SIEVE: a distributed, accurate, and robust technique to identify malicious nodes in data dissemination on MANET

Rossano Gaeta, Marco Grangetto, Riccardo Loti

SIEVE: a distributed, accurate, and robust technique to identify malicious nodes in data dissemination on MANET

Rossano Gaeta, Marco Grangetto, Riccardo Loti
Università di Torino, Dipartimento di Informatica
Corso Svizzera 185, 10129 Torino, ITALY
first.last@di.unito.it

***Abstract*—**
In this paper we consider the following problem: nodes in a MANET must disseminate data chunks using rateless codes but some nodes are assumed to be malicious, i.e., before transmitting a coded packet they may modify its payload. Nodes receiving corrupted coded packets are prevented from correctly decoding the original chunk. We propose *SIEVE*, a fully distributed technique to identify malicious nodes. *SIEVE* is based on special messages called *checks* that nodes periodically transmit. A check contains the list of nodes identifiers that provided coded packets of a chunk as well as a flag to signal if the chunk has been corrupted. *SIEVE* operates on top of an otherwise reliable architecture and it is based on the construction of a *factor graph* obtained from the collected checks on which an incremental belief propagation algorithm is run to compute the probability of a node being malicious. Analysis is carried out by detailed simulations using ns-3. We show that *SIEVE* is very accurate and discuss how nodes speed impacts on its accuracy. We also show *SIEVE* robustness under several attack scenarios and deceiving actions.

***Keywords*-MANET, data dissemination, malicious node identification, statistical inference.**

I. Introduction

Due to their open, fully distributed and dynamic architecture, shared wireless channel, and resource limited participants, MANET are vulnerable to attacks at any layer of the Internet model [1].

We consider a particular type of active, non-cryptography related attack: data corruption at the application level introduced by insider nodes. 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 distributed, 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 vertices are *checks*. 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 [2]. 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. 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 distributed and accurate solution based on the BP algorithm. Paper contributions are completed by a comprehensive experimental investigation of *SIEVE* capabilities. It is worth pointing out that the selected data dissemination application is just a quite popular use case [3], [4], whereas the proposed approach is by no means constrained to such particular 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.

SIEVE fits well two key MANET features that must be accounted for when devising any security solution: it is fully distributed 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., [5]) and does not have a high computational cost (as opposed to on-the-fly verification techniques in the area of network coding, e.g., [6], [7], [8], [9], [10]).

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

II. A use case for SIEVE

In this paper we consider a MANET composed of N wireless nodes on 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 $N = N_{fast} + N_{slow} + N_{still}$.

Nodes periodically produce a new data *chunk* to be disseminated to all others once every h seconds; to this end, all nodes cooperate by running a common application that assigns each node an unforgeable unique identifier, e.g., an integer value. When nodes produce or successfully obtain a chunk they turn into data sources for that chunk and help in spreading it. Data is transmitted by source nodes using rateless codes [11]: 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 forward coded packets using rateless LT codes [11], combining random subsets of the K blocks; the size of each coded packet is $S_{cb} = \frac{S}{K}$. According to [11] the original chunk can be obtained by any node able to collect any set of $K \cdot (1 + \epsilon)$ coded packets (ϵ is known as the code overhead). Coded packets are produced only by source nodes for a chunk and consist 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 (RSD) with parameters c and δ [11]. A coded packet conveys the XORed payloads of the corresponding original packets as well as a header signalling the indexes of the combined packets. Decoding is carried out by solving a system of linear equations. In [11] it is shown that the most simple decoder based the simplification of the equations of degree 1 exhibits asymptotically optimal overhead, provided that the RSD distribution is used to encode packets.

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. Similarly, each node can serve as data source for the w_t most recent received chunks. A simple round robin scheduling policy is used when transmitting coded packets of chunks in the upload window; coded packets are transmitted every T_{tx} msec using UDP over an 802.11g wireless communication interface yielding an average transmission range of r meters.

A subset of $P < N$ nodes (where $P = P_{fast} + P_{slow} + P_{still}$) is composed of *malicious nodes*, i.e., they deliberately modify the payload of coded packets they produce before transmission to prevent honest nodes from correctly decoding the original chunk. In presence of coding even a single corrupted coded packet can prevent an honest node from decoding the original chunk.

III. The SIEVE protocol

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 node is able to detect such condition as soon as an inconsistency is found in the solution of the underlying system of linear equations. In particular, according to the procedure [11] the decoder keeps cancelling 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. 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.

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. 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 node. 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 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

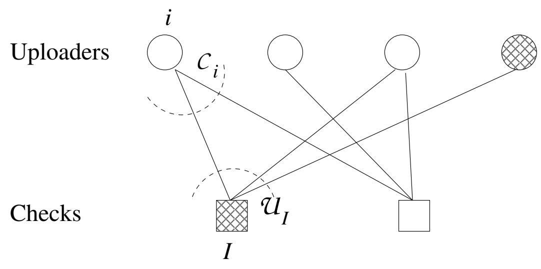

Figure 1: Example of factor graph.

uploader nodes by a witness node. In this paper we adopt the *Belief Propagation* (BP) algorithm [2].

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 messages, the belief, along the edges of the bipartite graph $\mathcal{G}_n$. In case of a Bayesian network BP represents a close form solution for the marginals. The same algorithm can be applied iteratively to a general factor graph that, as opposed to a Bayesian network, includes loops; in this case BP has proven to be a robust estimator for the variable marginals [2].

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 uploaders states have a separable distribution given by the probabilities $\{m^x_{i'I} : i' \in \mathcal{U}_I \setminus i\}$.

The messages are initialized to the values $m^0_{iI} = m^1_{iI} = 0.5$, i.e. all nodes 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. 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 it as shown in Table I for the four possible combinations of c_I and x. The next step is constituted by

Table I: Definition of m^x_{Ii} as a function of x and c_I.

$c_I\setminus x$	0	1
0	$\prod_{i'\in\mathcal{U}_I\setminus i} m^0_{i'I}$	0
1	$1-\prod_{i'\in\mathcal{U}_I\setminus i} m^0_{i'I}$	1

the updating of the probabilities m^x_{iI}, using information from previous computation in the checks: $m^x_{iI} = \prod_{I'\in\mathcal{C}_i\setminus I} m^x_{iI'}$. Finally, the marginal $P(x_i = x)$ can be computed as $P(x_i = x) = \prod_{I'\in\mathcal{C}_i} m^x_{iI'}$. To conclude the BP algorithm initializes the values of m_{iI}, then keeps updates messages from checks to nodes and from nodes to checks. A reliable estimate of $P(x_i = x)$ is computed after a certain number of iterations. In all the experiments reported in this paper 3 iterations have been used.

A. 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 $\mathcal{G}_{nt,w}$ 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 node shows up as an uploader for the first time in a check; then, the partial estimates of m^x_{iI} and m^x_{Ii} are propagated to all the computation windows where such values are needed.

After every BP algorithm ran 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.

IV. Results

In this section we present results on the accuracy and robustness of *SIEVE*. The performance of *SIEVE* have been investigated by simulations using ns-3 version 3.12 on a

Table II: Parameter values for the reference scenario.

N_{fast}, N_{slow}	100	K	100
N_{still}	50	S_{cb}	500 Bytes
P_{fast}, P_{slow}	5	T_{tx}	100 ms
P_{still}	0	c, δ	0.01
r	175 m	l	2000 m
V_{fast}	uniform [10-40] m/s	h	600s
V_{slow}	uniform [1-5] m/s	w_r	50
Mobility model	2D random walk	w_t	6

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 II and III. 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 terminating simulations (not steady-state) runs that lasted for 1 hour of simulated time. We used the independent replication method to obtain $N_{EXP} = 30$ realizations and computed 95% confidence intervals. Each realization is obtained with different streams of the random number 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 data sources only the still nodes; all N nodes contribute to the spreading of the information according to the policy described in Section II. All the system parameter values of the reference scenario considered in the following are summarized in Table II.

For each node n in the system we define the following performance indexes:

- $p(n)$ 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 $\mathcal{R}_n$.

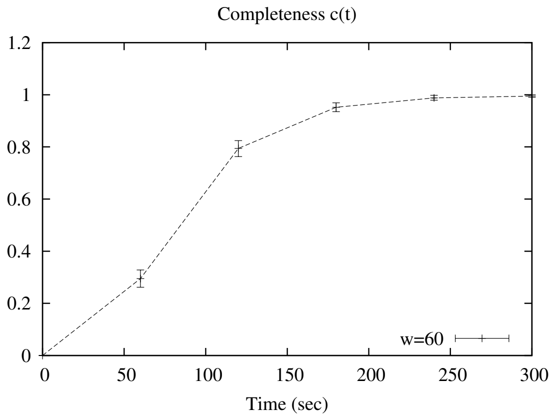

Figure 2: Completeness for $w = 60$s with 95% confidence intervals.

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} realizations and considered $c(t), a(t)$, and $tsi(x)$. In all computations we set $\eta = 0.99$ (η 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.

A. 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 III. Due to space limitations, in the following we omit graphs for $tsi(x)$. 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.

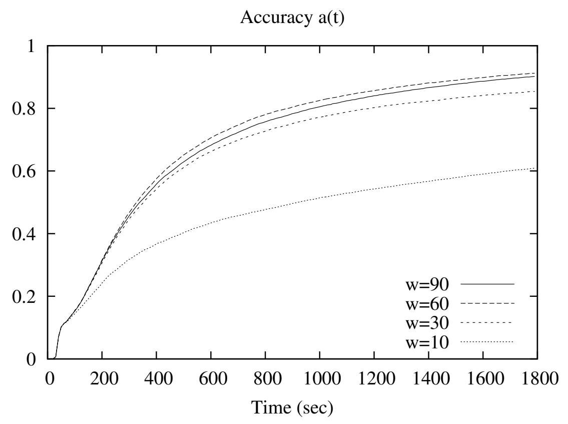

Figure 3: Accuracy as a function of w.

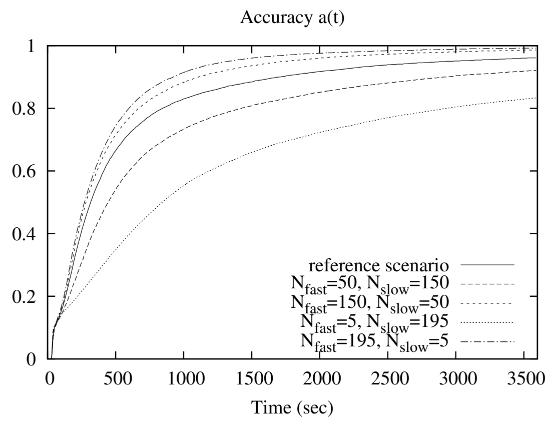

Figure 4: Accuracy for moving nodes ranging from all fast to all slow.

Figure 2 show the completeness $c(t)$ for $w = 60$s when $T = T_s = 10$s, and $Q = 10$ (all values of w yield very similar results so we chose a representative case to avoid cluttering the figures). It can 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. All experiments worked out in the following exhibit a very similar behavior for $c(t)$ so we avoid repeating similar figures. Furthermore, all following graphs compare different scenarios so we plot average results without confidence intervals for better readability.

Figure 3 shows the performance index $a(t)$ with the same settings (all 108 configurations yielded 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. Indeed, small window sizes do not allow the factor graph to include enough checks to accurately infer the node status; on the other hand, large values of w provide more checks in $\mathcal{G}_n$ but increase the number of short loops in the factor graph which in turn impact on the accuracy of the probability estimates yielded by the BP algorithm [12]. As for *SIEVE* reactivity, we observe that $w = 60$s yields the lowest values for $tsi(x)$ (we obtain $tsi(1) = 33s$).

We selected $w = 60$s as the value for all successive evaluation. Performance of *SIEVE* are less dependent on the values of other parameters. Actually, small values for T improve reactivity by decreasing $tsi(x)$ but increase the frequency of BP computation; this may cause concerns for battery operated nodes whose lifetime could be shortened. Battery lifetime can be prolonged by using large values for T_s and small values of Q. Nevertheless, performance of *SIEVE* deteriorates for the same reasons observed for small values of w. 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, $Q = 10$.

B. Mobility and SIEVE performance

The first interesting observations we made is that nodes mobility affects *SIEVE* performance. Figure 4 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 5 shows results in the reference scenario and in scenarios where all malicious nodes are either fast or slow. The best way for malicious nodes to delay identification by honest nodes is to move as slow as possible; indeed, 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 high 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 4) 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 vertices are nodes and an undirected edge between two vertices exist 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

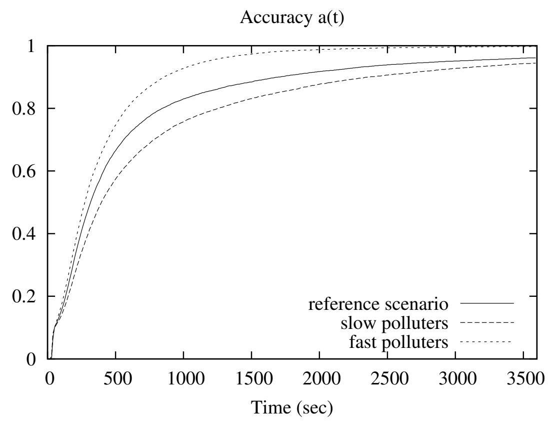

Figure 5: Accuracy for malicious nodes that are either all fast or all slow.

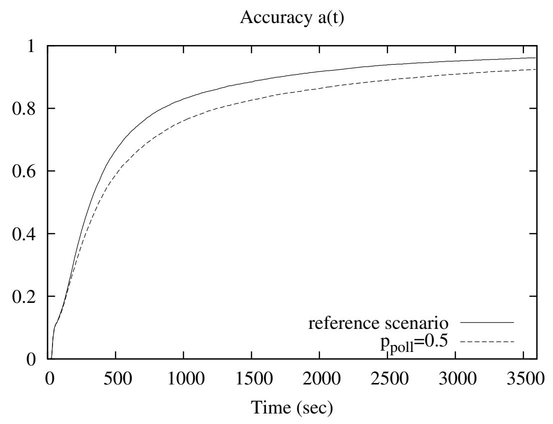

Figure 6: Accuracy for different pollution intensities.

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 loops in the factor graph used by *SIEVE* that is a well known cause of poor performance of the BP algorithm [12]. The shortest length of a loop 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. When the cardinality of the intersection is $x > 2$ then each of the $\binom{x}{2}$ pairs defines a length 4 loop.

When nodes move the geometric graph $\mathcal{O}$ becomes dynamic, i.e., for any node n its $\mathcal{N}(n)$ varies with time. Higher speed translates into higher rate of changing in $\mathcal{N}(n)$ that, in turn, lowers the number of short loops in $\mathcal{G}_n$ increasing accuracy of the BP algorithm. For the reference scenario we computed the average number of checks in $\mathcal{G}_n$ for fast, slow, and still nodes: we obtained 435, 423, and 742, respectively. We also computed the average total number of length 4 loops obtaining 29978, 41574, and 76563, respectively. It can be noted that still nodes must work on a larger $\mathcal{G}_n$ (the average neighborhood size is 7.37, 7.69, and 12.73, respectively so still nodes receive more random selections of Q checks from their neighbors) that results in a much higher number of short loops. Furthermore, the average number of checks for a node (the degree of a node) is higher for still nodes (24.75) with respect to fast and slow nodes (16.49 and 16.56, respectively). Of course, the higher the presence of a node in different checks, the higher the probability of short loops forming in $\mathcal{G}_n$ because of intersection of the set of uploaders.

C. Deceiving actions and SIEVE robustness

Besides following the data dissemination and *SIEVE* protocols, malicious nodes may also implement disturbing actions hoping to avoid or delay identification by honest nodes. Malicious nodes may deceive honest nodes by:

- reducing pollution intensity: a coded packet is modified before transmission only with probability p_{poll} by flipping a coin. Figure 6 shows how *SIEVE* performs when nodes lower their pollution intensity by 50% ($p_{poll} = 0.5$) in the attempt of making their identification harder. It can be noted that *SIEVE* is able to quickly identify all malicious nodes also for $p_{poll} = 0.5$; indeed, malicious nodes that reduce their pollution intensity are less effective in damaging honest nodes and do not succeed in substantially delay identification.
- lying on the status of the check: they flip a coin and with probability p_{lie} a positive check is sent out as negative and viceversa. Figure 7 shows that this trick is almost ineffective.
- disparaging honest nodes: malicious nodes always produce dummy positive checks involving a set of honest nodes. In this case malicious nodes flip a coin and with probability $p_{disparage}$ the actual uploader nodes of a check are replaced by honest nodes; the detection flag is marked as positive. Figure 8 shows *SIEVE* performance when $p_{disparage} = 1$ and 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

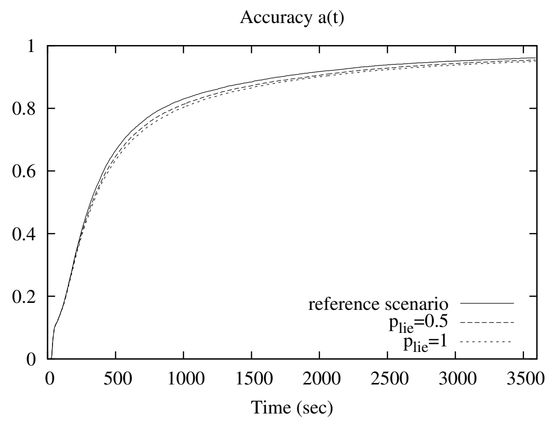

Figure 7: Accuracy for different lying intensities.

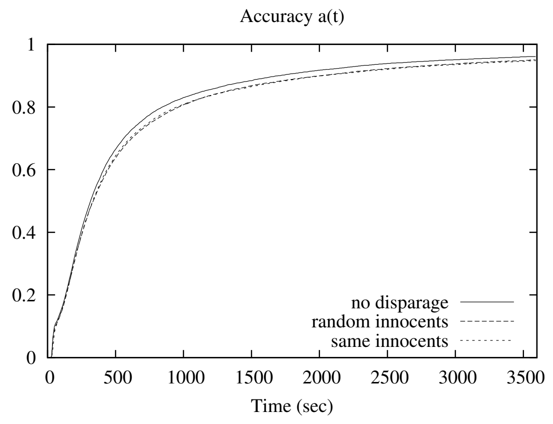

Figure 8: Accuracy for different disparaging attacks.

able to correctly identify all malicious nodes anyway at the end of the simulation experiments.

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 9 we show results for up to $P = 40$ malicious nodes in the system (in all cases, $P_{fast} = P_{slow} = \frac{P}{2}$). It can be noted that *SIEVE* accuracy reaches 0.9 at the end of the simulation experiments.

D. Computational cost

One of the main requirements for MANET 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 to be 25ms. Of course, MANET nodes do not have the computational power of a PC but newer CPUs equipping tablets and smart phones are reducing the gap, e.g., the ARM Cortex-A9 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.

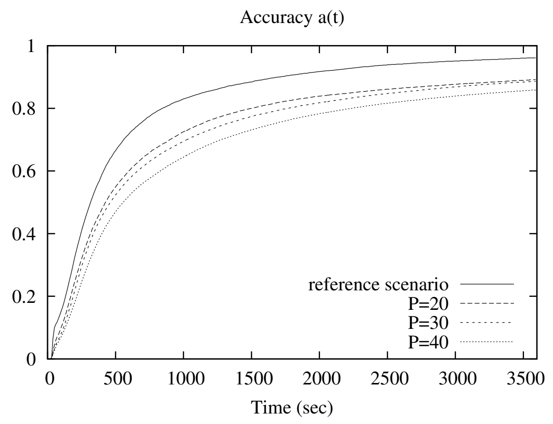

Figure 9: Accuracy for increasing number of colluding malicious nodes.

V. Related works

MANET are vulnerable to attacks at any layer of the Internet model [1]. In network coding several efforts have been devoted to devise on-the-fly verification techniques by participants [6], [7], [8], [9], [10] to identify the sources of corrupted data. Major drawbacks of these elegant methods are the high computational verification costs and communication overhead due to pre-distribution of verification information. Error correction is another approach to deal with data corruption attacks in network coding [13], [14], [15]; these methods introduce redundancy to allow receivers to correct errors but their effectiveness depends on the amount of corrupted information. 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 field are not easily adoptable in MANET. The work by Wang et al [5] proposes a detection scheme where each peer is able to detect receipt of corrupted blocks by checking the decoded chunk to the specific formats of the video stream. Peers detecting polluted chunks send alerts to the video server and the tracker. Upon receipt of an alert the server computes a checksum of the original chunk, used by peers to identify which uploader actually sent a corrupted block, and disseminates it to all peers in the overlay. Peers report their suspects to the server and 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. Lack of a centralized monitoring and management point makes it hard to adapt this solution to MANET.

The work by Li and Lui [16] presents a distributed detection algorithm and analyzes its performance based on simple intersection operations by peers: each starts with a suspects set equal to the entire neighborhood that shrinks 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 with a known number of malicious nodes in the neighborhood and an approximation is proposed when this quantity is unknown, it is attractive due to its simplicity and fully distributed nature. Performance deteriorates when multiple polluter exist and it is not clear if the technique can work when the neighborhood of a peer varies with time.

The work by Jin et al [17] proposed a monitoring reputation system that peers use to select neighbors. The focus of the paper is on reputation computation, storage and load balancing among monitoring nodes. Results show the system is able to detect malicious nodes up to a certain degree of lies. Nevertheless, the technique relies on assumption that each peer is able to compute the amount of corrupted blocks received by each uploader during a monitoring period. This appears to be a rather unrealistic assumption.

VI. Conclusions

In this paper we proposed the novel *SIEVE* technique for identification of malicious nodes performing an active, non-cryptography attack within a MANET. *SIEVE* is a fully 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 analyzed the sensitivity of *SIEVE* performance to the nodes mobility; we discovered that *SIEVE* is very robust to several deceiving actions, colluding attacks by malicious nodes, and amount of malicious nodes inside the network.

Acknowledgment

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

References

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

[2] 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.

[3] J.-S. Park, M. Gerla, D. Lun, Y. Yi, and M. Medard, “Codecast: a network-coding-based ad hoc multicast protocol,” *Wireless Communications, IEEE*, vol. 13, no. 5, pp. 76 –81, october 2006.

[4] D. Nguyen, T. Tran, T. Nguyen, and B. Bose, “Wireless broadcast using network coding,” *IEEE Transactions on Vehicular Technology*, vol. 58, no. 2, pp. 914 –925, feb. 2009.

[5] 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.

[6] 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.

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

[8] Z. Yu, Y. Wei, B. Ramkumar, and Y. Guan, “An efficient signature-based scheme for securing network coding against pollution attacks,” in *INFOCOM 2008, IEEE*, 2008.

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

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

[11] M. Luby, “LT codes,” in *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, IEEE FOCS 2002*, pp. 271–280.

[12] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, “Improved low-density parity-check codes using irregular graphs,” *Information Theory, IEEE Transactions on*, vol. 47, no. 2, pp. 585 –598, feb 2001.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.