R. Gaeta, M. Grangetto

Identification of malicious nodes in peer-to-peer streaming: a belief propagation based technique

IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS (2013) 24

DOI: 10.1109/TPDS.2012.342

Identification of malicious nodes in peer-to-peer streaming: a belief propagation based technique

Rossano Gaeta and Marco Grangetto

Abstract—Peer-to-peer streaming has witnessed a great success thanks to the possibility of aggregating resources from all participants. Nevertheless, performance of the entire system may be highly degraded due to the presence of malicious peers that share bogus data on purpose. In this paper we propose to use a statistical inference technique, namely Belief Propagation, to estimate the probability of peers being malicious. The detection algorithm is run by a set of trusted monitor nodes that receives notification messages (checks) from peers whenever they obtain a chunk of data; these checks contain the list of the chunk uploaders and a flag to mark the chunk as polluted or clean. Peers are able to detect if the received chunk is polluted or not but, since multi-party download is employed, they are not capable to identify the source(s) of bogus blocks. This problem definition allows us to define a factor graph of peers and checks on which an incremental version of the Belief Propagation algorithm is run by the monitor nodes to infer the probability of each peer being a malicious one. We evaluate the accuracy, robustness, and complexity of our technique by running a real peer-to-peer application on PlanetLab. We show that the proposed approach is very accurate and robust against malicious nodes misbehaving (different pollution intensity, presence of fake checks, churning, and total un-cooperation from malicious nodes), increasing number and colluding behavior of malicious nodes.

Index Terms—peer-to-peer, pollution attack, malicious node identification, streaming, belief propagation, statistical inference, PlanetLab.

1 Introduction

Peer-to-peer streaming architectures represent a mature area of research with several successful examples to date [1], [2]. Nevertheless, these systems have an important Achilles' heel: they are vulnerable to attacks carried out by peers that spread bogus data over the entire overlay network. These actions are commonly known as *pollution attacks* [3], [4] and the attackers are termed as malicious nodes. There are basically three tasks that a peer-to-peer streaming architecture should implement to get rid of malicious nodes: polluted content detection, malicious node identification, and removal/isolation. The first task can be carried out by each peer; since many peer-to-peer streaming architecture organize the content to be streamed in independently playable media units (usually denoted as *chunks*) each peer upon reconstruction of a chunk can check its state (valid or polluted) by verifying if the specific formats of the streamed media are matched. The second task is necessary to limit the origin of the polluted data to avoid continuous degradation of the architecture performance. The third step is the final result of the whole process where malicious nodes are made innocuous to restore acceptable streaming quality.

Among the three operations we briefly outlined the second one is by far the most complex. Complexity arises because peers usually collect *blocks* of a chunk from several of their neighbors (the chunk *uploaders*). Techniques to verify on the fly the state of a single block [5], [6], [7], [8], [9], [10], [11] or to correct a polluted block [12], [13], [14] have been proposed but their effectiveness is hampered by substantial computational costs, communication overhead due to pre-distribution of verification information, and the amount of corrupted information.

• *Dipartimento di Informatica,Università di Torino, Corso Svizzera 185, 10149, Torino, Italy, Email: first.last@di.unito.it*

A viable solution to solve the problem of identification of malicious nodes in peer-to-peer streaming is *peer monitoring*. In this context peers are required to collect information on the state of the chunks they obtain from their neighbors and to rely on a monitoring infrastructure that is able to feed peers back with information on malicious nodes identities. Upon reception of the suspects identity each peer is able to disconnect from the sources of polluted data.

Our contribution

In this paper we propose a technique to identify malicious nodes in a peer-to-peer streaming architecture; we employ a set of N_M trusted monitoring nodes collecting reports (that we denote as *checks*) sent by peers that have downloaded blocks of a given chunk from a set of other peers. Each peer is assigned to only one monitor node for all the time it joins the application. The resulting architecture is thus a two-level hierarchy reminiscent of many peer-to-peer applications, e.g., [15], [16], allowing the whole system to scale to large sizes.

We recast the problem of identifying the malicious peers from a given number of checks as an *inference* problem [17]. The goal of the inference is the estimation of the hidden state of the peers, i.e. being malicious or not, given a set of observations corresponding to the checks.

Each check can be interpreted as an accusation raised against a set of uploaders by a witness. Checks and uploaders are used by monitor nodes to build a bipartite graph (called the *factor graph*) on which an incremental version of the *Belief Propagation* (BP) algorithm [18] is run to compute the probability of a peer being malicious. BP is an iterative algorithm based on the exchange of probability messages (the *belief*) along the edges of the factor graph.

We evaluate the accuracy and the cost of our technique by conducting experiments on a real peer-to-peer live streaming architecture that we developed and deployed over the Planetlab network. We consider a wide range of malicious node misbehavior as well as their colluding attack: we observed that our technique is very accurate in identifying malicious nodes in all settings and it is very quick to determine the identity of actual misbehaving nodes.

Please note that the technique we propose is general and is not tailored to a particular class of peer-to-peer streaming architecture; it can be applied to any system where data units are obtained by peers through a multipart download of smaller blocks. It can be applied to both coded and uncoded systems provided that a mechanism to detect corruption of data units is available. Actually, it could be applied to file sharing systems as well.

The paper is organized as follows: Sect. 2 discusses previous work that is related to ours, Sect. 4 describes the formalization of the malicious node identification and presents the detailed solution, Sect. 5 discusses the performance results of the BP based technique, and Sect. 6 discusses the key finding of this paper and outlines directions for future research.

2 Related work

Defending peer-to-peer streaming systems from pollution attacks has been the subject of numerous research activity. In the area of network coding several efforts have been devoted to devise on-the-fly verification techniques carried out by participants [5], [6], [7], [8], [9], [10], [11]. The major drawback of these elegant methods is the high computational costs for verification and the communication overhead due to pre-distribution of verification information. Error correction is another approach to deal with pollution attacks in network coding based peer-to-peer streaming [12], [13], [14]; these methods introduce coding redundancy to allow receivers to correct errors but their effectiveness depends on the amount of corrupted information.

Three papers are more closely related to our work:

- the work by Wang et al. [19] proposes a detection scheme where each peer is able to detect receipt of corrupted blocks by checking the adherence of the decoded chunk to the specific formats of the video stream. Peers detecting polluted chunks send alert messages to the video server and the tracker. Upon receipt of an alert the server computes a checksum of the original chunk and disseminates it to all peers in the overlay. The checksum is used by peers to identify which uploader actually sent a corrupted block. Peers report their suspects to the server and a true polluters cannot lie (the authors develop a non repudiation protocol to ensure that peers cannot lie when reporting suspects to the servers). Sequence numbers are used to tag alerts to deal with cycles in the overlay.
The scheme is effective in quickly identify malicious nodes. Nevertheless, it is complex and relies on the existence of effective and efficient solutions to the broadcast authentication problem. Furthermore, it is not clear how the technique performance is affected if malicious nodes corrupt the checksum originated by the server and/or the results of the detection. The analysis has been conducted only through simulation with stable nodes; the maximum number of malicious nodes is 50, i.e., about 3% of the smallest overlay considered in the paper (1600 nodes).
- the work by Li and Lui [20] presents a distributed detection algorithm and analyzes its performance. The technique is based on simple intersection operations performed by peers: each peer starts with a set of suspects that is equal to the entire neighborhood that is shrunk as long as chunks are downloaded from a random subset of uploaders independently chosen from the entire set of neighbors. The scheme allows malicious nodes to send corrupted blocks using a pollution probability. The technique is analyzed when the number of malicious nodes in the neighborhood is known in advance and an approximation is proposed when this quantity is unknown. The technique is attractive thanks to its simplicity and fully distributed nature. Performance deteriorates when multiple polluter may exist and it is not clear how good performance is when peers churn. Furthermore, the entire set of results is obtained under the assumption that each chunk is obtained by a randomly chosen subset of uploaders and it is not known how the technique would perform if this assumption is relaxed.
- the work by Jin et al [21] proposed a monitoring architecture to build and maintain a reputation system that peers use to select neighbors. 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 malicious nodes 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 uploader during a monitoring period. This capability is not available in the system we consider and it is the main motivation for building our BP based identification technique.

3 System description

In this paper we consider a peer-to-peer streaming system composed of N peers. The streamed content is organized in *chunks* encoding independently reproducible audio/video units. Chunks are identified by a progressive number assigned by the content *server*; they are further partitioned in *blocks* that represent the smallest data unit that can be transferred between two peers. Peers organize in an *overlay* network and blocks are exchanged among neighbor peers. As soon as a peer has collected all blocks that allow a chunk to be reconstructed the peer is able to feed its local player with the obtained data and it shares the newly obtained chunk with its neighbors in the overlay network. 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 based on their IP address and the port number used for communication. Peers are allowed to churn, i.e., to alternate between connection and disconnection periods; nevertheless, peers retain their unique identifier over successive connections to the overlay.

3.1 Attack model and proposed enriched architecture

The system comprises N_P *malicious* peers: these peers join the swarm and follow the application level protocol for overlay organization and data exchange but they can deliberately modify the payload of blocks that are forwarded to their neighbors. The effect of this nasty action is to invalidate chunk reconstruction of the receiving peers thus preventing them from reproducing the original content.

The architecture also comprises a set of N_M trusted *monitor* nodes. The tracker assigns each peer to only one monitor node based on the chosen overlay-wide unique identifier: if we assume that the peer identifier is an integer i_p the tracker assigns it to the monitor whose identifier is equal to $i_p \bmod N_M$. It follows that a peer can only report its checks to the same monitor node for the whole duration of the video stream. The resulting architecture is thus a two-level hierarchy reminiscent of many peer-to-peer applications, e.g., [15], [16], allowing the whole system to scale to large sizes. We do not propose a specific mechanism to let monitor nodes exchange information: monitor nodes may either organize into a higher level unstructured overlay or refer to a common trusted central authority node.

As soon as a peer reconstructs a chunk it is assumed that it is able to detect if the chunk is polluted or not. If the chunk is polluted it is not shared with the peer neighbors and a *positive check* is sent to the monitor assigned by the tracker. On the other hand, a *negative check* is sent to the monitor upon reconstructing a valid chunk. A check contains the list of peer identifiers that uploaded blocks of that chunk and a binary flag to indicate a positive or negative outcome.

Malicious peers may misbehave in several ways:

- they can *modify* the payload of a block: on each block transmission a coin is flipped and with probability p_{poll} (that we denote as the *pollution intensity*) the content of the block is randomly modified before transmission;
- they can *lie* when sending checks to the monitor node: we let each malicious node flip a coin and with probability p_{lie} the value of the check is inverted and with probability $1 - p_{lie}$ the check is faithfully reported;
- they can *avoid* sending checks to their monitor;
- they can *churn* by alternating between connection and disconnection periods.

Since peers retain their overlay-wide unique identifier through several joins and departures we do not consider the case of a sybil attack where malicious nodes exploit several identities to pursue their goals.

4 Belief propagation formulation

The problem of identifying the malicious peers from a given number of checks, i.e. reports sent by peers that have downloaded blocks of a given chunk from a set of other peers, can be recast as an inference problem. The goal of the inference is the estimation of the hidden state of the peers, i.e. being malicious or not, given a set of observations corresponding to the checks. Each check can be interpreted as an accusation raised against a set of uploaders by a witness. In this paper we adopt the *Belief Propagation* (BP) algorithm [22], [17], [18], [23], that has been used to solve, at least approximatively, a number of inference problems in many different fields, e.g., iterative channel decoding [24], Bayesian networks [22] and computer vision [25] to mention a few.

Uploaders and checks can be graphically represented by a *factor graph* $(\mathcal{U}, \mathcal{C}, \mathcal{E})$ that is a bipartite graph, consisting of the uploaders $i \in \mathcal{U}$ (the variable nodes in the BP literature), checks $I \in \mathcal{C}$ (the factor nodes in the BP literature), and undirected edges $\{i, I\} \in \mathcal{E}$ between i and I if and only if check I depends on uploader i. In the following we will refer to the set of uploaders involved in check I as $\mathcal{U}_I$ and the set of checks that peer i contributes as an uploader as $\mathcal{C}_i$. An example of factor graph with four uploaders (circle nodes) and two checks (square nodes) is show in Fig. 1. Each uploader i can be in one of two states $x_i = 1$ or $x_i = 0$, depending on whether uploader i *is* or *is not* a malicious peer. Each check can report one of two observations $c_I = 0$ or $c_I = 1$ in case of negative or positive pollution detection, respectively. Coming back to the example of Fig. 1 the filled circle is used to represent a polluter that in turn causes a positive check (filled square).

The BP algorithm can be used to estimate from the factor graph the so called variable marginals $(P(x_i))_{i \in \mathcal{U}}$, i.e. the probability of peer i being malicious.

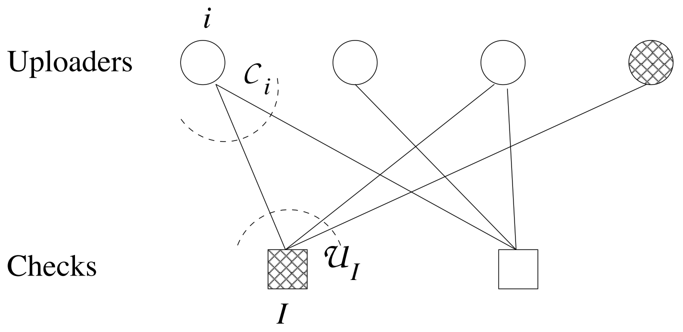


Fig. 1: Example of factor graph.

BP is an iterative algorithm based on the exchange of probability messages, the belief, along the edges of the bipartite graph. In our setting it is convenient to distinguish between two class of messages: message from uploader i to check I, m^x_{iI}, that is meant to be the probability that uploader i is in state x, given the information collected via checks other than check I ($\mathcal{C}_i \backslash I$); message from check I to uploader i, m^x_{Ii} is defined as the probability of check I having value c_I if uploader i is considered in state x and all the other uploaders states have a separable distribution given by the probabilities $\{m^x_{i'I} : i' \in \mathcal{U}_I \setminus i\}$. In our scenario we implement the BP algorithm iterating a few times the so called check pass, corresponding to the computation of all messages from checks to uploaders, followed by the node pass, where messages from uploaders to checks are computed and propagated.

The messages are initialized to the values $m^0_{iI} = m^1_{iI} = 0.5$, i.e. all peers are equally likely to be malicious in the first run. Then, the probabilities m^x_{Ii} are estimated using:

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

Equation (1) depends on the probability of observing a certain check value c_I, given the states of the uploaders of such check. We can compute the probability of observing a negative check $c_I = 0$, given a set of uploaders $\{x_i : i = 1, \ldots, k\}$ as:

$$P\left(c_I = 0 | \{x_i : i = 1, \ldots, k\}\right) = \prod_{x_i \neq 0} \left(1 - p_{poll}\right)^{u_{iI}} \tag{2}$$

where u_{iI} represents the number of packets of the chunk corresponding to the I-th check uploaded by peer i and p_{poll} is the pollution intensity. Previous equation represents the probability that all the uploaders that are supposed to be polluters ($x_i \neq 0$) do not pollute any of the u_{iI} uploaded blocks.

In general, the actual pollution intensity of the malicious nodes is not known by the monitor nodes. Therefore, we simplify Equation 2 by setting $p_{poll} = 1$. It follows that, when $p_{poll} = 1$, a check is negative if and only if all the uploaders are not malicious:

$$P\left(c_I = 0 | \{x_i : i = 1, \ldots, k\}\right) = \begin{cases} 1, & \text{if } x_i = 0, \forall i \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Analogously, a check is positive as soon as at least one of the uploaders is a malicious node. Therefore we get:

$$P\left(c_I = 1 | \{x_i : i = 1, \ldots, k\}\right) = \begin{cases} 0, & \text{if } x_i = 0, \forall i \\ 1, & \text{otherwise} \end{cases} \tag{4}$$

Plugging the last two expressions into Equation 1 we simplify it as shown in Equation 5 for the four possible combinations of c_I and x.

$$m^x_{Ii} = \begin{cases} \prod_{i' \in \mathcal{U}_I \setminus i} m^0_{i'I} & \text{if } c_I = 0, x = 0 \\ 0 & \text{if } c_I = 0, x = 1 \\ 1 - \prod_{i' \in \mathcal{U}_I \setminus i} m^0_{i'I} & \text{if } c_I = 1, x = 0 \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

In other words, given a certain check state c_I, Equation 5 allows one to compute two messages m^0_{Ii} and m^1_{Ii} at the cost of $|\mathcal{U}_I| - 1$ multiplications, where operator $|\cdot|$ evaluates the cardinality of a set. Since messages are associated to each edge of the bipartite graph we can conclude that the overall check pass takes on average $|E|(z_{\mathcal{U}} - 1)$, $z_{\mathcal{U}}$ being the average number of uploaders per check. It is worth pointing out that a brute force implementation of Equation 1 would require the summation over all the possible state configurations of $|\mathcal{U}_I| - 1$ uploaders, that is a well known combinatorial computational issue in BP implementations, especially for distributions with many states [18], [23].

The next step is constituted by the updating of the probabilities m^x_{iI}, using information from previous computation in the checks, according to (6):

$$m^x_{iI} = \prod_{I' \in \mathcal{C}_i \backslash I} m^x_{iI'} \tag{6}$$

This computation is performed on every edge from an uploader to its checks and it constitutes the so called node pass. The overall computational cost amounts to $|E|(z_{\mathcal{C}} - 1)$ multiplications, where $z_{\mathcal{C}}$ is the average number of checks per node.

After few iterations of check and node passes, the marginal $P(x_i = x)$ can be estimated as follows:

$$P(x_i = x) = \prod_{I' \in \mathcal{C}_i} m^x_{iI'} \tag{7}$$

This last step exhibits a computational cost of $|E|z_{\mathcal{C}}$ multiplications.

We recall that the probabilities estimates given by Equations (5), (6) and (7) are not guaranteed to be normalized; in the practical implementation we use proper normalization constants to avoid numerical issues as suggested in [17].

To conclude, the BP algorithm initializes the values of m_{iI}, then keeps iterating using Equations (5) (message from checks to nodes) and (6) (messages from nodes to checks). A reliable estimate of $P(x_i = x)$ is computed after a certain number of iterations according to Equation (7). The reader interested in the algorithmic details is referred to Appendix A, describing a simple numerical example.

4.1 Incremental BP estimation

In the previous description of the BP we have assumed that the factor graph $(\mathcal{U}, \mathcal{C}, \mathcal{E})$ is known in advance and kept fixed for all the iterations. In practice this assumption is not met in the scenario we consider. Nonetheless, the proposed algorithm can be implemented using an incremental (or sliding window) approach as follows.

A monitor nodes keeps receiving checks from the peers it is assigned and it is allowed to run the BP every T seconds considering only the checks received in a time window of the past w seconds. At time t, depending on the checks stored during a time window w, an updated factor graph $(\mathcal{U}, \mathcal{C}, \mathcal{E})_{t,w}$ is obtained by removing the old checks and adding the new ones; then, the corresponding estimates $P_{t,w}(x_i = x)$ are computed through BP. It is worth pointing out that the belief values are initialized only once, as soon as a peer shows up as an uploader for the first time. In particular, the probabilities m_{iI}^{x} are initialized to 0.5 when a previously unknown peer identifier is met for the first time. Then, the factor graph is updated dynamically without reinitializing the probability estimates of the peers that participates to multiple computation windows.

After every BP algorithm run a list of suspect peers is obtained by setting a threshold on the pollution probability $P_{t,w}(x_i = 1) \geq \eta$. A monitor node keeps a counter for each peer i: the peers in the list of suspects after the BP run have their counter increased by 1. Finally, a *suspects ranking* over peers is defined by sorting their counters in decreasing order. As an example, the first peer in the suspects ranking at time t is the uploader that more often has been included in the list of suspects after all the BP runs performed up to time t.

5 Experimental results

In this section we present results on the accuracy and effectiveness of the proposed technique. We first describe the test-bed and evaluation methodology we used in Sect. 5.1 and subsequently we present results for different types of peers misbehaving in Sect. 5.3.

5.1 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) [26]. As already mentioned in Sect. 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 packets and instead of transmitting the original data packets, LT coded packets [27] are encoded and forwarded by the peers. Coding has the following property: the original chunk can be obtained by any peer able to collect any set of $k \cdot (1 + \epsilon)$ coded packets (ϵ is known as the code overhead). LT encoded packets are produced only by peers that have already received the original chunk and consists in simple binary XOR operations among a random set of d original data packets, provided that d is selected according to the Robust Soliton Distribution [27]. A coded packet conveys the XORed payloads of the corresponding original packets as well as a header signaling the indexes of the combined packets. Decoding is carried out by solving a system of linear equations by a special purpose on the fly algorithm proposed in [28]. This decoder exhibits a very limited overhead, e.g. less than 4% in our settings.

We developed a full prototype that has been tested over the Planetlab network in [26]. In the present study we added to the prototype the class of malicious nodes that, on each transmission opportunity, flip a coin and with probability p_{poll} the content of a coded packet is randomly modified before insertion in the peer output queue.

LT coding of chunks, besides simplifying the content delivery mechanism, can be exploited to detect polluted data without an external verification tool. In fact, 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 malicious node that wants to prevent the LT decoding of the chunk will modify the coded packet (or equivalently the corresponding header used to signal the indexes of the XORed original packets) so as that the polluted packet does not represent a valid combination of the original data. A receiver is able to detect such condition as soon as an inconsistence is found in the solution of the underlying system of linear equations. In our prototype, this has been achieved running the LT decoder proposed in [28], that is an incremental version of the standard *Gaussian Elimination* technique for the solution of the system of coding equations. Given the sub-optimality of the coding approach (overhead $\epsilon > 0$), some redundant coded packets are always received. Every redundant packet is recognized as linearly dependent on the previously received ones using only the original packet indexes signaled in the packet header; if this is the case the coded packet payload must be obtained by XORing a subset of the already received packets. If this constraint does not apply the whole chunk is recognized as polluted. Unfortunately the receiver is not able to identify the malicious blocks but only that at least one of them as been maliciously manipulated. When a peer detects a polluted chunk it does not insert it in the set of chunks that can be provided to its neighbors so as to limit the propagation of corrupted information.

TABLE 1: Upload bandwidth distribution for TVS experiments (resource index=1.2).

Percentage	Upload bandwidth
0.46	128 Kbps
0.39	384 Kbps
0.15	1 Mbps

TABLE 2: TVS parameters for detection experiments.

Parameter	Value
k	120
$c = \delta$	0.01
packets size	1330 bytes
$N_{\max}$	30
$N_{\min}$	10
E	$[5, 20]$
$startup\_chunks$	5
$start\_buffering$	2
$end\_buffering$	5

Please note that each peers caches the identity received by the tracker upon its very first joining to the swarm and it subsequently uses that identifier if churning is allowed. This is to restrict our analysis to the case of non-sibyl attacks where peers keep on changing their identifier to escape control and detection activities.

5.2 The evaluation methodology

Since the aim of the following analysis is to assess the accuracy, robustness, and reactivity of the proposed technique we conduct experiment on a system with only one monitor node. Of course, for scalability, load balancing, and resilience reasons a set of monitor nodes should be deployed in case of a much larger systems.

Our reference scenario is composed of a set of $N = 1800$ peers and $N_P = 90$ malicious nodes (they are 5% of well behaving nodes). We conducted $N_{EXP} = 50$ independent trials for each scenario we considered and computed for all performance indexes the 95% confidence intervals (CI); each trial lasts for 1800s. In order to adopt a realistic churn model that is based on prior measurement studies 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 120s and permanently depart. Arrival of a new node (a node 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. We consider two activity models for the N_P malicious nodes: either they all join the swarm after 120s from the start of the experiment and stay connected until the end or they churn by alternating between active and idle periods; these periods are both exponentially distributed with average equal to $T_{on} = 120s$ and $T_{off} = 20s$. During this period peers join the system and start exchanging coded packets to decode video chunks.

Upon decoding of a chunk a peer logs a check containing a time stamp relative to the start of the experiment (each experiment starts at time 0), the chunk identifier, the number and identities (uniquely assigned by the tracker upon joining the swarm) of the uploaders of that chunk as well as the number of coded packets received by each of them. Finally, the check contains a flag to indicate the state of the decoded chunk (polluted or clean). Malicious nodes may lie when logging a check: with probability p_{lie} the value of the check is inverted and with probability $1 - p_{lie}$ the check is faithfully reported. All logs are then collected at the end of each experiment, merged, and sorted for increasing values of the time stamp so to emulate an approximate ordering of global arrival at the monitor node. An analysis software implementing the technique described in Sect. 4 is run on the resulting factor graph to compute the suspects ranking at each time t. The BP algorithm is run with 3 iterations and the after every BP run the list of suspect peers is obtained by setting the threshold on the pollution probability $\eta = 0.99$.

To evaluate the performance of our detection technique we defined two indexes:

- the *hit ratio* at time t (denoted as $h(t)$) that is a measure of accuracy. For the i-th trial we define an accuracy function $a_i(t)$ that is time dependent: $a_i(t) = x$ if x peers in the top N^i_{AP} positions of the suspects ranking are true malicious nodes. Here $0 \leq N^i_{AP} \leq N_P$ denotes the number of active malicious nodes in the i-th trial; indeed, some malicious nodes can be victims of other malicious nodes and hence may not be able to send other peers any corrupted block (or simply they are not able to decode any chunk and hence do not spread corrupted information). The last observation is related to our assumption that the malicious nodes are allowed to modify the packets in their output queue but for no reason they can modify the P2P sharing protocol; therefore, when a malicious node collects a chunk that has been compromised by other malicious nodes, has no mean to forward packets of such chunk. The hit ratio at time t is then defined as $h(t) = \frac{\sum_{i=1}^{N_{EXP}} \frac{a_i(t)}{N^i_{AP}}}{N_{EXP}}$.
- the *minimum time to remove x suspects* (denoted as $TSR(x)$) that is a measure of reactivity. For the i-th trial we first define a rank function $r_i(t)$ that is time dependent. We consider $r_i(t) = x$ if the top x peers in the suspects ranking are *all true malicious nodes* at time t ($0 \leq r(t) \leq N^i_{AP}$). This definition requires not only the detection technique to identify x malicious nodes but these nodes must be the top x elements in the suspects ranking. If the detection technique at time t has identified $N^i_{AP} - 1$ out of N^i_{AP} malicious nodes ($a_i(t) = N^i_{AP} - 1$) but the top peer in the suspect ranking is not a true malicious node then $r_i(t) = 0$. Indeed, this is a much more restrictive definition of true positiveness with respect to $a_i(t)$.

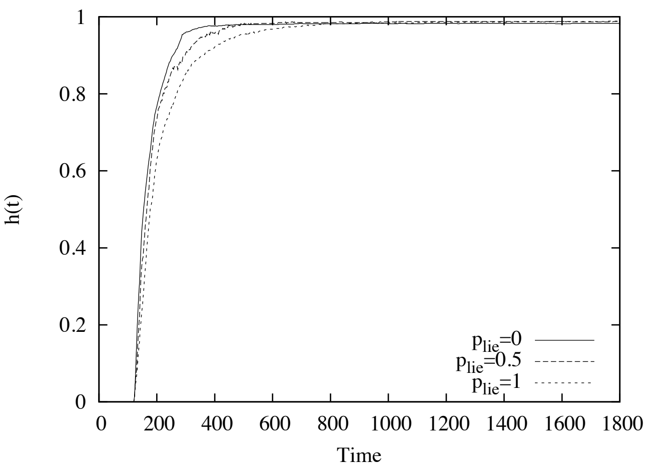


Fig. 2: $h(t)$ as a function of time for different values of p_{lie}.

If we denote as tf_i the arrival time to the monitor node of the first positive check[1] and as $tr_i(x) = \min_t\{t : r_i(t) \geq x\}$ the minimum time to identify at least x true malicious nodes in the top x positions of the suspect ranking, then we define $TSR(x) = \frac{\sum_{i=1}^{N_{EXP}} tr_i(x) - tf_i}{N_{EXP}}$.

The TVS application has been run to stream a 300 kbps bitrate video using a 2.1 Mbps server with upload bandwidth distribution summarized in Tab. 1 and yielding a resource index equal to 1.2, i.e., there is 20% average extra bandwidth with respect to the video bitrate. Tab. 2 reports the main parameters of TVS: we refer the reader to [26] for a detailed description of their precise meaning.

5.3 Single monitor performance

In this section a set of experimental results, worked out using a system with a single monitor ($N_M = 1$), are presented and discussed. The goal is twofold: from the one hand this study allowed us to optimally select several system parameters, from the other hand the single monitor case serves as a benchmark for the performance of the general case presented in Sect. 5.4. The technique we propose requires the use of two parameters: the analysis window size (w) and the analysis time interval (T), that are both expressed in seconds. Appendix B presents an extensive evaluation of the impact of several parameters on the performance of our technique. Following this analysis all subsequent results will be obtained for $w = 10$ s, and $T = 2.5$ s.

5.3.1 Effect of lying

In this set of experiments malicious nodes are allowed to send fake checks to a monitor node. The check value is inverted with probability p_{lie} and faithfully reported

1. Time tf_i represents the time instant when the monitor node realizes that the pollution attack has begun; as a consequence, we measure reactivity relative to the beginning of the attack.

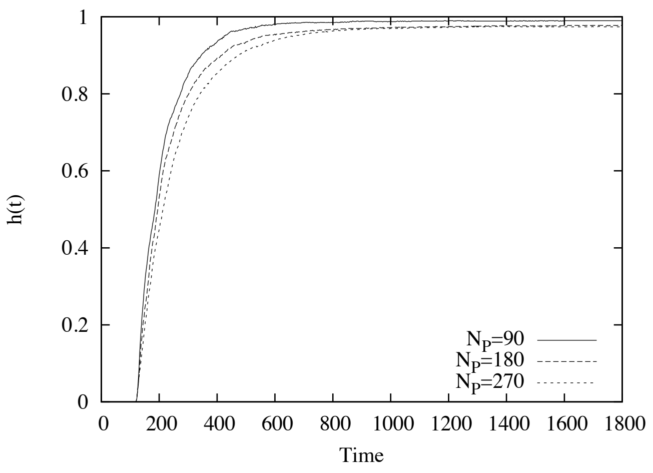


Fig. 3: $h(t)$ as a function of time for increasing number of malicious nodes.

TABLE 3: Reaction times for increasing number of malicious nodes.

N_P	$TSR(1)$ (CI)
90	20.6 ($[14.1, 27.1]$)
180	12.9 ($[8.8, 17.0]$)
270	12.7 ($[8.7, 16.7]$)

with probability $1 - p_{lie}$. In Fig. 2 we show $h(t)$ as a function of time for $p_{lie} = 0, 0.5, 1$; in all cases $p_{poll} = 1$. It can be noted that the detection technique still succeeds in identifying all malicious nodes. Lies only affect $TSR(1)$: $TSR(1)$ increases from $6.9s$, CI $[4.7, 9.1]$ for $p_{lie} = 0$ to $15.9s$, CI $[10.9, 21.0]$ for $p_{lie} = 1$ (in the case $p_{lie} = 0.5$ we obtain $TSR(1) = 8.7$, CI $[1.8, 15.7]$).

5.3.2 Effect of increasing the number of malicious nodes

All previous results have been obtained in the reference scenario comprising $N_P = 90$ malicious nodes. An interesting and important step to assess the accuracy of our technique is to evaluate its performance as the number of malicious nodes increases. To this end, Fig. 3 depicts $h(t)$ as a function of time for N_P up to 270, i.e., 15% of honest nodes. It can be noted that accuracy is high in all cases; the only difference is on the time to identify most of the malicious nodes. Furthermore, reactivity improves as the number of malicious nodes increases as summarized in Tab. 3.

We also conducted further evaluations to assess the impact of malicious nodes that do not comply to the protocol and of their upload bandwidth on the performance of our technique in Appendix C. We also evaluated the performance of our technique for different values of the polluting intensity p_{poll} and for churning malicious nodes that alternate between active and idle periods.

5.4 Accuracy with multiple monitor nodes

In this section we present accuracy results in the general case when more than one monitor node is employed to achieve scalability and load balancing. Regardless of the mechanisms used by the monitor nodes to share the collected information, we assume that they are able to merge the respective suspect rankings obtaining a single global ranking. This is achieved by summing the counters of all monitors for each peer i. The global suspect ranking is computed by sorting the peers' counters in decreasing order. Since each monitor runs BP estimation on a local version of the factor graph it is important to guarantee that the obtained global ranking is still accurate, thus proving that the proposed approach scales effectively to large overlays. In Fig. 4 we show the global $h(t)$ in the case $N_M = 4$ for the scenario considered to obtain Fig. 9 with $T = 2.5s$ and $w = N_M \cdot 10s = 40s$ to compensate for a reduction by a factor N_M of the number of checks to process. It can be noted that both single monitor and multiple monitor configurations yield comparable highly accurate performance.

5.5 Robustness to colluding attacks

In all previous analysis malicious nodes 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}. In this section we consider a more sophisticated strategy; in particular, we devise a two-pronged attack where malicious nodes upon decoding a chunk:

- send a positive check to their monitor node if the set of uploaders does not contain any other malicious node (a disparage action towards honest nodes);
- send a negative check if at least one of the uploaders is another malicious node (a protective action for other malicious nodes).

Fig. 5 compares the accuracy achieved in this case against that of the reference scenario. It can be noted that accuracy is very high even in presence of two-pronged attack. Reactivity is only slightly affected: $TSR(1) = 23.3$, CI $[15.9, 30.6]$, in the colluding attack while we obtain $TSR(1) = 20.6$, CI $[14.1, 27.1]$ in the reference scenario.

TABLE 4: Reaction times in case of a colluding attack for increasing number of malicious nodes.

N_P	$TSR(1)$ (CI)
90	23.3 ($[15.9, 30.6]$)
180	23.7 ($[16.2, 31.2]$)
270	25.9 ($[17.7, 34.1]$)

We also performed an evaluation of the accuracy of our technique in a system with an increasing number of malicious nodes. To this end, Fig. 6 shows that accuracy is high for all considered settings. Furthermore, reactivity slightly increases as the number of malicious nodes increases as summarized in Tab. 4.

5.6 Computational, communication, and memory costs

In this section we analyze the communication, computational, and memory costs of implementing our solution to identify malicious nodes at each monitor node.

5.6.1 Communication cost

The solution we propose is based on sending checks to a monitor node. If we assume that identifiers are 32 bits long then the average size of the payload of a check message is equal to $32z_{\mathcal{U}} + 1$ bits where $z_{\mathcal{U}}$ is the average number of uploaders for a check (see Sect. 4) and one bit is necessary to indicate the status of the check (positive or negative). On the other hand, for each peer the chunk reception rate cannot exceed the generation rate at the video server. From Tab. 2 we can compute the duration of a chunk as $T_{chunk} = \frac{(1330 \cdot 120 \cdot 8)}{300000} = 4.256s$. It follows that the rate of production of checks to send to the monitor node is upper bounded by $\frac{1}{T_{chunk}}$ checks/s.

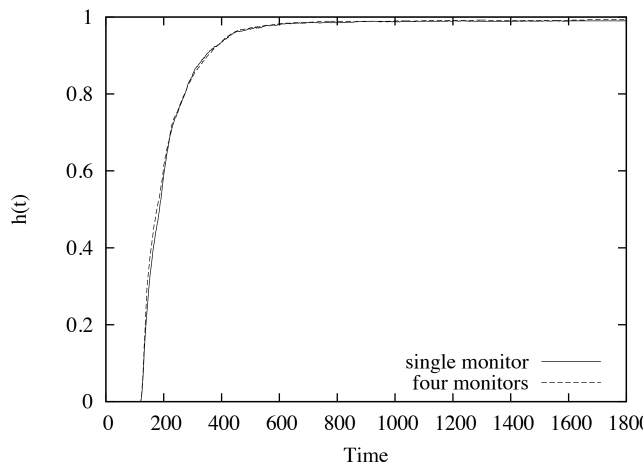


Fig. 4: $h(t)$ as a function of time for one and four monitors nodes.

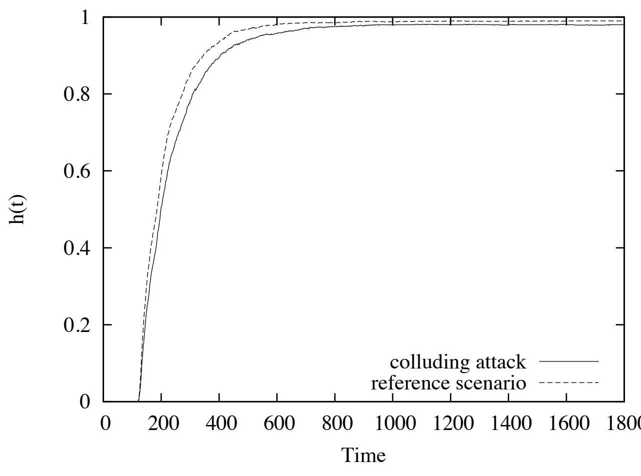


Fig. 5: $h(t)$ as a function of time in case of a colluding attack.

If we have $N + N_P$ peers that refer to the same monitor node then the number of checks per time unit is upper bounded by $\frac{N+N_P}{T_{chunk}}$. Please note that this is an upper bound since only 20% of the N honest nodes 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 120s. We can then compute an upper bound on the incoming communication bandwidth required at a monitor node in our system as $\frac{N+N_P}{T_{chunk}} \cdot \frac{(32 z_{\mathcal{U}}+1)}{1000} = 50.1$ kbit/s where we used $z_{\mathcal{U}} = 3.5$ in our computation (the average number of uploaders in all our experiment was equal to 3.27 as shown in Tab. 5), $N = 1800$, and $N_P = 90$. This is in accordance with the average measured effective bandwidth that is equal to 9.1 Kbit/s. The required bandwidth is lower than the theoretical upper bound also because when malicious nodes pollute data the system throughput reduces, i.e., the number of decoded chunks decreases, hence the generation rate of checks decreases accordingly.

We conclude that the communication cost of our technique is low and does not represent the limiting factor. As an example, a monitor node allowing only 1 Mbit/s for receiving checks from its peers would support up to $\frac{1000}{50.1} \cdot 1890 = 37,724$ total peers ($207,692$ if the measured bandwidth is used in the computation instead of the upper bound).

5.6.2 Memory 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, nodes, and arcs 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 Tab. 5; 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. For completeness, we also summarize in Tab. 6 the results for the multiple monitor scenario presented in Sect. 5.4; it can be noted that very similar results are obtained in this case.

TABLE 5: Average and maximum sizes for checks, nodes, and arcs of the factor graph.

	Average	Maximum
nodes	1000.6	1795
checks	881.6	2476
arcs	2872.4	15700
nodes per check	3.27	9.14
checks per node	2.73	12.41

TABLE 6: Average and maximum sizes for checks, nodes, and arcs of the factor graphs in the multiple monitor scenario.

	Average	Maximum
nodes	1060.3	1728
checks	874.3	2271
arcs	2853.4	9327
nodes per check	3.34	9.33
checks per node	2.57	10.71

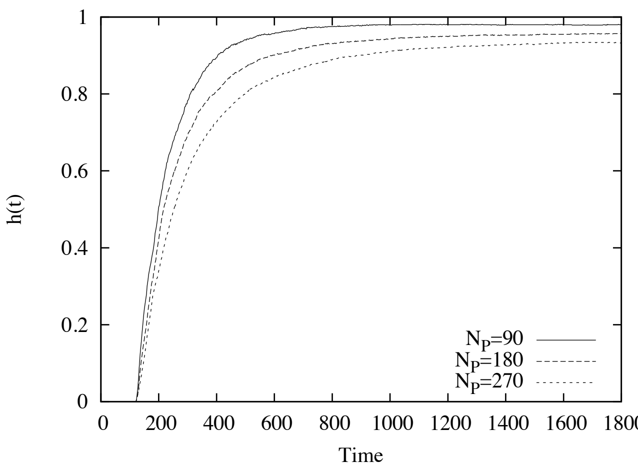


Fig. 6: $h(t)$ as a function of time in case of a colluding attack for increasing number of malicious nodes.

5.6.3 Computational cost

The measurements of the average number of arcs ($|E|$), nodes per check ($z_{\mathcal{U}}$) and check per nodes ($z_{\mathcal{C}}$) shown in Tab. 5 can be used to better understand the computational cost of the BP algorithm. Indeed, in Sect. 4 we have shown that 3 BP iterations takes $3(|E|(z_{\mathcal{U}} - 1) + |E|(z_{\mathcal{C}} - 1)) + |E| z_{\mathcal{C}}$ multiplications. Our measurements show that $z_{\mathcal{U}}, z_{\mathcal{C}} \ll |E|$ and therefore we can conclude that the BP complexity if 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 about 85 ms on an Intel(R) Core i5 2.80GHz CPU. Please note that such values are *two orders of magnitude* smaller than the time between any two BP runs, i.e., T.

6 Conclusion

In this paper we have shown that it is possible to recast the identification of malicious nodes attempting to pollute a P2P application as a problem of statistical inference. The proposed technique exploits the well known BP algorithm to iteratively estimate the probability of peers being malicious. To allow the system to scale to a large size, the current proposal is based on the use of a set of trusted monitor nodes that collect from the peers reports about the integrity of the downloaded chunks of data, the checks.

The technique has been validated in the framework of a complete P2P video streaming application using the Planetlab infrastructure. Promising results in terms of accuracy, robustness and reactivity have been reported in the presence of several misbehavior of the malicious nodes, namely different pollution intensity and bandwidth, fake reporting, churning and un-cooperation.

The experimental results show that is possible to identify all the polluters in a reliable way provided that enough checks are collected. The proposed identification technique has turned out to correctly detect up to 15% of polluters also in the case when malicious peers pollute the packets very seldom, periodically stay on idle, and send fake reports to the monitor to interfere with the estimation process. Furthermore, very good results have been obtained in the case of a sophisticated threat model where malicious nodes both disparage honest nodes and protect each other. The reactivity of the identification process has been shown to be a few seconds in the most favorable settings and no higher than 25s in any setting.

Clearly, these values also depend on the actual P2P application and in particular on the amount of checks that can be generated per unit of time. This latter in turn varies according to the P2P protocol parameters, e.g. chunk size. Therefore a joint optimization of quality of service offered by the P2P platform and a reduction of the polluter identification delay can be pursued.

Future works will tackle the design of a fully distributed detection mechanism where all the peers in the overlay can exchange checks and autonomously estimates suspect uploaders. The distributed version of our algorithm will allow each peer to rule out the malicious uploaders thus limiting the intensity of the pollution.

REFERENCES

[1] X. Zhang, J. Liu, B. Li, and T. Yum, "CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming," in *proceedings of IEEE Infocom*, vol. 3. Citeseer, 2005, pp. 13–17.

[2] G. Huang, "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*, 2007.

[3] P. Dhungel, X. Hei, K. Ross, and N. Saxena, "The pollution attack in P2P live video streaming: measurement results and defenses," in *Proceedings of the 2007 workshop on Peer-to-peer streaming and IP-TV, P2P-TV '07*, 2007, pp. 323–328.

[4] J. Liang, R. Kumar, Y. Xi, and K. Ross, "Pollution in P2P file sharing systems," in *IEEE INFOCOM 2005*, vol. 2, march 2005, pp. 1174 – 1185.

[5] M. N. Krohn, M. J. Freedman, and D. Mazieres, "On-the-fly verification of rateless erasure codes for efficient content distribution," *Security and Privacy, IEEE Symposium on*, 2004.

[6] C. Gkantsidis and P. Rodriguez, "Cooperative security for network coding file distribution," in *IEEE INFOCOM*, 2006.

[7] Q. Li, D.-M. Chiu, and J. Lui, "On the practical and security issues of batch content distribution via network coding," in *Network Protocols, 2006. ICNP '06. Proceedings of the 2006 14th IEEE International Conference on*, 2006.

[8] D. Kamal, D. Charles, K. Jain, and K. Lauter, "Signatures for network coding," in *In Proceedings of the fortieth annual Conference on Information Sciences and Systems*, 2006.

[9] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An efficient signature-based scheme for securing network coding against pollution attacks," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, 2008.

[10] E. Kehdi and B. Li, "Null keys: Limiting malicious attacks via null space properties of network coding," in *INFOCOM 2009, IEEE*.

[11] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, "An efficient scheme for securing xor network coding against pollution attacks," in *INFOCOM 2009, IEEE*.

[12] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger, "Byzantine modification detection in multicast networks with random network coding," *Information Theory, IEEE Transactions on*, vol. 54, no. 6, pp. 2798 –2803, june 2008.

[13] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, M. Medard, and M. Effros, "Resilient network coding in the presence of byzantine adversaries," *Information Theory, IEEE Transactions on*, vol. 54, no. 6, pp. 2596 –2603, june 2008.

[14] R. Koetter and F. Kschischang, "Coding for errors and erasures in random network coding," *Information Theory, IEEE Transactions on*, vol. 54, no. 8, pp. 3579 –3591, august 2008.

[15] B. Beverly Yang and H. Garcia-Molina, "Designing a super-peer network," in *Data Engineering, 2003. Proceedings. 19th International Conference on*. IEEE, 2003, pp. 49–60.

[16] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2003, pp. 407–418.

[17] D. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, 2003.

[18] J. Yedidia, W. Freeman, and Y. Weiss, "Constructing free-energy approximations and generalized belief propagation algorithms," *IEEE Transactions on Information Theory*, vol. 51, no. 7, pp. 2282 – 2312, 2005.

[19] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, "MIS: Malicious nodes identification scheme in network-coding-based peer-to-peer streaming," in *INFOCOM, 2010 Proceedings IEEE*, march 2010, pp. 1 –5.

[20] Y. Li and J. C. Lui, "Stochastic analysis of a randomized detection algorithm for pollution attack in P2P live streaming systems," *Performance Evaluation*, vol. 67, no. 11, pp. 1273 – 1288, 2010.

[21] X. Jin and S.-H. G. Chan, "Detecting malicious nodes in peer-to-peer streaming by peer-based monitoring," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 6, pp. 9:1–9:18, March 2010.

[22] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988.

[23] J. Yedidia, W. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," in *Exploring Artificial Intelligence in the New Millennium*, ser. Science & Technology Books. Elsevier, 2003, ch. 8.

[24] R. Gallager, *Low-Density Parity-Check Codes*. Cambridge: M.I.T. Press, 1963.

[25] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael, "Learning low-level vision," *International Journal of Computer Vision*, vol. 40, pp. 25–47, 2000.

[26] A. Magnetto, R. Gaeta, M. Grangetto, and M. Sereno, "P2P streaming with LT codes: a prototype experimentation," in *Proceedings of the 2010 ACM workshop on Advanced video streaming techniques for peer-to-peer networks and social networking*, ser. AVSTP2P '10, 2010, pp. 7–12.

[27] M. Luby, "LT codes," in *IEEE FOCS*, Nov. 2002, pp. 271–280.

[28] V. Bioglio, R. Gaeta, M. Grangetto, and M. Sereno, "On the fly gaussian elimination for LT codes," *IEEE Communication Letters*, vol. 13, no. 2, pp. 953–955, Dec. 2009.

[29] Y. Weiss and W. Freeman, "On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 736 – 744, feb 2001.

Rossano Gaeta Rossano Gaeta received his Laurea and Ph.D. degrees in Computer Science from the University of Torino, Italy, in 1992 and 1997, respectively. He is currently Associate Professor at the Computer Science Department, University of Torino. He has been recipient of the Best Paper award at the 14-th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2006) and at the 26th International Symposium on Computer Performance, Modeling, Measurements, and Evaluation (PERFORMANCE 2007). His current research interests include the design and evaluation of peer-to- peer computing systems and the analysis of compressive sensing and coding techniques in distributed applications.

Marco Grangetto M. Grangetto (S99-M03-SM09) received his Electrical Engineering degree and Ph.D. degree from the Politecnico di Torino, Italy, in 1999 and 2003, respectively. He is currently Associate Professor at the Computer Science Department, University of Torino. His research interests are in the fields of multimedia signal processing and networking. In particular, his expertise includes wavelets, image and video coding, data compression, video error concealment, error resilient video coding unequal error protection, and joint source channel coding. Prof. Grangetto was awarded the Premio Optime by Unione Industriale di Torino in September 2000, and a Fulbright grant in 2001 for a research period with the Department of Electrical and Computer Engineering, University of California at San Diego. He has participated in the ISO standardization activities on Part 11 of the JPEG 2000 standard. He has been a member of the Technical Program Committee for several international conferences, including the IEEE ICME, ICIP, ICASSP, and ISCAS.

APPENDIX A

In the following we clarify the algorithmic steps performed by the BP algorithm on a simple numerical example. To this end let us consider the simple factor graph with 4 uploaders (nodes) and 2 checks shown in Fig. 1 of the main paper.

The BP algorithm consists in the iterative evaluation of the so called check pass according to Equation (5), that computes messages from checks to nodes, followed by the node pass given by Equation (6), that updates messages in the opposite direction.

When an uplader i is added for the first time to the factor graph the messages towards its checks are set to $m^0_{iI} = m^1_{iI} = 0.5$; this amounts at assuming no prior knowledge on the uploader state, i.e. polluter or not. In Fig. 7 the check pass computations performed at the first iteration to update messages from the first check (a positive check) towards the rightmost uploader (shaded node) are graphically shown. The two incoming messages are set to $m^0_{iI} = m^1_{iI} = 0.5$ and are used to compute $m^0_{Ii} = 0.75$ and $m^1_{Ii} = 1.0$ (messages directed to the rightmost node) using Equation (5). Since (5) does not guarantee $m^0_{Ii}+m^1_{Ii} = 1$, the values are renormalized to 0.43 and 0.57, respectively.

Fig. 8 shows the values of all messages m^1_{Ii} towards the nodes after the first check pass. Such incoming messages can be used to compute the node pass according to Equation (6) multiplying all the incoming messages except the one corresponding to the check that one is updating. Analogously, the Equation (7) can be used to estimate the probability of every uploader being a polluter $P(x_i = 1)$; the estimate is evaluated multiplying all the messages m^1_{Ii} entering a node. In Fig. 8 the values of $P(x_i = 1)$, obtained after the first iteration, are shown above each uploader. It can be noted that in this simple example the polluter gets a probability 0.57 of being malicious after the first iteration whereas the other uploaders are not likely to be malicious. In our implementation 3 iterations of check and node passes are performed to obtain a reliable estimation of $P(x_i = 1)$.

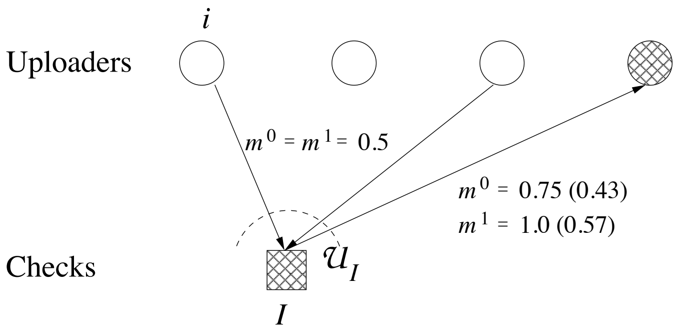


Fig. 7: Check pass computation example

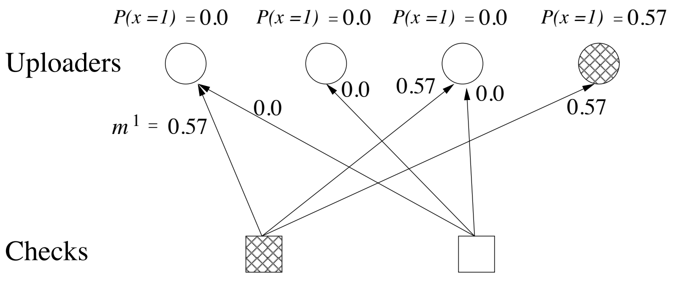


Fig. 8: Node pass computation example

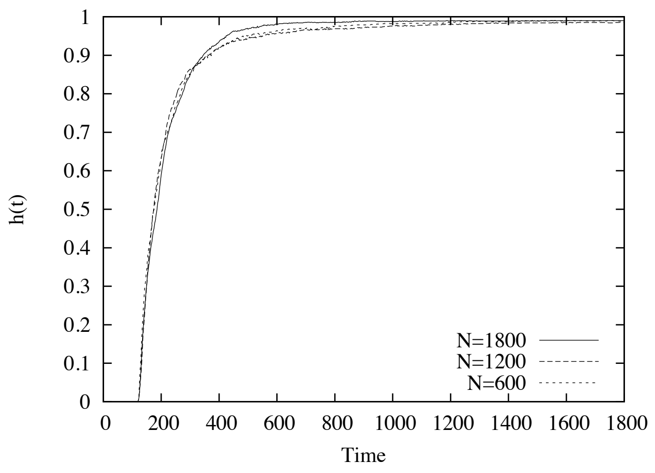
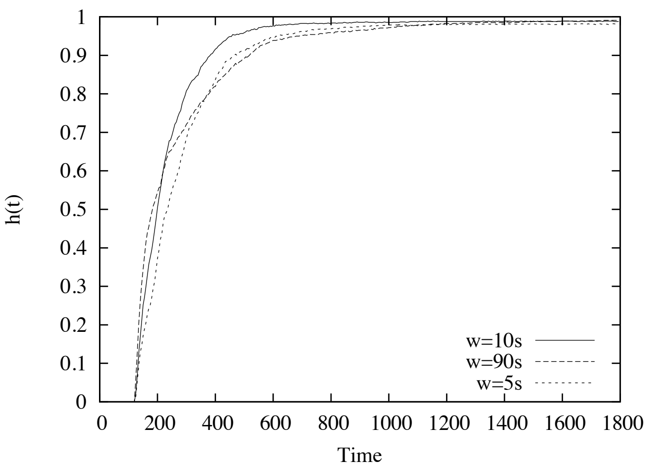


Fig. 10: $h(t)$ as a function of time for $T = 2.5s$ and $w = 10s$ for different system sizes.

APPENDIX B

The technique we propose requires the use of two parameters: the analysis window size (w) and the analysis time interval (T), that are both expressed in seconds. The first step is then to explore the performance of the detection technique for different combinations of these two parameters. To this end, we considered a system with $N = 1800$ churning peers and $N_P = 90$ stable (they do not churn) malicious peers; furthermore, we analyzed the performance of the detection technique for all nine combinations of $p_{poll} = \{0.1, 0.5, 1\}$ and $p_{lie} = \{0, 0.5, 1\}$. Since reactivity is an important quality factor for the detection technique we considered only two short time intervals: $T = \{2.5s, 5s\}$.

Fig. 9 (left graph) depicts the technique hit ratio h as a function of time for the configuration with $p_{poll} = 0.5$, $p_{lie} = 0.5$, and $T = 5s$ (all 18 configurations provided similar results so we selected a representative scenario). It can be noted that too small or too large values for w yield the worst performance for both values of T. 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 [29].



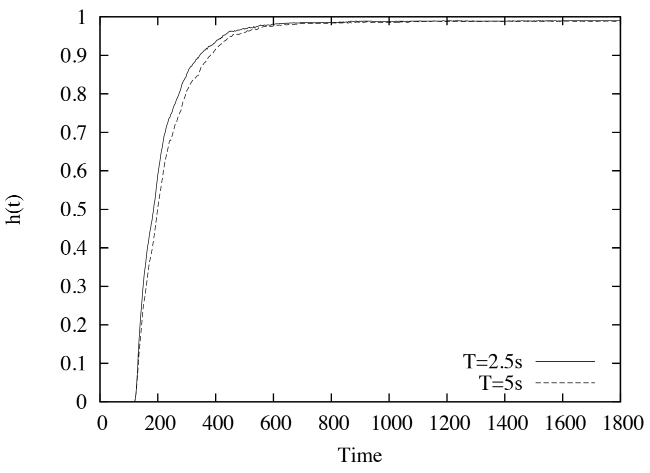


Fig. 9: $h(t)$ as a function of time for $p_{poll} = 0.5$, $p_{lie} = 0.5$, $T = 5s$ (left) and for $w = 10s$ (right) in the scenario with $N = 1800$.

We observed that values of w in the range $[10, 40]$ show comparable accuracy and reaction times; in turn the value of w determines the number of checks (and consequently the size) of the factor graph to be used in any BP computation window. Since, as shown in Sect. 4, the complexity of BP depends linearly on the number of edges in the factor graph, it follows that a small value of w is to be preferred. According to this reasoning we selected $w = 10s$ for all the following analysis. Fig. 9 (right graph) shows that a shorter time interval between successive runs of the detection algorithm yields the same accuracy while decreasing the time to safely remove the first identified malicious node ($TSR(1) = 23.4s$, CI $[16.0, 30.8]$ for $T = 5s$ and $TSR(1) = 20.6$, CI $[14.1, 27.1]$ for $T = 2.5s$).

Furthermore, Fig. 10 shows $h(t)$ as a function of time for different system sizes in the case $w = 10s$ and $T = 2.5s$. It can be noted that the accuracy is similar for smaller systems. Following this analysis all results will be obtained for $w = 10$ s, and $T = 2.5$ s.

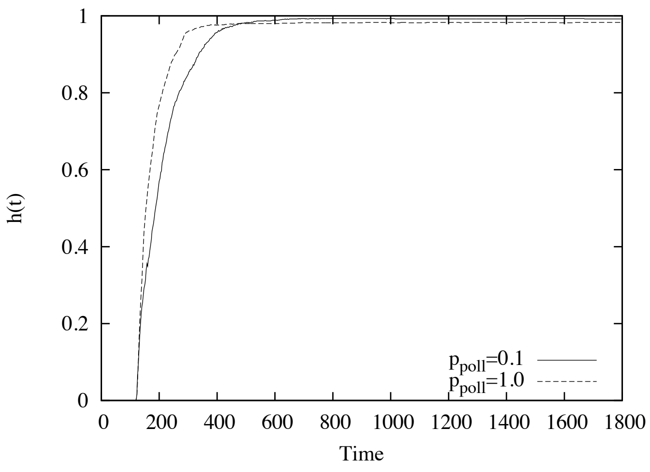


Fig. 11: $h(t)$ as a function of time for different polluting intensities.

Appendix C

C.0.1 Effect of polluting intensity

We consider the robustness of the detection technique against different levels of polluting intensity p_{poll}. Fig. 11 shows $h(t)$ as a function of time in the case when malicious nodes never lie. It can be noted that the detection technique is able to detect all active malicious nodes that do not lie regardless of the pollution intensity. As a further evidence of the robustness of the technique we observe that we obtain $TSR(1) = 6.9s$, CI $[4.7, 9.1]$ and $TSR(1) = 6.5s$, CI $[4.4, 8.6]$ for $p_{poll} = 1$ and $p_{poll} = 0.1$, respectively.

C.0.2 Effect of churning

In all previous experiments malicious peers always joined the overlay network and stayed connected until the end of the experiment. We evaluate the performance of the detection technique when malicious peers alternate frequent join to and leave from the overlay using the same average activity and silent period of normal peers. We consider malicious peers with the highest pollution and lies intensity ($p_{poll} = p_{lie} = 1$) that represent the toughest settings in our experimentation. In Fig. 12 we show $h(t)$. Please note that, despite the very challenging settings, our detection technique is able to identify all the active malicious peers: churning does not bring any advantage to malicious nodes. Reactivity is still very good: $TSR(1)$ is $15.9s$, CI $[10.9, 21.0]$ for stable malicious nodes while it takes $18.2s$, CI $[12.5, 24.0]$ when malicious nodes churn.

C.0.3 Effect of un-cooperation

Another type of misbehaving we consider is when malicious nodes do not comply to the protocol and never

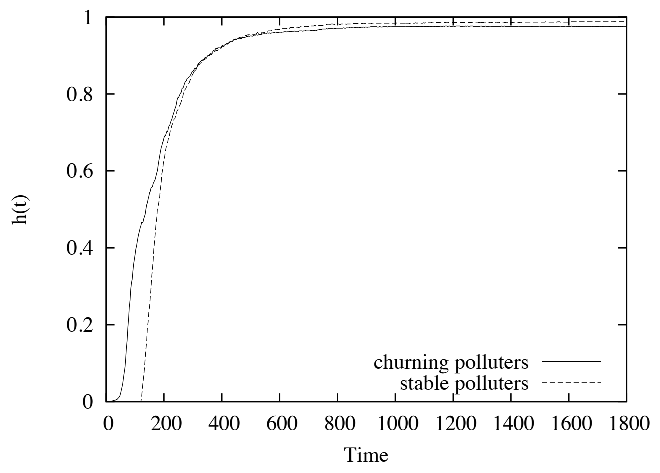


Fig. 12: $h(t)$ as a function of time for churning malicious peers.

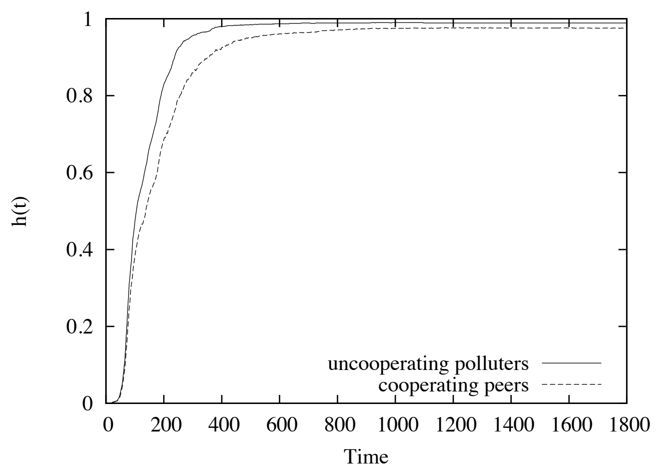


Fig. 13: $h(t)$ as a function of time for un-cooperating churning malicious peers.

send a check to a monitor node. In this case the task of the detection technique is even easier since no lies can deceive the monitor node. Indeed, Fig. 13 shows a comparison between these two cases when malicious nodes churn and pollution intensity is $p_{poll} = 1$ (co-operating malicious nodes always lie). It can be noted that a monitor node is able to detect all malicious nodes in less time. Furthermore, we obtain $TSR(1) = 8.7s$, CI $[6.0, 11.5]$ for un-cooperating malicious nodes and $TSR(1) = 18.2s$, CI $[12.5, 24.0]$ in the other case.

C.0.4 Effect of malicious nodes upload bandwidth

We also consider the impact of the upload bandwidth that malicious nodes use during their attack. To this end we consider the scenario

where churning stealth malicious nodes always lie, i.e., $p_{poll} = 0.1$ and $p_{lie} = 1$, for two values of the malicious nodes upload bandwidth: 300 kbps and 1 Mbps. Fig. 14 shows that higher upload bandwidth translates

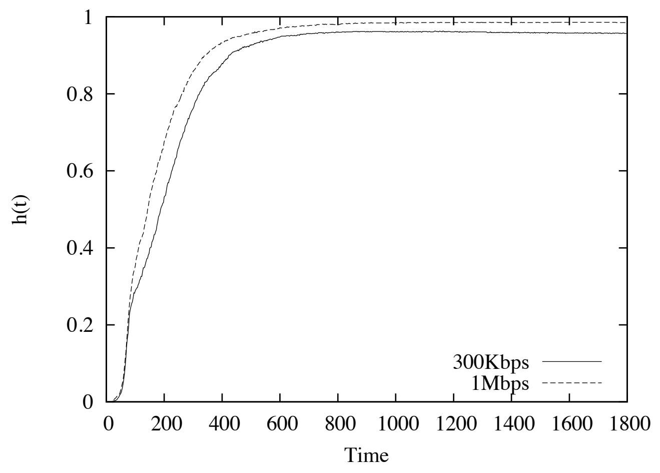


Fig. 14: $h(t)$ as a function of time for churning, always lying, stealth malicious peers.

into slightly higher value of the hit ratio. As far as the computation of $TSR(1)$ is concerned small differences can be observed: $TSR(1) = 27.0s$, CI $[18.5, 35.6]$ for 300 kbps and $TSR(1) = 22.5s$, CI $[15.4, 29.6]$ for 1 Mbps.