution is a particular concern in live P2P streaming as most popular applications do not employ any data encryption mechanism that could help pollution detection, transmitting data and control messages in clear text [1]. Thus, the impact of a pollution attack can be quite detrimental, as we further discuss next.

2.2. The impact of a pollution attack on SopCast

In order to assess the impact of a pollution attack against a real P2P live streaming system, we set up a controlled experimental environment with SopCast, a currently very popular P2P live application. Our setup consisted of a streaming media server and up to 400 peers running on PlanetLab nodes [19], spread all over the world. In order to restrict our attack to the controlled environment, therefore avoiding any harm to real external clients,

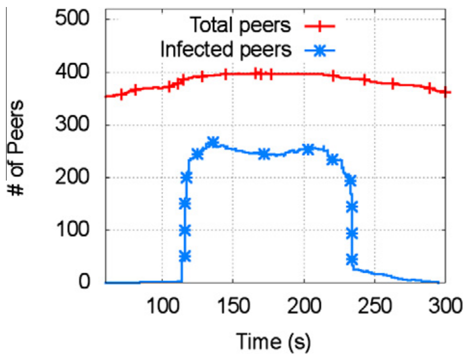


Fig. 1. Impact of a pollution attack on SopCast.

we created a private SopCast channel, and did not announce it in the SopCast web page. Neither the server nor any peer had any extra processing and bandwidth constraint (other than the constraints of the hardware available in each PlanetLab node), and the available resources were abundant for the private channel bitrate (120 kbps). Moreover, peers were configured to remain active in the system throughout the transmission.

We ran 22 experiments, varying the number of peers from 352 to 400. In each experiment, which consisted of a 5-min live transmission, one peer was randomly selected to be a polluter. During the first 2 min of transmission, the polluter acted just like a regular peer. After that, it started attacking the system, altering *every* received chunk before forwarding it to its partners.[3] The attack lasted for approximately 2 min. We measured the number of altered chunks received by each peer throughout the experiment.

We found that, on average, approximately 29% of all chunks received during the attack are polluted, although only around 30% of these chunks were uploaded by the polluter. Possible reasons for such large fraction of polluted chunks are: (1) legitimate peers passively forwarded polluted content; (2) due to network delays, peers may have requested the same chunk multiple times thus increasing the chance of receiving multiple copies of the same (polluted) chunk; and (3) if the polluter happens to be a node with many resources, chances are that it will establish a large number of partnerships, thus increasing the number of peers to which it will forward polluted content. We note that this result is consistent with those reported by Dhungel et al. [1] in a similar experiment with PPLive.

Moreover, as shown in Fig. 1, for one of the experiments, on average 260 out of the 400 peers were infected (i.e., received polluted chunks) during the attack. Thus, a single naive polluter was able to reach almost 65% of all peers in the system. These results provide evidence that SopCast does not implement any defense mechanism to pollution attacks, and peers do not perform any checking on the received chunks.

These numbers can be considered optimistic given that the attack was done by a single polluter and the server has

[3] Note that, in this experiment, the polluter neither created any fake data nor tried to promote itself by advertising more chunks than it actually had, which could further increase the pervasiveness of the attack. Rather, it simply added a watermark at each received chunk.

enough bandwidth to directly serve a large number of peers. The impact could be much greater in the case of multiple polluters and collusion. Thus, content pollution remains an open problem with severe implications for P2P live systems.

3. Fighting pollution in Live P2P systems

A variety of techniques to fight pollution in P2P live streaming systems are available in the literature, including hash-based signature and data encryption techniques [1,5,6,3], which are used to automatically detect polluted chunks before they are forwarded, as well as various centralized and decentralized reputation systems [2,7,8,1], including centralized black listing schemes [1], which, in turn, aim at identifying content polluters. Both data encryption and hash-based signature schemes avoid that tampered and fake chunks are forwarded over the network. However, by themselves, these techniques do not allow the identification of polluters, thus incurring in great processing and communication overheads (as we will see in Section 5.2) and long startup latencies due to frequent requests and retransmissions of damaged chunks [2].

In this section, we first review two previously proposed reputation mechanisms for detecting content polluters in live P2P streaming systems, namely, a centralized peer black list and a decentralized reputation mechanism called StRepS [2]. These two methods are described in Sections 3.1 and 3.2, respectively. Next, in Section 3.3, we introduce our new decentralized peer reputation mechanism, SimplyRep, which aims at quickly identifying and isolating content polluters.

Like other peer reputation mechanisms, including black list and StRepS, SimplyRep relies on the automatic identification of polluted chunks once they are received (or at least before they are forwarded). To that end, it should be applied jointly with a mechanism to detect polluted chunks. We do not assume any specific pollution detection method in the design of SimplyRep, but rather rely on the fact that the detection is possible and focus on identifying the peers that sent those polluted chunks. Thus, any previously proposed pollution detection method (e.g., hash-based signatures, star chaining, and Merkle Tree chaining) [6,5,1,16–18], could be used jointly with SimplyRep (as well as with the other methods). As we will discuss in Section 4, we here assume that a simple hash-based signature mechanism is jointly used with each polluter detection method to check data integrity of received chunks, although this assumption is made for evaluation purposes only, as any other such method could have been used.

Table 1 summarizes the main parameters and notations used in the three methods.

3.1. Centralized peer black listing

We here consider a black list system that works as follows. Each peer p_i maintains a list R_i with the locally computed reputation scores of all its partners. At regular time intervals with duration Tr_i, p_i sends the reputation scores in R_i to a centralized black list server called *LN*. *LN* may be any peer in the system and even the media server, although, due to performance, security and anonymity reasons, it is expected to be a reliable and independent entity. The *LN* server builds a global reputation score of each peer p_j, here referred to as $R_{LN}[p_j]$, by combining the local reputations sent by all its partners. This global reputation can then be checked by each peer p_i to decide whether to keep p_j as a partner: if the global reputation $R_{LN}[p_j]$ drops below a minimum reputation threshold R_i^{min} $\left(0 \leqslant R_i^{min} \leqslant 1\right)$, p_i chooses to remove p_j from its list of partners LP_i, denying future partnership and chunk requests from it.

The above description is common to most existing peer black listing strategies [20,2,21]. The definition of a peer's reputation score, in turn, depends on the specific solution. In our implementation of black list, we adopt a definition that is also used in StRepS [2], and shares great similarities with previous reputation mechanisms to detect content polluters and malicious hosts in P2P file sharing systems [9,20] and to detect free-riders in selfish overlay networks [10]. The same definition will be used also in our new system, SimplyRep (see Section 3.3). We intentionally used the same reputation definition in all three methods so as to evaluate the benefits from the specific mechanisms adopted by each of them (e.g., centralized lists, adaptive thresholds, etc.) regardless of the specific approach used to estimate the reputation of a peer p_j as perceived by another peer p_i.

A peer p_i assigns a reputation score to a partner p_j by checking, during each time interval with duration Tr_i, the quality of service provided by p_j. This is performed as follows. Let us say that, during this period, p_i requested r chunks to p_j, and p_j provided n *unsatisfying* responses to p_i $(0 \leqslant n \leqslant r)$. We consider any response that forces p_i to ask data again to another partner in the P2P network as unsatisfying. Thus, it could be a polluted chunk or a non-response from p_j, although, for the sake of simplicity, we refer to any such unsatisfying response as a polluted chunk. The ratio n/r is here taken as an estimate of the quality of the service provided by p_j as perceived by p_i. If the ratio n/r is above threshold T_i^{max}, p_i decreases p_j's local reputation. Otherwise, it increases its local reputation. Specifically, the reputation of p_j at p_i, $R_i[p_j]$, is updated as follows:

$$R_i[p_j] = \begin{cases} max(0, R_i[p_j] - \alpha_{p_i} * (1 + n/r)^{y_i}) & \text{if } n/r > \mathrm{T}_i^{max} \\ min(1, R_i[p_j] + \alpha_{g_i} * (1 - n/r)) & \text{otherwise} \end{cases} \tag{1}$$

where α_{p_i} and α_{g_i} are penalty and reward factors, respectively. The local reputation of p_j is initially set to R_i^{init}. In order to quickly identify and severely penalize polluters, we: (1) make the penalty factor larger than the reward factor (i.e., $\alpha_{p_i} > \alpha_{g_i}$), and (2) decrease the reputation of p_j exponentially with the fraction of polluted chunks received from it ($(1 + n/r)^{y_i}$) in case of penalties, where parameter y_i defines the exponential decay. Thus, a peer loses reputation much faster than it gains, which makes it possible to quickly detect polluters. A similar approach was also used in [9,10].

There are various strategies that one could adopt to penalize/raise a peer's reputation based on its prior contributions. Our approach is based on the *fraction* of polluted chunks received in response to previous requests (i.e., n/r). It is inspired on various previous strategies that adopted

Table 1
Parameters of content polluter detection mechanisms.

Common notation and parameters	
LP_i	List of partners of peer p_i
Tr_i	Duration of monitoring intervals at peer p_i
$R_i[p_j]$	p_j's Local reputation at peer p_i
n	Number of polluted chunks received in the last monitoring interval
r	Number of chunks requested in the last monitoring interval
T_i^{max}	Maximum polluted chunk rate accepted by $p_i (0 \leqslant T_i^{max} \leqslant 1)$
α_{p_i}	Multiplicative penalty factor applied to local reputations (individual experiences) at peer p_i
α_{g_i}	Multiplicative reward factor applied to local reputations (individual experiences) at peer p_i
y_i	Exponential penalty factor applied to local reputations (individual experiences) at peer p_i
R_i^{init}	Initial reputation assigned to a partner of p_i $\left(0 \leqslant R_i^{init} \leqslant 1\right)$
R_i^{min}	Minimum reputation threshold of peer p_i $\left(0 \leqslant R_i^{min} \leqslant 1\right)$
Centralized black listing approach	
LN	Black list server
$R_{LN}[p_i]$	p_j's Global reputation maintained by server LN
Decentralized StRepS reputation system	
T_i^{init}	Initial network testimony given to a partner of p_i $\left(0 \leqslant T_i^{init} \leqslant 1\right)$
$IE_i[p_j]$	Individual experience of peer p_i with partner p_j
$NT_i[p_j]$	Network testimony collected by p_i on partner p_j
M	Maximum number of reputation scores simultaneously kept by each peer[a]
New SimplyRep reputation system	
Tm_i	Duration of interval between consecutive R_i^{min} updates
γ_{p_i}	Additive increase of R_i^{min} in case system is in tempest
γ_{g_i}	Additive decrease of R_i^{min} in case system is in calm state
RT_i^{min}	Minimum and maximum threshold for R_i^{min}
RT_i^{max}	$\left(0 \leqslant RT_i^{min} \leqslant RT_i^{max} \leqslant 1\right)$

[a] The same parameter is also used in SimplyRep.

the average amount of dissatisfaction of a peer with previous transactions it performed in the system [20,2,10]. Instead, one could choose to update a peer's reputation based on the total amount of polluted chunks received (i.e., n), thus taking the effective contribution of peer p_j to peer p_i into account to compute p_i's individual experience with respect to p_j. Instead, we here focus on the *fraction* of polluted chunks so as to disregard differences in the total amount of chunks a peer requests/receives to/from each partner, which may vary depending on several factors including availability of needed chunks, bandwidth constraints, etc. We leave an evaluation of these alternative strategies to future work.[4]

The global reputation of peer p_j, $R_{LN}[p_j]$, is computed by the LN server as a weighted sum of all reported reputation scores, where weights are the global reputations of the senders:

$$R_{LN}[p_j] = \frac{\sum_{\forall k \neq j} R_k[p_j] * R_{LN}[p_k]}{\sum_{\forall k \neq j} R_{LN}[p_k]} \tag{2}$$

Thus, the global score computation favors the opinions of highly reputed peers, being thus robust to malicious peers with lower scores [20,22].

[4] We note that computing peer reputation based on the *number* of polluted chunks might lead to fewer false alarms (i.e., legitimate peers being wrongly penalized). However, we do not expect that such change should favor one polluter detector method more over the others. Thus, we expect that our main conclusions still hold if this alternative reputation metric is used, although this topic deserves future investigation.

3.2. StRepS: a decentralized reputation scheme

The literature has various decentralized reputation mechanisms for P2P file sharing applications, such as Credence [11], Scrubber [9] and XRep [12], as well as for selfish overlay networks [10]. In all these mechanisms, each peer p_i builds a reputation score for each partner p_j from two components, namely, the *individual experience* of p_i with respect to previous data exchanges with p_j, and the *network testimony* about p_j.

Inspired by these previous methods, we have previously proposed StRepS [2], a decentralized reputation scheme for P2P live streaming systems. StRepS works as follows. At regular intervals of Tr_i units of time, each peer p_i computes its *individual experience* with each partner p_j based on the number of chunks it requested to p_j and the number of polluted chunks received in response during that interval. Thus, p_i computes its individual experience with p_j, $IE_i[p_j]$ according to Eq. (1), like in the centralized approach. The individual experience with a new partner is set to R_i^{init}.

With the same frequency, peer p_i also collects the individual experiences of its partners with respect to p_j to build a *network testimony*, $NT_i[p_j]$, as a weighted sum of the collected measures. The weights are the individual experiences of p_i with each partner, so as to avoid peer defamation or promotion, which could happen if a partner of p_i tries to promote or defame p_j by reporting a fake individual experience. That is:

$$NT_i[p_j] = \frac{\sum IE_k[p_j] * IE_i[p_k]}{\sum IE_i[p_k]} \quad \forall (k \neq j) \wedge p_k \in \{LP_i \cap LP_j\} \tag{3}$$

In case p_i does not receive any testimonial about p_j then $NT_i[p_j] = T_i^{init}$, where T_i^{init} is a valid testimonial score.

Finally p_i builds a reputation score for partner p_j, $R_i[p_j]$, by combining the *individual experience* $IE_i[p_j]$ and the *network testimony* $NT_i[p_j]$ as follows:

$$R_i[p_j] = \beta * NT_i[p_j] + (1 - \beta) * IE_i[p_j] \tag{4}$$

where parameter $\beta\,(0 \leqslant \beta \leqslant 1)$ controls the weight given to each reputation component. Note that taking the network testimony into account (i.e., $\beta > 0$) helps a peer to identify potential polluters as well as to promote *peer rehabilitation*, since a peer currently considered untrustworthy by p_i may reacquire its trust by improving its reputation with other peers [9]. Facilitating peer rehabilitation is very important because the severe and quick punishments applied to polluters must be balanced with a mechanism that allows punished peers to recover themselves.

Like in the centralized approach, peer p_i considers partner p_j a polluter if its reputation $R_i[p_j]$ drops below a minimum threshold $R_i^{min}\left(0 \leqslant R_i^{min} \leqslant 1\right)$. However, unlike in the centralized black listing approach, where peer p_i makes such decisions based on a global reputation score computed from the opinion of *all* peers in the system, in StRepS, the final reputation of p_j at p_i reflects only the experiences of p_i and its partners with p_j.

We also note that, from a practical perspective, each peer reserves a fixed amount of memory to store other peers' reputations. Specifically, we assume that each peer can keep up to M reputation scores in a local buffer, managing this buffer according to a Least-Recently-Used (LRU) policy. A previously detected polluter p_j is recognized as such by peer p_i as long as its (low) reputation is kept in p_i's buffer. However, if its reputation score is removed from the buffer, past history is forgotten, and p_i may accept a new partnership request from p_j, resetting its individual experience to R_i^{init}.

3.3. SimplyRep: a novel decentralized reputation system

There are disadvantages in exploiting the network testimony to compute a peer's reputation, as performed by StRepS and other previous mechanisms [12,11,9,10]. In short time scales, a peer may behave very differently with different partners, uploading polluted content to only some of them. Thus, waiting until the network converges to a consistent opinion may take too long. Given that current live streaming rates often exceed 5 chunks per second, a damaged or fake chunk spreads much faster in a P2P live transmission, thus impacting the streaming quality much more heavily than, for instance, in a P2P file sharing application. Moreover, a peer p_i may not have many partners that share a partnership with a given peer p_j. Thus, the network testimony on p_j may not be reliable.

Therefore, we here propose a simpler decentralized reputation mechanism, called SimplyRep, that is built upon StRepS but, unlike the original method, relies only on the *individual experience* a peer had with each of its partners to compute their reputation scores. More precisely, at regular intervals of Tr_i units of time, peer p_i computes the reputation of each partner p_j, $R_i[p_j]$, as its individual experience with p_j measured in that interval, according to Eq. (1). Like in StRepS, the individual experience with a new partner is set to R_i^{init}. Moreover, each peer p_i has a minimum reputation threshold $R_i^{min}\left(0 \leqslant R_i^{min} \leqslant 1\right)$: if $R_i[p_j]$ drops below R_i^{min}, p_i removes p_j from its list of partners.

One potential problem of relying solely on the individual experience to compute a peer's reputation is that once peer p_i considers partner p_j a polluter due to a low reputation, p_j will not have the opportunity to exchange data with p_i as long as that low reputation is remembered by p_i. That is, from the point of view of p_i, p_j will be banished and will not be able to rehabilitate itself.[5]

Thus, we here propose a scheme to enable peer rehabilitation while relying only on the individual experiences to build reputation scores. The idea is to dynamically change the minimum reputation threshold R_i^{min} of each peer p_i in reaction to the network condition, as perceived by p_i. If p_i senses that the network is under attack, it increases R_i^{min} thus penalizing its bad partners more quickly. Otherwise, it decreases R_i^{min} to enable partnerships with previously punished peers. This is independently performed by all peers.

More precisely, we define two system states, namely *calm* and *tempest*. A system is in *calm* state from the perspective of a peer p_i, if p_i believes that there is no polluter in the system. Otherwise, from the perspective of p_i, the system is in *tempest*. At each Tm_i units of time, p_i checks the system state and updates R_i^{min} according to the following equation:

$$R_i^{min} = \begin{cases} max\left(RT_i^{max}, R_i^{min} + \gamma_{p_i}\right) & \text{if } \textit{tempest state} \\ min\left(RT_i^{min}, R_i^{min} - \gamma_{g_i}\right) & \text{if } \textit{calm state} \end{cases} \tag{5}$$

If the system is *calm*, p_i decreases its local threshold R_i^{min} by γ_{g_i}; otherwise R_i^{min} is increased by γ_{p_i}. We make $\gamma_{p_i} > \gamma_{g_i}$ so as to react faster to polluters. Note that we define limits RT_i^{min} and RT_i^{max} such that $0 \leqslant RT_i^{min} \leqslant R_i^{min} \leqslant RT_i^{max} \leqslant 1$.

System state is defined from the perspective of each peer, based only on its experiences with its partners. That is, if peer p_i receives *any* polluted data from one of its partners in the past Tm_i units of time, it suspects that the system is under a pollution attack: its local view of system state is set to *tempest*, and it reacts to that by adjusting its threshold R_i^{min} accordingly. Similarly, if all received chunks are legitimate, p_i perceives the system as in *calm* state, and reduces R_i^{min}.

Fig. 2 illustrates how a peer p_i reacts to changes in system state. The bar represents a reputation scale from low (left) to high (right) values, the arrow represents the current value of R_i^{min}, whereas numbers 1, 2 and 3 indicate the current reputations at p_i of three partners (here referred to p_1, p_2 and p_3). Note that, in Fig. 2a), all partners are located on the right side of the arrow, indicating that their reputations are above the minimum threshold. As soon as p_i receives any polluted chunk, it sets the system state to tempest, increasing its minimum reputation threshold, as shown in Fig. 2b). Since the current reputation of peer p_3 at p_i falls below the minimum, p_i understands that p_3 is a

[5] Recall that, in the previous StRepS system, the network testimony was taken into account to help a peer to rehabilitate itself in the eyes of an old partner.

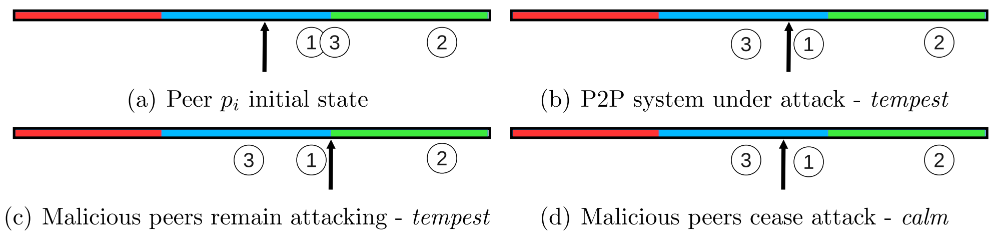

1
3
2
(a) Peer p_i initial state
(b) P2P system under attack - *tempest*
(c) Malicious peers remain attacking - *tempest*
(d) Malicious peers cease attack - *calm*

Fig. 2. Dynamic minimum reputation threshold of peer p_i (arrow indicates R_i^{min}, 1, 2 and 3 are p_i's partners, and their positions in the bar indicate their current reputations at p_i).

polluter, removes it from its list of partners, and stops interacting with it. Afterwards, p_i continues receiving polluted data and, once again, increases R_i^{min}, as shown in Fig. 2c). Now, also p_1 is removed from p_i's list of partners, as its reputation is below the minimum threshold. After that, p_i stops receiving polluted data, which makes it change the system state back to *calm* and decrease R_i^{min}. The new value of R_i^{min} allows p_i to accept p_1 as a partner once again. Note that changes in R_i^{min}, according to Eq. (5), are performed independently of the changes in the local reputations of each partner, defined in Eq. (1).

Finally, we note that, like in StRepS, each peer also has a fixed-size buffer to store the local reputations. Up to M reputations can be stored simultaneously, and the buffer is managed according to the LRU policy.

4. Evaluation methodology

We evaluated the polluter detection mechanisms described in Section 3 in various scenarios, with and without polluters acting jointly in a collusion attack. We also evaluated our new SimplyRep method under dissimulation and whitewashing attacks. Our evaluation was performed via simulation as well as via experiments in a real setup running on PlanetLab. In both cases, we included a bit marker in the packet header to indicate polluted chunks. Thus, we could track polluted chunks as they are transmitted over the network.

As discussed in Section 3, SimplyRep as well as the other polluter detection methods rely on each peer being able to automatically detect polluted chunks before forwarding them. Any existing hash-based signature and data encryption technique for pollution detection could be adopted [6,5,1,16–18]. For the sake of evaluation purposes, we here assume that a simple hash-based signature method is executed jointly with all three reputation systems analyzed. Each chunk is checked for data integrity after being received by a peer. When a peer detects a polluted chunk, it immediately discards it before forwarding it to other peers, and sends requests for chunk retransmission to its partners.

We describe our simulation environment in Section 4.1, whereas our PlanetLab setup is presented in Section 4.2. We also introduce our main metric of evaluation in Section 4.3.

4.1. Simulation environment

We used NS-2 [23] to design our simulator of a *mesh-pull* P2P live streaming system. Our simulator models the system at two levels, namely, the P2P overlay network (application level) and the underlying Internet-like network (routing level).

At the routing level, we generated underlying networks with 10,000 nodes using the Waxman model [24] of the BRITE Internet topology generator [25]. We created topologies by varying parameters α and β as suggested in [26]: α between 0.42 and 0.46, and β between 0.62 and 0.68. We used average node degrees uniformly distributed between 2 and 3. We also set the capacity and the delay of all links to 100 Mbps and 5 ms, respectively, thus modeling constraints on packet transmission in the underlying network.[6]

At the application level, we randomly selected 1000 nodes from the routing level to be participants of the P2P system: one participant represented the server, which generated the live streaming media, and the others represented peers. The remaining 9000 nodes acted as routers in the underlying network. All participants built a mesh-based overlay network. Partnership selection was performed by the bootstrap server (co-located with the media server), as discussed in Section 2.1, which used a random peer selection strategy to select the potential partners of a peer. Moreover, we adopted the Rarest First policy to schedule chunk requests, using the Earliest Deadline First policy to break ties, as previous work [27,28]. Chunk exchanges between two participants of the system were simulated by the routing of packets along the shortest path connecting both nodes in the underlying network. Thus, our simulator models all key components of a request-driven mesh-based P2P live streaming protocol, including the control messages between peers and bootstrap server to join the system and acquire (and reacquire) a list of potential partners as well as the control messages between peers to establish partnerships and exchange chunk maps, as described in Section 2.1.

Peers in the overlay network were randomly grouped into *good peers* and *polluters*. Good peers never forwarded polluted content received from others, as polluted chunks were detected and discarded once they were received. However, due to temporary network problems, good peers might unintentionally introduce delays or damaged chunks that were perceived as pollution by their partners. In that case, they might be (momentarily) seen as polluters

[6] We note that the ns-2 models for packet delay and losses, which assume uniform distributions and independence, are somewhat naive. However, our simulation results are validated in a real setup, subject to real-world network conditions.

and had their reputations reduced, being eventually penalized. Parameter p_i^{error} was thus defined as the probability of a chunk uploaded by good peer p_i being polluted. The goal of this parameter is to capture the probability of an error in the communication channel between two peers due to temporarily bad network conditions that cause a legitimate chunk at the origin to be corrupted or lost and thus be perceived as a polluted response in the destination. It basically reflects the quality of the communication channel between a good peer and its partners.

Polluters, in turn, are malicious peers that announced fake chunk maps containing all needed chunks, aiming at attracting more partners. During an attack, a polluter always answered requests from its partners, thus forging data. We simulated scenarios where polluters acted independently and scenarios of a joint collusion attack. When colluding, polluters tried to defeat the pollution defense mechanism by assigning each other high reputation scores, which are uniformly selected between R_i^{min} and 1.0,[7] and publicized those scores to the black list server or to their neighbors (in case of StRepS). We note that polluters were introduced in the network structure just like any other peer, thus occupying positions that were randomly distributed across the mesh.[8] They were also subject to the same partner selection policy as all other peers (i.e., random selection).

We simulated 60-min live transmissions at the rate of six chunks per second, a common value for these applications [1]. Moreover, we modeled the strict time restrictions of live transmissions by defining a fixed window of interest W on each chunk for each peer. In other words, when a peer was playing chunk i, it set a timer for receiving the chunks that are W seconds ahead in the video. The timer was set to W seconds into the future, corresponding to the deadline of those chunks, for playback purposes. Chunks that arrived after their deadlines were disregarded for the sake of computing the streaming rate (see Section 4.3), being considered as lost.

Good peers joined the system in the first 5 min of transmission with joining times uniformly distributed in the range 0–5. Similarly, polluters joined the system between the 2nd and 5th minute of transmission. Throughout the transmission, we probed the system every 30-s interval to measure the numbers of legitimate and polluted chunks as well as the number of retransmitted chunks received by each peer.

Our simulations were driven by a peer behavior model that was first presented at [29]. Our model was parameterized according to the results of a characterization of peer behavior in various live transmissions on SopCast. In particular, we focused on peer behavior during live transmissions of sport events broadcasted by one of the most popular TV channels in Brazil. The channels were monitored during two major events for the Brazilian audience, namely, the final matches of an important soccer championship. The live content was broadcasted at 250 kbps. We briefly describe the adopted peer behavior model next.

After joining the system, good peers may leave and rejoin the system dynamically (peer churn). To capture this behavior, we assumed an *ON/OFF* model in which each good peer alternates between an active ON state and an idle OFF state. While in ON state, a peer establishes partnerships, exchanging data with a number of other peers. We refer to this period as a session and to its duration as ON time. A peer may have various sessions in the system during a transmission. After a session finishes, it may either go to the OFF state with probability p_{off} or quit the system, never returning again. We refer to the period during which a peer remains idle by OFF time.

During each session, a peer establishes one or more partnerships. Thus, at the partnership level, client behavior was modeled in terms of number of simultaneous partnerships and partnership duration. The number of simultaneous partnerships serves as an upper-limit on the number of partners a peer may have at any time, which indirectly constraints the total bandwidth effectively used by the peer.

To simplify the generation of synthetic workloads, partnership duration was expressed as a percentage of the remaining peer ON time, which is upper-bounded by 100%. For example, if a peer p_i establishes a new partnership when its remaining ON time is 10 s and the partnership lasts for 5 s, we say that the partnership duration is 50%. The final component of our peer behavior model is the rate at which new peer sessions are initiated. Thus, we defined the session inter-arrival time as the time elapsed between consecutive sessions (of the same peer or of different peers).

We note that, unlike good peers, polluters remained in the system throughout the transmission (no churn), although the number of partnerships simultaneously established by each polluter as well as the duration of such partnerships were constrained, according to the aforementioned peer behavior model.

Table 2 summarizes the main findings of our previous characterization [29], presenting distribution models and parameter values for each component of our peer behavior model. We used these distributions to generate the synthetic workloads which, in turn, were used to drive our simulations.

When evaluating our new SimplyRep reputation mechanism, we also considered a scenario in which polluters may change their behavior dynamically to make the detection harder. Specifically, we simulated a *dissimulation attack* by having each polluter alternating between an attack state, during which it forwarded polluted content, and a non-attack state, during which it acted just like a good peer. All polluters started attacking the system at the same time and remained attacking it for T_{dissim} units of time. After that period, all polluters stopped forwarding polluted content, entering the non-attack state. While in this state, the polluters might jointly go back to the attack state with probability p_d.

We note that our simulator does not capture aspects related to specific media encodings. For example, in a real scenario, the loss (or pollution) of specific chunks (e.g., key frames) might have a more severe impact on the

[7] By doing so, polluters avoid being easily detected, which would occur if they always assigned the highest reputation score to each other.

[8] In other words, polluters *did not* necessarily occupy positions in the network that are strategically more important, which could lead to a more pervasive attack.

Table 2
Characterization of SopCast client behavior: summary.

	Distribution	Mean	Std. dev.	Parameters
Session inter-arrival times	Lognormal distrib.	1.417	1.113	$m = 0.108$ $\sigma = 0.693$
# Of sessions/peer	$\leqslant 2$ for 80% of peers			$P_{off} = 0.39$
ON times	Weibull distribution	23.593	34.986	$\alpha = 2.032$ $\beta = 0.233$
OFF times	Exponential distribution	18.491	16.171	$\lambda = 0.054$
# Of partnerships	Normal distribution			$\mu = 101.453$ $\sigma = 41.537$
Partnership duration	Gamma distribution	8.272	19.950	$\alpha = 0.118$ $\beta = 0.700$

Weibull: $p_X(x) = \alpha\beta x^{\beta-1} e^{-\alpha x^{\beta}} I_{(0,\infty)}(x)$, Lognormal: $p_X(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{\frac{-(\ln(x)-\mu)^2}{2\sigma^2}}$, Exponential: $p_X(x) = \lambda e^{-\lambda x}$, and Normal: $P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$.

streaming quality perceived by the user. Here, we assume that streaming quality is proportional to the fraction of unpolluted chunks received in time of playback (see Section 4.3), that is, all chunks are equally important to streaming quality. Despite the simplification, we believe that this assumption provides a good starting point for our investigation. Moreover, the adoption of a specific media encoding should not favor any particular polluter detector method considered here, and thus should not change our main conclusions.

4.2. Real setup

We also evaluated our novel SimplyRep reputation mechanism on a real setup running on PlanetLab. Towards that end, we implemented a mesh-pull based live streaming system, as described in [30], extended with the proposed mechanism. Like in our simulator, our real system uses the Rarest First chunk scheduling policy, and chooses partners for a peer based on random selection.

We set up a dedicated live streaming system with a media (and bootstrap) server in our campus network, and 133 peers running on PlanetLab nodes. Out of these peers, 120 acted as good peers and 13 as polluters. The media server transmitted a 60-min constant bit rate (CBR) stream at 120 kbps. Since server and PlanetLab nodes have bandwidth and processing constraints imposed by the available hardware, we did not introduce any extra resource restriction on any participant of the system. Moreover, PlanetLab nodes are heterogenous, which contributes to make this experimental setup more realistic.

All peers joined the live transmission during an initial period of 5 min, with joining times following an uniform distribution. We set all peers to remain active in the system throughout the transmission (i.e., no churn). We also set the maximum number of simultaneous partners and the duration of each partnership according to the distributions in Table 2, as in the simulation. If any partner failed or quit the system, a peer requested new partner candidates to the bootstrap server. At the beginning of each experiment, polluters acted just like good peers, forwarding only legitimate chunks. Polluters started attacking the system approximately 2 min after the beginning of the transmission, and remained attacking until the end of the transmission. During the attack, polluters announced a complete chunk map and forged chunks to answer their partners' requests. Like in the simulation, polluters also colluded by assigning each other reputation scores that were uniformly distributed between R_i^{min} and 1.0.

In our PlanetLab experiments, we also considered a scenario of a *whitewashing attack*. In this type of attack, polluters repeatedly leave and rejoin the P2P system with a new identity, which causes their reputation scores to be reset to R_i^{init}. In our experiments, each polluter left and rejoined the system every T_{ww} units of time.

4.3. Evaluation metric

Our main metric of evaluation was the mean instantaneous rate at which peers receive media (data) chunks, here referred to as simply *streaming rate*. We measured this rate at each peer at 30-s intervals, and normalized it by the video streaming rate.

The streaming rate consists of two main components: the rate of chunks that arrive in time and are played by their deadline and the rate of chunk retransmissions. The former is ideally 1, but may deviate from that mark (i.e., decrease) as the chunk loss rate[9] increases. Chunk retransmissions are triggered by chunk losses and the reception of polluted data, thus representing an overhead in terms of extra processing and bandwidth requirements. Both components are important: the former captures, though approximately, the streaming quality, whereas the latter captures the processing and bandwidth overhead caused by pollution. Thus, we analyze them both for each polluter detector method considered.

We note that streaming rate is only an approximate indicator of streaming quality. We chose this metric, following previous work [7,8], because it facilitates experimentation and automatic evaluation in a large number of different scenarios, our main goal here. Moreover, we here did not intend to capture any extra overhead introduced by the P2P streaming protocols (e.g., overhead due to control packets), as this would make our results specific to that protocol and its characteristics, which is not our goal. We also disregarded any overhead introduced by the specific method adopted to detect polluted chunks, as this overhead has been widely studied elsewhere [6,5]. Thus, we considered such overheads outside the present scope, focusing, instead, only on data exchanges and the overhead imposed by chunk retransmission.

We believe that, if we disregard the overheads introduced by the specific P2P streaming protocol and pollution detection method used, the streaming rate is a reasonable approximation of streaming quality. In particular, only chunks that arrived prior to their respective deadlines

[9] A chunk was considered lost when no legitimate copy of it arrives within its deadline.

were considered in our computation of streaming rate, and redundant (duplicate) chunks were discarded. In other words, a streaming rate equal to 1 implies in no chunk retransmission and no losses: all video chunks were received within the expected deadline, which means that video could have been played to the user with no interruptions (disregarding the aforementioned overheads). Larger media rates imply in larger retransmission rates and thus, larger processing and bandwidth overheads, whereas media rates lower than 1 imply in large chunk losses and thus worse streaming quality.

5. Experimental results

We here discuss some representative results of our evaluation of the new SimplyRep reputation system using both simulation and experiments with a real P2P live streaming system on PlanetLab. We start by motivating the need of a mechanism for identifying and isolating polluters. We do so by showing, in Section 5.1, simulation results that illustrate the pervasiveness of a pollution attack when the only defense mechanism used is to discard polluted chunks before forwarding them. In Section 5.2, we compare, using our simulation environment, SimplyRep against the black listing and StRepS approaches. Next, in Sections 5.3 and 5.4, we further evaluate SimplyRep, still using simulation, analyzing its sensitivity to some of its main parameters as well as its robustness to a peer dissimulation attack, respectively. Finally, in Section 5.5, we evaluate SimplyRep using experiments with a real system on PlanetLab, and also discuss its robustness to a whitewashing attack.

Unless otherwise noted, we assumed the following parameter values: Tr_i = 30 s, $T_i^{max} = 0.15–0.30$ (uniformly distributed, or u.d.), $\alpha_{p_i} = 0.07–0.1$ (u.d.), $\alpha_{g_i} = 0.07$, $y_i = 2$, $R_i^{init} = 0.6–0.7$ (u.d.),[10] and $R_i^{min} = 0.5$ for all peers. Moreover, we set $T_i^{init} = 0.6–0.7$ (u.d.) for StRepS, and for SimplyRep, assumed Tm_i = 5–30 s (u.d.), $\gamma_{p_i} = 0.6$, $\gamma_{g_i} = 0.3$, $RT_i^{min} = 0.3$, and $RT_i^{max} = 0.7$ for all peers. We also set M = 200 for all peers, for both StRepS and SimplyRep.

Moreover, in our simulations, we assumed that p_i^{error}, the probability of a chunk uploaded by a good peer being corrupted or lost, is uniformly distributed between 0 and 0.1. This is a reasonable range according to a previous study of live streaming systems that reported loss rates of up to 0.1 (and often 0.05) [31]. We also set W, the window of interest of each chunk, to 20 s, for both simulation and PlanetLab experiments.[11]

All simulation results, reported in Sections 5.1, 5.2, 5.3, 5.4, are averages of 35 replications, whereas results of PlanetLab experiments, reported in Section 5.5, are averages of five replications. To assess their variability, we also computed the coefficient of variation (CV), i.e., the ratio of the standard deviation to the average. For both simulation and real experiments, we obtained CV values below 2.9%, implying that reported averages are very representative of all replications.

5.1. Effectiveness of detecting and discarding polluted chunks

We start by assessing the effectiveness of using, as the only defense mechanism against pollution, a method to detect and discard polluted chunks before forwarding them. We here used a hash-based signature method to identify polluted chunks. Fig. 3 shows the instantaneous average streaming rate when 1% and 10% of the peers are polluters. These results were obtained via simulation.

Note that both curves show very similar behavior: the streaming rate increases and reaches a peak, which exceeds 1, very quickly, remaining close to the peak through the rest of the simulation. We note that the chunk loss rates during these experiments are very low, below 2%. This implies that the fraction of chunks that arrived in time for playback is very close to 1, and thus streaming rates exceeding 1 indicate chunk retransmissions due to pollution. In other words, the streaming rate minus 1 represents the retransmission overhead. Indeed, when 1% of the peers are polluters, the retransmission overhead is around 25% through most of the simulation. If the fraction of polluters increases to 10%, the overhead exceeds 100%.

We emphasize that this overhead is only due to chunk retransmission, as the overhead caused by the specific polluted chunk detection method adopted is disregarded (see Section 4.3). Thus, although the results shown in Fig. 3 refer to the use of a hash-based signature to identify polluted chunks, the results for other solutions based on simply checking received chunks and discarding polluted ones should be very similar. Clearly, this approach is not effective as polluters remain active in the system, forwarding pollution. It is also necessary to adopt a strategy to identify and isolate those polluters.

5.2. Comparison of polluter detection mechanisms

In this section we analyze the effectiveness of the three polluter detection methods considered, namely SimplyRep, StRepS and black list, comparing them using simulation. Fig. 4a and b shows the instantaneous average streaming rates for the three approaches when 1% and 10% of the peers are polluters, respectively, acting independently (i.e., no collusion). Once again, we find that the chunk loss rates are very low during simulation (below 2%), and thus fraction of chunks played by their deadline is very close to 1. Thus, we can infer the retransmission overhead associated with each method by assessing how much the instantaneous streaming rates exceed the ideal mark (i.e., 1).

In the scenario with only 1% of polluters, Fig. 4a shows that all methods incur in system overhead, in terms of streaming rate, below 20% (at peak). However, unlike when the only defense mechanism adopted is to discard polluted content (Fig. 3), the use of a polluter detection method allows the gradual reduction of the overhead as more polluters are identified and isolated. Comparing the three polluter detection methods, black list is the best approach, with a smaller peak overhead and a quicker convergence towards the ideal media rate (i.e., 1), which occurs when

[10] We also experimented with lower values of R_i^{init}, notably 0.51 for all peers, obtaining very similar results.

[11] We did experiment with other values of W (e.g., 30 and 60 s), reaching qualitatively similar results.

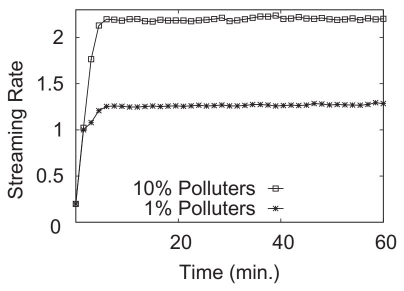


Fig. 3. Retransmission overhead for strategy based only on detecting and discarding polluted chunks.

all polluters are detected and isolated. SimplyRep is somewhat worse, followed by StRepS, the least effective method.

Fig. 4b shows the same trend for 10% of polluters. However, StRepS performs much more poorly in this case: the retransmission overhead reaches 70% at its peak and remains at around 30% until the end of the transmission. Thus, the system is not able to isolate all polluters. The main reason for that lies in the combination of three factors: (1) peers change their partners often, (2) peers have a maximum number of simultaneous partners, and (3) StRepS uses the individual experiences of a peer's partners (i.e., network testimony) to compute the reputations at that peer. Indeed, StRepS is very vulnerable to divergences in the individual experiences of various peers with respect to the same polluter. As peers change their partners often and have a limited number of partnerships at any time, polluters often manage to refresh their reputation scores by getting new partners. As consequence, network testimony may diverge as the collected individual experiences are not consistent. This implies that the aggregation of multiple individual experiences to compute the final reputation may end up favoring some polluters (i.e., leading to reputation scores above the minimum).

We note that the constraint on the number of reputation scores simultaneously kept by each peer (defined by parameter M) further contributes to some polluters escaping detection and isolation under StRepS. This is because previously detected polluters may be forgotten if their (low) reputation scores are removed from the local buffers. In this case, these polluters will have their reputations reset to R_i^{init}. As long as R_i^{init} is greater than or equal to R_i^{min}, partnership requests from those polluters may be accepted. However, recall that SimplyRep adopts the same constraint. Yet, the vast majority of all polluters are quickly isolated by that approach. Thus, we believe that the divergences in the individual experiences, particularly when peers often change their partners, is the main reason behind StRepS's poor performance when there is a reasonably large fraction of polluters in the system. In fact, the observation that the use of network testimony might actually be ineffective was the main motivation for us to propose a method that relies only on individual experiences.

Regarding the other two methods, we find that the black list approach again has the best performance, thanks to the global reputations computed by the centralized server. Interestingly, in this scenario, the difference between the black list and SimplyRep is very marginal, and smaller than in Fig. 4a: the peak overhead is about the same (i.e., 40%), and SimplyRep takes only slightly longer to isolate all polluters. This is because as the number of polluters increases, more peers quickly notice the attack, changing system state to tempest, which increases the chances of them denying future partnerships with other polluters. When there are few polluters, system convergence takes longer as most peers do not sense the attack, being more susceptible to maintaining partnerships with polluters.

We now turn to scenarios where polluters act jointly by assigning high reputations to each other, in a collusion attack. Fig. 5a shows that, with only 1% of polluters, all three methods behave very similarly with or without the collusion. In contrast, Fig. 5b shows a different trend when the number of polluters in the collusion represents 10% of the system. In this case, the black list is the least effective approach, with the retransmission overhead reaching almost 90%. Indeed, this overhead is comparable to that when no polluter detection mechanism is used and defense is done by simply discarding polluted chunks (Fig. 3). Thus, a collusion attack from a reasonably large peer population renders a black list approach useless, as the global reputations are biased by polluters' reported opinions. In contrast, both StRepS and SimplyRep are very

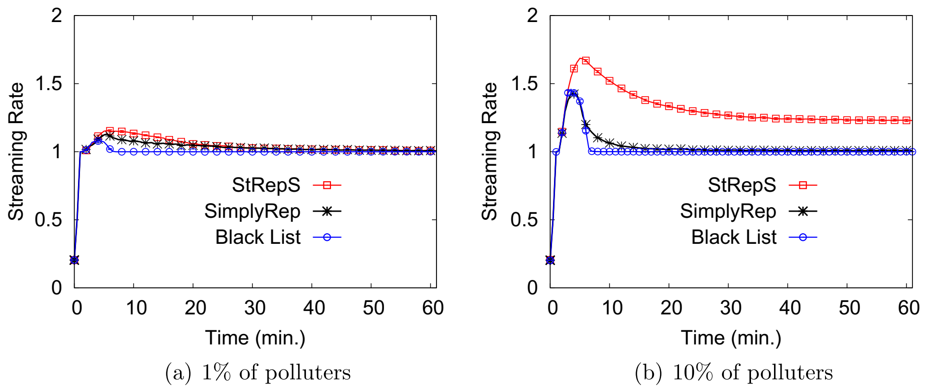


Fig. 4. Retransmission overhead of polluter detection methods (no collusion).

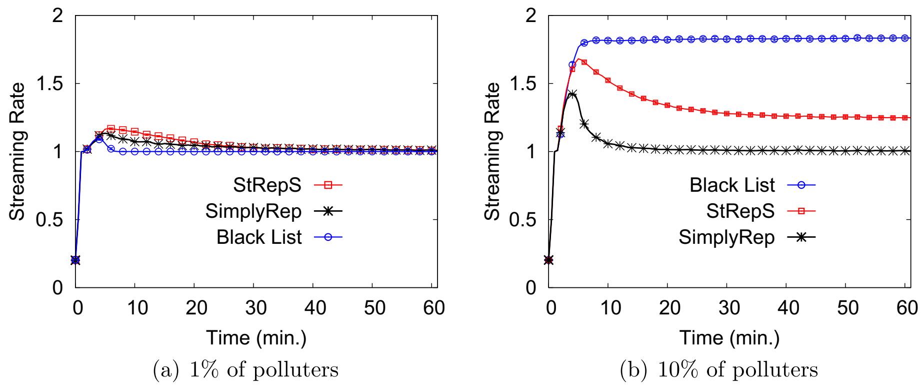


Fig. 5. Retransmission overhead of polluter detection methods (with collusion).

robust to the collusion attack, producing results very similar to those without the collusion. SimplyRep, in particular, should be insensitive to collusion as peers rely only on their local experiences to repute their partners.

In sum, our new SimplyRep mechanism greatly outperforms the previous StRepS method in all analyzed scenarios. Moreover, their results are very close to those of the centralized black list approach, the best performer, in case of small number of polluters or no collusion. In case of collusion attacks from a reasonably large number of polluters, SimplyRep greatly outperforms the black list strategy, quickly detecting and isolating all polluters in the system.

5.3. Sensitivity of SimplyRep to key parameters

We further evaluate SimplyRep, again using simulation, by analyzing the impact of some of its key parameters on polluter detection. In particular, we focus on parameters y_i, α_{p_i} and α_{g_i}, which are used to update (increase/decrease) a peer's local reputation (Eq. (1)), and thus play central role not only on SimplyRep but also on the other two methods. Focusing on SimplyRep, we analyze how it reacts as we vary these parameters. In these experiments, we consider a scenario where 10% of the peers are polluters.[12] Moreover, we set the same value of each analyzed parameter for all peers in the network.

Starting with y_i, the exponential factor applied to penalize a polluter's reputation, Fig. 6a shows the instantaneous average streaming rate obtained with SimplyRep during the first 20 min of the transmission, for various values of y_i, and for α_{p_i} and α_{g_i} equal to 0.07 and 0.04, respectively. As y_i increases, the penalties become more severe and polluters are isolated more quickly. As consequence, the peak overhead is somewhat lower, and the streaming rate approaches the ideal rate of 1 more quickly. However, note that results are very similar for y_i values between 1 and 1.4, in which case polluters loose reputation almost linearly, remaining active, harming the system, for longer. The results are also similar for values between 1.6 and 2.0, where reputation decreases more quickly, and a polluter is identified and isolated with fewer interactions.

Fig. 6b shows results for various values of α_{p_i}, the multiplicative penalty factor. For all considered scenarios, we set $y_i = 2$ and $\alpha_{g_i} = 0.04$ for all peers. Similar results were also obtained for other values of α_{g_i}, and for values of y_i between 1.6 and 2.0. Once again we see that larger values of α_{p_i} lead to faster convergence and lower peak overheads. In particular, values of α_{p_i} between 0.07 and 0.1 produce very similar results. We note that very large values of either α_{p_i} or y_i increase the chances of the system penalizing good peers that are experiencing temporary network problems,[13] which is not desired. Thus, we recommend to set $\alpha_{p_i} = 0.07$ and $y_i = 2$, as we observed that the chance of penalizing good peers due to temporary errors becomes non-negligible for larger values.

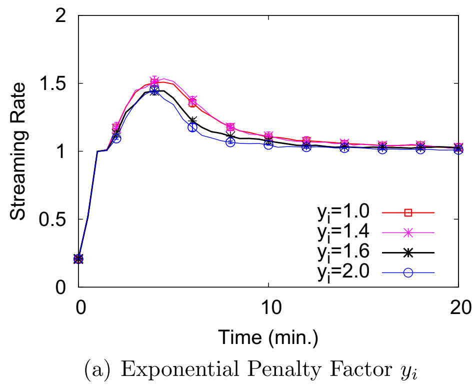
Finally, Fig. 6c shows that SimplyRep is very insensitive to the value assigned to α_{g_i}, the multiplicative reward factor applied to a good peer's reputation. The results shown in the figure were obtained for $y_i = 2.0$ and $\alpha_{p_i} = 0.07$, although the same holds for scenarios with various values of y_i and α_{p_i}. Thus, we recommend to set α_{g_i} to close to half of α_{p_i}, such as $\alpha_{g_i} = 0.04$, so as to penalize reputations more quickly.

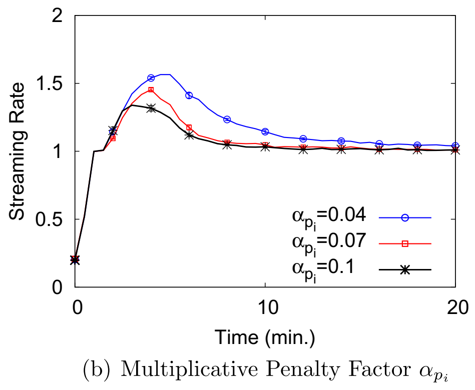
5.4. SimplyRep under peer dissimulation attack

So far we have considered that once a polluter starts attacking the system, it remains attacking it until detected and isolated. However, a smart polluter may change its behavior dynamically to make it harder for the defense mechanism to identify it. Thus, we here analyze the robustness of SimplyRep under a dissimulation attack, in which polluters alternate between attack and non-attack states, as described in Section 4.1. To that end, we use our simulator, fixing T_{dissim}, the period during which polluters remain attacking the system each time they are in the attack state, at 3 min, and varying the probability p_d that polluters, once acting as good peers (i.e., in the non-attack state), go back to the attack state. Moreover, we consider a

[12] We here consider polluters that act jointly, in a collusion attack. However, as discussed in Section 5.2, SimplyRep is very insensitive to collusion, as modeled by us. Thus, similar results are obtained in a scenario without collusion, fixing the same fraction of polluters.

[13] Recall that, in the simulation, we model these problems by assigning probability p_i^{error} of a chunk uploaded by a good peer p_i being perceived as an error (i.e., being lost or corrupted).





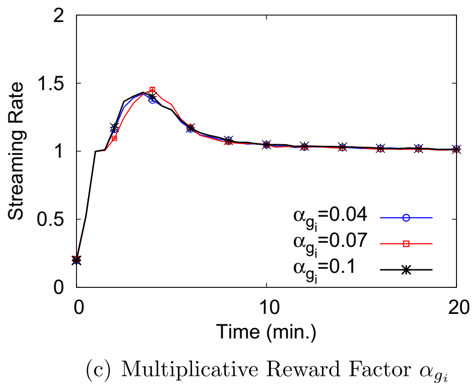


Fig. 6. Sensitivity of SimplyRep to key parameters (10% of polluters).

scenario with 10% of polluters. Once again, like in all simulation experiments, the chunk loss rates were very low (up to 2%). Thus, our main metric of interest is the overhead due to chunk retransmission experienced by the peers.

Fig. 7 shows results for values of p_d equal to 0.25, 0.5 and 0.75. Note that, in all 3 scenarios, SimplyRep is able to identify and isolate almost all polluters very quickly, with the retransmission overhead falling below 2% by the end of the simulation. Obviously, the initial peak overhead is larger for larger values of p_d, which reflect more aggressive polluters. However, after the initial peak, the curves are very similar for all values of p_d. As soon as polluters are identified in the first attack, the system reacts very similarly, regardless of p_d. This is because once a polluter is identified, it will have to forward unpolluted chunks to other peers (unaware that they are polluters) for a long time until it is allowed to interact again with previously attacked peers, that is, until these peers, sensing the system is back to calm state, decrease their minimum reputation thresholds enough to enable such interactions. In that case, as soon as the polluter starts a new attack, it is quickly detected, as its reputations are close to the minimum reputation thresholds.

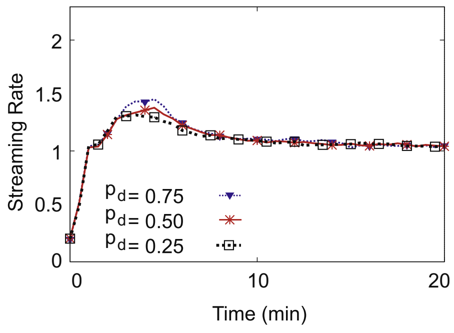


Fig. 7. SimplyRep under dissimulation attack (p_d is the probability of polluters starting an attack, 10% of polluters).

5.5. PlanetLab experimental results

Finally, we evaluate SimplyRep in a realistic setup, running experiments in PlanetLab. As discussed in Section 4.2, we configured the system with 10% of the peers as polluters. PlanetLab nodes have natural constraints on CPU, memory and bandwidth. In such real environment, the P2P live system is susceptible to network delays and packet losses that further contribute to increase the rate of chunk retransmissions. In fact, unlike in the simulation experiments, discussed in Sections 5.1, 5.2, 5.3, 5.4, we did observe a non-negligible fraction of chunk losses during our experiments in PlanetLab. Thus, we here show separate results in terms of overhead imposed by chunk retransmissions and chunk losses. In this environment, the retransmission overhead is caused by the reception of polluted data and by network delays, which force peers to request a chunk before its deadline expires.

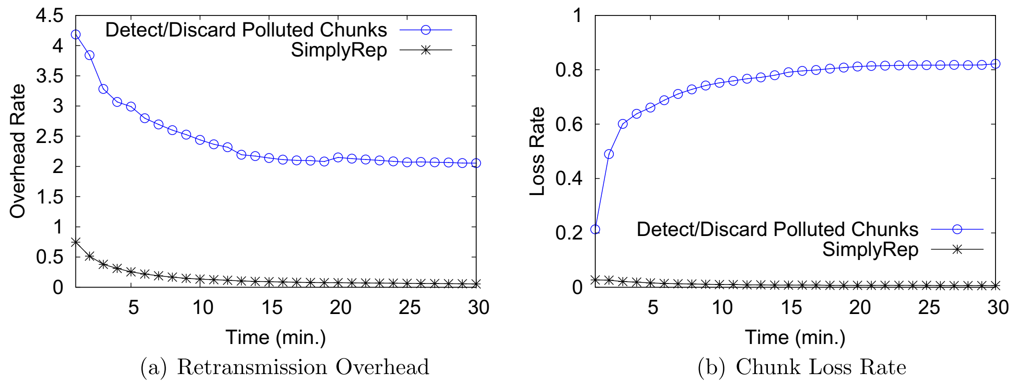


Fig. 8. Impact of pollution attack in PlanetLab setup (10% of polluters).

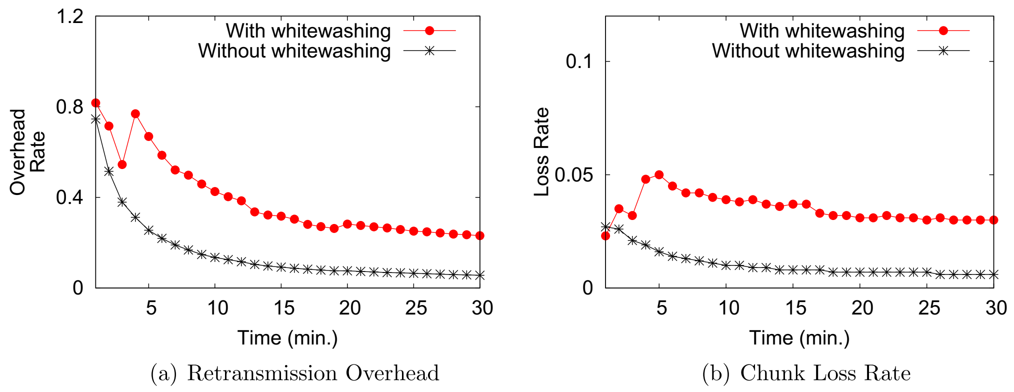


Fig. 9. Impact of pollution with and without whitewashing attack (10% of polluters).

Figs. 8a and b shows the instantaneous average retransmission rate and average chunk loss rate, respectively, experienced by peers during the first 30 min of transmission. For comparison purposes, both figures also show the results when the only defense mechanism used is to detect and discard polluted chunks before forwarding them.

Note that simply detecting and discarding polluted chunks leads to a retransmission overhead that exceeds 400% at the beginning of the experiment and remains at 214% even after 30 min of transmission. In other words, the download bandwidth requirements of each peer exceeds three times the streaming rate of the live transmission. Note that this overhead is much larger than the one observed in our simulations (see Section 5.1). This increase can be credited mostly to delayed chunks, which force peers to request data to different partners. Moreover, as shown in Fig. 8b, the chunk loss rates experienced under this method is very high, exceeding 80%. As polluters are not isolated, they keep flooding the system with polluted chunks, which contribute to increase the retransmission overhead and the time taken until a genuine copy of a chunk is received. A good peer may have to request a chunk to several partners (some of which are polluters) until a genuine copy of it is received, often too late for playback.

In contrast, our SimplyRep reputation system performs much better, with results that are similar to those obtained in our simulations. Although the overhead reaches a peak of 80% at the beginning of the attack, it drops to only 8% after this initial period, and the loss rate falls to below 1%.[14] Thus, SimplyRep is able to quickly detect and isolate polluters also in the more realistic PlanetLab environment.

We also ran experiments in PlanetLab to assess the robustness of SimplyRep to a whitewashing attack. In this scenario, each polluter leaves and rejoins the system every $T_{ww} = 215$ s, as described in Section 4.2. Figs. 9a and b shows the results obtained with SimplyRep under whitewashing. For comparison purposes, we also show the results without whitewashing, i.e., the same results shown in Figs. 8a and b.

Note that whitewashing causes a significant impact on the system. In comparison with the scenario without whitewashing, the retransmission overhead experienced by peers after 30 min of transmission raises from 8% to around 29%. The chunk loss rate also increases, although remains at an acceptable rate (3.3% after 30 min). Thus, whitewashing is still a challenge for reputation systems. Nevertheless, despite the larger overhead and loss rates, we note that SimplyRep is reasonably robust to it as these results are even better than those obtained, *via simulation*,

[14] Despite some quantitative differences, particularly in the beginning of the attack, which are due to the longer network delays, the results after this initial period are very similar to those obtained with simulation.

with both black list and StRepS approaches under collusion (Fig. 5b).

6. Related work

The literature contains a great body of work on malicious and opportunistic behaviors in P2P systems. In this section, we discuss some previous efforts to analyze and fight these actions, focusing first on P2P resource sharing systems in general (Section 6.1), and then discussing specific studies on P2P live streaming systems (Section 6.2).

6.1. P2P Resource sharing systems

Various types of malicious and opportunistic behavior patterns have been widely studied in P2P resource sharing systems as well as other decentralized systems. For instance, various reputation mechanisms have been proposed to reduce the detrimental impact of free-riding in CPU-sharing grids [32] and selfish behavior in overlay networks [10].

Focusing particularly on pollution attacks in P2P file sharing systems, Liang et al. proposed an efficient measurement methodology for identifying and black listing the sources of pollution and estimating the levels of polluted content in P2P file sharing applications [4]. Similarly, Kamvar et al. proposed Eigentrust [33], a system that computes and maintains a global reputation for each peer based on the opinions of all other peers, weighting each opinion by the global reputation of its source. Eigentrust requires a set of pre-trusted peers, thus raising issues on its practical deployment.

There have also been various proposals of decentralized reputation systems for fighting pollution in these applications. For instance, in Credence [11], users assign reputations to the objects they download. The system is based on a distributed vote gathering protocol for disseminating the object reputations in the network, and on a correlation scheme that gives more weight to votes from like-minded peers. In Scrubber [9], in turn, peers assign reputations to other peers as sources of unpolluted content. A reputation is locally computed at a peer based on a linear combination of the individual experiences of the local peer and the testimony of a number of other peers. The testimony of other peers is used to allow the rehabilitation of passive polluters (i.e., peers that obliviously forwarded polluted content). Scrubber inherited some of its key components from a reputation system previously proposed to mitigate selfish behavior in overlay networks [10].

Extending Scrubber, the same authors also proposed a hybrid peer and object reputation system that combines the benefits of both strategies [9]. Similarly, in Xrep, peers vote on the authenticity of objects (polluted or not) and assign reputations to each other as sources of unpolluted content [12]. Reputation sharing is performed through a distributed polling algorithm by which resource requestors can assess the reliability of a resource offered by a participant before initiating the download.

More generally, PeerTrust is a reputation-based trust supporting framework for P2P systems, which includes an adaptive trust model that quantifies and compares the trustworthiness of peers based on a transaction-based feedback system [13]. Similarly, in [14], the authors proposed a distributed reputation mechanism to detect malicious or unreliable peers which combines local and aggregate opinions. The authors also discussed how to aggregate noisy (i.e., dishonest and inaccurate) votes using weighted majority techniques. Wang and Vassileva proposed a Bayesian network based trust model and a method for building reputation based on recommendations in P2P file/service sharing networks [15]. Like other methods [9,10,2], these three mechanisms combine local experiences with the opinions collected from other peers (i.e., network testimony) to compute a local reputation, weighting the opinions of other peers by some measure of their credibilities at the local peer.

We here are focused on P2P live streaming systems. Thus, although our new SimplyRep reputation system shares similarities with some of the aforementioned mechanisms [10,9] (particularly when it comes to how local reputations are updated), our method was designed with the characteristics of live transmissions (e.g., strict time constraints) in mind. In particular, unlike most discussed methods, it does not rely on aggregate opinions, but rather uses only individual experiences to compute local reputations. This was done to avoid the period of convergence of opinions, which could be too long for current transmission rates and, thus, could lead to great spread of pollution over the network.

6.2. P2P live streaming systems

In the specific context of P2P live streaming systems, Gheorghe et al. presented a survey of security and privacy related issues for these applications, discussing common attacks as well as security practices [34]. They also discussed general aspects and features that novel P2P streaming systems should consider in order to minimize the chances of an attack. Dhungel et al. reported results of an experiment in a real system showing that a single polluter may severely reduce the perceived streaming quality [1]. The authors also suggested some techniques to check the integrity of the data stream, aiming at automatically identifying polluted chunks. In [3], the authors carried out a formal analysis of content pollution and discussed its implications in P2P live video streaming systems. They proposed a probabilistic model to capture the progress of content pollution, validating it based on a real system. They showed that the number of passive polluters can grow exponentially, which can sharply decrease the effective bandwidth utilization due to the transmission of polluted chunks. They also showed that increasing the number of polluters does not necessarily lead to a faster progress of content pollution. Finally, they examined several techniques to fight content pollution, advocating for the use of a hash-based signature scheme.

As mentioned, a number of previous studies discussed various strategies to detect polluted chunks upon reception, exploiting techniques such as hash-based signatures, linear digests, star chaining and Merkle Tree chaining [1,3,16–18]. In particular, Haridasan and van Renesse proposed SecureStream [5,6], a P2P live streaming system implemented on top of an intrusion-tolerant membership protocol that addresses various types of attacks, including pollution (or forgery) and Denial-of-Service (DoS). SecureStream ensures the integrity of received data by applying a linear digests approach, where the hashes of n packets are computed and combined into a single digest packet, which is then signed by the sender. The signed message needs to be sent to the receivers prior to the dissemination of data that it corresponds to. This approach incurs in an overhead of one hash per packet plus the cost of a single signature/verification operation over n packets. Wong and Lam, in turn, proposed that the sender computed the hashes of a limited number of consecutive packets in the stream, using them as leaves in a Merkle Tree where each internal node consists of the hash of its children [16]. Each packet can be verified upon receipt, as it carries the signed root node and the hashes of all needed interior nodes in the path from the root to itself in the Merkle Tree. SimplyRep can be applied jointly with any existing method to detect polluted chunks. For evaluation purposes only, we here assumed a hash-based approach that allows us to check the integrity of each chunk immediately after it is received, although any other strategy, such as the ones discussed above, could be adopted.

Some other studies tackled the pollution attack in P2P live streaming systems by proposing reputation mechanisms. In [20], for instance, the authors proposed a black list based reputation system to detect malicious hosts, focusing particularly on the scenario where some hosts may lie by submitting forged reports to the black list server. They formulated the problem of computing a peer's reputation in the presence of lying hosts as a minimization problem, solving it using the Levenberg–Marquardt algorithm. In [2], we proposed and evaluated StRepS, a decentralized reputation system that computes the reputation of a peer i with respect to a peer j based on the individual experience of j with i as well as the individual experiences of j's partners (network testimony) with i. StRepS shares great similarities with Scrubber [9], particularly in terms of its key components, with adaptations to the context of live streaming. Both StRepS and the system proposed in [20] update the reputation of a peer based on the fraction of polluted/corrupted chunks received, a definition that we keep in our design of SimplyRep (see Eq. (1)).

In [35], the authors focused on a particular type of pollution attack, namely a typhoid adware attack, where polluters only partially alter the content by, for instance, inserting advertisements. They studied the impact of a pollution attack in popular streaming models, under various network settings and configurations. They observed that the feasibility of the attack is sensitive to the speed at which an attacker can modify content. Finally, they also discussed some defense mechanisms, including hash-based verification and a decentralized reputation system based on a Bayesian trust model.

Hu and Zhao proposed a trust management system to identify attackers in P2P live streaming systems [8]. They also investigated possible attacks against the trust management system and analyzed its robustness to those attacks. In [7], the same authors proposed a joint pollution detection and attacker identification system that relies on the previously proposed trust management system to quickly identify polluters. They considered different strategies to detect polluted chunks aiming at investigating the tradeoff between pollution resistance and system overhead. One such strategy was based on a combination of Merkle-Tree chaining, Reed-Solomon based authentication and early decoding,[15] whereas the other was based on an adaptive early decoding strategy.

In [21], Kang and Wu proposed to fight pollution attacks in P2P streaming systems by introducing a trust management system that, like other reputation systems (including StRepS), combines both individual and aggregate trust estimates, referred to as direct and indirect trusts, using a linear function. One interesting aspect of the proposed method is a confidence factor that can dynamically adjust the weights given to the direct and indirect trusts. The authors also proposed a new method to model the direct trust, which unlike our approach (Eq. (1)), has an exponential decay with the exponent being the number of polluted chunks received. The indirect trust, in turn, is similar to the computation of network testimony in StRepS (Eq. (3)).

Focusing on other types of malicious and opportunistic behaviors in P2P live streaming systems, Ripple-Stream [22] is a framework to improve the resilience of these systems to DoS attacks. It exploits existing credit systems to introduce credit constraints in the construction of the overlay, such that malicious nodes are pushed to the fringe of the network. Similarly, Oversight [36] aims at preventing both selfishness and DoS attacks by running a separate P2P download rate enforcement protocol applied to each participating peer. Finally, Tang et al. proposed an incentive mechanism for peer cooperation in live streaming that offers service differentiation to users with different contributions [37]. The system was evaluated on PlanetLab.

Our focus here is on quickly detecting and isolating content polluters. To that end, we proposed a novel decentralized reputation system which, unlike existing approaches, rely only on the individual experiences of each peer, and thus is not impacted by possible divergences among the opinions of multiple peers. To allow peer rehabilitation, we also proposed a decentralized dynamic reputation threshold mechanism such that each peer can independently adapt its local threshold of minimum reputation based on whether it perceives that the system is under attack or not. To our knowledge, our approach is original and, based on the results discussed in Section 5, leads to superior performance in comparison to both a centralized black list and a decentralized reputation method that uses both individual and aggregate opinions to compute reputations (e.g., StRepS).

7. Conclusions and future work

P2P live streaming has become a popular method of transmitting live content. Due to its popularity, it may be the target of attacks and opportunistic behavior. In this

[15] Received chunks are decoded before their playback time so that polluted chunks are detected as early as possible.

article, we studied a particular type of attack, a pollution attack, where polluters collude and forge data. We showed that simply relying on a method to detect and discard polluted chunks is not an effective strategy, as polluters remain active in the system, leading to great retransmission overhead. Effective anti-pollution techniques for live streaming must quickly isolate polluters, and should be simple and flexible. To that end, we here presented a new decentralized and light-weight reputation system, namely SimplyRep, to identify and penalize attackers as fastly as possible. Unlike most previous reputation mechanisms for P2P systems, SimplyRep relies only on individual experiences of a peer to compute the reputation of its partners, and thus is more robust to divergences in the opinions of multiple peers with respect to a given participant. We also designed a mechanism to dynamically adjust the minimum reputation threshold of each peer, which is important to allow peer rehabilitation. Our experimental results, based on both simulation and experiments on PlanetLab, indicate that SimplyRep greatly outperforms both a centralized black list and a previously proposed decentralized reputation system, particularly in case of a collusion attack from a reasonably large number of polluters, being also very robust under dissimulation attacks. Our results also show that SimplyRep is reasonably resilient to the challenging whitewashing attack.

A lot of work remains to be done, but the results obtained here are promising and display the potential of our simple approach to fight attacks in P2P live streaming systems. Possible directions for future work include further evaluating SimplyRep with larger and more dynamic peer populations, considering other metrics of streaming quality in our evaluation, as well as investigating alternative heuristics to compute the local reputations and to dynamically adjust the minimum reputation thresholds.

Acknowledgements

This research is partially funded by the Brazilian National Institute of Science and Technology for Web Research (MCT/CNPq/INCT Web Grant No. 573871/2008-6), and by the authors' individual grants from CNPq, CAPES and FAPEMIG.

References

[1] P. Dhungel, X. Hei, K. Ross, N. Saxena, The pollution attack in P2P live video streaming: measurement results and defenses, in: Proc. of the SIGCOMM Peer-to-Peer Streaming and IP-TV Workshop, 2007, pp. 323–328.

[2] A.B. Vieira, J.M. Almeida, S.V.A. Campos, Fighting pollution in P2P live streaming systems, in: Proc. of the IEEE International Conference on Multimedia and Expo (ICME), 2008, pp. 481–484.

[3] S. Yang, H. Jin, B. Li, X. Liao, H. Yao, X. Tu, The content pollution in peer-to-peer live streaming systems: analysis and implications, in: Proc. of the International Conference on Parallel Processing, 2008, pp. 652–659.

[4] J. Liang, N. Naoumov, K. Ross, Efficient blacklisting and pollution-level estimation in P2P file-sharing systems, in: Proc. of the 1st Asian Internet Engineering Conference on Technologies for Advanced Heterogeneous Networks (AINTEC), 2005, pp. 1–21.

[5] M. Haridasan, R. van Renesse, Defense against intrusion in a live streaming multicast system, in: Proc. of the 6th International Conference on Peer-to-Peer Computing (P2P), 2006, pp. 185–192.

[6] M. Haridasan, R. van Renesse, SecureStream: an intrusion-tolerant protocol for live-streaming dissemination, Computer Communications 31 (3) (2008) 563–575.

[7] B. Hu, H. Zhao, Joint pollution detection and attacker identification in peer-to-peer live streaming, in: Proc. of the IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP), 2010, pp. 2318–2321.

[8] B. Hu, H. Zhao, Pollution-resistant peer-to-peer live streaming using trust management, in: Proc. of the 16th IEEE International Conference on Image Processing (ICIP), 2009, pp. 3057–3060.

[9] C. Costa, J.M. Almeida, Reputation systems for fighting pollution in peer-to-peer file sharing systems, in: Proc. of the 7th IEEE International Conference on Peer-to-Peer Computing (P2P), 2007, pp. 53–60.

[10] B. Rocha, V. Almeida, D. Guedes, Increasing QoS in selfish overlay networks, IEEE Internet Computing 10 (3) (2006) 24–31.

[11] K. Walsh, E.G. Sirer, Experience with an object reputation system for peer-to-peer filesharing, in: Proc. of the 3rd Conference on Networked Systems Design & Implementation (NSDI), 2006.

[12] E. Damiani, D. di Vimercati, S. Paraboschi, P. Samarati, F. Violante, A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks, in: Proc. of the 9th ACM Conference on Computer and Communications Security, 2002, pp. 207–216.

[13] L. Xiong, L. Liu, PeerTrust: supporting reputation-based trust for peer-to-peer electronic communities, IEEE Transactions on Knowledge and Data Engineering 16 (7) (2004) 843–857.

[14] B. Yu, M. Singh, K. Sycara, Developing trust in large-scale peer-to-peer systems, in: Proc. of the IEEE 1st Symposium on Multi-Agent Security and Survivability, 2004, pp. 1–10.

[15] Y. Wang, J. Vassileva, Trust and reputation model in peer-to-peer networks, in: Proc. of the 3rd International Conference on Peer-to-Peer Computing (P2P), 2003, pp. 150–157.

[16] C. Wong, S. Lam, Digital signatures for flows and multicasts, IEEE/ACM Transactions on Networking 7 (4) (1999) 502–513.

[17] R. Gennaro, P. Rohatgi, How to sign digital streams, in: Proc. of the 17th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO), 1997, pp. 180–197.

[18] R.C. Merkle, A digital signature based on a conventional encryption function, in: Proc. of the Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology (CRYPTO), 1987, pp. 369–378.

[19] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, M. Bowman, PlanetLab: an overlay testbed for broad-coverage services, ACM SIGCOMM Computer Communication Review 33 (2003) 3–12.

[20] X. Jin, S. Chan, W. Yiu, Y. Xiong, Q. Zhang, Detecting malicious hosts in the presence of lying hosts in peer-to-peer streaming, in: Proc. of the IEEE International Conference on Multimedia and Expo, 2006, pp. 1537–1540.

[21] X. Kang, Y. Wu, Fighting pollution attack in peer-to-peer streaming networks: a trust management approach, Information Security and Privacy Research 376 (2012) 537–542.

[22] W. Wang, Y. Xiong, Q. Zhang, S. Jamin, Ripple-stream: safeguarding P2P streaming against DoS attacks, in: Proc. of the IEEE International Conference on Multimedia and Expo, 2006, pp. 1417–1420.

[23] S. McCanne, S. Floyd, K. Fall, The LNBL Network Simulator. <http://www-nrg.ee.lbl.gov/ns/>.

[24] B.M. Waxman, Routing of multipoint connections, IEEE Journal on Selected Areas in Communications 6 (9) (1988) 1617–1622.

[25] A. Medina, A. Lakhina, I. Matta, J. Byers, BRITE: an approach to universal topology generation, in: Proc. of the 9th International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2001.

[26] O. Heckmann, M. Piringer, J. Schmitt, R. Steinmetz, Generating realistic ISP-level network topologies, IEEE Communications Letters 7 (7) (2003) 335–337.

[27] A. Vlavianos, M. Iliofotou, M. Faloutsos, BiToS: enhancing BitTorrent for supporting streaming applications, in: Proc. of the 25th IEEE International Conference on Computer Communications, 2006, pp. 1–6.

[28] Y. Zhou, D. Chiu, J. Lui, A simple model for chunk-scheduling strategies in P2P streaming, IEEE/ACM Transactions on Networking (TON) 19 (1) (2011) 42–54.

[29] A. Borges, P. Gomes, J. Nacif, R. Mantini, J.M. Almeida, S. Campos, Characterizing SopCast client behavior, Computer Communications 35 (8) (2012) 1004–1016.

[30] X. Hei, Y. Liu, K.W. Ross, IPTV over P2P streaming networks: the mesh-pull approach, IEEE Communications Magazine 46 (2) (2008) 86–92.

[31] R. Caceres, N. Duffield, S. Moon, D. Towsley, Inferring Link-Level Performance from End-to-End Multicast Measurements, Global Internet, Rio de Janiero.
[32] N. Andrade, F. Brasileiro, W. Cirne, M. Mowbray, Discouraging free riding in a peer-to-peer CPU-sharing grid, in: Proc. of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC), 2004, pp. 129–137.
[33] S. Kamvar, M. Schlosser, H. Garcia-Molina, The eigentrust algorithm for reputation management in P2P networks, in: Proc of the 12th International Conference on World Wide Web (WWW), 2003, pp. 640–651.
[34] G. Gheorghe, R. Lo Cigno, A. Montresor, Security and privacy issues in P2P streaming systems: a survey, Peer-to-Peer Networking and Applications 4 (2) (2011) 75–91.
[35] E. Lin, D. de Castro, M. Wang, J. Aycock, SPoIM: a close look at pollution attacks in P2P live streaming, in: Proc. of the 18th International Workshop on Quality of Service (IWQoS), 2010, pp. 1–9.
[36] W. Conner, K. Nahrstedt, I. Gupta, Preventing DoS attacks in peer-to-peer media streaming systems, in: Proc. of the 13th Annual Multimedia Computing and Networking Conference (MMCN), 2006.
[37] Y. Tang, L. Sun, M. Zhang, S. Yang, Y. Zhong, A novel distributed and practical incentive mechanism for peer to peer live video streaming, in: Proc. of the IEEE International Conference on Multimedia & Expo (ICME), 2006, pp. 1533–1536.

Alex Borges Vieira received his Bachelors degree in Computer Science from the Universidade Federal de Minas Gerais (UFMG), in Brazil, in 2001. He received his M.S. degree in 2004 and his Ph.D. degree in the same institution in 2010. He is currently an Associate Professor at the Computer Science Department in the Universidade Federal de Juiz de Fora (UFJF), Brazil. His research interests are in the areas of P2P, dynamic networks and wireless sensor networks emphasis on system modeling and performance analysis.

Rafael Barra de Almeida received his Bachelor and Master degrees from the Computer Science Department from the Universidade Federal de Juiz de Fora, Brazil. His area of interest is computer networks, more specifically, in security in P2P live streaming systems.

Jussara Marques de Almeida has a Bachelor and Master degrees from the Computer Science Department of the Universidade Federal de Minas Gerais (UFMG), in Brazil. She also has an M.Sc. and a Ph.D. degrees from the University of Wisconsin-Madison, US. She is currently an Associate Professor of Computer Science at UFMG. Her research interests are primarily on performance modeling of large-scale distributed systems, workload and user behavior modeling, and social computing.

Sérgio V. Campos received the Bachelors degree in Computer Science from the Universidade Federal de Minas Gerais (UFMG), Brazil in 1986, the Masters degree in Computer Science from the same university in 1990 and the Ph.D. degree in Computer Science from Carnegie Mellon University in 1996 on the topic of formal verification of real-time systems. He is currently an Associate Professor at the Computer Science Department of UFMG. His interests include design, analysis and verification of real-time systems, hardware and software in general.