# Exploiting rateless codes and belief propagation to infer identity of polluters in MANET

# Exploiting rateless codes and belief propagation to infer identity of polluters in MANET

Rossano Gaeta, Marco Grangetto, Riccardo Loti

**Abstract**—
In this paper, we consider a scenario where nodes in a MANET disseminate data chunks using rateless codes. Any node is able to successfully decode any chunk by collecting enough coded blocks from several other nodes without any coordination. We consider the problem of identifying malicious nodes that launch a pollution attack by deliberately modifying the payload of coded blocks before transmitting. It follows that the original chunk can only be obtained if there are no malicious nodes among the chunk providers.
In this paper we propose *SIEVE*, a fully distributed technique to infer the identity of malicious nodes. A node creates what we termed a *check* whenever a chunk is decoded; a check is a pair composed of the set of other nodes that provided coded blocks used to decode the chunk (the chunk uploaders) and a flag indicating whether the chunk is corrupted or not. *SIEVE* exploits rateless codes to detect chunk integrity and belief propagation to infer the identity of malicious nodes. In particular, every node autonomously constructs its own bipartite graph (a.k.a. *factor graph* in the literature) whose vertexes are checks and nodes, respectively. Then, it periodically runs the belief propagation algorithm on its factor graph to infer the probability of other nodes being malicious.
We show by running detailed simulations using ns-3 that *SIEVE* is very accurate and robust under several attack scenarios and deceiving actions. We discuss how the topological properties of the factor graph impacts *SIEVE* performance and show that nodes speed in the MANET plays a role on the identification accuracy. Furthermore, an interesting trade-off between coding efficiency and *SIEVE* accuracy, completeness, and reactivity is discovered. We also show that *SIEVE* is efficient requiring low computational, memory, and communication resources.

**Keywords**—MANET, rateless codes, belief propagation, pollution attack, malicious node identification, statistical inference.

## 1 Introduction

Mobile Ad-hoc Networks (MANET) are characterized by open, distributed and dynamic architectures, built on top of the shared wireless medium; all features contribute to make MANET very vulnerable to attacks at any layer of the Internet model [1], [2].

We consider a particular type of active, non-cryptography related attack, where insider nodes corrupt data at the application level (this is also known as *pollution attack*). In this paper we deem as a use case a data dissemination application over a MANET. Nodes generate data chunks to be disseminated to all participants using rateless codes; some malicious nodes deliberately modify coded packets of a chunk before relaying them to prevent honest nodes from obtaining the original information.

In this paper we propose *SIEVE* a decentralized, accurate and robust technique to identify malicious nodes on top of an otherwise reliable and attacker-free architecture. Each node in *SIEVE* dynamically creates a bipartite graph (*factor graph*) whose vertexes are *checks* and uploading nodes. A check is a report created by a node upon decoding a data chunk; a check contains a variable length list of nodes identifiers that provided parts of the data as well as a flag to signal if the data chunk has been corrupted. Detection of the compromised chunks is achieved exploiting the constraints imposed by linear channel coding. The factor graph is periodically and independently analyzed by each node running an incremental version of the Belief Propagation (BP) algorithm [3], [4], [5], [6]. The proposed algorithm allows each node to compute the probability of any other node being malicious; these latter probabilities are used to derive a *suspect ranking* of nodes in the MANET. Each node updates its local factor graph using the checks obtained by its own decoding operations as well as checks that are periodically gossiped by neighbor nodes.

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

• *This work was partially supported by Università degli Studi di Torino and Compagnia di San Paolo under project AMALFI (ORTO119W8J).*

### Our contributions

The major contributions of the paper are the recasting of the problem of malicious nodes identification in terms of the estimation of the marginal probabilities on a bipartite graph and the proposal of a decentralized and accurate solution based on the BP algorithm. It is worth pointing out that the selected data dissemination application is just a quite popular use case [7], [8], whereas the proposed approach is by no means constrained to a single scenario. In particular, *SIEVE* can be used in any application that uses multi-party download or collaboration, provided that is possible to detect that a given set of collaborating entities is compromised by at least one malicious node. As opposed to cryptographic/algebraic techniques proposed in the area of network coding based wireless mesh networks, e.g., [9], [10], [11], [12] *SIEVE* does not rely on verification tools to check the integrity of every coded block. In *SIEVE* the BP algorithm is used to infer the identity of the malicious nodes resting upon only on a simple pollution detection mechanism; as an example, in our reference scenario, pollution detection is achieved as a by product of the data dissemination protocol based on rateless codes. Furthermore, the aforementioned solutions may not be suitable for MANET since mobility is likely to affect key predistribution, routing mechanisms, attack behavior, etc.

*SIEVE* fits well two key MANETs features that must be accounted for when devising any security solution: it is fully decentralized and does not rely on any infrastructure (as opposed to some solutions in the area of peer-to-peer streaming where special well known nodes are necessary, e.g., [13]). Furthermore, *SIEVE* requires small computational, storage, and communication costs for implementation.

Paper contributions are completed by a comprehensive experimental investigation of *SIEVE* capabilities. Our analysis is carried out by detailed simulations using ns-3; we show that *SIEVE* is accurate in letting each honest node identify all malicious nodes under several scenarios. We analyze the sensitivity of *SIEVE* performance to the nodes speed and we stress test *SIEVE* under several deceiving actions, colluding attacks launched by malicious nodes, and increasing number of malicious nodes. We also discuss an interesting trade-off between rateless code efficiency and *SIEVE* performance. The *SIEVE* technique has been partially presented in [14]. The current paper includes a richer set of experimental results and a more detailed analysis of the obtained performance indexes, worked out on a larger set of system settings.

The paper is organized as follows: Section 2 describes the system model we consider, Section 3 presents the *SIEVE* technique, Section 4 discusses the simulation methodology and the accuracy, reactivity, and robustness results we obtained, Section 5 summarizes other works related to *SIEVE*, finally Section 6 draws conclusions and outlines directions for future developments.

## 2 A USE CASE FOR SIEVE

In this paper we consider a MANET composed of $N$ wireless nodes moving in a given area. A set of $N_{source}$ nodes periodically produces a new data *chunk* to be disseminated to all others once every $h$ seconds. All nodes cooperate to the diffusion of the data chunks by running a distributed dissemination algorithm based on *Luby Transform* (LT) codes [15]. Data is transmitted by source nodes using LT codes [15]: a chunk (whose size in bytes is fixed and is denoted as $S$) is divided in $K$ equally sized blocks. The source node then creates and forwards coded packets using LT codes [15], combining random subsets of the $K$ blocks; the size of each coded packet is $S_{cb} = \frac{S}{K}$.

### 2.1 LT codes

LT codes have been proposed in [15] and represents one of the first embodiment of the class of rateless codes: these are a particular family of erasure codes where the rate is not fixed by design, so that the number of coded packets can be decided and changed on the fly. LT codes are rateless based on the binary Galois field GF(2), i.e., coded packets are computed with simple binary XOR of random subsets of the $K$ original data blocks. In [15] it is shown that selecting the number of blocks to be combined, termed as the packet degree $d$, according to the Robust Soliton Distribution (RSD)[1], one gets optimal asymptotic decoding performance. By optimal performance we mean that the so called decoding overhead, i.e. the number of coded packets to be received in excess of $K$, turns to be negligible for asymptotically large $K$. In particular, the original chunk can be obtained by any node able to collect any set of $M = K \cdot (1 + \epsilon)$ coded packets (on average), where $\epsilon$ is defined as the code overhead. The decoding algorithm can be viewed as the solution of a system of linear equations with $K$ unknowns (the $K$ original data blocks) and $M \geq K$ equations. In [15] it is shown that a simple method based on the recursive cancellation of equations corresponding to packets with degree 1, i.e. representing one original data blocks, guarantees the desired asymptotic performance.

### 2.2 LT based dissemination protocol

The rateless principle and randomness of LT codes is used for spreading data in the MANET by letting source nodes transmit novel coded packets that can be generated randomly and on the fly. A coded packet conveys the XORed payloads of the corresponding original packets as well as a header signaling the indexes of the combined packets. The original chunk can be obtained by any node able to collect any set of $K \cdot (1 + \epsilon)$ coded packets without requiring any coordination among the source nodes. In turn, whenever a node is able to successfully decode a chunk, it can behave as a new source for the chunk, generating and disseminating novel coded blocks.

In Fig. 1 the algorithmic steps of the LT based dissemination protocol implemented by a node are graphically sketched. Each node can simultaneously collect coded packets for different chunks; to this end a window based mechanism is used where the $w_r$ most recent chunks can be concurrently downloaded and progressively decoded. As soon as a given chunk is decoded (this event happens on average when $K \cdot (1 + \epsilon)$ coded blocks of the chunk have been buffered), the corresponding data blocks are moved into a buffer storing the $w_t$ data chunks that have been decoded most recently. A simple round robin scheduling policy is used on the transmitter side, where

1. RSD is derived by the Ideal Soliton Distribution and depends on two free parameters usually identified with constants $c$ and $\delta$ .

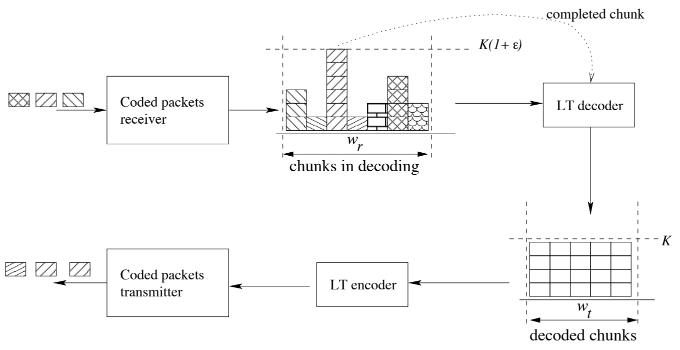

Fig. 1: Node operations: LT encoding, decoding and dissemination protocol.

one of the data chunk is selected, a novel LT coded packet is encoded and transmitted. A new coded packet is transmitted every $T_{tx}$ ms using UDP over an 802.11g wireless communication interface yielding an average transmission range of $r$ meters.

### 2.3 Malicious nodes

The proposed dissemination protocol is an example of a distributed and collaborative approach that has the potential to simplify and accelerate the spreading of the information in the MANET thanks to node mobility. On the other hand, few malicious nodes may try to break the system by polluting, i.e. modifying some coded packets. In this paper we assume that a subset of $P < N$ nodes is composed of *malicious nodes*, that deliberately modify the payload of the coded packets to prevent honest nodes from correctly reconstructing the original chunk. In the presence of coding even a single corrupted coded packet can prevent an honest node from decoding the original chunk.

## 3 The SIEVE protocol

*SIEVE* uses LT codes decoding mechanism to detect modified chunks and exploits the *Belief Propagation* (BP) algorithm [3] to identify malicious nodes.

### 3.1 LT codes verification mechanism

According to the dissemination strategy described in Sect. 2 every node keeps collecting from different uploaders sets of coded blocks corresponding to different chunks. LT codes can be exploited to detect if modified blocks have been collected without the need of any supplementary verification mechanism.

Indeed, a node is able to detect pollution as soon as an inconsistency is found in the solution of the underlying system of linear equations. In particular, according to the procedure [15] the decoder keeps canceling out all the already known data packets. This is achieved by observing that a coded packet with a degree 1 equation represents a data packet in the clear. Such data packet can be simplified from all the incoming equations. Since LT codes have a certain overhead some coded packets that are linearly dependent on the ones received previously are always collected before successful decoding; this amount to the reception of some equations whose terms are all already known. As soon as this condition is met the LT decoder can check the consistency of the payload carried by the coded packet; in other words, the same linear combination must be obtained combining a set of already known packets. If this constraint is violated the whole chunk is recognized as corrupted. Please note that the receiver node is not able to identify the corrupted block(s) but only that at least one of them has been maliciously manipulated.

### 3.2 Check construction and reporting

*SIEVE* is based on the concept of *checks* that are reports created by nodes upon decoding a chunk. A check contains the list of the identifiers of nodes that provided coded blocks of a chunk and a flag to label such chunk as corrupted or not. A check describing a corrupted chunk is called a *positive check* while it is termed a *negative check* otherwise. Each nodes $n$ maintains a list of all checks created that is denoted as $\mathcal{L}_n$.

Each node, besides accumulating the checks from its local decoding operations, gossips them in the neighborhood. Each node $n$ in the MANET transmits its checks in two cases:

- as soon as $n$ decodes a chunk it inserts it in $\mathcal{L}_n$ and broadcasts it;
- once every $T_s$ seconds $n$ randomly selects $Q$ checks in $\mathcal{L}_n$ and transmits them.

### 3.3 Identification based on belief propagation

The checks in $\mathcal{L}_n$ and all checks received by $n$ are used to build a *factor graph* $\mathcal{G}_n$=$(\mathcal{U}, \mathcal{C}, \mathcal{E})$. $\mathcal{G}_n$ is a bipartite graph where the vertex set $\mathcal{U}$ is the set of uploader nodes, the vertex set $\mathcal{C}$ is the set of checks, and an undirected edge $\{i, I\} \in \mathcal{E}$ exists if and only if check $I \in \mathcal{C}$ depends on uploader $i \in \mathcal{U}$.

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 node $i$ contributes as an uploader as $\mathcal{C}_i$. An example of factor graph with four uploaders (circles) and two checks (squares) is show in Fig. 2. The factor graph can be progressively created while decoding the chunks. In Fig. 2 we assume that a node has decoded two chunks corresponding to two checks in its local factor graph. The filled square represents a *positive check* due to the detection of a modified chunk concurrently downloaded by three nodes (in the figure we assume that the rightmost node is malicious). The remaining check is a negative one, appended when a chunk has been decoded successfully. From the point of view of any decoding node, each uploader $i$ can be in one of two hidden states $x_i = 1$ or $x_i = 0$, depending on whether uploader $i$ *is* or *is not* a malicious node. Each check can report one of

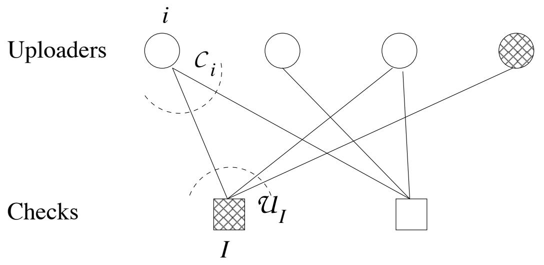

Fig. 2: Example of factor graph.

two observations $c_I = 0$ or $c_I = 1$ in case of negative or positive pollution detection, respectively.

The problem of identifying the malicious nodes from a given number of checks can be recast as an inference problem. The goal of the inference is the estimation of the hidden state of the nodes, 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 uploader nodes by a witness node. In this paper we adopt the BP algorithm [3], [4], [5], [6], that has been used to solve a number of inference problems in many different fields, e.g., iterative channel decoding [16], Bayesian networks [3] and computer vision [17] to mention a few.

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 node $i$ being malicious. BP is an iterative algorithm based on the exchange of probability estimates (also called messages or beliefs), along the edges of the bipartite graph $\mathcal{G}_n$. In case of a Bayesian network BP represents a closed-form solution for the marginals. Nonetheless, the same algorithm has proven to be a robust estimator for the variable marginals of general factor graph [5], [6].

In our setting it is convenient to distinguish between two classes 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 $I$ ($\mathcal{C}_i \setminus I$); 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 uploader states have a separable distribution given by the probabilities $\{m^x_{i'I} : i' \in \mathcal{U}_I \setminus i\}$.

The BP algorithm is based on iterative refinements of the check messages $m^x_{Ii}$ based on the current values of the node messages $m^x_{iI}$ (check pass), followed by updating of $m^x_{iI}$ as a function of $m^x_{Ii}$ (node pass). For the first check pass the messages are initialized to the values $m^0_{iI} = m^1_{iI} = 0.5$, since we assume that no prior information on the number and identity of the malicious nodes is available.

The check pass is based on the estimation of the probabilities $m^x_{Ii}$ as follows:

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

Equation (1) depends on the probability of observing a certain check value $c_I$, given the states of the uploaders of such check.

Given that a check turns out to be positive as soon as at least one of the uploaders is a malicious node, we can write:

$$P\left(c_I = 1 | \{x_i : i = 1, \ldots, k\}\right) = \begin{cases} 0, & \text{if } x_i = 0, \forall i \\ 1, & \text{otherwise} \end{cases} \tag{2}$$

Analogously, observing that a check can be negative if and only if all the uploaders are not malicious, we get

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

Plugging the last two expressions into Equation (1) we can simplify the check pass computation as follows:

$$m^x_{Ii} = \begin{cases} \prod_{i'\in\mathcal{U}_I\setminus i} m^0_{i'I} & \text{if } c_I = 0, x = 0 \\ 0 & \text{if } c_I = 0, x = 1 \\ 1 - \prod_{i'\in\mathcal{U}_I\setminus i} m^0_{i'I} & \text{if } c_I = 1, x = 0 \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

The next BP step is constituted by the updating of the probabilities $m^x_{iI}$, using information from previous computation in the checks (node pass), according to (5):

$$m^x_{iI} = \prod_{I'\in\mathcal{C}_i\setminus I} m^x_{I'i} \tag{5}$$

After any node pass it is possible to get an estimate of the marginal $P(x_i = x)$ according to:

$$P(x_i = x) = \prod_{I'\in\mathcal{C}_i} m^x_{I'i} \tag{6}$$

We recall that the probability estimates given by Equation (5) and (6) are not guaranteed to be normalized; in the practical implementation we use proper normalization constants to avoid numerical issues as suggested in [4].

To conclude, the BP algorithm initializes the values of $m^x_{iI}$, then keeps iterating using Equations (4) and (5). A reliable estimate of $P(x_i = x)$ is computed after a certain number of such iterations using (6). In all the experiments reported in this paper 3 iterations have been used to strike a balance between estimation accuracy and complexity.

## 3.4 BP complexity

The computational cost of single message update using Equation (4) amounts to $|\mathcal{U}_I| - 1$ 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 $|\mathcal{E}|(z_\mathcal{U} - 1)$ multiplications, $z_\mathcal{U}$ being the average number of uploaders per check.

Using the same reasoning from (5) one can compute the cost of the node pass as $|\mathcal{E}|(z_\mathcal{C} - 1)$ multiplications, where $z_\mathcal{C}$ is the average number of checks per node.

Analogously, the final estimate in (6) takes $|\mathcal{E}|z_{\mathcal{C}}$ multiplications.

Finally, taking into accounts all the BP steps and assuming 3 iterations the overall computational complexity amounts to $3(|\mathcal{E}|(z_{\mathcal{U}}-1)+|\mathcal{E}|(z_{\mathcal{C}}-1))+|\mathcal{E}|z_{\mathcal{C}}$ multiplications.

### 3.5 Incremental BP estimation

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

Each node $n$ keeps receiving checks from the other ones and it is allowed to run the BP on $\mathcal{G}_n$ every $T$ seconds considering only the checks received and created 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 $\mathcal{G}_{n,t,w}$ 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. The belief values $m^x_{iI}$ are initialized to 0.5 when the $i$-th node is met for the first time; then, the partial estimates of $m^x_{iI}$ are stored in memory and propagated to all the iterations and computation windows where such values are needed.

After every BP estimation of $P(x_i = x)$, a list of suspect nodes is obtained by setting a threshold on the probability $P_{t,w}(x_i = 1) \geq \eta$. Each node $n$ keeps a counter for each of its uploader nodes $i$: the nodes in the list of suspects after the BP run have their counter increased by 1. Finally, a *suspects ranking* $\mathcal{R}_n$ over uploader nodes is defined by sorting their counters in decreasing order. As an example, the first node 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 $t$.

## 4 Results

In this section we describe the simulation methodology and the indexes we defined to evaluate the accuracy, reactivity, and robustness of *SIEVE*. In particular, we show how the structural properties of the factor graph $\mathcal{G}_n$ and mobility impact the performance of *SIEVE* as well as how robust it is with respect to several deceiving actions operated by malicious nodes. Finally, we show a trade-off between coding efficiency and *SIEVE* accuracy.

### 4.1 Factor graph and performance

According to [18] when short cycles are absent in the factor graph $\mathcal{G}_n$ it is best to have high degree nodes (i.e., uploaders that contribute to many checks) and low degree checks (i.e., checks obtained from a small number of uploaders). High degree nodes (both malicious and honest) tend to their correct marginal probability $(P(x_i))_{i\in\mathcal{U}}$ (i.e. the probability of node $i$ being malicious) quickly; on the other hand, the lower the number of uploaders in a check the more valuable the information it conveys.

Unfortunately, the actual factor graphs computed by nodes do contain cycles and it is well known that cycles in the factor graph negatively impact on the accuracy of the probability estimates yielded by the BP algorithm [18]. The shortest length of a cycle in a bipartite graph with at most one edge between any two nodes is 4 and is due to the presence of at least two common uploaders in a pair of checks, i.e. the cardinality of the intersection between the uploaders of a pair of checks is at least 2. If the cardinality of the intersection is $x > 2$, then each of the $\binom{x}{2}$ pairs defines one length 4 cycle. Equivalently, a cycle forms if any two nodes contributed to provide blocks of at least two common checks. It is easy to note that the higher the degree of nodes and/or checks the higher the number of cycles in the factor graph.

In this context, the benefits of high degree nodes and low degree checks are counter-balanced by the negative effects of cycles. Indeed, increasing the average number of checks per node improves the performance of the BP algorithm up to a point where the number of cycles increases too much and lowers the accuracy of the BP algorithm. Similarly, the benefits of a lower average check degree are offset by a high number of cycles.

### 4.2 Simulation methodology

The performance of *SIEVE* have been investigated by simulations using ns-3 version 3.12 [19] on a Red Hat 4.4.6-3 machine using standard variable settings. In particular, we developed a node object implementing the sender and receiver functionalities according to the protocol described in Sections 2 and 3. The MANET is composed of $N$ wireless nodes placed in a square area whose side length is $l$ meters. Nodes move using different average speeds. $N_{fast}$ nodes moving at $V_{fast}$ m/s (e.g., cars, buses, motorcycles), $N_{slow}$ nodes at $V_{slow}$ m/s (e.g., pedestrian, bikes), and $N_{still}$ fixed nodes (e.g., sensors, relay stations, shops). As a consequence we have $N = N_{fast} + N_{slow} + N_{still}$. All non-fixed nodes movements are described by the same mobility model and moving nodes are randomly distributed across the simulation area at the beginning of the experiment. Still nodes are distributed on a grid to avoid clustering in a particular region. We used the ns-3 default transmission model composed of a propagation delay model (a constant value model) and a propagation loss model (based on a log distance model with a reference loss of 46.677 dB at a distance of 1 meter). Under these settings we derived the maximal transmission range $r$ as 175 meters.

We conducted simulation experiments that terminate after 1 hour of simulated time. We used the independent replication method to obtain $N_{EXP} = 30$ executions and computed 95% confidence intervals. Each execution is obtained with different streams of the random number

| Parameter | Description | Value |
|---|---|---|
| $N_{fast}$, $N_{slow}$ | Number of fast/slow nodes | 100 |
| $N_{still}$ | Number of still nodes | 50 |
| $P_{fast}$, $P_{slow}$ | Number of fast/slow polluters | 5 |
| $P_{still}$ | Number of still polluters | 0 |
| $K$ | Number of blocks | 100 |
| $S_{cb}$ | Size of blocks | 500 Bytes |
| $T_{tx}$ | Time between transmissions of blocks | 100 ms |
| $c$, $\delta$ | RSD parameters for LT encoder | 0.01 |
| $l$ | Square area side length | 2000 m |
| $h$ | Time between data chunk generation | 600 s |
| $w_r$ | Window size for concurrent chunk downloading | 50 |
| $w_t$ | Window size for concurrent chunk uploading | 6 |
| $r$ | Maximal transmission range | 175 m |
| $V_{fast}$ | Fast nodes speed | [10-40] m/s |
| $V_{slow}$ | Slow nodes speed | [1-5] m/s |
| - | Mobility model | 2D RW |

TABLE 1: Parameter values for the reference scenario.

generator provided by ns-3. The simulation output is made of a log file containing all checks received by each node that is post-processed by our analysis software to run the *SIEVE* protocol to detect malicious nodes.

All results have been obtained by setting as initial data sources only the $N_{still}$ still nodes; all $N$ nodes contribute to the spreading of the information according to the policy described in Section 2. A subset of nodes ($P = P_{fast} + P_{slow} + P_{still} < N$) are simulated as malicious nodes that modify the coded packets. All the system parameter values of the reference scenario considered in the following are summarized in Table 1.

## 4.3 Performance indexes

For each node $n$ in the system we define:

- $p(n)$ an indicator function whose value is equal to 1 if node $n$ is not malicious and 0 otherwise;
- $c_n(t)$ an indicator function whose value is equal to 1 if $\mathcal{R}_n$ is non empty at time $t$ and 0 otherwise;
- $a_n(t)$ the number of actual malicious nodes correctly identified by node $n$ at time $t$. More precisely, $a_n(t) = x$ if $x$ nodes in the top $P$ positions of $\mathcal{R}_n$ are truly malicious;
- $r_n(t)$ the number of actual malicious nodes correctly identified in the top positions of $\mathcal{R}_n$ at time $t$. More precisely, $r_n(t) = x$ if *top* $x$ nodes in $\mathcal{R}_n$ are all actually malicious. If *SIEVE* at time $t$ has identified $P-1$ out of $P$ malicious nodes ($a_n(t) = P-1$) but the first node in $\mathcal{R}_n$ is not malicious then $r_n(t) = 0$.
- $t_{n,trigger}$ the time node $n$ received the very first positive check that triggered the *SIEVE* activation;
- $t_{n,hit}(x) = min_t\{t : r_n(t) \geq x\}$, that is, the minimum time to identify at least $x$ malicious nodes in the top $x$ positions of $\mathcal{R}_n$.

In each independent replication trial we compute the following averages:

- $c^{(i)}(t) = \frac{\sum_{n=1}^{N} p(n)c_n(t)}{\sum_{n=1}^{N} p(n)}$, i.e., the fraction of honest nodes that detected the attack and have a non empty $\mathcal{R}$;
- $a^{(i)}(t) = \frac{1}{P}\frac{\sum_{n=1}^{N} p(n)c_n(t)a_n(t)}{\sum_{n=1}^{N} p(n)c_n(t)}$, i.e., the average accuracy of honest nodes with non empty $\mathcal{R}$. When $a^{(i)}(t) = 1$ it means that *all* honest nodes with non empty $\mathcal{R}$ have correctly identified *all* malicious nodes.
- $tsi^{(i)}(x) = \frac{\sum_{n=1}^{N} p(n)(t_{n,hit}(x) - t_{n,trigger})}{\sum_{n=1}^{N} p(n)}$, i.e., the average time honest nodes require to unambiguously identify $x$ malicious nodes.

Finally, we computed averages and 95% confidence intervals over $N_{EXP}$ executions and considered $c(t)$, $a(t)$, and $tsi(x)$ that we call completeness, accuracy, and time to identification, respectively. In all computations we set $\eta = 0.99$ ($\eta$ is the threshold on the probability $P_{t,w}(x_i = 1)$ for suspect identification) and iterated each run of the BP algorithm on a specific factor graph three times.

## 4.4 Sensitivity results

The first analysis we conducted is on the sensitivity of $c(t), a(t)$, and $tsi(x)$ to parameters $w$, $T$, $T_s$, and $Q$ used by *SIEVE* as defined in Section 3. To this end we considered integer multiples of 10s values for $w$ ranging from 10s to 90s, $T = \{5, 10\}$s, $T_s = \{5, 10, 15\}$s, and $Q = \{5, 10\}$. We analyzed all 108 combinations and observed that *SIEVE* performance is most sensitive to the value $w$.

Figure 3 shows the performance indexes $a(t)$, $c(t)$, and $tsi(x)$ for some values of $w$, when $T = T_s = 10$ s and $Q = 10$ (all 108 configurations yielded similar results so we selected a representative set of curves). It can be noted that too small or too large values for $w$ yield the worst performance for all the performance indexes. Indeed, small window sizes do not allow the factor graph to include enough checks and nodes to accurately infer the node status; on the other hand, large values of $w$ provide more checks and nodes in $\mathcal{G}_n$ but increase the number of cycles in the factor graph which in turn impact on the accuracy of the probability estimates yielded by the BP algorithm, as discussed in Section 4.1.

We summarized the structural properties of $\mathcal{G}_n$ in Table 2. It can be noted that the average number of nodes per check $z_{\mathcal{U}}$ is not dependent on $w$ (it mostly depends on the overall number of nodes, the mobility patterns, and the data exchange protocols). On the contrary, the size of $\mathcal{G}_n$ ($|\mathcal{C}|$ and $|\mathcal{U}|$) increases as $w$ increases although the number of nodes saturates since it is upper bounded by $N$. Accordingly, the average number of checks per node $z_{\mathcal{C}}$ increases and it positively impacts on *SIEVE* accuracy up to the value $w = 60$; for larger values the average number of cycles in the factor graph increases and nullifies the increase of $z_{\mathcal{C}}$.

It can also be noted that $c(t)$ approaches 1 after about 4 minutes; the behavior of $c(t)$ clearly depends on the delay after which a node has collected a sufficient amount of checks to start its own suspect ranking. As for *SIEVE* reactivity, we observe that $w = 60$s yields

| $w$ | $\lvert\mathcal{C}\rvert$ | $z_{\mathcal{U}}$ | $\lvert\mathcal{U}\rvert$ | $z_{\mathcal{C}}$ | length 4 cycles |
|---|---|---|---|---|---|
| 10 | 83.1688 | 8.53656 | 173.7 | 3.8633 | 1175.57 |
| 30 | 247.617 | 8.55307 | 216.824 | 9.29914 | 8899.69 |
| 60 | 492.807 | 8.5262 | 231.687 | 17.3395 | 32021.6 |
| 90 | 735.944 | 8.49615 | 236.444 | 25.3386 | 68064.4 |

TABLE 2: Structural properties of $\mathcal{G}_n$ for increasing values of $w$.

the lowest values for the time to identification $tsi(x)$ (all graphs for $tsi(x)$ are computed for the maximum $x$ such that for each honest node $n$ it is true at the end of the simulation experiment that $r_n = x$). Based on the above discussion on the impact of $w$ on the performance of SIEVE, we selected $w = 60$s as the value for all the following experiments.

Figure 4 shows a comparison among different choices of the values for parameters $T$, $T_s$, and $Q$ (please recall that each node executes a BP estimation once every $T$ seconds and transmits a selection of $Q$ checks once every $T_s$ seconds). In particular, we selected three settings for three different values of the number of checks transmitted per seconds ($\frac{Q}{T_s} = 2, 1, 0.33$). It can be noted that performance of SIEVE achieve their best when the BP estimation is run more frequently ($T = 5$s) and the the number of checks transmitted per seconds is the highest. Nevertheless, improved performance come at the cost of increased computation and communication burden on nodes; this may cause concerns for battery operated nodes whose lifetime could be shortened. Based on the above reasoning, we selected a compromise between performance and energy consumption awareness using the following setting for the rest of the paper: $w = 60$s, $T = T_s = 10$s, $Q = 10$.

## 4.5 Mobility and SIEVE performance

The first interesting observations we made is that mobility affects *SIEVE* performance. In particular, the performance depends not only on the speed but also on the mobility model adopted for nodes.

Figure 5 shows the performance of *SIEVE* in the reference scenario with different mobility models available in ns-3, i.e. 2D random walk, random direction, Gauss-Markov and steady state random waypoint; clearly, all mobility models have been compared for the same node speeds. It can be noted that the 2D random walk mobility model selected for the reference scenario yields slightly worse performance with respect to other mobility models that more closely represent mobility in a urban environment.

Figure 6 shows the performance of *SIEVE* in the reference scenario but averaged over nodes in the same speed class (fast, slow, still). It can be noted that accuracy of fast nodes is higher with respect to slow and still nodes. Reaction times are also lower for fast moving nodes.

Figure 7 shows the overall performance of *SIEVE* in the reference scenario and in scenarios with different mixes of fast and slow nodes. The system where all moving nodes are fast yields much higher accuracy and much lower reaction times with respect to other extreme case where all nodes move slowly. Nevertheless, *SIEVE* accuracy is 0.83 at the end of the one hour long experiments and approaches 1 for longer runs.

Figure 8 shows results in the reference scenario and in scenarios where all malicious nodes are either fast or slow. This results show that malicious nodes can delay their identification if they move slowly; on the other hand if all malicious nodes move fast all of them are quickly identified.

Although *SIEVE* is able to identify all malicious nodes in the long run in any setting, in all three cases we can conclude that *high speed is key to obtain both high accuracy and low reaction delays by honest nodes*.

Why is higher speed beneficial for *SIEVE* performance of honest nodes but detrimental for malicious nodes? Consider the case where different mixes of fast and slow nodes (Figure 7) are compared and assume an extreme scenario composed of all still nodes. Since nodes do not move it is possible to define a static geometric graph $\mathcal{O}$ describing connections among nodes where vertexes are nodes and an undirected edge between two vertexes exists if the corresponding nodes are within the transmission radio range $r$. For any node $n \in \mathcal{O}$ let $\mathcal{N}(n)$ denote the set of nodes connected to $n$ (the neighborhood of $n$).

Consider one data source $s$ and a chunk $c$; clearly, when $s$ has transmitted $K \cdot (1 + \epsilon)$ coded packets, $c$ is decoded by all nodes in $\mathcal{N}(s)$; in turn, these nodes start transmitting fresh coded packets for $c$ and, if $\mathcal{O}$ is connected, all nodes will be able to decode $c$ after some time. It can be noted that all coded packets produced by $s$ will follow the same paths in $\mathcal{O}$. This means that for any node $n \in \mathcal{O}$ chunks originating from $s$ will be provided to $n$ by *the same set of uploaders* $\mathcal{U}_s(n)$ that is a subset of $\mathcal{N}(n)$. Of course, the same reasoning is valid for all checks describing chunks produced by any data source other than $s$. The final effect is that the set of checks created by each node $n$ upon decoding chunks ($\mathcal{L}_n$) is such that strong intersections exist among the uploaders of different checks (given that $\mathcal{O}$ is static and $\mathcal{N}(n)$ does not change over time); this translates into a high number of short cycles in the factor graph used by *SIEVE* that is a well known cause of poor performance of the BP algorithm, as discussed in Section 4.1.

Instead, when nodes move the geometric graph $\mathcal{O}$ becomes dynamic, i.e., for any node $n$ its $\mathcal{N}(n)$ varies with time. Speed translates into higher rate of changes in $\mathcal{N}(n)$ that, in turn, reduces the number intersections among the uploaders of different checks (hence of short cycles in $\mathcal{G}_n$) and increases the accuracy of BP algorithm. For the reference scenario we computed the structural properties of $\mathcal{G}_n$ that are summarized in Table 3. It can be noted that the average number of nodes per check ($z_{\mathcal{U}}$) and the average number of checks per node ($z_{\mathcal{C}}$) for still nodes would suggest better performance with respect to

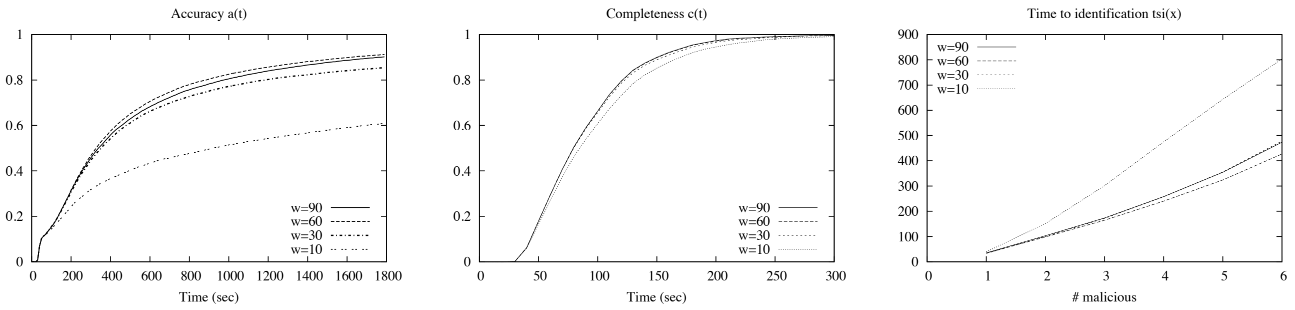

Fig. 3: Representative accuracy (left), completeness (middle), and time to identification (right) of *SIEVE* as a function of $w$.

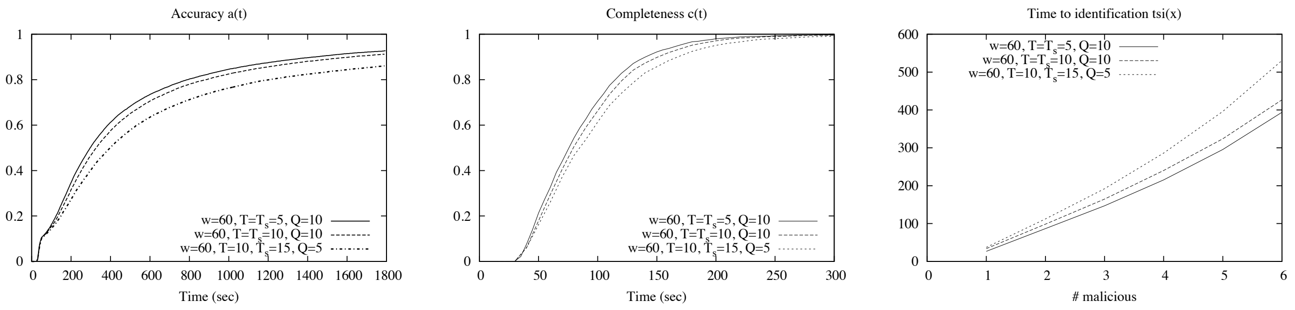

Fig. 4: Accuracy (left), completeness (middle), and time to identification (right) of *SIEVE* for best performance, lower energy consumption, and compromise setting.

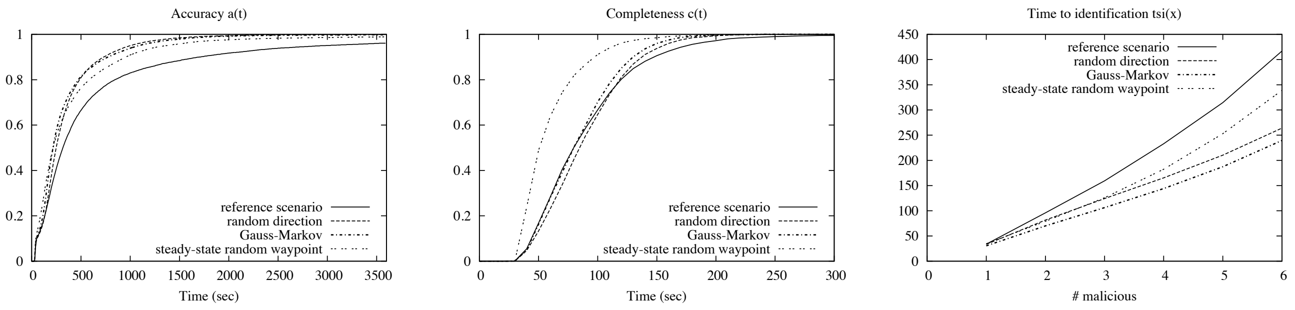

Fig. 5: Accuracy (left), completeness (middle), and time to identification (right) of *SIEVE* for different nodes mobility models.

| speed | $\|\mathcal{C}\|$ | $z_{\mathcal{U}}$ | $\|\mathcal{U}\|$ | $z_{\mathcal{C}}$ | length 4 cycles |
|---|---|---|---|---|---|
| fast | 434.14 | 8.6891 | 232.594 | 15.8041 | 23240.5 |
| slow | 428.937 | 8.91507 | 230.292 | 15.8437 | 29242.7 |
| still | 737.88 | 7.42264 | 232.663 | 23.4016 | 55141.4 |

TABLE 3: Structural properties of $\mathcal{G}_n$ for different nodes speed in the reference scenario.

fast and slow nodes. Nevertheless, the average number of length four cycles is *much* higher for still nodes and this is the main reason for worse performance.

### 4.6 Deceiving actions and SIEVE robustness

Besides following the data dissemination and *SIEVE* protocols, malicious nodes may also implement disturbing actions aiming at preventing or delaying their identification. In the following a number of deceiving actions are investigated.

- Reduced pollution intensity at the coded packet level: a malicious node modifies a coded packet with probability $p_{poll}$ (the reference scenario is analyzed with $p_{poll} = 1$). Figure 9 shows how *SIEVE* performs when nodes lower their pollution intensity to $p_{poll} = 0.5, 0.25, 0.125$ in the attempt of making their identification harder. It can be noted that *SIEVE* is able to spot malicious nodes in all cases although very low pollution intensities delay identification. Nevertheless, it is worth pointing out that tuning the pollution intensity yields a natural trade-off between speed of identification and average damage

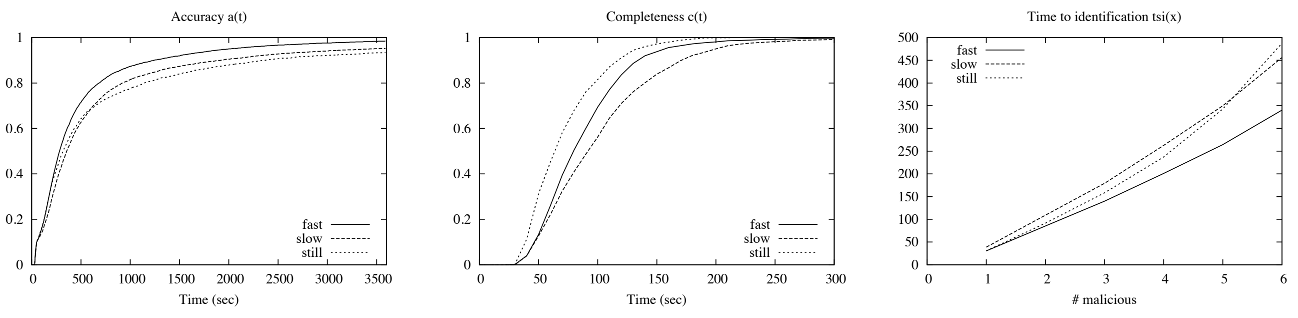

Fig. 6: Accuracy (left), completeness (middle), and time to identification (right) of *SIEVE* for different nodes speeds in the reference scenario.

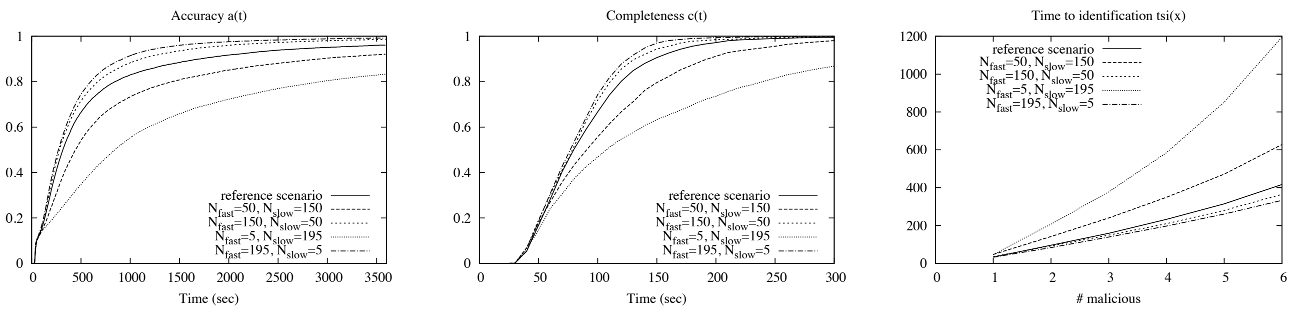

Fig. 7: Accuracy (left), completeness (middle), and time to identification (right) of *SIEVE* where moving nodes range from all fast to all slow.

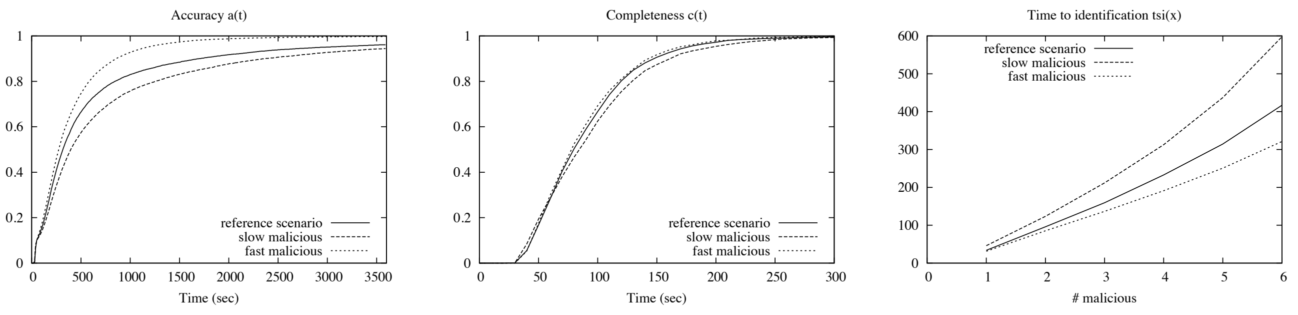

Fig. 8: Accuracy (left), completeness (middle), and time to identification (right) of *SIEVE* where malicious nodes are either all fast or all slow.

caused to honest nodes, i.e., the lower $p_{poll}$ the lower the number of polluted chunks and the longer the time to identification.

- Reduced pollution intensity at the chunk level: a malicious node pollutes a chunk with probability $c_{poll}$ and the corresponding coded packets are modified with probability $p_{poll}$. In Figure 10 we show results for $c_{poll} = 0.75, 0.5$ and $p_{poll} = 1$, comparing them to the reference scenario (analyzed with $c_{poll} = p_{poll} = 1$ ). Also in this case, we observe that malicious nodes are identified by *SIEVE* albeit at the price of some delay for low values of $c_{poll}$.
- Check status falsification: with probability $p_{lie}$ a positive check obtained by a malicious node is forwarded as negative and viceversa. Figure 11 shows that this trick is ineffective: indeed, the only effect is a limited increase in the values of $tsi(x)$. Figure 12 also shows that *SIEVE* is able to achieve high accuracy even in extreme cases when half of the moving nodes is malicious. Indeed, provided that the system operates for long enough time, *SIEVE* is able to allow every node to identify all existing malicious nodes.
- Honest nodes disparaging: malicious nodes always produce dummy positive checks involving a set of honest nodes. In this case malicious nodes flip a coin and, with probability $p_{disparage}$, they replace the actual uploaders of a check with a set of honest nodes; the detection flag is marked as positive. Figure 13 shows *SIEVE* performance when $p_{disparage} = 1$ and

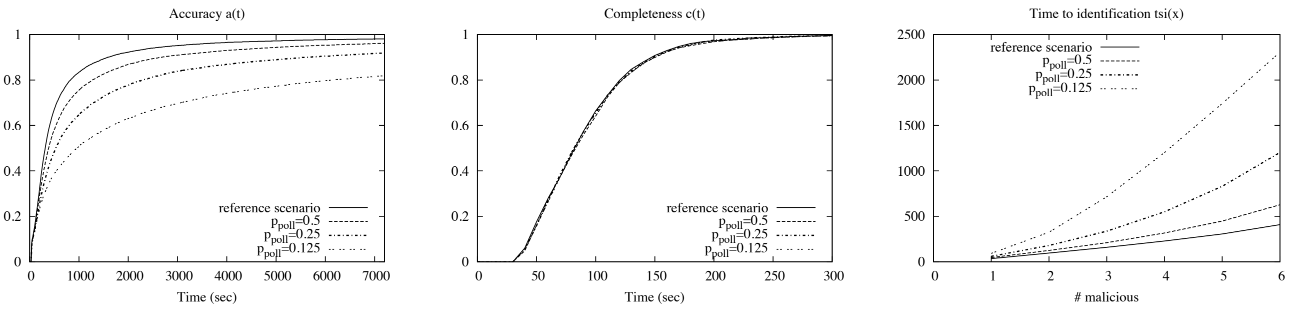

Fig. 9: Accuracy (left), completeness (middle), and time to identification (right) of *SIEVE* for different pollution intensities at the coded block level ($p_{poll}$).

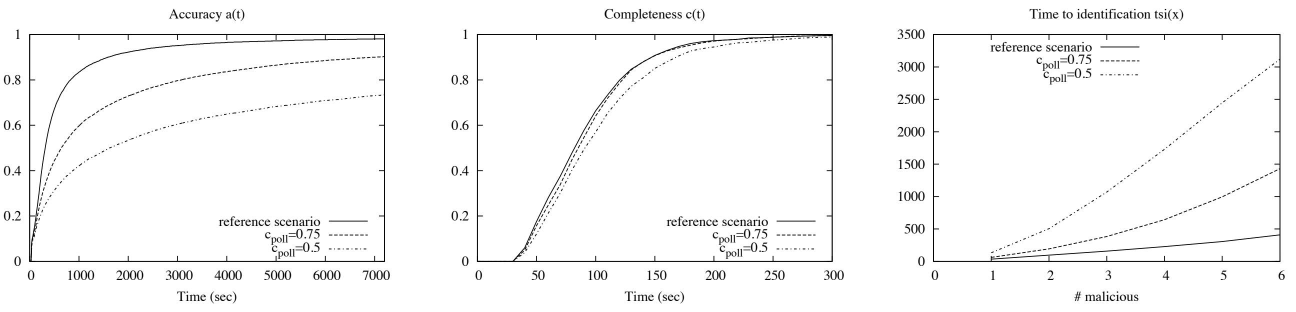

Fig. 10: Accuracy (left), completeness (middle), and time to identification (right) of *SIEVE* for different pollution intensities at the chunk level ($c_{poll}$).

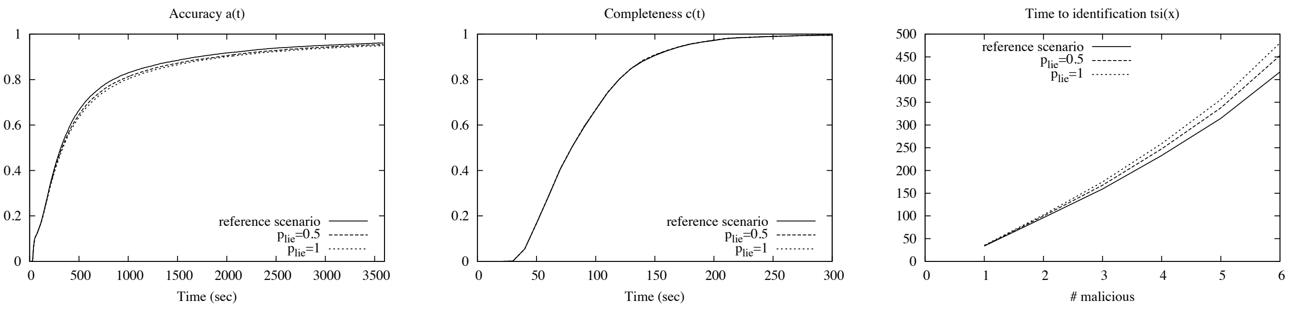

Fig. 11: Accuracy (left), completeness (middle), and time to identification (right) of *SIEVE* for different lying intensities.

honest nodes are either randomly chosen or chosen in the same fixed order followed by all malicious nodes (this is a colluding attack to honest nodes). It can be noted that, by colluding, malicious nodes only succeed in slightly delay their identification. As a matter of fact, *SIEVE* is able to correctly identify all malicious nodes anyway at the end of the simulation experiments.

- Increasing the number of malicious nodes: the last stress test for *SIEVE* is to consider its performance for an increasing number of malicious nodes that coordinate to launch the colluding attack. In Figure 14 we show results for up to $P = 100$ malicious nodes in the system (in all cases, $P_{fast} = P_{slow} = \frac{P}{2}$). It can be noted that *SIEVE* reaches high accuracy at the end of the simulation experiments with an increase of the $tsi(x)$ values, provided that the system runs for a long enough time.

## 4.7 Coding efficiency vs. SIEVE performance

Coding efficiency is defined as the capability of decoding with the smallest possible overhead $\epsilon$. It is well known that LT codes overhead decreases as $K$ increases [15]; therefore high values of $K$ are preferred in most cases. We compared the *SIEVE* performance for different values of $K$ and $S_{cb}$ to keep the chunk size constant; in particular, Figure 15 shows results for $K = 50, S_{cb} = 1000$ and $K = 200, S_{cb} = 250$. It can be noted that small values of $K$ yield better accuracy and lower reaction times.

Table 4 summarizes the structural properties of $\mathcal{G}_n$ for different values of $K$. It can be noted $K = 50$ yields both lower average number of nodes per check and length

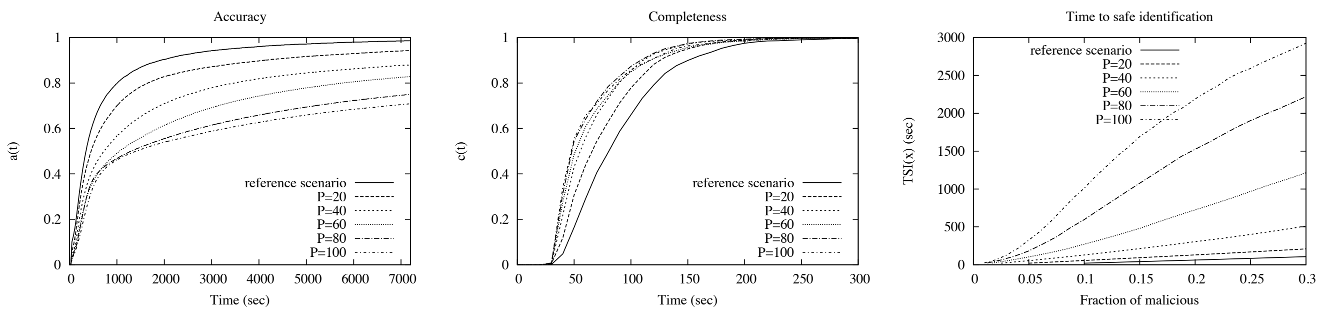

Fig. 12: Accuracy (left), completeness (middle), and time to identification (right) of *SIEVE* for $p_{lie} = 1$ and increasing number of malicious nodes.

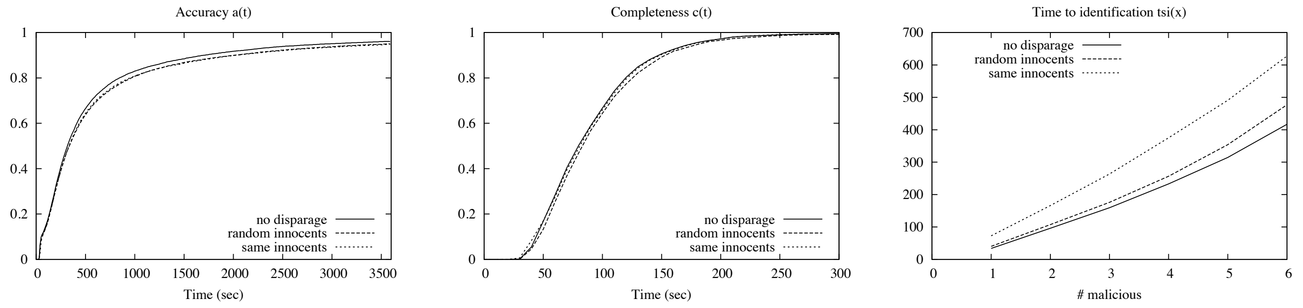

Fig. 13: Accuracy (left), completeness (middle), and time to identification (right) of *SIEVE* for different disparaging attacks.

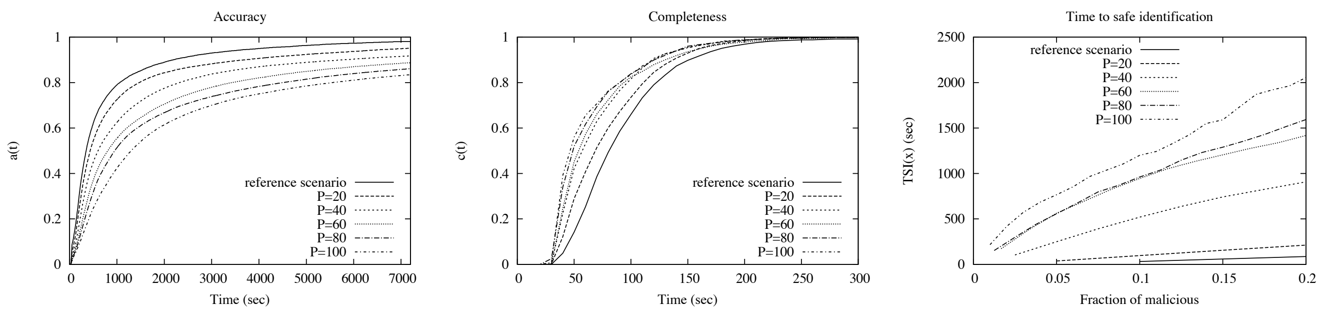

Fig. 14: Accuracy (left), completeness (middle), and time to identification (right) of *SIEVE* for increasing number of colluding malicious nodes.

| $K$ | $\lvert\mathcal{C}\rvert$ | $z_{\mathcal{U}}$ | $\lvert\mathcal{U}\rvert$ | $z_{\mathcal{C}}$ | length 4 cycles |
|---|---|---|---|---|---|
| 50 | 499.447 | 6.10647 | 230.457 | 12.9598 | 11508.5 |
| 100 | 492.807 | 8.5262 | 231.687 | 17.3395 | 32021.6 |
| 200 | 480.232 | 11.3097 | 229.752 | 22.1949 | 94461.8 |

TABLE 4: Structural properties of $\mathcal{G}_n$ for different values of $K$.

four cycles.

As a side effect of using a smaller value for $K$, the probability of receiving a coded packet from a malicious node is lower as confirmed by the attack damage, i.e., the average fraction of corrupted decoded chunks, that is equal to 0.20 for $K = 50$ and 0.33 for $K = 200$.

## 4.8 Bandwidth, memory and CPU costs

The bandwidth overhead required by a node to implement SIEVE is determined by the number of bits necessary to transmit a check when decoding a data chunk plus the bits required by the broadcasting of $Q$ checks every $T_s$ seconds. In our simulations a check $I$ is represented by using a message whose payload size is equal to $b_I = 32|\mathcal{U}_{\mathcal{I}}| + 1$ bits; in this expression $32|\mathcal{U}_{\mathcal{I}}|$ bits are required to store the node identifiers represented as integers and one bit is used for the chunk corruption flag.

From the simulations of the reference scenario, we measured an average value of the bandwidth overhead equal to 301 bps per node (95% confidence interval $[193, 408]$). We also measured the average throughput, defined as the number of decoded bits (of data chunks)

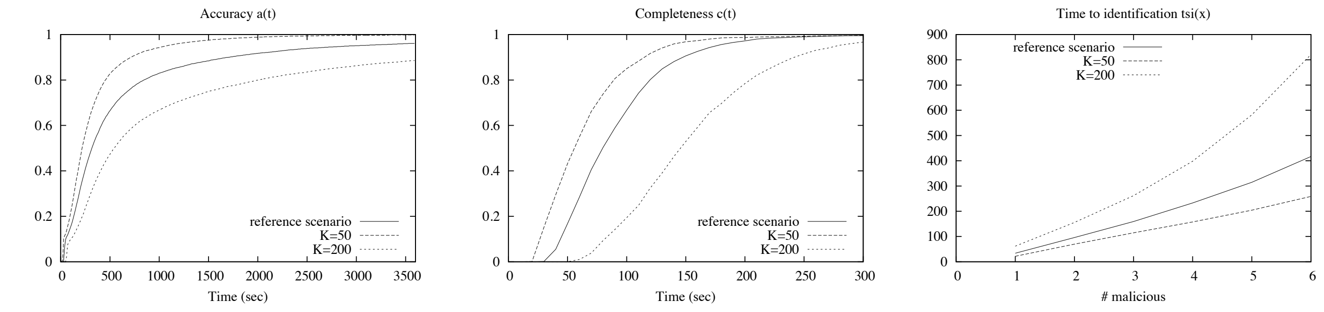

Fig. 15: Accuracy (left), completeness (middle), and time to identification (right) of *SIEVE* for different values of $K$.

per second, that is equal to 20483 bps (95% confidence interval $[13162, 27803]$). It follows that SIEVE communication overhead, defined as the ratio between bandwidth overhead and throughput, is limited to about $1.5\%$. It is worth noticing that one contribution to the bandwidth overhead depends on $Q$ and $T_s$ and can therefore be traded-off. As an example, the SIEVE communication overhead for $Q = 5$ and $T_s = 15$s (see Fig. 4) turns out to be about the $0.5\%$.

One of the main requirements a solution for MANET must satisfy is to have a low computational and memory cost. We measured the average CPU time experienced to run *SIEVE* on a single factor graph in our C++ implementation on an Intel(R) Core i5 2.80GHz CPU: we obtained 25ms. Of course, MANET nodes do not have the same computational power of a desktop PC but newer CPUs equipping tablets and smart phones are reducing the gap, e.g., the ARM Cortex-A9 MPCore has up to 4 cores, 2 GHz clock, and 10,000 DMIPS.

Furthermore, storage requirements are very low: the average number of checks in $\mathcal{G}_n$ for fast, slow, and still nodes is 435, 423, and 742, respectively. The average number of nodes in $\mathcal{G}_n$ is equal to 227, 224, and 228, respectively. The factor graph must be represented as a dynamic undirected bipartite graph. Several data structures can be used. In our implementation for each check $I$ (received during the last $w$ seconds) we store the set of uploaders $i \in \mathcal{U}_I$, each one representing an arc of the factor graph (represented as the pair of identifiers $(I, i)$). Each node/check identifier is represented as a 32 bit integer; therefore, the number of bits for storing the factor graph is equal to $2 \cdot 32 \cdot |\mathcal{E}|$ bits. In our experiments, the average number of arcs of the factor graph is equal to 3823, 3962, and 5790 (for fast, slow and still nodes respectively). It follows that the average memory requirements is equal to 30.5 kB, 31.7 kB, and $46.3$ kB, respectively.

## 5 Related work

MANET are vulnerable to attacks at any layer of the Internet model [1], [2]. In the area of network coding several efforts have been devoted to devise on-the-fly verification techniques carried out by participants [20], [21], [22], [23], [24], [25], [26] to identify the sources of corrupted data. 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. In [9], [10], [11] previous works have been extended to limit the communication overhead and achieve high level of robustness to pollution in the context of network coding applied to static wireless mesh networks.

Error correction (and algebraic) approaches have been devised to deal with data corruption attacks in network coding [27], [28], [29]; these methods introduce coding redundancy to allow receivers to correct errors but their effectiveness depends on the amount of corrupted information. Recently, [12] has proposed an extension of previous work which aims at limiting the communication overhead required for verification of coded packets.

The data corruption attack we consider in this paper is a well-known plague in peer-to-peer streaming systems. Unfortunately, all solutions developed in that research are not easily adoptable in MANET. The work by Wang et al. [13] 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. This solution requires a centralized monitoring and management point that is not available in MANET. Therefore, it is hard (if not impossible) to adapt in the context we consider in our work.

The work by Li and Lui [30] 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 same approach has been also adopted in network coding wireless mesh networks [31], [32], where the analysis is restricted on the backbone network consisting of stationary (with minimal mobility) mesh routers. The technique is attractive thanks to its simplicity and fully distributed nature although performance deteriorates when multiple polluter exist. Nonetheless, the technique works when then neighborhood of a node does not vary with time. Also in this case a comparison with our work is quite difficult.

The work by Jin et al [33] 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. Unfortunately, this capability is not available in the system we consider in our work: the original chunk can only be obtained if there are no malicious nodes among the chunk providers and if corruption is detected the honest node only knows that at least one of the chunk providers is malicious. In this case, comparison with our work is simply not possible.

## 6 Conclusions

In this paper we have proposed the novel *SIEVE* technique for the identification of malicious nodes performing a pollution attack within a MANET. A data dissemination application based on the distribution of LT coded data packets has been considered as a use case, where malicious nodes aim at preventing the delivery of the information corrupting, i.e., polluting, coded packets. In turn, the reception of a single corrupted packet may break down the decoding of a whole data chunk. Fortunately, the LT decoding procedure can be used by each node to detect that a data chuck has been attacked; nonetheless, since parallel downloading form multiple nodes is used, such detection does not allow to identify the malicious nodes. This latter represents the core problem solved by *SIEVE*. In *SIEVE* node collaborations are represented by a bipartite graph linking nodes and detection opportunities, the checks. It is worth pointing out that such representation is quite general and can be used in many other collaborative scenarios other than the data dissemination use case analyzed in this paper.

*SIEVE* is a completely distributed technique that, using statistical inference based on the belief propagation algorithm, allows each node to independently analyze local snapshots of the bipartite graph of the collected checks to estimate the probability of nodes being malicious and to perform a progressive ranking of the suspect nodes. Our results, worked out using detailed ns-3 simulations, show that *SIEVE* is accurate in letting each honest node identify all malicious nodes under several scenarios. We have analyzed the sensitivity of *SIEVE* performance to the nodes mobility; we have discovered that *SIEVE* is very robust to several deceiving actions, colluding attacks launched by malicious nodes, and amount of malicious nodes inside the network.

Future works will be focused in two main directions. From the one hand we will complete the design and experimentation of a full system to counteract an active attack within MANET, including both malicious node identification and their subsequent isolation or removal. The techniques adopted for the identification and the following removal of malicious nodes clearly require a joint and careful design to optimize the overall performance. The other research line will be devoted to better understand the features of the bipartite graph representation, e.g. presence of loops, average node and check degrees, that affect *SIEVE* accuracy. Such study is expected to let us further improve its performance by adding the checks in a smart and adaptive way guaranteeing better convergence of the BP algorithm.

## References

[1] B. Wu, J. Chen, J. Wu, and M. Cardei, "A survey of attacks and countermeasures in mobile ad hoc networks," in *Wireless Network Security*, ser. Signals and Communication Technology, Y. Xiao, X. S. Shen, and D.-Z. Du, Eds. Springer US, 2007, pp. 103–135.

[2] H. Yang, H. Luo, F. Ye, S. Lu, and L. Zhang, "Security in mobile ad hoc networks: challenges and solutions," *IEEE Wireless Communications*, vol. 11, no. 1, pp. 38–47, 2004.

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

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

[5] 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.

[6] ——, "Understanding belief propagation and its generalizations," in *Exploring Artificial Intelligence in the New Millennium*, ser. Science & Technology Books. Elsevier, 2003, ch. 8.

[7] T. Schierl, S. Johansen, A. Perkis, and T. Wiegand, "Rateless scalable video coding for overlay multisource streaming in manets," *Journal of Visual Communication and Image Representation*, vol. 19, no. 8, pp. 500–507, 2008.

[8] V. R. Syrotiuk, C. J. Colbourn, and S. Yellamraju, "Rateless forward error correction for topology-transparent scheduling," *IEEE/ACM Transactions on Networking*, vol. 16, no. 2, pp. 464–472, 2008.

[9] S. Agrawal and D. Boneh, "Homomorphic macs: Mac-based integrity for network coding," in *Applied Cryptography and Network Security*. Springer, 2009, pp. 292–305.

[10] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical defenses against pollution attacks in intra-flow network coding for wireless mesh networks," in *Proceedings of the second ACM conference on Wireless network security*. ACM, 2009, pp. 111–122.

[11] Y. Li, H. Yao, M. Chen, S. Jaggi, and A. Rosen, "Ripple authentication for network coding," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[12] A. Newell and C. Nita-Rotaru, “Split null keys: A null space based defense for pollution attacks in wireless network coding,” in *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012 9th Annual IEEE Communications Society Conference on*, 2012, pp. 479–487.
[13] Q. Wang, L. Vu, K. Nahrstedt, and H. Khurana, “MIS: Malicious nodes identification scheme in network-coding-based peer-to-peer streaming,” in *IEEE INFOCOM 2010*, march 2010, pp. 1–5.
[14] R. Gaeta, M. Grangetto, and R. Loti, “SIEVE: a distributed, accurate, and robust technique to identify malicious nodes in data dissemination on manet,” in *IEEE ICPADS 2012*, 2012, pp. 331–338.
[15] M. Luby, “LT codes,” in *43rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2002*, 2002, pp. 271–280.
[16] R. Gallager, *Low-Density Parity-Check Codes*. Cambridge: M.I.T. Press, 1963.
[17] 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.
[18] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, “Improved low-density parity-check codes using irregular graphs,” *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 585 –598, feb 2001.
[19] G. F. Riley and T. R. Henderson, “The ns-3 network simulator,” in *Modeling and Tools for Network Simulation*. Springer Berlin Heidelberg, 2010, pp. 15–34.
[20] M. N. Krohn, M. J. Freedman, and D. Mazieres, “On-the-fly verification of rateless erasure codes for efficient content distribution,” *IEEE Symposium on Security and Privacy*, 2004.
[21] C. Gkantsidis and P. Rodriguez, “Cooperative security for network coding file distribution,” in *IEEE INFOCOM 2006*, 2006.
[22] Q. Li, D.-M. Chiu, and J. Lui, “On the practical and security issues of batch content distribution via network coding,” in *14th IEEE International Conference on Network Protocols, ICNP ’06.*, 2006.
[23] D. Kamal, D. Charles, K. Jain, and K. Lauter, “Signatures for network coding,” in *40th Annual Conference on Information Sciences and Systems, 2006*, 2006.
[24] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, “An efficient signature-based scheme for securing network coding against pollution attacks,” in *IEEE INFOCOM 2008*, 2008.
[25] E. Kehdi and B. Li, “Null keys: Limiting malicious attacks via null space properties of network coding,” in *IEEE INFOCOM 2009*.
[26] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, “An efficient scheme for securing xor network coding against pollution attacks,” in *IEEE INFOCOM 2009*.
[27] T. Ho, B. Leong, R. Koetter, M. Medard, M. Effros, and D. Karger, “Byzantine modification detection in multicast networks with random network coding,” *IEEE Transactions on Information Theory*, vol. 54, no. 6, pp. 2798 –2803, june 2008.
[28] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, M. Medard, and M. Effros, “Resilient network coding in the presence of byzantine adversaries,” *IEEE Transactions on Information Theory*, vol. 54, no. 6, pp. 2596 –2603, june 2008.
[29] R. Koetter and F. Kschischang, “Coding for errors and erasures in random network coding,” *IEEE Transactions on Information Theory*, vol. 54, no. 8, pp. 3579 –3591, august 2008.
[30] 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.
[31] Y. Li and J. Lui, “Identifying pollution attackers in network-coding enabled wireless mesh networks,” in *20th International Conference on Computer Communications and Networks (ICCCN)*, aug 2011, pp. 1 –6.
[32] ——, “Epidemic attacks in network-coding enabled wireless mesh networks: Detection, identification and evaluation,” *IEEE Transactions on Mobile Computing*, vol. PP, no. 99, p. 1, 2012.
[33] 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.

**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 (IFIP PERFORMANCE 2007). His current research interests include the design and evaluation of interconnected overlay networks and the analysis of compressive sensing and coding techniques in distributed applications.

**Marco 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.

**Riccardo Loti** received his Laurea degree in Computer Science from the University of Torino, Italy, in 2010. He is currently Ph.D. student at the Computer Science Department, University of Torino. His research interests are the evaluation of distributed protocols on MANET and the analysis of interconnected overlay networks.