DIP: Distributed Identification of Polluters in P2P live streaming

ROSSANO GAETA, MARCO GRANGETTO and LORENZO BOVIO, Università di Torino

Peer to peer live streaming applications are vulnerable to malicious actions of peers that deliberately modify data to decrease or prevent the fruition of the media (pollution attack). In this paper we propose *DIP*, a fully distributed, accurate and robust algorithm for the identification of polluters. *DIP* relies on checks that are computed by peers upon completing reception of all blocks composing a data chunk. A check is a special message that contains the set of peer identifiers that provided blocks of the chunk as well as a bit to signal if the chunk has been corrupted. Checks are periodically transmitted by peers to their neighbors in the overlay network; peers receiving checks use them to maintain a factor graph. This graph is bipartite and an incremental belief propagation algorithm is run on it to compute the probability of a peer being a polluter. Using a prototype deployed over PlanetLab we show by extensive experimentation that *DIP* allows honest peers to identify polluters with very high accuracy and completeness even when polluters collude to deceive them. Furthermore, we show that *DIP* is efficient requiring low computational, communication, and storage overhead at each peer.

Categories and Subject Descriptors: C.2.4 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems—*Distributed applications*

General Terms: Security, Design, Experimentation

Additional Key Words and Phrases: peer-to-peer, pollution attack, malicious node identification, P2P streaming, belief propagation, statistical inference, PlanetLab

ACM Reference Format:

Rossano Gaeta, Marco Grangetto, and Lorenzo Bovio. 2013. DIP: Distributed Identification of Polluters in P2P live streaming. *ACM Trans. Multimedia Comput. Commun. Appl.* 00, 00, Article 00 (2014), 20 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Pollution attacks [Dhungel et al. 2007; Liang et al. 2005] represent a powerful threat for peer-to-peer live streaming architectures [Zhang et al. 2005; Huang 2007]: malicious peers (the polluters) can modify data on purpose with the aim of decreasing or preventing the fruition of the media. To protect these systems from pollution attacks identification of polluters is the key issue: it relies on the capability of detecting polluted content and it enables attackers isolation/removal.

In peer-to-peer live streaming architectures the media to be streamed is organized in independently playable data units called chunks; each chunk is divided in small transmittable data units called

Author's address: Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, 10149, Torino, Italy email: first.last@di.unito.it;

DOI:http://dx.doi.org/10.1145/0000000.0000000

blocks. If a chunk is obtained from only one uploader then identification of polluters is greatly simplified by assuming a chunk integrity check is available, e.g., a verifiable hash associated to each chunk. Nevertheless, frequent trusted publication of hash-based identifiers for each chunk incurs a notable overhead for peer-to-peer live streaming.

Identification of polluters becomes quite complex when chunks are obtained by collecting blocks from more than one neighbor peer (multi-part download) because an honest node can only suspect multiple uploaders when a chunk is detected as corrupted. Effective solutions in this scenario must be:

—scalable since P2P live streaming systems can comprise millions of participants,

—accurate to allow for correct identification of misbehaving peers and to minimize damage to honest ones,

—robust to guarantee high performance against several types of attacks even in highly dynamic peer settings,

—and efficient requiring low computational, communication, and memory costs for implementation.

In this paper we propose *DIP*, a fully distributed, accurate, robust, and efficient algorithm for the identification of polluters that is based on peer monitoring. In *DIP* peers are required to collect information on the state of the chunks they obtain from their neighbors. Information is summarized in *checks* that are computed by peers upon completing reception of a chunk. A check is a special message that contains the set of peer identifiers that provided blocks of the chunk as well as a bit to signal if the chunk has been corrupted. The main contributions of this paper are:

—identification of polluters has been recast as an *inference* problem [MacKay 2003] from a given number of checks. The goal of the inference is the estimation of the hidden state of the peers, i.e. being polluters or honest, given a set of observations corresponding to the checks. Each check can be interpreted as an accusation raised against a set of peers providing blocks by a witness. In *DIP* each node periodically transmits/receives checks to/from peer neighbors in the overlay network; peers receiving checks use them to maintain a bipartite graph (called factor graph) on which an incremental belief propagation algorithm [Yedidia et al. 2005] is run to compute the probability of a peer being a polluter;

—definition of a fully distributed algorithm to allow each honest peer to infer polluters identity by exploiting both own checks and those shared by their neighbors;

—thorough evaluation of the performance of *DIP* carried out by conducting experiments on our rateless codes based P2P live streaming architecture called ToroVerde Streaming (TVS) [Magnetto et al. 2010]. The usage of rateless codes allowed us to detect chunk corruption by exploiting the rateless decoding algorithm thus avoiding the overhead represented by hashes. Using a prototype deployed over PlanetLab we show by extensive experimentation that *DIP* allows honest peers to identify polluters with very high accuracy and completeness even when polluters collude to deceive them. Furthermore, we show that *DIP* is efficient requiring low computational, communication, and storage overhead at each peer.

As a final remark, please note that the *DIP* inference core can be customized for any system where multi-part download and a mechanism for detecting corruption of data units is available.

The paper is organized as follows: Section 2 describes the P2P live streaming architecture we consider, an overview of *DIP* protocol, and the attack model; Section 3 describes the formalization of *DIP* and presents the detailed solution; Section 4 discusses the performance results; Section 5 discusses previous work that is related to ours, and finally Section 6 discusses the key finding of this paper and outlines directions for future research. To ease the task of the reader, Table I summarizes all the notation introduced in the following sections.

Table I. : Paper notation.

Parameter	Description
T_h	Time between transmission of most recent checks
T_{BP}	Time between runs of the incremental BP
w	Size of the time sliding window
T_S	Threshold to classify peers
$S_{i,h,t}$	Suspect counter of peer i for peer h at time t
$z_{\mathcal{U}}$	Average number of peers per check
$z_{\mathcal{C}}$	Average number of checks per peer
N	Overlay size
N_P	Number of polluters
p_{poll}	Pollution intensity
p_{lie}	Lie intensity
vbr	Video bitrate
k	Number of blocks composing a chunk
B_b	Block size
N_{neigh}	Average number of neighbors

2. SYSTEM DESCRIPTION

In this section we define the peer-to-peer live streaming architecture we consider, the *DIP* protocol we designed, and the attack model describing polluters actions. Presentation is general and rather abstract; Section 3 will describe all details related to the statistical inference method we devised while a practical P2P application, based on rateless codes [Magnetto et al. 2010], including the corruption detection feature will be used as a testbed in Section 4.

2.1 The peer-to-peer architecture

In this paper we consider a peer-to-peer streaming system composed of N peers. The content is a video whose bitrate is vbr that is originated from a streaming server. The content is organized in *chunks* carrying independently reproducible audio/video units. Chunks are further partitioned in k *blocks* whose size is B_b bytes that represent the smallest data unit that can be transferred between two peers. Peers organize in a random mesh overlay network and blocks are exchanged among neighbor peers; the average size of one peer's neighborhood is N_{neigh}. As soon as a peer has collected all blocks of a chunk, it is able to feed its local player with the obtained data. Every peer shares with the neighbors only blocks of its completed chunks. The architecture we consider is completed by a rendez-vous point, i.e., a tracker. The task of the tracker is to provide joining peers with the address of a random subset of participants in order to start following the protocol to obtain the streamed content. The tracker also assigns peers an overlay-wide unique identifier. We assume this identifier is unforgeable therefore although peers are allowed to alternate between connection and disconnection periods they retain their unique identifier over successive connections to the overlay. It follows that we do not consider the case of a sybil attack where polluters exploit several identities to pursue their goals: this is an assumption made in all related works on this topic (see Section 5). Furthermore, countermeasures to sybil attack and whitewashing that can be used to improve the security of *DIP* have been proposed in the literature [Levine et al. 2006; Dinger and Hartenstein 2006; Gupta and Singh 2013].

2.2 The *DIP* protocol

DIP has been designed to counteract a polluting attack in a completely distributed fashion. As for the attack model we assume that N_P polluters join the overlay network. Polluters are malicious peers that follow the streaming protocol for overlay organization and data exchange but deliberately modify the payload of blocks that are uploaded to the neighbors. The effect of this nasty action is to invalidate chunk reconstruction thus preventing the fruition of the original content.

The *DIP* identification mechanism is based on the concept of *check*, that is a report created by every peer when a chunk has been completed. Every check contains a variable length list of peer identifiers that provided blocks of a chunk and a binary flag declaring the chunk as polluted or not. To this end, we assume that peers are able to detect if the chunk is polluted and the check flag is set to true in this case (in Section 4 we describe a practical detection technique that we used in our experiments). Chunks that are detected as polluted are not shared with the other peers to avoid a potentially quick propagation of bogus data. *DIP* goal is to discover which chunk contributors are misbehaving by exploiting the information carried by checks. In order to build a completely distributed system each peer, besides accumulating the checks from its local downloading operations, gossips in the neighborhood its most recent checks. In particular, the *DIP* protocol is run independently by each peer of the overlay and it is characterized by the following operations:

—parallel downloading of multiple blocks from the overlay, pollution detection for a completed chunk, and check creation on every chunk completion;

—checks storage in a local buffer;

—periodic checks sharing in the neighborhood: once every T_h seconds a peer sends its most recent checks stored in its local buffer to all its neighbors. The local buffer is then emptied to ensure that innovative checks are sent each time;

—statistical inference on stored checks (a peer exploits both checks it creates and checks received from its neighbors) for polluter identification.

The last item represents the main computational core of *DIP* where a list of suspected peer is computed as presented in details in Section 3. Such a list could be used to implement reactive actions by honest peers, e.g., honest peers can exploit the results of identification to ban polluter from their neighborhood (local blacklisting), to discard ongoing downloads from them, and to prune future positive checks by removing other honest peers to shrink the set of suspected peers. Global blacklisting could also be considered although we believe it would require special solutions to counteract malicious behavior of polluters with respect to a central authority. In this paper we only focused on the performance of *DIP* in identifying polluters and we leave the evaluation of countermeasures as future works.

2.3 The attack model

Polluters may try to escape identification by misbehaving in several ways. In this paper we considered the following attacks scenarios.

—Polluters *modify* the payload of a block non deterministically: on each block transmission opportunity, a coin is flipped and with probability p_{poll} (that we denote as the *pollution intensity*) the content of the block is randomly modified.

—Polluters *churn* by alternating between connection and disconnection periods. Since we assume the identifier is unforgeable churning polluters retain their unique identifier over successive connections to the overlay.

—Polluters *lie* on the checks they send to their neighbors: since the goal of the inference is the identification of polluters given a set of observations (corresponding to the checks) an effective misleading action for polluters is to let them introduce noise in checks. Indeed, it is well known that the accuracy of the belief propagation algorithm (the inference tool we use) is affected by noisy (erroneous) checks. In our context, noisy checks are obtained by modifying the check flag, an action that we termed as

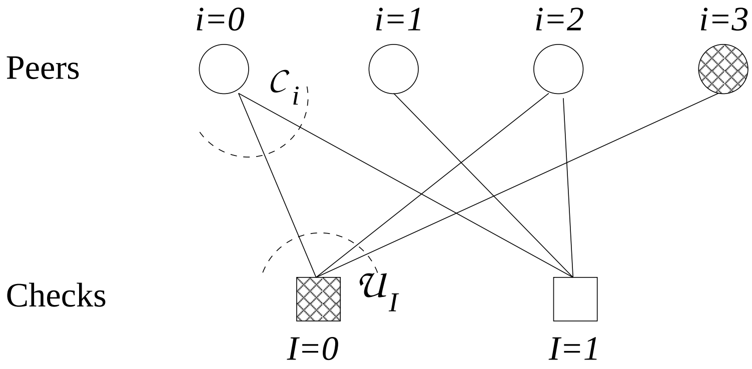

Fig. 1: Example of factor graph.

lying. We considered both random lying and intelligent lying by collusion. In the first case, each polluter flips a coin and with probability p_{lie} the check flag is inverted. Clearly, when $p_{lie} = 1$ lying is not random anymore. In the second case, we assume that polluters collude by setting to true the check flag if the set of peers providing blocks of a chunk does not contain any other polluter (a disparage action towards honest peers). Analogously, they set the check flag to false if at least another polluters has provided blocks for the chunk (a protective action for other polluters).

Of course, there could be other intelligent strategies for lying. Nevertheless, by using random lying with $p_{lie} = 1$ every single check sent out by polluters is a fake thus maximizing the amount of noisy data for the belief propagation inference tool. It turns out that this scenario represents an upper bound for any other lying strategy.

3. *DIP* STATISTICAL INFERENCE

The core of the identification mechanism used in *DIP* exploits statistical inference to estimate the probability of a peer being a polluter. Every peer is able to set up a statistical inference problem, whose inputs are both the checks available from the chunks it completes and those sent by its neighbors, and whose outputs are probabilities (of being a polluter) associated to every known peer. The goal is the estimation of the hidden state x_i of the i-th pees, i.e. being a polluter ($x_i = 1$) or not ($x_i = 0$), given the set of collected checks. The I-th check reports a list of peer identifiers $\mathcal{U}_I$ that have jointly uploaded blocks of a given chunk and the corresponding pollution detection flag c_I, with $c_I = 1$ (positive check) in case of pollution detection and $c_I = 0$ (negative check) in case of successful decoding.

Such problem can be graphically represented as a bipartite graph $(\mathcal{U}, \mathcal{C}, \mathcal{E})$ called *factor graph*, consisting of peers $i \in \mathcal{U}$, checks $I \in \mathcal{C}$ and undirected edges $\{i, I\} \in \mathcal{E}$ between i and I. An edge interconnects peer i and check I if and only if the list $\mathcal{U}_I$ contains the i-th peer. An example of factor graph (constructed by an arbitrary peer) connecting four peers (circles) and two checks (square) is shown in Figure 1. The circle filled with a pattern (the rightmost peer in Figure 1) represents a polluter that, in turn, determines a positive check (filled square) involving the true polluter and two innocent peers that were used in parallel to download a given chunk. Clearly, every peer can be included in a variable number of checks, depending on the overlay structure and protocol behavior; in the following the set of checks that peer i is involved in will be referred to as $\mathcal{C}_i$.

In *DIP* we are interested in detecting polluters based on the collected checks; given the previous factor graph interpretation, this amounts to the estimation of the hidden state of the nodes, i.e. $x_i = 1$ for polluters and $x_i = 0$ for honest nodes, from the observable variables, i.e. the checks. Clearly, a negative check ($c_I = 0$) makes it likely that all the corresponding uploaders are honest and, on the contrary, a positive check ($c_I = 1$) states that at least one of the involved uploaders is a polluter. This inference problem can be recast as the estimation of the marginal probability $P(x_i)$, defined as the probability of peer i being a polluter, from all the available checks $\mathcal{C}$. This goal can be achieved with the well-known *Belief Propagation* (BP) algorithm [MacKay 2003; Yedidia et al. 2005], that is a practical solution to compute marginal probabilities defined on a factor graph. It is worth noting that computing marginal probability is in general very hard since it requires summing an exponentially large number of terms of the corresponding joint probability. BP is an iterative algorithm based on the exchange of conditional probability updates, called the *beliefs* or *messages*, along the edges of the factor graph. The message from the peer i to the check I can be considered as a statement about the probability of the check being positive or negative. In turn, the message from the check I to the peer i can be interpreted as a statement about the probability of the peer being a polluter. The BP algorithm defines the rules allowing to iteratively refine such messages so as to converge to the exact evaluation of the useful marginal probability in the case of direct acyclic graphs, e.g. Bayesian networks. Nonetheless, BP provides robust and accurate estimates also in the case of arbitrary graphs [Yedidia et al. 2005; Weiss and Freeman 2001] as in our scenario. BP has been already proposed in [Gaeta and Grangetto 2013] for P2P polluters identification by a centralized monitor. On the contrary, in *DIP* , a completely distributed approach is followed, where each peer keeps modifying dynamically its own factor graph using both self-generated and collected checks.

In *DIP* every peer constructs is own factor graph as follows. Each peer keeps receiving checks and discovering the peer identifiers carried by the checks. BP inference is run every T_{BP} seconds, considering only the checks received in a time window of the past w seconds. The purpose of this windowed approach is twofold. First of all, it allows one to keep the factor graph updated when the P2P overlay network changes dynamically, e.g., in presence of peer churn. Second, the memory required for the storage of the factor graph remains limited during the peer lifetime. In the following, we will use notation $(\mathcal{U}, \mathcal{C}, \mathcal{E})_{t,w}$ to identify the factor graph updated at time t and constructed using the checks stored during a time window w by a given peer. The BP iterative algorithm is run on $(\mathcal{U}, \mathcal{C}, \mathcal{E})_{t,w}$ to estimate $P_{t,w}(x_i = x)$, i.e. the probability of peer i being in state $x = 0, 1$ at time t and window w.

The BP algorithm is defined as follows. The message pair m^x_{iI}, $x = 0, 1$ is associated to each directed edge from peer i to check I. The message m^x_{iI} represents the probability that peer i is in state x, given the information collected via checks other than check I, i.e. checks $I' \in \mathcal{C}_i \setminus I$. In turn, the message pair m^x_{Ii}, $x = 0, 1$ is associated to each directed edge from check I to peer i. The message m^x_{Ii} represents the probability of check I reporting pollution detection c_I, if peer i is considered in state x and all the other uploaders states, i.e. $x_{i'} : i' \in \mathcal{U}_I \setminus i$, have a separable distribution given by the probabilities $m^x_{i'I}$. BP algorithm is based on iterative application of the so called check and node passes. In check pass all messages directed from checks to the peers are refined according to the following equation:

$$m^x_{Ii} = \sum_{\{x_{i'}:i' \in \mathcal{U}_I \setminus i\}} P\left(c_I | x_i = x, \{x_{i'} : i' \in \mathcal{U}_I \setminus i\}\right) \prod_{i' \in \mathcal{U}_I \setminus i} m^{x_{i'}}_{i'I} \tag{1}$$

where $\{x_{i'} : i' \in \mathcal{U}_I \setminus i\}$ is used to denote the set of states of the uploaders of check I other than i. This message depends on the probability of observing check flag c_I, given the states of all the uploaders and their pollution intensities. Since the pollution intensity of an attacker is in general unknown, we assume $p_{poll} = 1$ in all the following derivations. This simplifies the combinatorial nature of Equation 1 allowing for an efficient implementation and relieving honest nodes from knowing actual values of p_{poll} used by polluters. Clearly, this assumption yields an approximation whenever polluters use $p_{poll} \neq 1$ as will be the case in some our experiments in Section 4.

Under the above assumption, a check is negative if all its uploaders are not malicious and a check is positive if at least one of the uploaders is malicious and we obtain:

$$m_{Ii}^{x} = \begin{cases} \prod_{i' \in \mathcal{U}_I \setminus i} m_{i'I}^{0} & \text{if } c_I = 0, x = 0 \\ 0 & \text{if } c_I = 0, x = 1 \\ 1 - \prod_{i' \in \mathcal{U}_I \setminus i} m_{i'I}^{0} & \text{if } c_I = 1, x = 0 \\ 1 & \text{otherwise} \end{cases} . \tag{2}$$

that can be used to update m_{Ii}^{x} (message from check to peer) as a function of m_{iI}^{x} (message from peer to check). Messages m_{iI}^{x} are whether computed from previous iterations, as detailed in the following, or initialized setting $m_{iI}^{0} = m_{iI}^{1} = 0.5$ before the BP computation over a factor graph that involves peer i for the first time. This event will happen when peer i is discovered for the first time as an uploader in a certain check and it amounts to initially consider all peers as equally likely to be polluters.

In turn, the updated values of m_{Ii}^{x} are used to refresh the messages m_{iI}^{x} (node pass) by using equation:

$$m_{iI}^{x} = \prod_{I' \in \mathcal{C}_i \setminus I} m_{I'i}^{x}. \tag{3}$$

In the *DIP* implementation previous check and node passes are iterated 3 times on every factor graph before getting marginal probabilities that are computed from messages m_{Ii}^{x} as follows:

$$P(x_i = x)_{t,w} = \prod_{I' \in \mathcal{C}_i} m_{I'i}^{x}. \tag{4}$$

Due to the adopted approximations previous BP equations are not guaranteed to yields normalized probabilities, e.g. $P(x_i = 0) + P(x_i = 1) = 1$; therefore, in practical implementations they are usually scaled to easy the interpretation of the results and to avoid numerical issues. In Table II some significant numerical results, obtained running the proposed BP estimation algorithm on the toy factor graph shown in Figure 1, are reported. In particular, the table shows the values of the messages from the (positive) check $I = 0$ to the i-th node computed by Equation 2, the messages from the node $i = 0$ to the I-th check computed by Equation 3 and the partial estimates of the probability of being a polluter $P(x_i = 1)$ computed by Equation 4. The values of such terms are reported for 3 iterations of the algorithm, where iteration 0 represents the initialization phase. It can be observed that in the case of the simple factor graph in Figure 1 the BP algorithms converges to the exact solution, i.e. node $i = 3$ is correctly recognized as a polluter with probability 1, in just 2 iterations.

Every T_{BP} s every peer h infers thorough BP a set of suspect peers $L_{h,t}$ that is obtained by setting a threshold on the estimated probability of being a polluter $P_{t,w}(x_i = 1) \geq \eta$. Moreover, every peer i known by h is associated with a suspect counter $S_{i,h,t}$, and the peers that belong to $L_{h,t}$ have their counter increased by 1 at time t. Finally, the i-th peer is considered a polluter by h as soon as $S_{i,h,t} \geq T_S$, where T_S is a proper threshold. To summarize, each peer in the overlay network keeps monitoring all the other peers that it has downloaded from or it has somehow known indirectly though other checks and is able to perform polluter identification in a completely distributed fashion.

The computational complexity of the BP inference phase depends on the evaluation of Equations 2, 3, and 4. Given a certain check state c_I, Equation 2 permits to compute two messages m_{Ii}^{0} and m_{Ii}^{1} at the cost of $|\mathcal{U}_I| - 1$ multiplications. Since messages are associated to each edge of the factor graph we can conclude that the overall check pass takes on average $|\mathcal{E}|(z_{\mathcal{U}} - 1)$, $z_{\mathcal{U}}$ being the average number of peers involved in a check. The node pass (Equation 3) is performed on every edge from any peer to its checks with a computational cost of $|\mathcal{E}|(z_{\mathcal{C}} - 1)$ multiplications, where $z_{\mathcal{C}}$ is the average number of checks per peer. Analogously, the final estimate (Equation 4) takes $|\mathcal{E}|z_{\mathcal{C}}$ multiplications.

Table II. : BP computation example on factor graph in Figure 1.

BP iteration	BP message	i/I			
		0	1	2	3
0	$m^0_{I=0,i}$	-	-	-	-
	$m^0_{i=0,I}$	0.5	0.5	-	-
	$P(x_i = 1)$	-	-	-	-
1	$m^0_{I=0,i}$	0.43	-	0.43	0.43
	$m^0_{i=0,I}$	1.0	0.0	-	-
	$P(x_i = 1)$	0.0	0.0	0.0	**0.57**
2	$m^0_{I=0,i}$	0.33	-	0.33	0.0
	$m^0_{i=0,I}$	1.0	0.0	-	-
	$P(x_i = 1)$	0.0	0.0	0.0	**1.0**
3	$m^0_{I=0,i}$	0.01	-	0.01	0.0
	$m^0_{i=0,I}$	1.0	0.0	-	-
	$P(x_i = 1)$	0.0	0.0	0.0	**1.0**

Recalling that in *DIP* we use 3 check/node passes, the overall cost of each computation of the marginal turns out to be:

$$3\,[|\mathcal{E}|(z_{\mathcal{U}}-1)+|\mathcal{E}|(z_{\mathcal{C}}-1)]+|\mathcal{E}|z_{\mathcal{C}}. \tag{5}$$

4. EXPERIMENTAL RESULTS

In this section we describe the experimental testbed we used for our experimentation, the reference scenario we considered, the metrics we defined to assess the performance of *DIP*, and results showing its accuracy, completeness, robustness, and efficiency.

4.1 The experimental testbed

We evaluated the performance of our detection technique by conducting experiments on our rateless codes based peer-to-peer live streaming architecture called ToroVerde Streaming (TVS) [Magnetto et al. 2010]. As already mentioned in Section 1, the proposed identification mechanism is quite general and it is not constraint to a particular P2P application; nevertheless we believe that providing experimental evidence of its effectiveness within a real prototype represents an added value. TVS is mesh based architecture and exploits the following main idea: the content is organized in *chunks* composed by k blocks and instead of transmitting the original data block, LT [Luby 2002] coded blocks are created and forwarded by the peers. Coding has the property that the original chunk can be obtained by any set of $k \cdot (1+\epsilon)$ coded blocks (ϵ is known as the code overhead). LT blocks are encode only by peers that have already received the original chunk and consists in simple binary XOR operations among a random set of blocks. Decoding is carried out by solving a system of linear equations by a special purpose on the fly algorithm proposed in [Bioglio et al. 2009]. This decoder exhibits a very limited overhead of about 3% in our settings.

LT coding of chunks, besides simplifying the content delivery mechanism, can be exploited to detect polluted data without an external verification tool. Indeed, given a chunk, all the received coded blocks are required to belong to the same vector subspace defined by all the possible linear combinations of the original k data packets. A polluter that wants to prevent the LT decoding of the chunk will modify the coded packet. We exploit the property that a polluted packet does not represent a valid combination of the original data to perform detection. A receiver is able to detect pollution as soon as an inconsistency is found in the solution of the underlying system of linear equations. Given the sub-optimality of the random coding approach (overhead $\epsilon > 0$), some redundant coded blocks are always received. Every

Table III. : Parameters setting in the reference scenario.

Parameter	Value
η	0.99
N	2000
N_P	100
p_{poll}	0.5
p_{lie}	1
vbr	600 kbps
k	80
B_b	1330 bytes
N_{neigh}	50 ($[40 - 60]$)
Upload bandwidth	42% 256 kbps, 40% 768 kbps, 18% 2 Mbps

redundant block is recognized as linearly dependent on the previously received ones. A redundant coded block must be re-obtained by XORing a subset of the already received blocks. If this constraint does not apply the whole chunk is recognized as polluted. Unfortunately, the receiver is not able to identify the corrupted blocks but only that at least one of them as been maliciously manipulated. When a peer detects a polluted chunk, besides creating a positive check, it does not use it to produce novel coded blocks for the neighbors so as to limit the propagation of corrupted information.

A complete prototype implementing *DIP* has been developed in C++ and deployed over PlanetLab for experimentation. Among all the host available on PlanetLab only about 450 can be reliably used for experimenting since they are free of all the following problems: frequently unreachable due to downtimes and other issues, unable to reach (or be reached) by other Planetlab nodes, experienced a very bad connection quality, suffering from DNS problems, under very high load, varying SSH-keys, not enough disk space to write log files. It follows that the maximum size of the overlay in our experiment could not scale to our will: allocation of several peers processes on the same host allowed us to evaluate systems composed only of a few thousands of peers.

Although the number of reliable PlanetLab host limits the size of the system we can analyze, we preferred not to resort to simulation. This choice is based on the observation that *DIP* requires the knowledge of peer identities and the amount of coded blocks exchanged among them to decode a chunk at a given time. This implies that the actual interactions and the exact timing among peers in both the overlay management and data exchange have to be taken into account in a simulation. We believe that simulations would inevitably be very detailed (and complex) and therefore very close to a real implementation.

4.2 The reference scenario

The parameters characterizing our reference scenario are summarized in Table III. We conducted $N_{EXP} = 30$ independent trials for each scenario we considered and computed for all performance indexes the 95% confidence intervals (CI); each trial lasts for 1800s.

A realistic churn model should assume that a fraction of the users join the application and remain until the end, others permanently depart, new peers continuously join the network, and sessions durations are of at least a few minutes and drawn from an heavy-tailed distribution [Silverston et al. 2009]. Of course, there are plenty of ways to implement these criteria: to ease deployment on PlanetLab, we partition the N peers in two subsets: 20% are stable, i.e., they join the overlay network and stay connected until the end of the experiment, while the remaining 80% join the swarm, stay connected for an average active period equal to 90s (a uniform random variable in the range $[60, 120]$s) and permanently depart. Arrival of a new peer (a peer with a new identifier) is triggered by a permanent departure after an average delay equal to 20s. The $20 - 80$ partition is intended to represent session duration whose distribution exhibits a heavy tail. Polluters churn by alternating between active and idle periods with the same average lengths of non stable peers.

The TVS application has been run to stream a 600 kbps bitrate video using a 4.2 Mbps server with upload bandwidth distribution summarized in Table I and adapted from [Zhang et al. 2007] that yields a resource index equal to 1.3, i.e., there is 30% average extra bandwidth with respect to the video bitrate. Polluters upload bandwidth is equal to 768 kbps. All key parameters for the TVS application (video bitrate, block size and number of blocks per chunk, neighborhood size, upload bandwidth distribution) are included in Table III. We refer the reader to [Magnetto et al. 2010] for a detailed description of TVS and for the complete list of its parameters that have minor relevance in the context of this paper. Please note that ***DIP*** results would not be greatly affected (improved or worsened) by choosing other values for these parameters. Indeed, ***DIP*** relies on the most recently decoded chunks to create and send checks to neighbors; in other words, it relies on system throughput that is in turn dependent on the resource index. Therefore, scaling up both peers upload bandwidths and video bitrate while maintaining the same resource index would yield comparable system throughput hence comparable ***DIP*** performance.

4.3 The performance metrics

Let us consider the i-th trial and define $I^{(i)}_{poll}(p)$ as the indicator function that is equal to 1 if peer p is a polluter and 0 otherwise and let $w^{(i)}(p)$ denote the fraction of polluted chunks where polluter p was among the block providers; clearly, the larger $w^{(i)}(p)$ the higher the polluting activity of polluter p in trial i. If $w^{(i)}(p) = 1$ polluter p contributed to the corruption of *all* polluted chunks in trial i. On the opposite extreme, $w^{(i)}(p) = 0$ suggests that polluter p has been inactive during the experiment, e.g., because it never decoded a chunk. By definition, $w^{(i)}(p) = 0$ if p is an honest peer. We also define the indicator function $I^{(i)}_{T_S}(p, h, t)$ that is equal to 1 if $S^{(i)}_{p,h,t} \geq T_S$, i.e., if the suspect counter of peer p for honest peer h at time t has reached or exceeded the accuracy threshold T_S in trial i, and 0 otherwise.

The performance of ***DIP*** have been evaluated on the basis of several indexes. For each honest peer h we compute (sums are defined over all peers p such that $S^{(i)}_{p,h,t} > 0$):

—the *completeness* at time t defined as $c^{(i)}(h,t) = \frac{\sum w^{(i)}(p) I^{(i)}_{poll}(p) I^{(i)}_{T_S}(p,h,t)}{\sum w^{(i)}(p) I^{(i)}_{poll}(p)}$. We have that $\forall h, t : 0 \leq c^{(i)}(h,t) \leq 1$; this index represents the overall activity of polluters that have been identified by honest peer h at time t. For instance, $c^{(i)}(h,t) = 0.9$ means that the polluters identified at time t have been collectively involved in corrupting 90% of all polluted chunks corrupted by all polluters that honest peer h knows during the i-th trial[1].

—the *accuracy* at time t defined as $a^{(i)}(h,t) = \frac{\sum I^{(i)}_{poll}(p) I^{(i)}_{T_S}(p,h,t)}{\sum I^{(i)}_{T_S}(p,h,t)}$. We have that $\forall h, t : 0 \leq a^{(i)}(h,t) \leq 1$; this index can be interpreted as the probability that honest peer h correctly classifies a peer whose suspect counter reaches or exceeds the accuracy threshold T_S as a true polluter.

We compute ***DIP*** completeness at time t ($c^{(i)}(t)$) by averaging $c^{(i)}(h,t)$ over all honest peers. Similarly, we compute ***DIP*** accuracy at time t ($a^{(i)}(t)$) by averaging $a^{(i)}(h,t)$ the same way. Finally, the values of $c(t)$ and $a(t)$ are obtained by averaging all values of $c^{(i)}(t)$ and $a^{(i)}(t)$ over the N_{EXP} trials. All graphs we present show $a(t)$ and $c(t)$ as a function of the peer lifetime, i.e., the time elapsed since their joining

[1] In general, the number of polluter that an honest peer knows about during a trial is less than or equal to N_P because polluters churn.

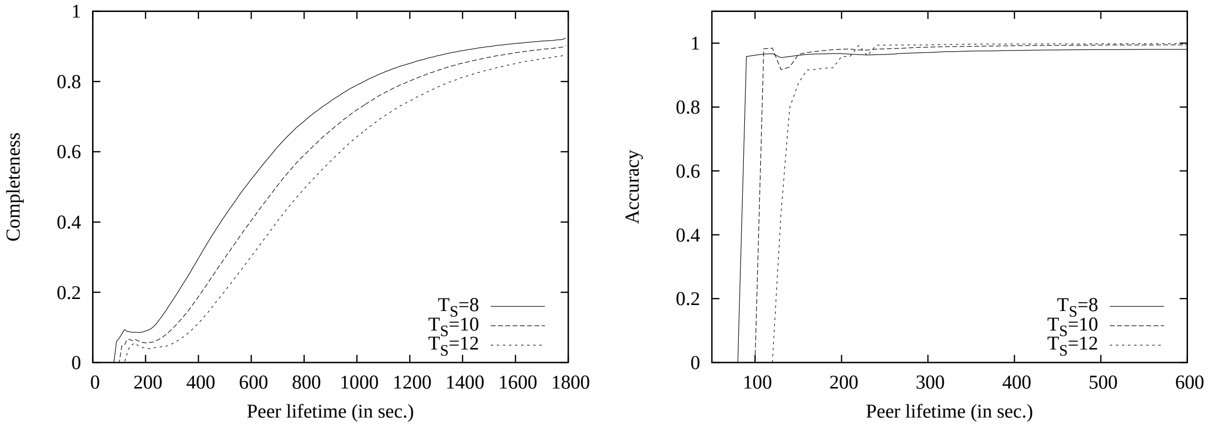

Fig. 2: $c(t)$ (left) and $a(t)$ (right) as a function of peer lifetime for different values of T_S.

the overlay network. In other words the performance indexes are always presented as a function of the local clock of every peer.

We also estimate the average and compute 95% confidence intervals for the CPU time required for every BP run, communication overhead, number of checks, peers, and edges of the factor graph in the reference scenario over all the N_{EXP} experiments. For the size of the factor graph we also report the maximum values

4.4 *DIP* parameters selection

In all the following experiments ***DIP*** employs the BP algorithm with 3 iterations; after every BP iteration the set of suspect peers $L_{h,t}$ is obtained by setting the threshold on the pollution probability $\eta = 0.99$. Indeed, a low threshold probability η would yield too many false positives, i.e., honest nodes erroneously identified as malicious. On the other hand, a high threshold probability would increase the number of false negatives. Since ***DIP*** periodically runs the BP estimation and update a suspect ranking we observed that setting $\eta = 0.99$ represents a good compromise to balance the two issues mentioned above.

DIP is characterized by four additional parameters: the analysis window size (w in the range $[30, 90]$s), the analysis time interval (T_{BP} in the range $[5, 15]$s), the time between successive transmissions of the latest reconstructed chunks (T_h in the range $[5, 20]$s), and the threshold T_S to classify peers in the set of suspects $L_{h,t}$ as a true polluter. The first step has been to explore the performance of ***DIP*** for all combinations of these parameters. We observed that too small or too large values for w yield the worst performance: indeed, small window sizes do not allow the factor graph to include enough checks to accurately infer the peer status. On the other hand, large values of w increase the number of loops in the factor graph which, in turn, impact on the accuracy of the probability estimates yielded by the BP algorithm [Weiss and Freeman 2001]. The value that yielded the best performance is $w = 60$s. Performance were slightly affected by T_{BP} and T_h therefore we selected the values $T_{BP} = 10$s and $T_h = 15$s to strike a balance between computational and communication overhead and reactivity. The selection of T_S has been found to have a significant impact on the performance and therefore it will be analyzed in details in the following. A low value allows for high reactivity (each peer can start classifying peers in L_h after $T_{BP} \cdot T_S$ s) but accuracy can be low since many misclassified honest peers can be erroneously

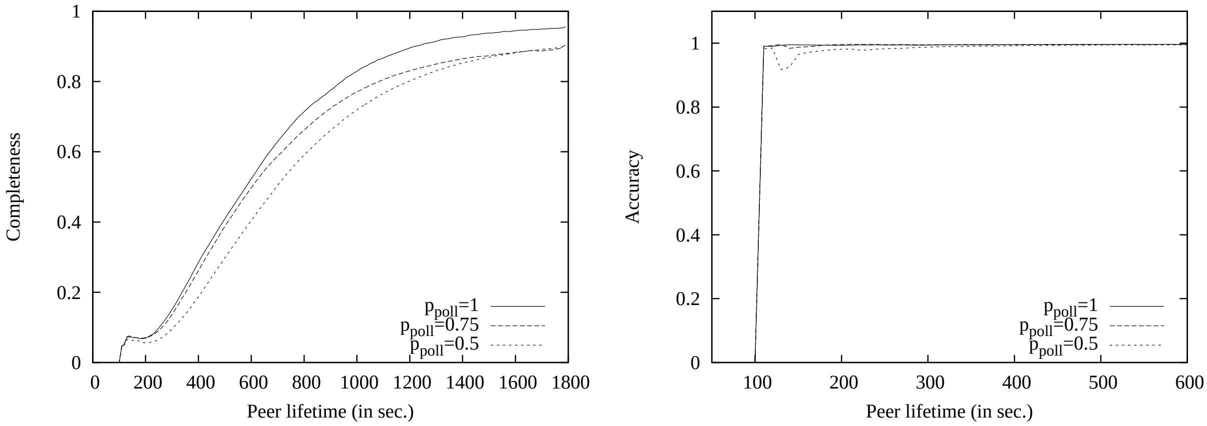

Fig. 3: $c(t)$ (left) and $a(t)$ (right) as a function of peer lifetime for different polluting intensities.

inferred as polluters. To show the trade-off Figure 2 depicts $a(t)$ (left) and $c(t)$ (right) for several values of T_S. We selected $T_S = 10$ as a balance between accuracy, completeness, and reactivity.

It can be noted that throughout all the experiments presented in the paper $c(t)$ curves rise and after a short plateau they rise again. Similarly, $a(t)$ curves rise and after a small dip they rise again. The particular shape of $c(t)$ and $a(t)$ curves is due to the churn model we adopted. Indeed, both are composed of a first part from time 0 to the maximum lifetime for short connections and a remaining part for longer peer lifetimes (stable connections). The resulting curves are then a weighted average of two different peer classes. The weigh of short connections is much greater than the weigh of long (stable) connections. Indeed, the average number of short connections in our experiments can be computed as $0.8 \cdot N \cdot \frac{1800}{90+90}$ where the fraction represents the average number of times a peer joins the overlay network during the experiment. On the other hand, the number of long (stable) connections is only $0.2 \cdot N$.

4.5 Experimental evaluation of *DIP* accuracy and robustness

In this section we present results from a set of experiments to show that *DIP* is accurate and robust with respect to pollution intensity, misleading actions performed by polluters, and colluding attacks. Moreover, we show how *DIP* performance scale for larger system sizes and overall number of polluters. To this end all the following results are worked out by using the parameters values selected in Section 4.4.

4.5.1 *Polluting intensity.* First of all we analyze the effect of the pollution intensity p_{poll} as defined in Section 2.3 on the performance of *DIP* . Figure 3 shows that both *DIP* accuracy and completeness are very high and increase as the pollution intensity gets close to 1. Lowering p_{poll} has virtually no impact on $a(t)$ (it reaches 1 a few minutes after an honest peer joins the overlay network) while it slightly reduces $c(t)$; on the other hand, a lower pollution intensity translates into lower damage for the honest peers. It can be noted that the pollution intensity yields a natural trade-off between speed of identification and average damage caused to honest nodes, i.e., the lower p_{poll} the lower the number of polluted chunks and consequently the number of positive checks. Since positive checks represent accusations against polluters it also follows that the polluters identification mechanism depends on the number and the rate of positive checks produced in the system; clearly, time to identification gets

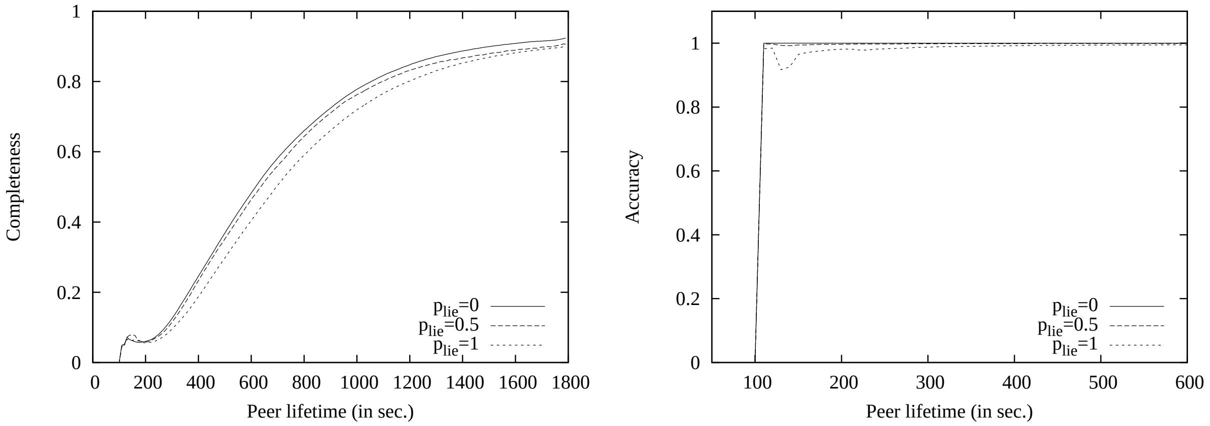

Fig. 4: $c(t)$ (left) and $a(t)$ (right) as a function of peer lifetime for different lying intensities.

longer for lower pollution intensity. From the reported results, we can conclude that polluters may slow down (but not avoid) identification only by using a *very* low pollution intensity at the price of imposing a smaller damage to the system.

4.5.2 *Lying intensity.* As described in Section 2.3, it is well-known that the accuracy of the BP algorithm is affected by noisy (erroneous) checks that in our attack model polluters create by lying, i.e., by modifying the check flag with probability p_{lie}. Figure 4 shows that polluters cannot shield from being identified by lying. For all the possible range of p_{lie} ***DIP*** accuracy reaches 1 and completeness exceeds 0.9. Indeed, by setting $p_{lie} = 1$ the amount of noise polluters can add is maximized; it follows that lying on every check ($p_{lie} = 1$) is the best (but not enough) polluters can do to hide from honest peers.

4.5.3 *Increasing the number of polluters.* All previous results have been obtained in the reference scenario comprising $N_P = 100$ polluters. An interesting and important step to assess the accuracy and completeness of ***DIP*** is to evaluate its performance as the number of polluters increases. To this end, Figure 5 shows ***DIP*** performance for N_P ranging from 50 to 300, i.e., 15% of honest peers. It can be noted that accuracy still reaches 1 in all cases with completeness exceeding 0.8 (it gets close to 1 for longer experiments). When N_P increases the damage to the system shows up as a reduced throughput, i.e., number of chunks completed in the time unit. It follows that less checks are available to peers that run the ***DIP*** protocol on factor graphs whose average number of checks is lowered, thus slightly slowing down its performance. Nevertheless, we conclude that ***DIP*** can satisfactorily handle configurations with a large number of polluters.

4.5.4 *Increasing the system size.* We conducted experiments for larger systems to evaluate how ***DIP*** performance scale as the system size increases. To overcome the lack of a large number of reliable PlanetLab nodes we ran several peer processes on the same PlanetLab host to increase the overall system size.

Figure 6 shows ***DIP*** performance for the reference scenario and for $N = 3000$ and $N_P = 150$, i.e. we increased the size of the system while keeping the polluter density fixed to the 5% as in the reference scenario. It is worth noticing that the larger system turns out to be more complex from the point of view of polluter identification task since, locally, every honest peer is affected by a larger number of

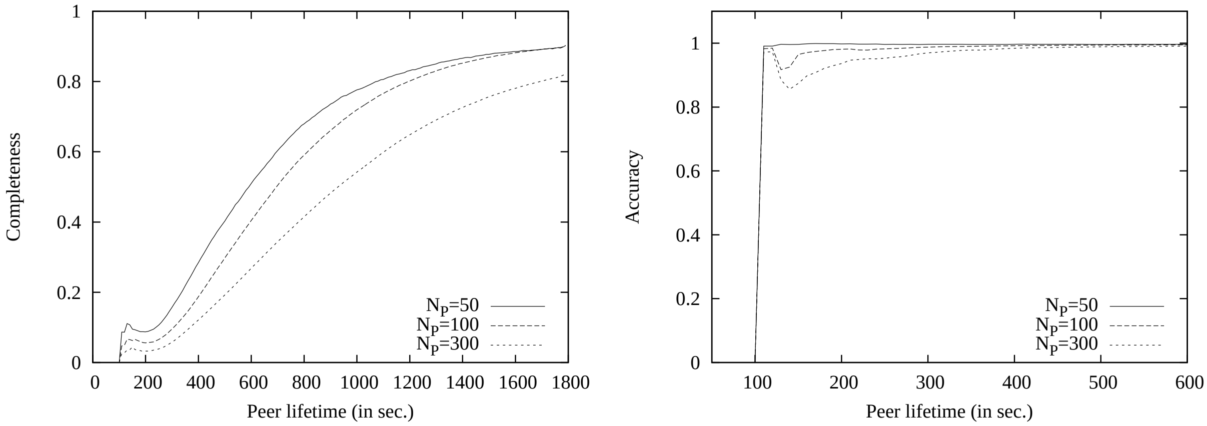

Fig. 5: $c(t)$ (left) and $a(t)$ (right) as a function of peer lifetime for increasing N_P.

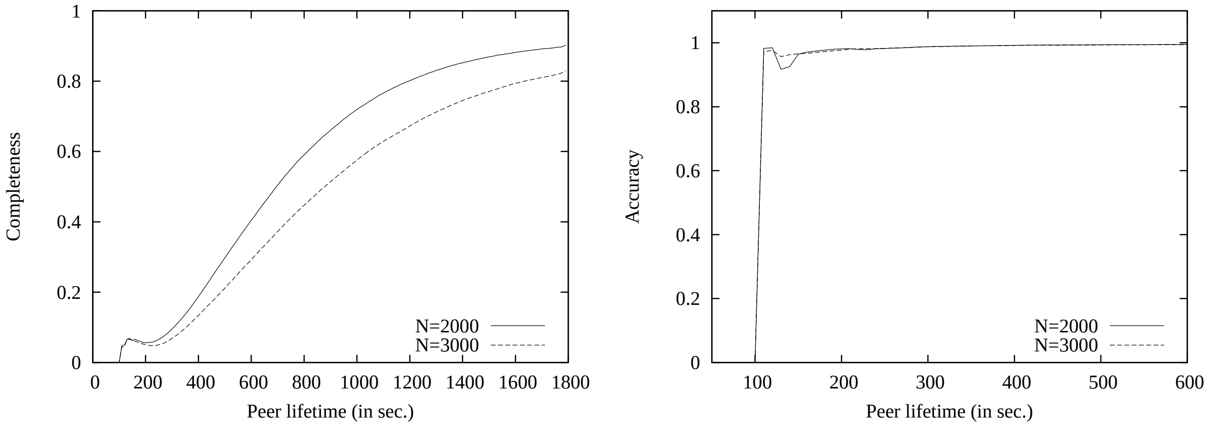

Fig. 6: $c(t)$ (left) and $a(t)$ (right) as a function of peer lifetime for increasing N.

polluters. This phenomenon is due to the fact that the network size is finite and relatively small. In our settings each peer has an average neighborhood size of 50; therefore, the average number of second neighbors is equal to $50 \cdot 49 = 2450$. This number is comparable to the system sizes N we considered and it follows that any peer knows almost all participants (including polluters) within two hops. It turns out that the number of polluters to be identified by every honest peer is larger in the case $N = 3000$ with respect to the reference scenario with $N = 2000$. On the other hand, the computational and communication efforts employed by every honest peer is kept constant in the two experiments, thus making the case $N = 3000$ a more difficult task. The set of malicious peers identified by honest peer h can be partitioned in two disjoint subsets: those that were neighbors of h in the overlay at least once during the experiment and those that h never knew of. The former set represents malicious peers that h can spot because they might have provided polluted blocks (direct polluters) while the latter set represents malicious peers h can identify only thanks to checks sent by its neighbors according

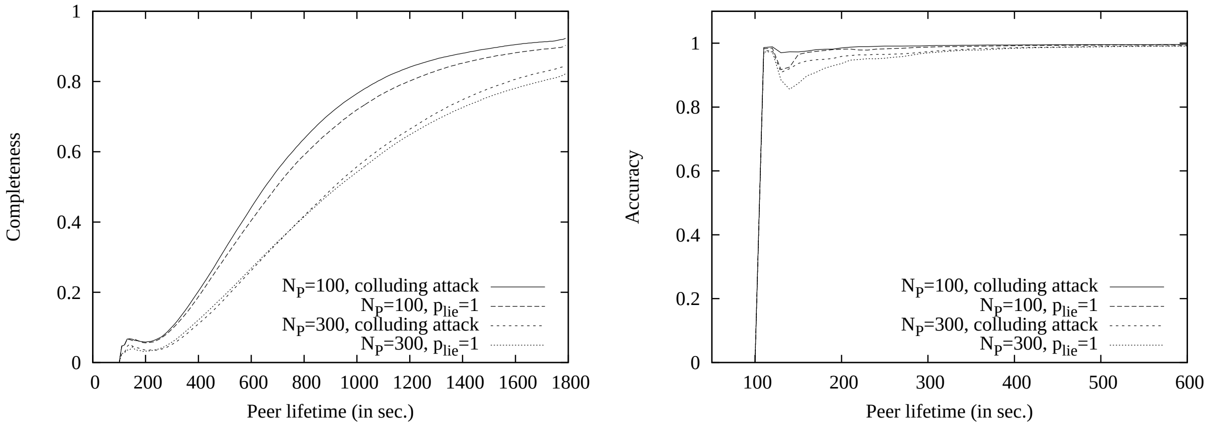

Fig. 7: $c(t)$ (left) and $a(t)$ (right) as a function of peer lifetime for colluding attack and increasing N_P.

to the DIP protocol (indirect polluters). Indeed, indirect polluters of h are the direct polluters of h's neighbors (the ”two hops away” polluters of h). In particular, we have noted that the number of direct polluters we obtained in our experiments are almost the same for $N = 2000$ and $N = 3000$: 19.9 and 20.5, respectively. Nevertheless, the number of indirect polluters we obtained in our experiments are 63.3 and 98.9 for $N = 2000$ and $N = 3000$, respectively. This experimental data clearly point out that the case $N = 3000$ represents a tougher scenario where more polluters must be identified.

Despite the larger number of polluters to be identified, the results in Figure 6 show that *DIP* accuracy remains high (it is equal to 1 also in the case $N = 3000$) whereas completeness reaches 0.84 (the completeness for $N = 2000$ reaches 0.89 therefore the relative difference is only 5.6%).

Indeed, the $N = 3000$ scenario we present in Figure 6 is qualitatively similar to that we considered in Figure 5, where we analyzed the reference scenario for several values of N_P, i.e. increasing the number of polluters for the same system size. In that case, we observed that DIP obtains good performance for all values of N_P we considered although for $N_P = 300$ more time is necessary to achieve completeness values similar to those of the reference scenario ($N_P = 100$). In light of this analysis we conclude that DIP can scale to larger system sizes allowing honest peer to identify polluters by both direct and indirect means. Furthermore, it is important to note that by using the same average amount of information (the number of checks sent by a peer according to DIP depends on the number of decoded chunks per time unit and on the number of neighbors in the overlay: they both remained the same for the values of N we considered) honest peers are able to identify more polluters in the same time window (the length of our PlanetLab experiments) for $N = 3000$.

4.5.5 *Colluding attacks.* In all previous analysis polluters tried to escape identification by introducing noise in the shape of fake checks, i.e., a check whose value is inverted with probability p_{lie}. As discussed in Section 2.3, we also considered a more sophisticated strategy; in particular, we devise a two-pronged attack where polluters upon sending checks to their neighbors send a positive check if the set of block providers does not contain any other polluter (a disparage action towards honest peers), and send a negative check if at least one of the block providers is another polluter (a protection for other polluters). Figure 7 compares the accuracy and completeness achieved in this case against that of the reference scenario for increasing N_P. It can be noted that accuracy is even slightly better in presence of two-pronged attack (it reaches 1 in all cases) while completeness slightly slows down for

$N_P = 300$ (it gets close to 1 for longer experiments, anyway). As discussed in the comment to results in Figure 4, setting $p_{lie} = 1$ maximizes the amount of noise polluters can add to mislead honest peers. On the other hand, the colluding attack does not maximize noise. Indeed, when a polluter sends a negative check if at least one of the block providers is another polluter (the protection for other polluters) it could send out an unmodified check if the decoded chunk is clean (this is the case when the polluters involved in the checks provided clean blocks due to $p_{poll} \neq 1$). It follows that the colluding attack does not maximize the amount of noisy data and yields (slightly) better performance. Nonetheless, performance improvements observed in Figure 7 are very small: for $N_P = 100$ the final rounded $c(t)$ values are 0.92 and 0.90 for the colluding attack and the $p_{lie} = 1$ case, respectively. For $N_P = 300$ the final rounded $c(t)$ values are 0.84 and 0.82 for the colluding attack and the $p_{lie} = 1$ case, respectively. We can therefore conclude that *DIP* turns out to be robust to collusion strategies.

4.6 Computational, communication, and memory costs

Here we analyze the communication, computational, and memory costs of implementing *DIP*.

- communication cost: the communication overhead of *DIP* can be upper bounded by the following reasoning. A check I contains an integer number representing the identifier of the sender, and integer number representing the number of peers providing blocks of the chunk, an indicator of the chunk status (polluted or clean), and as many integers as the number of peers representing the provider identifiers. It follows that a check is represented in our prototype as a message whose payload size in bytes is equal to $B_c = 9 + 4|\mathcal{U}_I|$ where $\mathcal{U}_I$ is the set of peers that provided blocks for the chunk (integers and peer identifiers require 4 bytes while the chunk corruption flag is represented using one extra byte). The average size of a check is then equal to $\overline{B_c} = 9 + 4z_{\mathcal{U}}$.
  The time duration of a chunk is given by $T_{chunk} = \frac{(8 \cdot B_b \cdot k)}{1000vbr}$s. In an ideal, stable, system at steady-state (a system where peers do not leave and the throughput of each peer is equal to the throughput of the video server) the number of bytes sent by each peer to its neighborhood is given by $\frac{T_h N_{neigh} \overline{B_c}}{T_{chunk}}$ yielding an overhead throughput equal to $\frac{8 N_{neigh} \overline{B_c}}{1000 T_{chunk}}$ kbps.
  From the system parameters characterizing our reference scenario we obtain 6.98 kbps by assuming that $\overline{B_c}$ is computed with $z_{\mathcal{U}}$ obtained from Table IV. Please note that this is a loose upper bound since only 20% of the honest peers stays connected for the whole video duration while the remaining 80% disconnects (and reconnects as a new peer) after an average connection time equal to 90s. This also implies that not all peers succeeds to create a fully populated neighborhood and startup delays may reduce throughput. Furthermore, when polluters corrupt data the system throughput reduces, i.e., the number of decoded chunks decreases, hence the generation rate of new checks decreases accordingly.
  This results is in accordance with the average measured effective bandwidth in our experiment that is equal to 1.174 kbps with 95% confidence interval $[0.457, 1.892]$ that represents only 0.19% of the video bitrate. We conclude that the communication cost of *DIP* is *very* low and does not represent a limiting factor.
- storage cost: the size of the factor graph is a key element to evaluate the complexity and memory requirements of our technique. We measured the average number of checks, peers, and edges of the factor graph $(\mathcal{U}, \mathcal{C}, \mathcal{E})$ in the reference scenario in all N_{EXP} experiments. We obtained the results summarized in Table IV; it can be noted that the memory requirements of our technique are low for both average and maximum number of components of the factor graph.
- computational cost: Equation 5 in Section 3 evaluates the computational cost of BP as function of the average number of arcs ($|E|$), peers per check ($z_{\mathcal{U}}$) and checks per peer ($z_{\mathcal{C}}$). The measurements

Table IV. : Average and maximum sizes for checks, peers, and edges.

	Average	Maximum
peers	535.535 CI [208.3,862.7]	1629.1 CI [633.6,2624.3]
checks	355.2, CI [138.1,572.3]	1841.3 CI [716.2,2966.3]
edges	1398.5, CI [543.9,2253.1]	6922.9 CI [2692.7,11153.1]

reported in Table IV show that $z_{\mathcal{U}}, z_{\mathcal{C}} \ll |E|$ and allow us to conclude that the BP complexity is of the order $\mathcal{O}(|E|)$. For completeness we also measured the average CPU time required for every BP run in the reference scenario for all the N_{EXP} experiments: our C++ implementation takes on average 229.4 ms with 95% confidence intervals $[89.2, 369.6]$ on usually overloaded PlanetLab hosts running several instances of our prototype. Please note that such values are orders of magnitude smaller than the time between any two BP runs, i.e., T_{BP}.

5. RELATED WORK

Pollution attack to P2P streaming applications, where malicious peers inject bogus data in the network to deteriorate the streaming performance of all honest peers is a well-known issue. In [Lin et al. 2010] it is shown that a limited number of polluters can easily deteriorate the quality of a streaming application; a preliminary analysis of the possible defense mechanisms, whether based on cryptographic tools or reputation based approaches, is presented in [Lin et al. 2010], as well. As opposed to our work in [Lin et al. 2010] a pull-based P2P streaming system without coding is used and the focus is more on the analysis of the effects of pollution in P2P streaming, whereas known techniques are evaluated as possible defenses.

A reputation system is exploited in [Borges et al. 2008] to fight pollution attack in a standard P2P streaming application, where it is assumed that every peer can detect the malicious behavior of its neighbors; to this end it is assumed that a whole chunk is downloaded by a single peer and therefore a direct experience of a malicious behavior can be reported building up the reputation system. On the contrary, in this paper we face the more complex problem of polluter identification in presence of parallel downloading of the chunk from multiple peers. SymplyRep [Vieira et al. 2012] is another recent example of a design of a reputation system to counteract pollution in P2P streaming. Also in this case a chunk pull-based P2P approach is assumed and the proposed solution is evaluated using simulation and PlanetLab experiments.

Identification of polluters in non coded peer-to-peer systems has been dealt with in [Wang et al. 2010]. Alerts are sent to the video server and to the tracker upon detecting a corrupted chunk. The server computes and broadcasts a checksum of this chunk that peers use to identify the polluter. Peers notify the server about their suspects and true polluters cannot lie to it thanks to a non repudiation protocol. The scheme is quick in identifying the polluters. The drawbacks are many: the technique might not scale due to the centralization of the key operations in the server and the tracker, it is complex and relies on the existence of effective and efficient solutions to the broadcast authentication problem, and it is vulnerable to the corruption of the checksum originated by the server and/or the results of the detection. Furthermore, its performance have been addressed only by simulation of overlays with stable peers (churn is not considered), and the maximum number of polluters (50) represent only about 3% of the smallest overlay considered in the paper (1600 peers).

In [Jin and Chan 2010] a monitoring architecture for a reputation system that peers use to select neighbors is proposed. The focus of the paper is on reputation computation, storage and load balancing among monitoring nodes. The results show that the system is able to detect polluters up to a certain degree of lies. Nevertheless, the technique relies on the assumption that each peer is able to compute the amount of corrupted blocks received by each neighbor during a monitoring period. This capability is not available in the system we consider and it is the main motivation for building our technique. Furthermore, the monitoring architecture is assumed to be composed of trustworthy nodes and it is not clear how the performance of the technique would be if this assumption is relaxed.

In the area of push based P2P streaming systems, that generally rely on channel coding and parallel chunk downloading for faster information spreading, cryptographic or algebraic approaches have been proposed. On-the-fly verification and/or correction of blocks [Krohn et al. 2004; Gkantsidis and Rodriguez 2006; Yu et al. 2008; Kehdi and Li 2009; Yu et al. 2009; Ho et al. 2008; Jaggi et al. 2008; Koetter and Kschischang 2008] have been proposed in network coding based systems to identify polluters; their drawback are the computational cost, communication overhead due to pre-distribution of verification information, and the limit on the maximum amount of corrupted information.

The work in [Li and Lui 2010] presents a fully distributed detection algorithm and analyzes its performance. The technique is based on simple intersection operations performed by peers on the set of neighbors providing chunks: each peer starts with a set of suspects that is equal to its neighborhood that is shrunk as long as chunks are downloaded from a random subset of neighbors independently chosen from the whole neighborhood. The technique is analyzed when the number of polluters in the neighborhood is known in advance and an approximation is proposed when this quantity is unknown. The technique is applicable only under the (unrealistic) assumptions that the neighborhood of peers does not change over time and that each chunk is obtained by a randomly chosen subset of neighbors (neighbors to retrieve blocks from are usually selected with some non random criteria).

DIP follows the approach presented in [Gaeta and Grangetto 2013] although there are numerous and significant differences between them. In particular:

—*DIP* is fully distributed hence each peer autonomously infers identity of polluters. Instead, in [Gaeta and Grangetto 2013] inference is carried out by a set of special, globally known nodes (the monitor nodes) that add to the streaming infrastructure, i.e., the tracker node and the video server;

—monitor nodes in [Gaeta and Grangetto 2013] are assumed to be trusted and secure while no security assumptions are necessary in *DIP*. Indeed, the assumption that monitor nodes are trustworthy can be strong and can invalidate the accuracy and robustness characteristics of [Gaeta and Grangetto 2013] when one or more monitoring nodes behave maliciously or inconsistently;

—the number of monitor nodes in [Gaeta and Grangetto 2013] must be fixed a priori. A fixed number of monitoring nodes and a mapping between peer identifiers and reference monitor node avoids the conflicting behavior attack, i.e., the attack where a coalition of malicious peers might act differently depending on the monitoring node. On the other hand, scalability problems may arise during sudden rises in the number of participant peers since monitor nodes cannot be dynamically added or removed;

—in *DIP* each peer uses a threshold based criterion to spot malicious nodes while monitor nodes in [Gaeta and Grangetto 2013] relay on minimum amount of time before correct identification is guaranteed to occur with high probability;

—counter reaction in [Gaeta and Grangetto 2013] would probably be global blacklisting; this means that false positiveness could be an issue since an honest peer erroneously classified as malicious would be banned by all participant peers while in *DIP* such a peer would only be locally blacklisted by the honest peer that failed correct identification.

6. CONCLUSIONS AND FUTURE WORK

In this paper we proposed *DIP*, a fully distributed technique to allow honest peers to identify polluters launching a pollution attack in peer-to-peer live streaming. We recast this problem as a problem of statistical inference to estimate the probability of a peer being a polluter. At the heart of *DIP* lies the concept of check, i.e., a summary containing the set of peer identifiers that provided blocks of the chunk as well as a bit to signal if the chunk has been corrupted. Checks are computed by peers upon completing reception of a chunk and are exchanged among neighboring peers to maintain the factor graph (a bipartite graph) on which an incremental belief propagation algorithm is run to compute the probability of a peer being a polluter.

The technique has been evaluated using our rateless codes based peer-to-peer live streaming architecture; we extensively experimented with a prototype deployed over PlanetLab and we showed that *DIP* allows honest peers to identify polluters with very high accuracy and completeness even when polluters collude to deceive them. Furthermore, we show that *DIP* is efficient requiring low computational, communication, and storage overhead at each peer.

In this paper we only focused on the performance of DIP in identifying polluters. Future works will evaluate the effectiveness of reactive actions by honest peers, e.g., honest peers can exploit the results of identification to ban polluter from their neighborhood (local blacklisting), to discard ongoing downloads from them, and to prune future positive checks by removing other honest peers to shrink the set of suspected peers. Global blacklisting could also be considered although we believe it would require special solutions to counteract malicious behavior of polluters with respect to a central authority and to cope with the effects of false positives (false positiveness could be an issue to deal with since an honest peer erroneously classified as polluter would be banned by all participant peers in global blacklisting).

Furthermore, the current version of *DIP* performs only one hop gossiping although extension to a multi-hop gossiping scenario is possible by also devising solutions to cope with check modification made by polluters. Also, it would be interesting to blend *DIP* with trust management mechanisms in P2P networks to obtain more efficient and effective solutions to the pollution attack.

REFERENCES

V. Bioglio, R. Gaeta, M. Grangetto, and M. Sereno. 2009. On the Fly Gaussian Elimination for LT Codes. *IEEE Communication Letters* 13, 2 (Dec. 2009), 953–955.

Alex Borges, Jussara Almeida, and Sergio Campos. 2008. Fighting pollution in p2p live streaming systems. In *Multimedia and Expo, 2008 IEEE International Conference on*. IEEE, 481–484.

P. Dhungel, X. Hei, K.W. Ross, and N. Saxena. 2007. The pollution attack in P2P live video streaming: measurement results and defenses. In *Workshop on Peer-to-peer streaming and IP-TV, P2P-TV '07*. 323–328.

Jochen Dinger and Hannes Hartenstein. 2006. Defending the sybil attack in p2p networks: Taxonomy, challenges, and a proposal for self-registration. In *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*. IEEE, 8–pp.

Rossano Gaeta and Marco Grangetto. 2013. Identification of Malicious Nodes in Peer-to-Peer streaming: A Belief Propagation Based Technique. *IEEE Transactions on Parallel and Distributed Systems* 24, 10 (2013), 1994–2003.

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

Ruchir Gupta and Yatindra Nath Singh. 2013. Avoiding Whitewashing in Unstructured Peer-to-Peer Resource Sharing Network. *arXiv preprint* (2013).

Tracey Ho, Ben Leong, R. Koetter, M. Medard, M. Effros, and D.R. Karger. 2008. Byzantine Modification Detection in Multicast Networks With Random Network Coding. *Information Theory, IEEE Transactions on* 54, 6 (june 2008), 2798 –2803.

G. Huang. 2007. PPLive: A practical P2P live system with huge amount of users. In *Proceedings of the ACM SIGCOMM Workshop on Peer-to-Peer Streaming and IPTV Workshop*.

S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, M. Medard, and M. Effros. 2008. Resilient Network Coding in the Presence of Byzantine Adversaries. *Information Theory, IEEE Transactions on* 54, 6 (june 2008), 2596 –2603.

Xing Jin and S.-H. Gary Chan. 2010. Detecting malicious nodes in peer-to-peer streaming by peer-based monitoring. *ACM Trans. Multimedia Comput. Commun. Appl.* 6 (March 2010), 9:1–9:18. Issue 2.

E. Kehdi and Baochun Li. 2009. Null Keys: Limiting Malicious Attacks Via Null Space Properties of Network Coding. In *IEEE INFOCOM 2009*.

R. Koetter and F.R. Kschischang. 2008. Coding for Errors and Erasures in Random Network Coding. *Information Theory, IEEE Transactions on* 54, 8 (august 2008), 3579 –3591.

Maxwell N. Krohn, Michael J. Freedman, and David Mazieres. 2004. On-the-Fly Verification of Rateless Erasure Codes for Efficient Content Distribution. *Security and Privacy, IEEE Symposium on* (2004).

Brian Neil Levine, Clay Shields, and N Boris Margolin. 2006. A survey of solutions to the sybil attack. *University of Massachusetts Amherst, Amherst, MA* (2006).

Yongkun Li and John C.S. Lui. 2010. Stochastic analysis of a randomized detection algorithm for pollution attack in P2P live streaming systems. *Performance Evaluation* 67, 11 (2010), 1273 – 1288.

J. Liang, R. Kumar, Y. Xi, and K.W. Ross. 2005. Pollution in P2P file sharing systems. In *IEEE INFOCOM 2005*. 1174 – 1185.

E. Lin, D.M.N. de Castro, Mea Wang, and J. Aycock. 2010. SPoIM: A close look at pollution attacks in P2P live streaming. In *Quality of Service (IWQoS), 2010 18th International Workshop on*. 1–9.

M. Luby. 2002. LT codes. In *IEEE FOCS 2002*. 271–280.

D.J.C. MacKay. 2003. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.

Andrea Magnetto, Rossano Gaeta, Marco Grangetto, and Matteo Sereno. 2010. P2P streaming with LT codes: a prototype experimentation. In *ACM workshop on Advanced video streaming techniques for peer-to-peer networks and social networking, AVSTP2P '10*. 7–12.

Thomas Silverston, Olivier Fourmaux, Alessio Botta, Alberto Dainotti, Antonio Pescapé, Giorgio Ventre, and Kavé Salamatian. 2009. Traffic analysis of peer-to-peer IPTV communities. *Computer Networks* 53, 4 (2009), 470–484.

Alex Borges Vieira, Jussara Marques de Almeida, and Sérgio Vale Aguiar Campos. 2012. SimplyRep: A simple and effective reputation system to fight pollution in P2P live streaming. *Computer Networks* (2012).

Qiyan Wang, Long Vu, K. Nahrstedt, and H. Khurana. 2010. MIS: Malicious Nodes Identification Scheme in Network-Coding-Based Peer-to-Peer Streaming. In *IEEE INFOCOM 2010*. 1 –5.

Y. Weiss and W.T. Freeman. 2001. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Trans. on Information Theory* 47, 2 (feb 2001), 736 –744.

J.S. Yedidia, W.T. Freeman, and Y. Weiss. 2005. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Trans. on Information Theory* 51, 7 (2005), 2282 – 2312.

Zhen Yu, Yawen Wei, B. Ramkumar, and Yong Guan. 2008. An Efficient Signature-Based Scheme for Securing Network Coding Against Pollution Attacks. In *IEEE INFOCOM 2008*. 1409 –1417.

Zhen Yu, Yawen Wei, B. Ramkumar, and Yong Guan. 2009. An Efficient Scheme for Securing XOR Network Coding against Pollution Attacks. In *IEEE INFOCOM 2009*.

Meng Zhang, Qian Zhang, Lifeng Sun, and Shiqiang Yang. 2007. Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better? *Selected Areas in Communications, IEEE Journal on* 25, 9 (december 2007), 1678 –1694.

X. Zhang, J. Liu, B. Li, and T.S.P. Yum. 2005. CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming. In *IEEE INFOCOM 2005*. 13–17.