Uniwersytet Warszawski
Faculty of Mathematics, Computer Science and Mechanics
Vrije Universiteit Amsterdam
Faculty of Sciences

Anna Chmielowiec
Student id. no.: 209198 (UW), 1637576 (VU)

Node Isolation in Wireless Ad Hoc Networks

Master's Thesis
in COMPUTER SCIENCE
in the field of DISTRIBUTED SYSTEMS

Supervisors:

Maarten van Steen and **Daniela Gavidia**
Dept. of Computer Science,
Vrije Universiteit Amsterdam

and

Janina Mincer-Daszkiewicz
Institute of Informatics,
Uniwersytet Warszawski

August 2007

Abstract

In ad hoc networks nodes rely on forwarding packets for each other. Yet, in large heterogeneous networks the willingness of all nodes to decently cooperate towards a common goal of effective communication cannot be taken for granted. Malicious nodes can compromise the integrity of the messages they forward by replacing their contents with unsolicited data (hereinafter referred to as spam) and without any countermeasures such malpractices may disrupt the performance of some of the nodes or even the whole network.

This thesis presents a mechanism for detecting and isolating malicious nodes from the network. The decision whether to cease communicating with another node is made individually by every node, thus, no exchange of reputation information is necessary.

Firstly, all nodes continuously perform probabilistic verification of the data they relay, removing the spam they find and marking the verified messages that proved to be undistorted with the 'checked' flag. The algorithm is designed so that the frequency of checks adapts itself to the intensity of the spam received.

Secondly, every node gathers during the sampling of its traffic information on the estimated fractions of spam and 'checked' messages received from every neighbour. It appears that these two values should be close to each other if only a node adheres to the protocol. Thereby, if a node observes discrepancies between these two values, the node will regard the corresponding neighbour as a spammer and refuse to communicate with it. The divergence between the values that raises the suspicion and the number of data exchanges needed before a node can make a decision on neighbour isolation were set experimentally in order that all visibly malicious nodes are detected and isolated, yet the number of false accusations is minimized.

Keywords

ad hoc networks, epidemic protocols, gossiping protocols, probabilistic verification, security, spam, wireless networks

Thesis domain (Socrates-Erasmus subject area codes)

11.3 Informatics, Computer Science

Subject classification

C. Computer Systems Organization
C.2. Computer - Communication Networks
C.2.2. Network Protocols
C.2.4. Distributed Systems

Contents

List of Figures

Chapter 1

Introduction

Among many characteristics of wireless ad hoc networks such as dynamic nature and often unreliable links the following two are especially interesting: the absence of trusted infrastructure for routing and the lack of a central authority. Due to the first property, all networking functions must be performed by the nodes themselves. Thus, if a node wants to send a packet, it has to rely on the willingness of other nodes in the network to forward it. The lack of a central authority makes it difficult to guarantee or execute this willingness from the nodes and results in wireless networks being a tempting environment for selfish or malicious behaviours.

I have focused on networks that use gossiping as a communication protocol and on sending *spam* as a way of misbehaving. Because excessive production of messages can be easily prevented by restricting the id space of messages per node, malicious node would have to make use of the id space belonging to other nodes in the network and compromise the content of their messages if it wants to place more spam in the network. Another motivation behind corrupting messages created by other nodes instead of publishing spam under own identity is the desire to remain anonymous. Therefore, hereinafter the term *spam* refers to any message whose content has been compromised.

The conventional approach to deal with corrupted messages is to require every message to be signed by its creator. Then, the receiver can check the integrity of the message by verifying its digital signature. However, this solution may not be appropriate in all scenarios. Firstly, it makes impossible to detect and isolate a spammer, because the message might have been corrupted by any of the intermediate nodes that were engaged in its forwarding. Secondly, sometimes a message does not have a particular receiver, as it happens when gossiping is used for dissemination of information.

Other solution is to check the integrity of a message at every hop. This means that each node in the network has to verify every message it relays. The advantages are that corrupted messages are removed immediately from the network and malicious nodes can be easily detected. But this solution may be computationally too expensive.

1.1. Contribution of the Thesis

I focused on the isolation of malicious nodes that corrupt relayed messages. The assumed design constraints constitute prohibitive costs of verifying every forwarded message and lack of any reputation information exchanged among the nodes. Under these assumptions the goal was to minimize the probability of accidental isolation of good nodes while isolating as many malicious ones as possible. The thesis presents a step by step path of the isolation mechanism.

The thesis can be regarded as a continuation and further development of the work by Gavidia et al. (see [9] and [10]) on the subject of enforcing data integrity in ad hoc networks. In her publications, Gavidia presented mechanisms to prevent the spread of the spam and to detect malicious nodes in the network. The subject on spammer isolation discussed in this thesis is taking this research one step further.

1.2. Related Work

Most of previous work addressing the security problems in wireless ad hoc networks focuses on securing the routing layer. In great part this stems from the fact that the majority of protocols that have been proposed for wireless ad hoc networks (AODV [19], DSDV [20], DSR [15], TORA [17], etc.) lack mechanisms to enforce cooperation and non-selfish behaviour. As assuming non-hostile environments is utopian, the researchers have developed secure routing protocols that mostly are modifications of already existing protocols. Several of these protocols are SAODV [24], SecAODV [18], SEAD [13], Ariadne [14], SRP [16], BSAR [2], SBRP [23].

Apart from the above protocols several solutions have been proposed for discouraging selfish behaviour and punishing malicious nodes that operate on the forwarding layer. We can distinguish two main approaches: *Payment Systems* and *Reputation-based Systems*. The latter can be additionally divided into systems where nodes exchange reputation information among themselves (*Cooperative Reputation-based Systems*) and systems where each node evaluates its neighbours reputation basing only on its individual observations (*Autonomous Reputation-based Systems*).

1.2.1. Payment Systems

In payment schemes, the use of tamper-proof hardware or outside infrastructure is required for securing the accounting. Both of these solutions may not be realizable in actual ad hoc network scenarios.

In [7] (the continuation of [6]) Buttyan and Hubaux explore the use of the former solution. Each node is equipped with a tamper resistant hardware module — *security module*. This module maintains a counter (referred to as *nuglet counter*). The counter is increased by one every time the node forwards a packet and decreased by the number of intermediate nodes when the node wants to send its own packet. As the value of the nuglet counter must remain non-negative, nodes are motivated to forward packets for other nodes.

In Sprite [25] the use of tamper resistant hardware is substituted by an external infrastructure in the form of a *Credit Clearance Service* (CCS). When a node sends its own messages, it loses some fraction of *credit*. In order to receive more credit, every node would forward messages of other nodes and store the receipts of these messages. The receipts are reported to the CCS, which is responsible for computing the value of gained credit.

1.2.2. Reputation-based Systems

When neither the use of tamper-proof hardware nor outside infrastructure is available, a reputation-based scheme may be appropriate.

Cooperative Reputation-based Systems

In cooperative reputation-based systems nodes not only evaluate the behaviour of their neighbours but also share this information with other nodes.

The CONFIDANT system [3] consists of four modules that are deployed at every node. A *monitor* is responsible for detecting and registering deviations from normal behaviour. When malicious action is observed a *trust manager* sends out a so-called *ALARM message*, to inform other nodes of this misbehaviour. A *reputation system* manages a table of node ratings that is updated on the basis of first-hand and trusted second-hand (ALARM messages) observations. A *path manager* uses this table to adapt a node's routing behaviour. The biggest problem in such system is the threat of the false accusations. It was mitigated in the following works [4], [5] by the use of Bayesian statistics.

In SORI [12] the propagation of reputation information is limited only to the direct neighbours. This reduces the communication overhead and lessens the problem of false accusations. In more detail, a node evaluates the reputation of its neighbour by weighting the information received from all other neighbours and its own observations. As weights used for calculations depend on the *credibility* of each node, the accusations of a suspected node would have low impact on the evaluation.

Autonomous Reputation-based Systems

In autonomous reputation-based systems no reputation information is spread, a node has to evaluate a neighbour's behaviour and make an eventual decision about punishment based solely on its own observations.

Every node in OCEAN [1] maintains ratings for each of its neighbours in a *RouteRanker* component. The rating is incremented (decremented) on observing positive (negative) action of the neighbour. Based on these ratings a node decides which route to choose for forwarding a packet. A route is assumed to be good if the next-hop node is non-*faulty* (has enough high rating). Besides route selection, ratings are also used for rejection of malicious traffic; all traffic from a misleading node is rejected.

The mechanism proposed in [21] lets a node autonomously evaluate the reputation of each of its neighbours, not on the basis of observing a neighbour just forwarding the packet, but on the completion of the requested service. In general, when a node forwards a packet to one of its neighbours, this neighbour (from the point of view of the node) is held responsible for the delivery of the packet to the destination.

Adopting the above classification, the isolation mechanism presented in this thesis lies close to the third group — Autonomous Reputation-based Systems — as the decision about punishing a node relies only on the statistical analysis of incoming traffic. But unlike the systems presented above, which focus mainly on the malicious behaviour of dropping packets, our mechanism is designed to fight the nodes that corrupt the packets they forward. From this perspective, our work is more related to the Quarantine Regions [8] for wireless sensor networks or fighting spam in peer-to-peer networks [22].

1.3. Overview

The remainder of this thesis is organized as follows. In the next chapter, the system model for the data integrity enforcement and malicious node isolation mechanism is described and the platform used for testing and basic modelling of the malicious behaviour are presented.

Chapter 3 is focused around the probabilistic verification of the data that is relayed by the nodes in the network. Brief descriptions of previous models are presented in Sections 3.1 and 3.2, and in Section 3.3 the initial probabilistic verification scheme developed by the author is explained. Chapter 3 is completed with remarks on the effects of spammer activity in the network depending on the probabilistic verification scheme used. Chapter 4 is solely devoted to detecting and isolating malicious nodes in the network. The results of extensive simulations are presented that highlight not only the efficiency in malicious nodes isolation (minimising the number of false negatives), but also avoidance of false accusations (namely false positives). In Chapter 5 possible attempts of malicious nodes to avoid isolation are analysed. The conclusions are presented in Chapter 6 along with the future work possibilities.

Chapter 2

System Model

2.1. General Description

This thesis is focused on store-and-forward systems with wireless communication medium. Every node in such system devotes a limited amount of space to store messages, which are disseminated through the network in a multi-hop manner. We assume that nodes forward a batch of messages at a time.

The Gossip-based News Service, as introduced in [11], can be an example of the system presented above and will serve as the experimental platform for evaluation of the data integrity enforcement and node isolation measures described in this thesis. The service is provided by a mesh backbone formed of a large number of wireless routers communicating through gossiping (the protocol used for gossiping is described in detail in Section 2.3). Owners of the routers are eligible to publish events, called *news items*, which are gossiped through the mesh backbone in the form of *news entries*. While a news item is a piece of information, a news entry is the representation of the news item in the network and for each news item several news entries may exist. The users are then able to retrieve the news matching their interests by connecting to the mesh backbone by means of portable mobile devices (such as smart phones, laptops, PDAs). The relevance of the news items received by the users is ensured by providing a nearby router with the user's preferences.

2.2. Assumptions

Each node is assigned a unique id. Thus, every entry in the network can be uniquely identified by a combination of the publisher's id and a sequence number. A limited number of these entries is stored by each node in its local cache. In experiments conducted in this thesis, all nodes have the same cache size c and hence, every node can store at most c entries. Nodes gossip periodically swapping some of their entries with their neighbours. An interval in which each node initiates an exchange once is referred to as *round*.

Items can be published by any node and are propagated through the network in the form of entries (just like news items and news entries in Gossip-based News Service). Replication may occur naturally if a node has available storage space to keep a copy of received entry, resulting in many entries for the same item being present in the network.

Furthermore, every item has to be digitally signed by the node which originally has published it, so every entry can be a subject to integrity check. As execution of a public key signature verification may require considerable computational workload, checking all entries received during a gossip exchange is assumed to be prohibitively expensive.

2.3. Shuffle Protocol

The data exchange between nodes follows a predefined structure, presented in Figure 2.1. Each node initiates an exchange once every round undertaking the role of an active party. The node that is contacted assumes the passive role.

```
/*** Active thread ***/                   /*** Passive thread ***/
// Runs periodically every T time units   // Runs when contacted by another node

Q = selectPeer()                          receive buff_recv from any P
buff_send = selectItemsToSend()           /* Place for the isolation mechanism */
send buff_send to Q                       buff_send = selectItemsToSend()
receive buff_recv from Q                  send buff_send to P
cache = selectItemsToKeep()               cache = selectItemsToKeep()
```

Figure 2.1: Skeleton of the shuffle protocol.

The core of the protocol is represented by three methods: `selectPeer()`, `selectItemsToSend()` and `seletItemsToKeep()`. By implementing different policies in these methods, various epidemic protocols, each with its own distinctive characteristics, could be instantiated. Yet, in this thesis the shuffle protocol is used (described in [11]), in which each node agrees to keep the entries received from a neighbour for the next round. Given the limited storage space available in each node, keeping the entries received during an exchange implies discarding some entries that the node has in its cache. By picking the entries to be removed from the ones that have been sent to the neighbour the conservation of the data in the network is ensured.

Furthermore, the basic shuffle protocol is enriched with the added measures to enforce data integrity and to isolate maliciously behaving nodes. First of them — **probabilistic verification** — has already been presented in papers [10] and [9] and in this thesis an author's modification of probabilistic verification is presented and evaluated in comparison to previous solutions. **Isolation mechanism** introduced in this paper is a lightweight mechanism that uses information gathered in probabilistic verification phase and by the means of which a node can decide to refuse to communicate with obviously malicious nodes. In the passive thread the isolation mechanism is implemented just after receiving `buff_recv` from P (see the comment in Figure 2.1). P is then checked if it should be isolated and when the answer is positive then all entries in `buff_recv` are dropped and an empty `buff_send` is sent to P. In the active thread, isolation mechanism is enclosed in `selectPeer()` method.

The policies for the main shuffle protocol's methods are summarized as follows:

- `selectPeer()` randomly selects a neighbour for gossip exchange. If the **isolation mechanism** is switched on then only a node that has not been discovered as being malicious is chosen,

- `selectItemsToSend()` randomly selects s entries from the local cache and sends their copies (`buff_send`) to the selected peer,

- `selectItemsToKeep()` performs **probabilistic verification** of the data integrity and then adds the received entries from `buff_recv` to the local cache removing repeated entries. If the number of entries exceeds the size c of the node's cache, entries among the ones that were previously sent are removed (unless they are also in `buff_recv`) to make the room for the new ones.

2.4. Basic Attack Model

Wireless ad hoc networks that use gossiping for data dissemination constitute a perfect environment for the activity of spammers. Knowing only one node in the network is enough for a malicious node to start operating. Moreover, as all peers collaborate to propagate messages, the effort that a spammer has to make in order to spread its spam resolves itself simply into inserting messages into the network and all other nodes will make sure that the messages are delivered.

This vulnerability of gossip network could easily lead to flooding the whole network with the items produced excessively by a small fraction of nodes, which can be considered a form of a distributed denial-of-service attack.

Fortunately, excessive production of items can be easily prevented by restricting the id space of items per node. For instance, the id space of items per node could be set to n bits resulting in 2^n items. In such situation, after a node has published 2^n items with different ids, the next item will have to reuse the id of the one of previously published items. Since nodes are allowed to hold at most one entry per item (based on item's id) entries of the more recently published item will overwrite entries of the older item in the network. Taking into consideration the specific characteristics of the shuffle protocol that ensure that each item has on average the same number of entries in the network, a node could only occupy limited fraction of collective storage space.

Hence, in order to place more spam in the network, a malicious node would have to make use of the items id space belonging to other nodes. In essence, a spammer would be replacing the content of the other nodes' entries with its own while keeping the metadata (id, signatures,. . .) of the entries unchanged.

By this means, the malicious node steals the storage space of other nodes and is able to flood the network with its messages afresh. Especially, when executing an integrity check for every message received is assumed to be prohibitively expensive for a node, tracking down and isolating a spammer becomes nontrivial as a compromised entry might have been shuffled around several times before its integrity has been verified.

With the aim of testing the effectiveness of proposed methods of probabilistic verification and node isolation a relatively small number of nodes in the network is assumed to be malicious. These nodes, also referred to as *spammers*, execute a slightly different version of the shuffle protocol. Their basic attack model is to corrupt entries before forwarding them to their neighbours. In the simplest case, a spammer would corrupt outgoing entries with a constant probability P_{spam} (also called *spamming rate*) while executing `selectItemsToSend()`. Moreover, to save the computational resources, a malicious node does not perform any probabilistic verification of relayed traffic.

Although it appears that the higher the P_{spam} is the more effective attack becomes, the results presented in Chapters 4 and 5 prove that high spamming rates are actually counter productive and lead to faster detection and isolation of the malicious node. Furthermore, in Chapter 5 we illustrate through simulation results that not only should a malicious node lower (and maybe differentiate) the percentage of spam inserted into the network in order to avoid isolation but it also have to adhere to the rules of probabilistic verification. What is more, it appears that spammers have to check the entries they receive more frequently than an ordinary well-behaving node in order not to fall under suspicion because of relaying spam of other nodes and to compensate for the spam they place in the network.

2.5. Simulation Setup

In the experiments, whose results are presented in this thesis, nodes are arranged in a square grid topology, with 50 nodes on each side over an area of 50×50 units. The range of each of 2500 nodes is set to 1 unit, making communication possible with the node's adjacent neighbours to the North, South, West and East. Every node is provided with a cache size of $c = 100$ and during each gossip exchange swaps $s = 50$ of stored entries. From among 2500 nodes present in the network 250 malicious nodes are selected randomly at the beginning of each experiment. Up till the 50th round of an experiment spammers behave as every well-behaving node. After the 50th round they start their malicious activity by corrupting outgoing entries with a constant probability as described in Section 2.4 unless specified otherwise.

Chapter 3

Probabilistic Verification

Checking all messages at every hop has its undeniable advantages. Spam is discovered immediately after it has been created and thus has no opportunity to spread over the network. At the same time malicious nodes are detected and can be isolated from the network the moment they start operating. Unfortunately, in a system where the entries are constantly being gossiped, verifying all entries that node receives would be computationally too expensive which makes this approach unpractical.

As an alternative solution to checking all entries at every hop, each node could just check some randomly selected fraction of the entries it receives. In more detail, the scheme for this kind of data integrity checking can be incorporated into the shuffle protocol in the following way:

- In `selectItemsToKeep()`, before a node merges received entries with the entries in its cache it executes probabilistic verification of the received entries.
- Each of the received entries is checked with a probability P_{check}. If a selected entry passes the digital signature verification, it is marked as `checked`. Otherwise, the entry is spam and is discarded.
- Lastly, entries that were not selected for checking and the ones that received a `checked` mark are merged into the local cache.

Depending on the way the value of P_{check} is handled (whether it is fixed and constant for all the nodes in the network or changes in time individually for every connection depending on the quantity of spam received from the neighbour on the other side of this connection) the probabilistic verification may have different characteristics. But regardless whether at all or how P_{check} is modified, it would be recommended that some lower bound for P_{check} (greater than 0) is set as every node should be on the guard in case any spam appears. This necessity for some $P_{check_{min}}$ derives directly from the properties of wireless ad hoc networks, where nodes may join or leave the network at any given time (intentionally or due to failure or falling out of reach of its neighbours). This results in wireless environments being generally very unstable. A node that joins the network has no knowledge of its environment besides the identity of its immediate neighbours. It has no preconceptions about its neighbours nor about the spam traffic level it will encounter. Thus, to be on the safe side, the node should apply at least the minimal level of checking at the beginning by setting its P_{check} to $P_{check_{min}}$. Since the behaviour of neighbours may change over time, the value of P_{check} should never drop below that lower bound. The implication of this is that there is a minimum workload imposed on the network but at the same time we have a probabilistic guarantee that every spam entry would be detected and removed from the network. We just have to realise that the probability of an entry not being checked in n hops is:

$$
\begin{aligned}
\mathbb{P}(\text{entry not checked in n hops}) &= \prod_{i=1}^{n}(1 - P_{check,i}) \\
\text{where } P_{check,i} \text{ was used for probabilistic verification at hop } i & \\
&\leq \prod_{i=1}^{n}(1 - P_{check_{min}}) \\
&= (1 - P_{check_{min}})^{n} .
\end{aligned}
$$

Thus, the probability of corrupted entry traversing further drops with every hop and the further from a spam source (in number of hops) a node in the network is placed, the lower the probability that it receives a corrupted entry.

In all our experiments $P_{check_{min}}$ is set to 0.05.

3.1. Constant Probability Scheme

The simplest scenario for probabilistic verification is to keep P_{check} fixed in the entire network. This solution was evaluated in [10]. The results presented in that paper show that deploying even the simplest probabilistic verification into the gossip protocol is enough to stop the number of corrupted entries from increasing until all entries in the network are corrupted. Checking entries with constant probability P_{check} will both reduce the overall percentage of spam in the network and decrease its expansion. Moreover, the effectiveness of probabilistic verification in fighting spam depends directly on the value of P_{check}.

Firstly, the value of P_{check} is indirectly proportional to the value to which the amount of spam in the network converges. Secondly, entries are restricted from spreading too far from the source. With every hop it becomes more likely that the corrupted entry will be removed by a non-malicious node. It is not surprising that the average distance (in hops) from the source that spam travels is also reciprocally proportional to the value of P_{check}.

The experimental results described in [10] showing that higher values of P_{check} reduce both the amount of spam in the network and the area affected by corrupted entries derive from the properties of the probabilistic verification.

To start with we note that the corrupted entry can be created only in two ways:

- by the spammer that compromises the content of the original entry,
- by the replication during a gossip exchange.

We will also be taking no account of the fact that the corrupted entry may be relayed by other spammers while being gossiped (we omit in our computations hops where a spammer is a receiver). This omission has no effect on the correctness of the computations below.

If we now denote by the random variable X the number of hops that the corrupted entry travels until it is checked and discarded and provided P_{check} is fixed for all nodes in the network then X has a geometric distribution with parameter P_{check} and

$$\mathbb{P}(X = i) = P_{check} \cdot (1 - P_{check})^{i-1}$$

where $i \in 1, 2, \ldots$, and

$$\begin{aligned}\mathbb{E}(X) &= \sum_{i=1}^{\infty} i \cdot P_{check} \cdot (1 - P_{ckeck})^{i-1} \\ &= \frac{1}{P_{check}}.\end{aligned}$$

From the formula for $\mathbb{E}(X)$ we can safely infer that the higher the value of P_{check} is, the shorter the time the corrupted entry exists in the network and the shorter the distance it can travel through the network. In addition, the probability of the entry being replicated drops as well along with the increase of P_{check} (and decrease of $\mathbb{E}(X)$).

Unfortunately, this simply solution of P_{check} being a fixed network-wide parameter has a huge drawback of not being flexible. Regardless of the size of the threat faced by a node from its neighbours (being flooded with spam or not) constant P_{check} requires a fixed amount of work to be performed.

3.2. Ageing Probability Scheme

The conclusions from [10] led to a more advanced way of handling P_{check} that has been presented in [9]. In that paper, nodes are given the autonomy to adjust the P_{check} values to their needs, namely to the intensity of spam they receive from their neighbours. The motivation behind giving the nodes freedom to control the level of probabilistic integrity checks is to transform the network into a self-policing system where the high security measures are engaged where needed while at the same time nodes in areas where spam is rare keep their work to a minimum. A node cannot completely stop performing integrity checks, because it should always keep a watchful eye in case situation in the network changes.

In more detail, every node maintains a different P_{check} for each of its neighbours. When a node engages in a gossip exchange with one of its neighbours the scenario of updating P_{check} proceeds as follows:

- executing probabilistic verification
  The node checks each of the received entries with the probability of the current value of P_{check} (P_{check_t}). If we assume that the neighbour selects entries to send randomly from its cache, then the fraction of corrupted entries received by the node in the current exchange should reflect the fraction of spam in the neighbour's cache. Consequently, the entries chosen for verification should give the node an estimated level of corrupted entries at the neighbour's side. We denote by P' the ratio of the number of the entries that have not passed the integrity check (*numRemoved*) to the number of checked entries in the probabilistic verification (*numChecked*). Thus,

$$P' = \frac{numRemoved}{numChecked}.$$

- updating the checking probability P_{check}
  The value of P_{check} for the next gossip exchange with this neighbour is updated as a weighted sum of the current value of P_{check} and the value of P':

$$P_{check_{t+1}} = (1 - \alpha) P_{check_t} + \alpha P' \tag{3.1}$$

  where parameter $\alpha \in [0, 1]$ determines the sensitivity of P_{check} to the changes in the value of P'.

In this probabilistic verification scheme when a node joins the network it sets P_{check}'s for all it neighbours to $P_{check_{min}}$. Thus, when it starts gossiping it would probably need some time to adjust its filters (P_{check}'s) to the right values. Ideally, if the node always observed P' equal to the percentage of spam actually sent by the neighbour i (denoted by P_i), the value of a $P_{check}[i]$ would finally converge to the level of spam received (on the grounds of equation 3.1). Consequently, if $P_{check}[i] \approx P_i$, the fraction of spam received from i and detected by the node would on average be equal to P_i^2 and the fraction of spam that goes undetected into node's cache would be equal on average $P_i(1 - P_i)$. This simple observation already suggests that for a spammer spamming with a P_{spam} higher then 0.5 would be unprofitable because more than half of the spam would get discarded at the very first hop (compare to Figure 3.1).

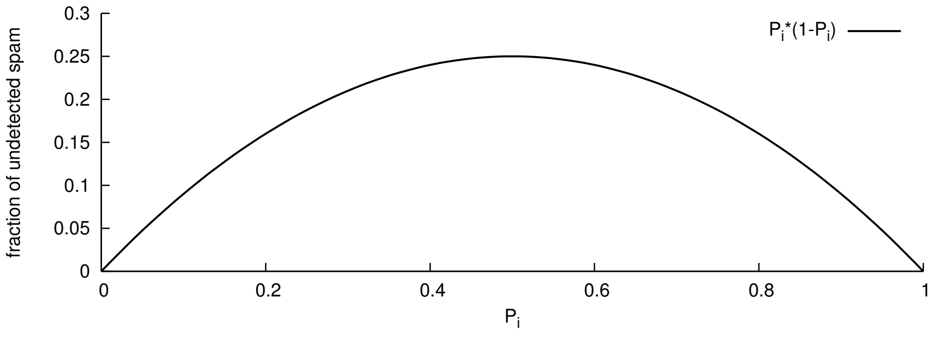


Figure 3.1: Fraction of spam received from a neighbour that is not detected by the node (P_i — the probability that a received entry is corrupted and $P_i \approx P_{check}[i]$).

3.3. Probability based on Conditional Expectation

In the course of the characteristics analysis of equation 3.1, the following observation came into focus. The difference between $P_{check_{t+1}}$ and P_{check_t} is directly proportional to the difference between P' and P_{check_t}, which can be regarded as an advantage — it causes that the more significantly the current value of P_{check} (which is assumed to estimate the intensity with which a neighbour sends out corrupted entries) deviates from the amount of spam received in the latest gossip exchange, the more substantially the updated value of P_{check} will be modified to reduce this divergence. If the value of P_{check_t} is low and on the contrary the value of P' is considerable one can consider the above as a fast alarm rise. Unfortunately, at the same time such a rapid increase of the P_{check} value may be overhasty behaviour and the related growth of the sampling frequency can only result in higher computational resources consumption by the node.

It is important to investigate under what circumstances it is justifiable for a node to take dramatic precautions against its neighbour and raise the value of P_{check} and when this kind of behaviour may be premature. Let us compare the following two situations that the node executing probabilistic verification may encounter. In the first one, the node is checking the received entries with the probability $P_{check} = 0.05$. If the number of entries the node has received is 50 then on average the node will check only a few messages (mean value of the number of verified entries equals exactly 2.5). In the second case, $P_{check} = 0.50$ and the mean value of the number of checked entries equals 25 out of 50. If we now assume that in both cases all the checked entries turned out to be spam then in both cases to update P_{check} the value of $P' = 1.0$ will be taken as the observed spamming rate of the current message exchange. Supposing $\alpha = 0.1$ the new values of P_{check} would equal accordingly 0.145 (growth by 0.095) and 0.55 (growth by just 0.05). But intuition says that in the first situation it could be just by chance that the node has observed this high P' and the real fraction of spam sent by the neighbour could be much smaller. The contrary situation has place in the second case, where the value P' seems to reflect the actual fraction of spam received in the last gossip exchange more reliably.

The above observation led to a new scheme of updating P_{check}. To obtain the enhanced equation the information provided by the probabilistic verification mechanism itself was used but to explain how the equation was derived we have to look at the gossip exchange from the mathematical perspective.

We can look at the N entries received by a node from its neighbour as a probabilistic space of N statistically independent Bernoulli trials, where an entry being spam denotes success and uncorrupted entry denotes failure. The node does not know the real probability with which an entry is compromised, it can only estimate it by the value of P_{check}. (Note that the node does not know the real distribution of spam among the entries it receives from the spammer-neighbour in the course of gossip exchanges and the P_{check} value may be discordant from the real rate at which the spammer sends out corrupted entries so this is just a simplistic model).

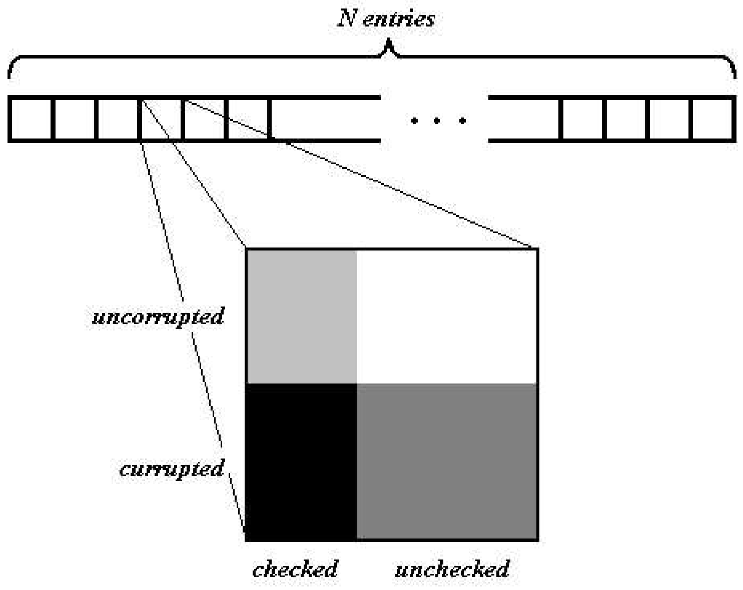


Figure 3.2: Probabilistic model of a gossip exchange.

While the node executes the probabilistic verification of the received entries it checks every entry with the probability P_{check}. Thus, there are another N Bernoulli trials (statistically independent from previous Bernoulli trials) where checking an entry denotes a success.

We introduce now random variable X_i that represents the number of corrupted entries among i received ones. Suppose now that:

- N is the number of received entries,
- M is the number of checked entries ($0 \leq M \leq N$ and $M/N \approx P_{check}$),

- S is the number of checked entries that turned out to be spam ($0 \leq S \leq M$ and $S/M = P'$).

Then, the conditional expected value of the number of the spam entries among all N entries provided that among M entries we found S spam entries can be estimated by:

$$\mathbb{E}(X_N | X_M = S) = S + P_{check} \cdot (N - M) \; . \tag{3.2}$$

$$\begin{aligned}
\mathbb{E}(X_N|X_M = S) \;&= \\
&\text{from the definition of conditional expected value} \\
&= \sum_{i=0}^{N} i \cdot \frac{\mathbb{P}(X_M = S \wedge X_N = i)}{\mathbb{P}(X_M = S)} \\
&\text{from statistical independence of the Bernoulli trials} \\
&= \sum_{i=0}^{N} i \cdot \frac{\mathbb{P}(X_M = S) \cdot \mathbb{P}(X_{N-M} = i - S)}{\mathbb{P}(X_M = S)} \\
&= \sum_{i=0}^{N} i \cdot \mathbb{P}(X_{N-M} = i - S) \\
&\text{replacing variables — } j = i - S \\
&= \sum_{j=-S}^{N-S} (j + S) \cdot \mathbb{P}(X_{N-M} = j) \\
&\text{changing summing ranges, removing components of 0 value} \\
&= \sum_{j=0}^{N-M} (j + S) \cdot \mathbb{P}(X_{N-M} = j) \\
&\text{Bernoulli distribution of spam entries among all entries} \\
&= (N - M) \cdot \mathbb{P}(X_1 = 1) + S \\
&\text{as said above, we do not know the exact probability of the entry being a spam, so we will use } P_{check} \text{ as the estimation of it} \\
&= (N - M) \cdot P_{check} + S \; .
\end{aligned}$$

Hence, if we accept that in the last gossip exchange the node has received $(N-M)P_{check} + S$ spam entries then the spamming rate of the neighbour can be assumed to be close to $((N - M)P_{check} + S)/N$. This value can be taken as an updated value of P_{check}:

$$P_{check_{t+1}} = \frac{(N - M)P_{check_t} + S}{N} \; . \tag{3.3}$$

Although we have all information needed to use the equation $P_{check_{t+1}} = ((N-M)P_{check} + S)/N$ (S, M and N are the products of probabilistic verification), we will transform the equation to rely solely on P' and P_{check_t}. To achieve this, we notice that $P' = S/M$ and that $P_{check_t} \approx M/N$. Consequently,

$$S = M \cdot P' \approx N \cdot P_{check_t} \cdot P'$$

and

$$N - M \approx N - N \cdot P_{check_t} = N(1 - P_{check_t}) \; .$$

After inserting the above values into equation 3.3 we obtain:

$$\begin{aligned} P_{check_{t+1}} &\approx \frac{N(1-P_{check_t}) \cdot P_{check_t} + N \cdot P_{check_t} \cdot P'}{N} \\ &= (1-P_{check_t}) \cdot P_{check_t} + P_{check_t} \cdot P' \\ &= P_{check_t}(1+P'-P_{check_t}) \end{aligned}$$

Thus, our new equation for updating P_{check} would be:

$$P_{check_{t+1}} = P_{check_t}(1+P'-P_{check_t})\,. \tag{3.4}$$

The 3-dimensional graph of the $P_{check_{t+1}}$ function is presented in Figure 3.3.

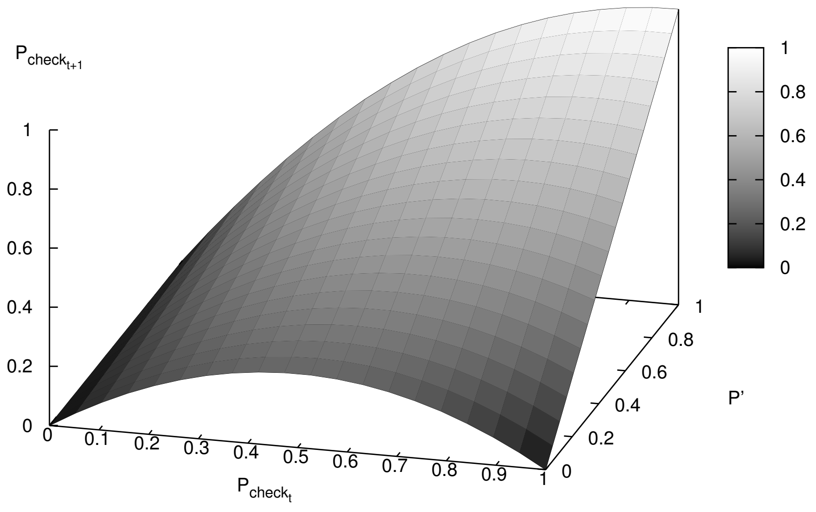


Figure 3.3: 3-dimensional graph of the new $P_{check_{t+1}}$ update function.

The last remark that should be made about the above equation is whether the P_{check} converges to the P' value provided P' is constant. We can easily show that if $P_{check_t} < P'$ then $P_{check_t} < P_{check_{t+1}} < P'$ and similarly if $P' < P_{check_t}$ then $P' < P_{check_{t+1}} < P_{check_t}$. The pace of the convergence is dependent both on the value of P_{check_t} as well as on the value of $(P' - P_{check_t})$ which results directly from transformed equation 3.4 ($P_{check_{t+1}} - P_{check_t} = P_{check_t}(P' - P_{check_t})$).

3.4. Probabilistic Verification vs. Workload Caused by Spammers

Workload that the nodes in the network face while using probabilistic verification with a fixed P_{check} does not depend on the size of the threat imposed by spammers. Only the value of P_{check} is responsible for the additional amount of work that well-behaved nodes must perform while gossiping. Yet, the situation looks completely different when the nodes deploy probabilistic verification that allows P_{check} adjustments (like the P_{check} updates schemes described in Sections 3.1 and 3.3).

As long as there is no spam in the network, nodes keep their P_{check} value to a minimum (namely $P_{check_{min}}$) so the costs of probabilistic verification computations are low. When spammers appear, nodes in the network, especially those in the vicinity of malicious nodes, start to adjust their P_{check} values to the actual amounts of spam observed through each of the links to their neighbours. Owing to the properties of equations 3.1 and 3.4 the strength of the filtering is fine-tuned to be proportional to the amount of corrupted entries received. This explains the inability of spammers to disseminate corrupted data while using a very high spamming rate but at the same time it gives the malicious nodes an opportunity to perform a kind of denial-of-service attack on their neighbours by depleting neighbours of energy and thus, disrupt the performance of the network.

Figure 3.4 illustrates the imbalance in the workload (in average values of P_{check} as a metric) placed on the nodes when 10% of the nodes in the network are malicious and they spam 90% of their outgoing traffic. The majority of nodes have a low average P_{check} (performing less than 10% of checking) but a considerable number of nodes check more than a quarter of their incoming traffic and a few dozen nodes check around 50% or even more of the entries they receive. These nodes have malicious nodes as their neighbours and they face the danger of performance difficulties if the computational workload caused by probabilistic verification does not decrease.

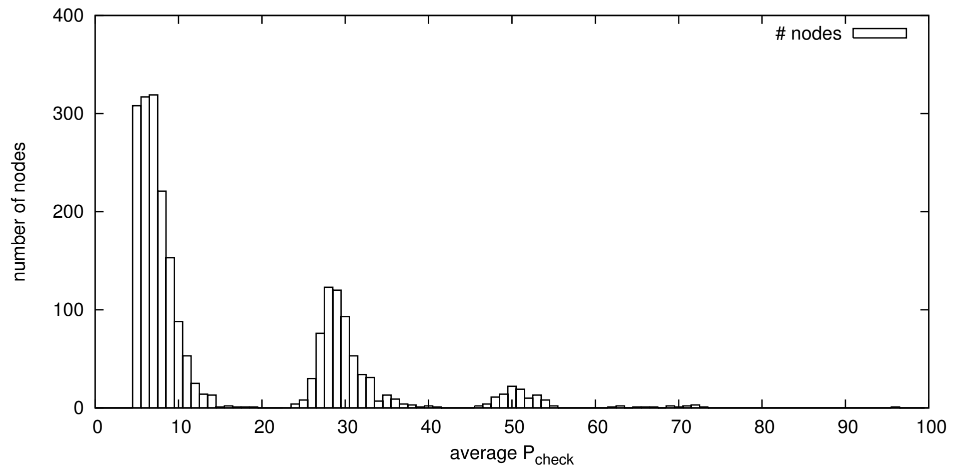


Figure 3.4: Histogram of average values of P_{check} for every node in the network (except for spammers). 250 malicious nodes with spamming rate 0.90. P_{check} updates according to the equation 3.4.

These considerations lead to the question whether the nodes can differentiate malicious nodes from well-behaved ones and isolate spammers, yet at the same time keeping the probability of isolating a good node to a minimum.

Chapter 4

Detecting and Isolating Malicious Nodes

Before a node is able to isolate malicious neighbours it needs a method of differentiating spammers from well-behaved nodes. Since nodes do not verify every entry they receive, they cannot assume that a neighbour that has sent them a corrupted entry is malicious. Actually, the probabilistic verification makes it completely acceptable that some fraction of entries relayed by nodes in the network is spam. Comparably, if a node sends out checked entries it is not enough to assume that it is well-behaved. In fact, it may be a spammer which tries to avoid detection by executing probabilistic verification.

The following section presents what the cache contents of a well-behaved node that adheres to the rules of probabilistic verification looks like and thus, what properties characterize the outgoing traffic of a well-behaved nodes. Section 4.2 describes what should raise suspicion when a node analyzes incoming traffic from one of the neighbours and in Section 4.3 the isolation mechanism is introduced and experiments concerning its performance are presented.

4.1. Cache Contents of a Well-behaved Node

When a node wants to examine the traffic coming from one of its neighbours it can observe two figures: the P_{check} related to that neighbour and the fraction of received entries marked as `checked`. It was already described in [9] how these two values should behave if a neighbour is a well-behaved node and in this section these results are briefly presented.

If a node is adhering to the rules of probabilistic verification while gossiping, part of the entries in its cache would be marked as `checked`. If spammers are present in the network, every node may have corrupted entries in its cache as well.

The probability of a corrupted entry to sneak through probabilistic verification into a node's cache ($P_{spam\ in\ cache}$) depends on both the probability with which the node's neighbour i sends spam, P_i, and the probability of checking an entry received from this neighbour, $P_{check}[i]$. More precisely, the probability of a corrupted entry making its way into a node's cache equals:

$$P_{spam\ in\ cache} = P_i(1 - P_{check}[i])\,. \tag{4.1}$$

In a similar way the probability of a node marking an entry as `checked`, $P_{checked\ in\ cache}$, can be determined by calculating the probability of an entry being selected to be checked, $P_{check}[i]$, and not being corrupted at the same time, $(1 - P_i)$:

$$P_{checked\ in\ cache} = P_{check}[i](1 - P_i)\,. \tag{4.2}$$

If a node is executing the probabilistic verification properly, $P_{check}[i]$ should approximate the probability that an entry received from neighbour i is corrupted (P_i). Therefore, we can expect the values $P_{spam\ in\ cache}$ and $P_{checked\ in\ cache}$ to be approximately the same.

Thus, every well-behaved node should have in its cache similar amounts of corrupted and `checked` entries. This statement holds even if a node has neighbours that forward different amounts of spam. For example, neighbour A may be a spammer and send many corrupted entries and therefore, be responsible for a large number of both corrupted and `checked` entries in nodes cache, while neighbour B may pass corrupted entries only once in a while and as a consequence, be responsible for only a few corrupted and `checked` entries.

Moreover, when $P_{check}[i] \approx P_i$ and we approximate $P_{spam\ in\ cache}$ and $P_{checked\ in\ cache}$ by $P_i(1-P_i)$, we can tell even more about these values. The function $f(P_i) = P_i(1-P_i)$ is increasing in the interval $[0.0, 0.5]$ and decreasing in the interval $[0.5, 1.0]$. Thus it has its maximum in 0.5 equal to 0.25. Therefore, if a node has adjusted its filters (P_{check}'s) to approximate the values of P_i's, then it can be expected that this node does not have in its cache more than 25% of spam and more than 25% of `checked` entries alike.

In this light, if a protocol used by a node to exchange data selects entries from a node's cache randomly (like the shuffle protocol does) its neighbours can expect to receive similar amounts of spam and `checked` entries and these values should not exceed 0.25 each. Significant discrepancies in the difference of the amount of corrupted and `checked` entries or fractions of corrupted or `checked` entries higher than 0.25 should raise concern.

4.2. Detection mechanism

The previous section provides us with enough information to make an attempt on detecting spammers in the network. We know that we have to pay attention not only to the number of corrupted and `checked` entries but also on the relation between these two.

We have already explained how nodes compute P_{check} values: they can use one of the equations 3.1 or 3.4. We will next motivate why the latter equation is to be used for detection and isolation of malicious nodes. However, there are some matters that should be addressed before.

Keeping track of the number of `checked` entries received from neighbours is much easier than estimating the number of corrupted entries by P_{check}. Marking an entry as `checked` can be implemented by setting one bit. Then a node that receives entries can just count the number of entries flagged as `checked` — this operation is computationally inexpensive. As a node wants to have an estimate of the overall number of `checked` entries received from the neighbour i, it can compute it with the equation similar to equation 3.1 used for updating P_{check}. The new estimated fraction of `checked` entries ($checked_{t+1}[i]$) is updated as a weighted sum of its previous value ($checked_t[i]$) and a fraction of checked entries in the current gossip exchange with neighbour i (*fracChecked*):

$$checked_{t+1}[i] = (1-\alpha)checked_t[i] + \alpha fracChecked\,. \tag{4.3}$$

We cannot use equation 3.4 here because the model for this equation assumes that we have only partial information about the number of entries of certain properties (like being corrupted).

The spammer detection mechanism introduced in [9] is based on monitoring only the difference between $P_{check}[i]$ and $checked[i]$. After every gossip exchange with neighbour i a node calculates

$$absDiff[i] = |P_{check}[i] - checked[i]| \tag{4.4}$$

and the parameter δ defines the acceptable difference:

- if $\mathit{absDiff} \leq \delta$, neighbour i is most probably behaving properly,
- if $\mathit{absDiff} > \delta$, neighbour i is a suspected spammer.

Following intuition confirmed by the experiments presented in [9], the value of the threshold δ affects the detection of spammers in the following manner.

- A small δ allows fast detection of spammers and makes it possible to detect malicious nodes even if they don't spam severely. But at the same time, a small δ can result in well-behaved nodes being mistakenly taken for spammers i.e. we will have false positives.
- When larger values of δ are used, spammers can operate for longer period of time before being identified or even not being identified at all if the spamming rate is low. Nevertheless, a larger δ threshold makes it less possible to confuse a well-behaved node with a spammer.

In our experiments we have set δ equals to 0.10 after initial calibrations.

Choosing equation for updating P_{check}

The motivation for abandoning equation 3.1 based on the weighted sum in favour of equation 3.4, which is based on conditional expectation, is closely related to the discernment between spammers and well-behaved nodes.

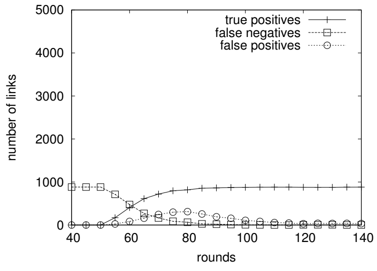
When spammers appear in the network, well-behaved nodes (especially those in the vicinity of the malicious nodes) need to adjust their values of P_{check} for their neighbours to the appropriate level. And as adjusting takes some time, during this period good nodes may relay more corrupted entries than expected. As a result, they can easily be confused with spammers, in particular, if the threshold δ is low. The duration of the regulating phase for P_{check} to match the incoming rate of spam depends highly on the equation used for updating P_{check}.

In Figure 4.1 the results of applying the detection function $\mathit{absDiff} > \delta$ where $\delta = 0.10$ are presented. The experiments were organized as described in Section 2.5. They were conducted for the network of 2500 nodes organized in a square grid topology. Each node could communicate with its adjacent neighbours to the North, South, West and East. From among all nodes present in the network, 250 malicious nodes were selected randomly and they started spamming at the 50th round.

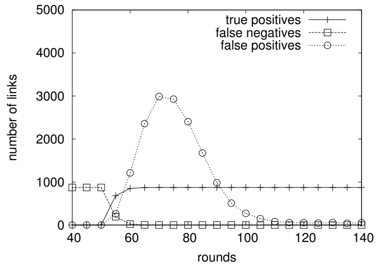
Since nodes perform separate analyses for each of their neighbours and the decision whether the neighbour is behaving suspiciously or not are made individually, we count the results *per link*, not per node. Thus, the number of *true positives* denotes the number of connections between well-behaved node and malicious one that has been rightly detected by the good node, the number of *false positives* denotes the number of connections between well-behaved node and malicious one where the good node has not detected the spammer on the other end and the number of *false positives* denotes the number of connections between two well-behaved nodes where one of the nodes has mistakenly suspected a good node on the other side (if both nodes suspect each other the connection is counted twice).

In the experiments there are on average approximately 880 links between good and malicious nodes. As a consequence, there is around 760 well-behaved nodes which have at least one spammer as a neighbour. Further, there are roughly 7930 connections between two good nodes (counted separately for each of the two directions).

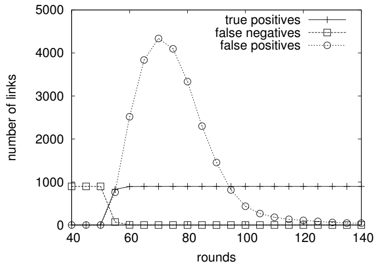
$$P_{check_{t+1}} = (1-\alpha)P_{check_t} + \alpha P', \text{ where } \alpha = 0.10$$



$P_{spam} = 0.10$

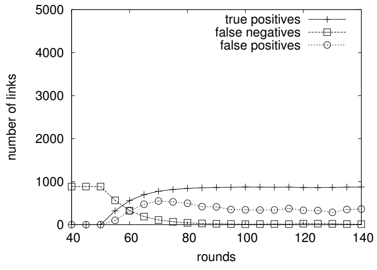


$P_{spam} = 0.50$

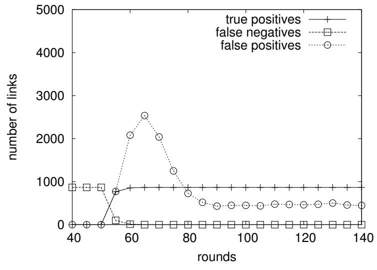


$P_{spam} = 0.90$

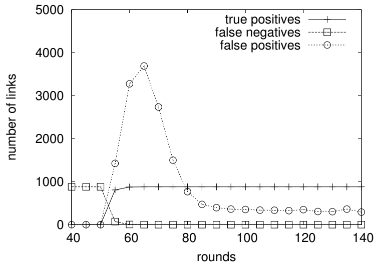
$$P_{check_{t+1}} = (1-\alpha)P_{check_t} + \alpha P', \text{ where } \alpha = 0.20$$



$P_{spam} = 0.10$

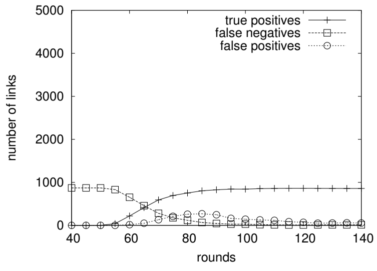


$P_{spam} = 0.50$

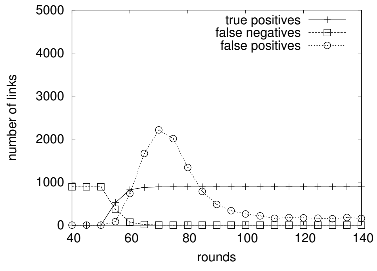


$P_{spam} = 0.90$

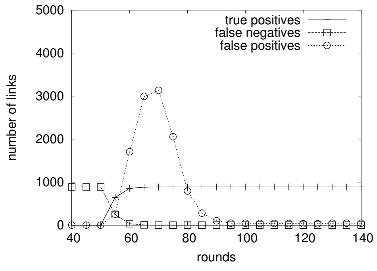
$$P_{check_{t+1}} = P_{check_t}(1 + P' - P_{check_t})$$



$P_{spam} = 0.10$



$P_{spam} = 0.50$



$P_{spam} = 0.90$

Figure 4.1: Results of applying detection function $absDiff > \delta$ ($\delta = 0.10$) over time in the network of 2500 nodes (10% of them malicious).

We easily notice that the number of false negatives drops fast to 0; our detection mechanism has no problems with detecting spammers even if the spamming rate is as low as 0.10. More alarming are the results for false positives.

What can be observed is the change over time in the number of false positives. Obviously, before the appearance of spammers in the network, there are no false positives. After the 50th round, when spammers start to operate, the number of falsely suspected good nodes grows rapidly and after some time decreases until it reaches a stabilized level, forming a mountain-like shape.

We can see in every row, which corresponds to a different P_{check} update scheme, that the number of false positives in its highest point is directly proportional to the value of P_{spam} used by malicious nodes. Therefore, from the point of view of minimizing the probability of isolating a well-behaved node, the higher P_{spam} is, the higher the probability of false accusation of a well-behaving node. Thus, while choosing the function for updating P_{check}, we should focus on the ability to adjust as fast as possible, so a well-behaved node may avoid being misjudged.

We can see that the equation using weighted sums (3.1) with $\alpha = 0.10$ performs poorly for high values of P_{spam}. When $P_{spam} = 0.10$, the number of false positives reaches over 4300 ($\sim$ 55%) and it takes over 55 rounds for nodes to reach a state where the number of mistakenly suspected links stabilizes (at around 0.5%). Using $\alpha = 0.20$ instead of $\alpha = 0.10$ results in an improvement in the decrease of the number of false positives, which is distinctly below 3700 ($\sim$ 46%), as well as a reduction in the time needed by the network to stabilize the number of false positives to around 35 rounds. But the level to which the number of false positives converges is around 4% which is much higher than in case of $\alpha = 0.10$. Further increase of α causes even further increase in the level to which false positives converge. Equation based on the conditional expectation (3.4) seems to consolidate the advantages of the equation based on a weighted sum with $\alpha = 0.10$ and $\alpha = 0.20$. The number of false positives does not reach the high values as in the case of equation 3.1 with $\alpha = 0.10$ or even $\alpha = 0.20$ stopping at the level of 39% and the number of false positives stabilizes after around 35 rounds as in the case of equation 3.1 with $\alpha = 0.20$, but at the same time, the level to which the number of false positives converges is comparable to the level in the case of equation 3.1 with $\alpha = 0.10$.

Modification of detection function

Apart from using a different equation for updating P_{check} values than in [9], a different formula for separating spammers from well-behaved nodes is proposed.

Firstly, we will not take the absolute value as in 4.4:

$$diff[i] = P_{check_t}[i] - checked[i]\,. \tag{4.5}$$

We introduce this change because we do not want to punish well-behaved nodes that happen to send out at some point more checked entries than corrupted ones. Besides, we imagine that this kind of situation, where $checked[i] > P_{check}[i]$ would be actually more common for well-behaved nodes than malicious ones. A spammer has no reason for performing any kind of verification of its traffic except for avoiding detection and isolation. Thus, it would try to keep additional work to the minimum (not performing or performing only an indispensable minimum of checks). Therefore, when a spammer has to make a decision whether to send out more spam and less `checked` messages or less spam but more `checked` messages (while still wanting to avoid detection) it is obvious that it will choose the former. We can safely assume that a spammer does not mark a corrupted entry as `checked`. Such an act would automatically identify a spammer as being malicious.

Secondly, aside from using now in our detection mechanism inequality $\mathit{diff} > \delta$ instead of $\mathit{absDiff} > \delta$, we add another inequality:

$$P_{\mathit{check}}[i] > \gamma \tag{4.6}$$

which imposes an upper bound on the value of P_{check} which now cannot be higher then γ if a node does not want to risk isolation. This inequality stems from equation 4.1, where we have shown that the amount of spam in a well-behaved node's cache should not exceed 25%. Thus, the parameter γ should be equal to or higher than 0.25. In our experiments we set for $\gamma = 0.30$ to avoid an unnecessary raise in the number of false positives.

To sum up, to detect spammers nodes will use, instead of inequality $\mathit{absDiff}[i] > \delta$, the alternative inequalities:

$$\mathit{diff}[i] > \delta \text{ or } P_{\mathit{check}}[i] > \gamma\,. \tag{4.7}$$

As our experiments confirm, the changes introduced to the detection mechanism do not lower the number of detected spammers. This is understandable as the spammers in our experiments perform a basic type of attack where they do not send any checked messages. Yet, the level to which the number of false positives converges is lower, which is a big advantage because it lowers the possibility of isolating a well-behaved node.

4.3. Isolation mechanism

The desirability of malicious nodes isolation is beyond all question. If nodes are capable of detecting and isolating spammers then once malicious nodes are isolated they are no longer able to insert any spam into the network. The network is then able to recover from a spammer's attack and in a short period of time discard corrupted entries. Besides, in the context of probabilistic verification, isolation of malicious nodes is important for every single node. If a node has a neighbour that is a spammer which produces vast amounts of corrupted entries, then the P_{check} value for this neighbour is high and the node has to devote more time to verify incoming traffic from the associated link. This is strictly connected to higher computational workload imposed on the node and higher consumption of its CPU time. And when the sensor networks are concerned, this can result in fast depletion of the node's energy resources. Therefore, the ability to isolate spammers has both local and global advantages.

In the previous section we have presented formula (4.7) to differentiate spammers from well-behaved nodes. This formula has its origin in the properties of the contents of a good node's cache. But as it was explained, the assumptions about these properties would be correct only if the node has adjusted all its filters (P_{check}'s) to the levels of spam received from neighbours. Therefore, a node cannot isolate its neighbour i just on the grounds of one gossip exchange after which the values of $P_{\mathit{check}}[i]$ and $\mathit{checked}[i]$ comply with the detection formula 4.7. The node must take into account that his neighbour i may be just struggling to adjusts its own P_{check} value for its own neighbour spammer which just started operating.

Thus, the simplest idea to isolate spammers and at the same time minimize the probability of isolating a well-behaved node is to wait for some rounds and see if the way the neighbour i is perceived improves and $P_{\mathit{check}}[i]$ and $\mathit{checked}[i]$ values no longer satisfy detection formula 4.7. In more detail, every node keeps counters (`counter`) for each of its neighbours. After every gossip exchange with neighbour i, the node checks whether updated the $P_{\mathit{check}}[i]$ and $\mathit{checked}[i]$ values satisfy formula 4.7 or not:

- if $P_{\mathit{check}}[i] \leq \gamma$ and $P_{\mathit{check}}[i] - \mathit{checked}[i] \leq \delta$ (formula is not satisfied) the counter for this neighbour is reset to 0 — `counter[i] := 0`,

- otherwise (formula is satisfied), a node is suspected spammer the counter is increased by 1 — `counter[i] := counter[i] + 1`.

If the value of `counter[i]` crosses some threshold the neighbour i is assumed to be a spammer and the node refuses any further communication with it. The node does not choose neighbour i for gossip exchanges any more and drops all the packets with entries received from i.

In Table 4.1 we present the results of an experiment on isolating nodes according to the above scheme with different values for the isolation threshold. The experiments are setup in the same way as the experiments from the previous section. What becomes evident after studying the results is that avoiding isolation of well-behaved nodes is problematic. Naturally, one could settle for the high isolation threshold but this solution leaves too much space for the spammers to change their behaviour (like stop sending spam for some time) and avoid detection.

Isolation threshold	$P_{spam} = 0.10$		$P_{spam} = 0.50$		$P_{spam} = 0.90$	
	isolated spammers (in %)	false positives	isolated spammers (in %)	false positives	isolated spammers (in %)	false positives
5	100.0	106 (1.3%)	100.0	1648 (20.8%)	100.0	2825 (35.6%)
10	99.9	23 (0.2%)	100.0	608 (7.7%)	100.0	935 (11.8%)
15	99.8	3 (0.0%)	100.0	91 (1.1%)	100.0	81 (1.0%)
20	99.4	0 (0.0%)	100.0	9 (0.1%)	100.0	4 (0.0%)
25	98.9	0 (0.0%)	100.0	1 (0.0%)	100.0	0 (0.0%)

Table 4.1: Results of simple node isolation (neighbours which counter crosses the network-wide threshold are isolated) for different threshold values.

We can see from the results presented in Table 4.1 that if we want to avoid isolating more than 1% of links connecting good nodes while using a constant isolation threshold, we should choose it to be equal or higher than 20. But at the same time, we would like to be able to isolate spammers as fast as possible.

To be able to improve the decision process of node isolation, one should understand better what happens to a node when one of its neighbours starts to send spam. Assume that node P has a malicious neighbour Q and Q has just started operating. In this situation, P has to adjust $P_{check}[Q]$ to match the level of spam received from Q. Regulating $P_{check}[Q]$ takes some time and during this period the number of corrupted entries that sneak into P's cache can be much higher than in the situation where all P_{check} values are already adjusted. Thus, P forwards to its neighbours more spam than it normally would. This may raise suspicion and, as consequence, P may face the threat of being isolated. This threat is even higher when P has more than one spammer as a neighbour. Hence, if P could isolate its maliciously behaving neighbours before being isolated itself, it would have higher chances to clear its cache (by the means of communicating only with well-behaved nodes) to an acceptable level and eventually avoid isolation.

Note, that on average node P forwards less spam than it receives from its neighbours (because part of it is discarded during the probabilistic verification phase). As a result, the $P_{check}[P]$ values at P's neighbours should be lower than the P_{check} values at node P for spammers that are neighbours of P. Bearing this in mind, if the isolation threshold depended upon the value of P_{check}, we could isolate malicious nodes fast and at the same time minimize the number of isolated good nodes.

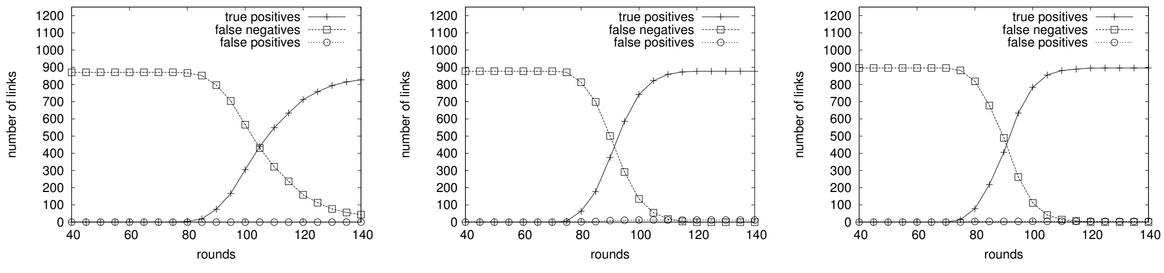
Experiments were conducted to check the performance of a few heuristic functions for node isolation where the isolation threshold depended linearly or quadratically on the value of P_{check}. The result were the most promising when nodes were isolated if

$$\texttt{counter[i]} \geq \textit{isolationThreshold}(1 - (P_{check}[i])^2) \tag{4.8}$$

with $\textit{isolationThreshold} = 20$. In this scheme, the number of links between two well-behaved nodes that were isolated by one of the nodes (false positives) was around 0.1%. This is the same level of false positives as in experiments with constant isolation threshold equal to 20.

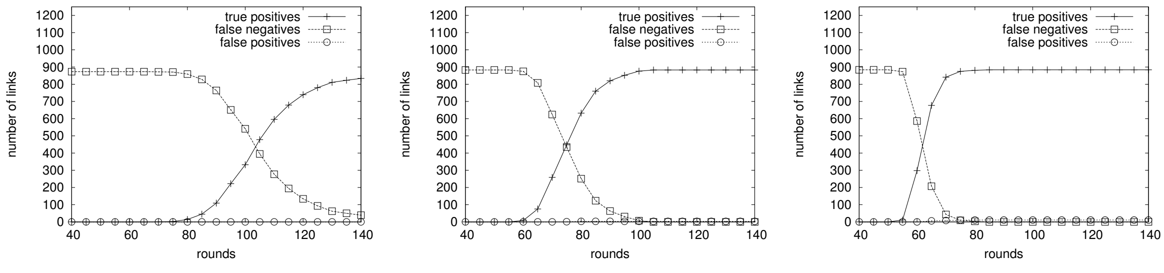
Figure 4.2 compares the results of isolating nodes with the constant isolation threshold and for an isolation threshold dependent on P_{check} (following formula 4.8). The graphs depict the number of isolated links between spammers and good nodes, as well as the number of links to malicious nodes that avoided detection (false negatives) and the number of links between two good nodes that were mistakenly isolated by one of them (false positives). The results are counted per link, since nodes do separate analyses for each of their neighbours. We can easily notice how the curves denoting the number of isolated links with spammers for $P_{spam} = 0.50$ (and even more for $P_{spam} = 0.90$) are shifted to the left when we use P_{check} dependent isolation threshold. This clearly indicates that while using formula 4.8 we isolate spammers faster than in the scheme with constant isolation threshold, keeping at the same time the number of mistakenly isolated good nodes to a minimum.

a) isolating nodes with the constant isolation threshold equal 20



$P_{spam} = 0.10$ $P_{spam} = 0.50$ $P_{spam} = 0.90$

b) isolating nodes according to inequality 4.8 with isolation threshold equal 20



$P_{spam} = 0.10$ $P_{spam} = 0.50$ $P_{spam} = 0.90$

Figure 4.2: Results of applying the isolation mechanism with constant isolation threshold or following formula 4.8 for different spamming rates.

Chapter 5

Ways of Avoiding Isolation

To have a thorough insight into the efficiency of proposed solutions for malicious node isolation we should examine how a spammer can avoid isolation. In order to prevent its neighbours from refusing communication with it, a malicious node needs to modify its behaviour enough to be confused with a well-behaved node. In this chapter three possible strategies are presented:

- Checking entries just as a normal node would do and spamming with lower probability P_{spam} in the unchecked entries.
- Imitating good node behaviour by controlling the properties of outgoing traffic (carefully managing numbers of corrupted and checked entries that are sent to each neighbour).
- Changing the number of sent corrupted and checked entries over time in order to manipulate the values of P_{check} and *checked* on a neighbour's side.

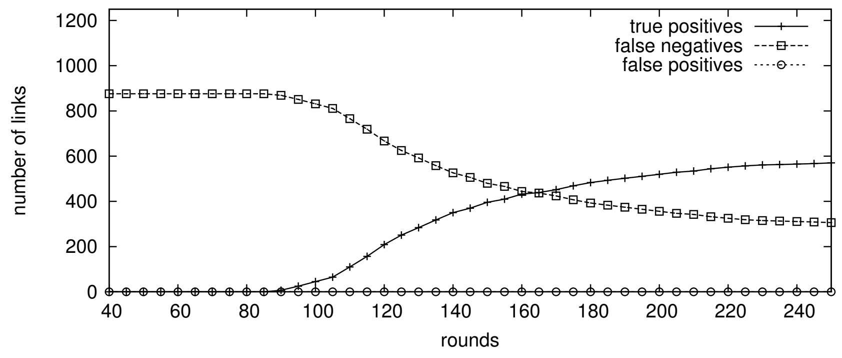
5.1. Checking and Lowering Spamming Rate

When a spammer is aware how nodes in the network detect malicious behaviour, it can make an attempt to avoid isolation. Knowing the inequalities 4.7 by means of which good nodes determine which of neighbours behaves suspiciously, a spammer may try to abide to the rules more. We identify two possible ways for a malicious node to achieve this:

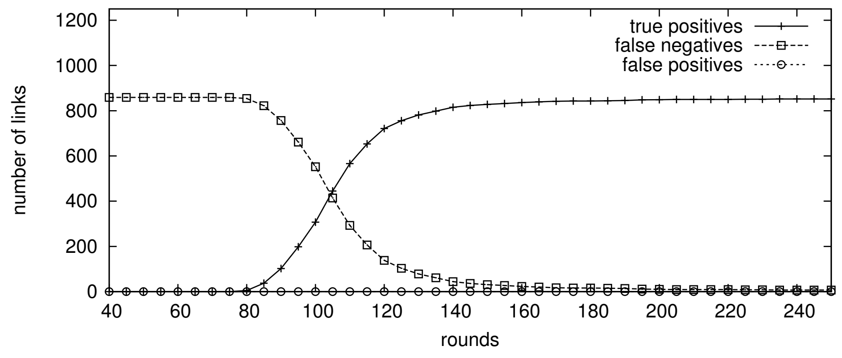
- Checking entries just as a well-behaved node would in the hope of reducing the difference between P_{check} and *checked* at its neighbours enough so that *diff* $\leq \delta$.
- Lowering P_{spam} to reduce both *diff* and P_{check} enough so that the former complies with *diff* $\leq \delta$ and the latter does not exceed the upper bound γ.

A spammer may use only one of the above strategies or both. We will consider the following four scenarios:

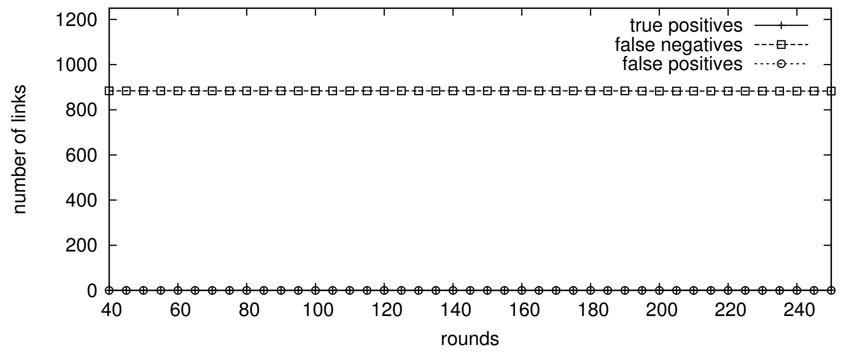
- (a) a spammer lowers its spamming rate to **0.05** and does not perform any checking,
- (b) a spammer lowers its spamming rate to **0.10** and does not perform any checking,
- (c) a spammer lowers its spamming rate to **0.05** and performs probabilistic verification as every well-behaved node,
- (d) a spammer lowers its spamming rate to **0.10** and performs probabilistic verification as every well-behaved node.



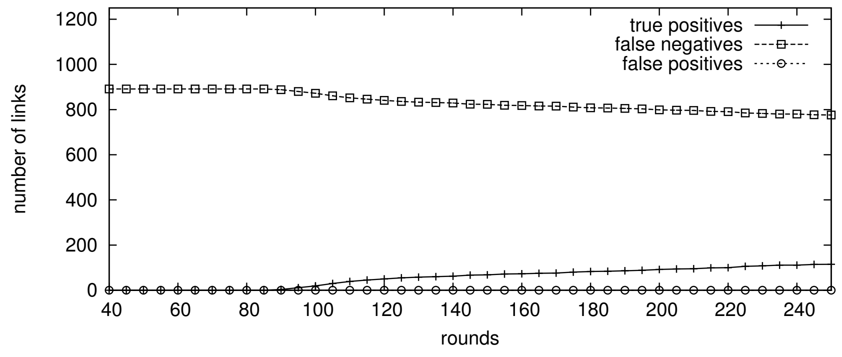
(a)



(b)



(c)



(d)

Figure 5.1: Spammers attempts at avoiding isolation by lowering spamming rate or performing probabilistic verification: a) $P_{spam} = 0.05$, no checking, b) $P_{spam} = 0.10$, no checking, c) $P_{spam} = 0.05$ and performing probabilistic verification, d) $P_{spam} = 0.10$ and performing probabilistic verification.

The results of applying the isolation mechanism for the networks of 2500 nodes in which 250 spammers try each of the above methods: (a), (b), (c) and (d) are presented in Figure 5.1.

The very first observation is that lowering P_{spam} is not enough to avoid isolation. Graphs (a) and (b) of Figure 5.1 depict the number of isolated links between spammers and good nodes when P_{spam} equals 0.05 and 0.10 respectively. In both situations malicious nodes fail at avoiding isolation. When $P_{spam} = 0.05$ after 200 rounds of spammers operating in the network (that is around 250th round), 65% of links connecting malicious nodes with well-behaved ones is already isolated. When $P_{spam} = 0.10$ almost all links to spammers are isolated during first 200 rounds of operating. When spammers perform probabilistic verification the number of malicious nodes that gets isolated drops. During the same period of time almost no link connecting spammer and good node is isolated when $P_{spam} = 0.05$ and just 13% of links with spammers at one of the ends is isolated when $P_{spam} = 0.10$.

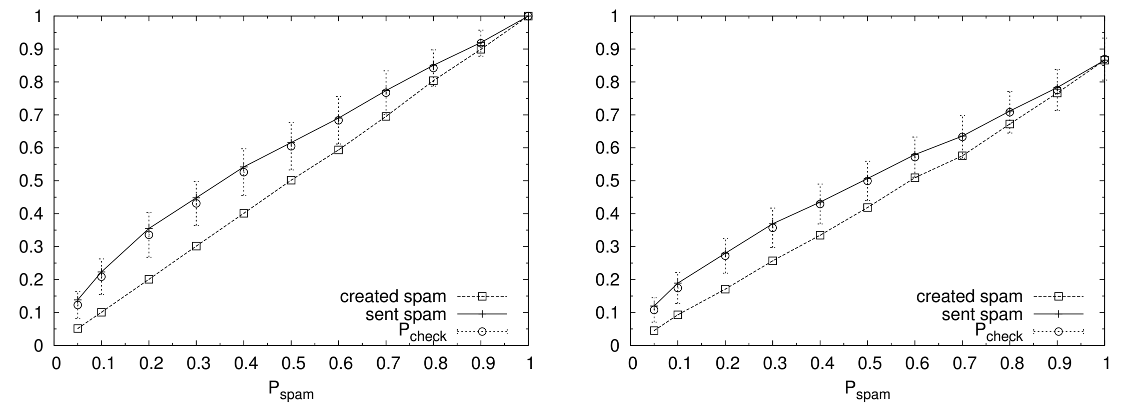
In conclusion, even though malicious nodes execute probabilistic verification and spam with rate as low as 0.10, good nodes are still able to detect and isolate them. To lower the risk of isolation while using the approach proposed in this section, a spammer has to lower its spamming rate to around 0.05. At first this may seem a little bit unintuitive. Consider a malicious node P that spams with $P_{spam} = 0.10$ and performs probabilistic verification. From the back-of-the-envelope calculations, in P's cache the number of checked entries should be equal to at least a few per cent (even if P's neighbours don't send out any spam, P checks at least $P_{check_{min}}$ entries, which in our experiments amounts to 0.05). Thus, P's neighbours should perceive the traffic coming in from P to have the following properties:

- $\mathit{checked}[P] > 0.0$,
- $P_{check}[P] < 0.10$ (because the corrupted entries are inserted with probability P_{spam} among unchecked entries).

Therefore, P should never be isolated, because $P_{check}[P] \leq \gamma$, $\gamma = 0.30$, and $\mathit{diff} \leq \delta$, $\delta = 0.10$. Yet, our experiments suggest quite the contrary. This inconsistency between our simple calculations and the experimental results derives from the misconception that in the traffic outgoing from the spammer only a fraction P_{spam} is corrupted. In fact, a malicious node also forwards the spam that it has received from its neighbours. Part of the received spam consists of the corrupted entries created earlier by this malicious node (which just came back to it), the other part consists of the corrupted entries created by other malicious nodes in the network. Figure 5.2 presents the average number of corrupted entries created by the spammers per every gossip exchange, the average number of corrupted entries actually sent out by a spammer to its neighbour and the mean and standard deviation of the P_{check} values for spammers.

We can see that especially for small values of P_{spam} the fraction of created corrupted entries is visibly lower than the fraction of corrupted entries that are sent by the spammer. This occurs regardless of performing probabilistic verification by the spammer or not, although when a spammer performs integrity checks the difference between these two fractions is a little smaller, because some of the corrupted messages that a spammer receives are discarded during the probabilistic verification.

Of course $P_{check}[P]$ at the spammer's neighbours estimates the level of spam that is received by a node and not the number of spam created by the spammer (see Figure 5.2). Thus, ironically the spam created by the spammer that traversed the network and returned to its creator's cache puts the spammer at higher risk of being discovered and isolated. Additionally, the chances of a spammer to be isolated are increased by the spam that the spammer receives and that was created by other malicious nodes operating in the network.



(a) (b)

Figure 5.2: Divergence between percentage of spam that is created or sent by malicious nodes in one gossip exchange a) spammers do not perform any integrity checks, b) spammers do probabilistic verification. 2500 nodes are in the network, 10% of them is malicious.

And for these reasons we presume that the higher the density of spammers in the network the higher the chances to isolate spammers. Therefore, a spammer should control its outgoing traffic more carefully if it wants to avoid detection and isolation.

5.2. Imitating a Well-behaved Node

When a spammer is aware of the dangers related to the lack of control over the outgoing traffic, it can decide to perform the integrity check on every incoming entry. This way it can make sure that it does not forward any unwanted corrupted entries. Of course, this may entail a higher workload on the spammer but by means of careful management of outgoing traffic, a malicious node can realize a much more dangerous attack than the one presented in the previous section. A spammer can both stay undetected by its neighbours and send more spam.

Firstly, consider a situation where our detection mechanism does not take into account the value of P_{check} but only the difference between P_{check} and *checked*. Thus, a neighbour is suspected only when $\mathit{diff} > \delta$. In this scenario spammer P can send in every gossip exchange 50% of checked entries and another 50% of corrupted ones. P's neighbours would never isolate it because the values of $P_{check}[P]$ and $\mathit{checked}[P]$ would be both around 0.5 and consequently $\mathit{diff} \leq \delta$.

We can clearly see that the previous detection mechanism introduced in [9] that uses an absolute value of *diff* would also be helpless in the presence of this type of a spammer attack. Therefore, this malicious behaviour is a good example that motivates usage of the upper bound for P_{check} aside from the upper bound for *diff*. When $P_{check} > \gamma$ and $\gamma = 0.3$, a spammer is no longer able to perform an attack in which it sends in every gossip exchange 50% of corrupted entries. But still a spammer is able to spam with P_{spam} smaller than γ and not be detected and isolated as long as it sends at the same time the same fraction of checked entries. For example, 25% of spam and 25% of checked entries in every gossip exchange.

5.3. Manipulating Spamming Rate

The previous examples of spammer behaviour assume that spammers spam with a constant spamming rate; in every gossip exchange malicious nodes send P_{spam} newly corrupted entries. Yet, it turns out that a malicious node can impose a much bigger threat on the network when it diversifies the spamming rate over time. For instance, a spammer may change the P_{spam} per every gossip exchange with a particular neighbour.

Consider the following spammer behaviour:

- Every two gossip exchanges with a particular neighbour the spammer sends only corrupted entries ($P_{spam} = 1.00$).
- Every other two gossip exchanges the spammer sends 44% of entries marked as `checked` and no corrupted ones ($P_{spam} = 0.00$).

Thus, on average the outgoing traffic comprises 50% spam.

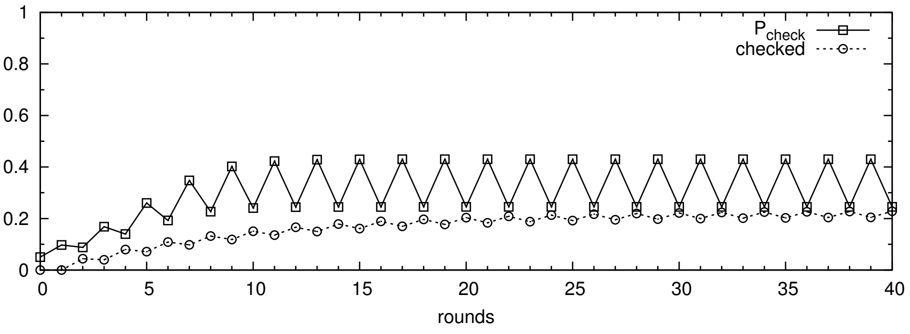


Figure 5.3: Fluctuations of P_{check} and *checked* values when a spammer changes its *spamming rate* and checked entries fraction every two gossip exchanges (equation 3.4 is used for updating P_{check}).

Figure 5.3 models the changes in the P_{check} and *checked* values on the spammer's neighbour side. These values where computed using equations 3.4 for P_{check} and 4.3 with $\alpha = 0.10$ as in all our experiments for node isolation. In even rounds the spammer sends only corrupted entries so we can observe the decrease in the *checked* value and increase in the P_{check} value for the next round. In odd rounds the malicious node sends 44% of its entries marked as `checked` and no spam. Thus, there is an increase of *checked* and a decrease of P_{check} for the next round. After the initial period when a good node tries to adjust P_{check} and *checked*, the situation stabilizes and P_{check} equals around 43% and 24% alternately and *checked* equals around 20% and 23%, respectively. Every two rounds P_{check} is smaller than 25% (and as a consequence smaller than γ) and at the same time $P_{check} - checked \approx 0.01$ so that $diff < \delta$ ($\delta = 0.10$). Therefore, our isolation mechanism would not be able to isolate this kind of a spammer.

Results of experiments for the network with 2500 nodes (see Figure 5.4) confirm our concerns. During 200 rounds of 250 spammers operating in the network (spammers start to operate at the 50th round), no link to a malicious node is isolated. Yet, we can see the growing number of false positives. During these 200 rounds there are more than 100 (1.2%) such links . This phenomenon is caused by the fact that a neighbour of the spammer is not able to adjust its P_{check} value to match the spam coming in from a malicious node. In every

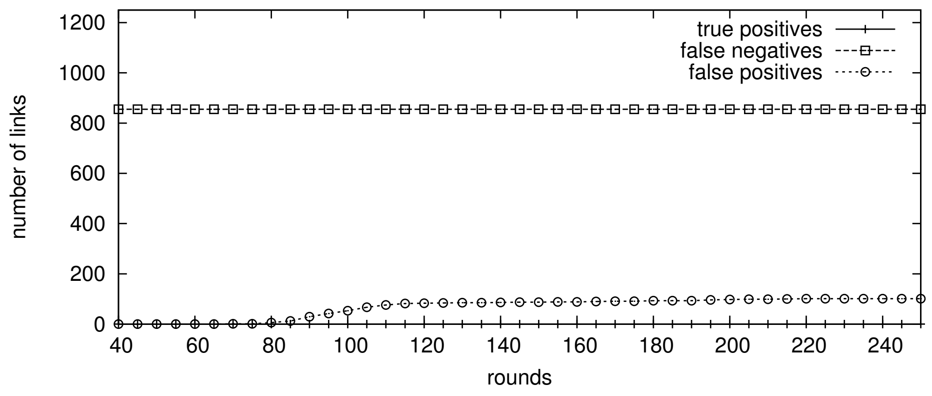


Figure 5.4: Results of applying the isolation mechanism when spammers change their *spamming rates* and `checked` entries fractions every two gossip exchanges.

other gossip exchange when the spammer sends only corrupted entries, P_{check} is lower than 25% and as a result more than 75% of corrupted entries from this gossip exchange sneak into its neighbour's cache. That is much more than 25% when a spammer spams with constant $P_{spam} = 0.50$. Hence, the adjacent nodes of this well-behaved neighbour may confuse it with a spammer and isolate it.

Another consequence of the fact that the neighbours of the spammers are not able to adjust their P_{check} values is the higher number of corrupted entries in the network. Figure 5.5 compares the number of corrupted entries in the network over time when spammers spam with constant spamming rate equal to 0.50 and do not perform any integrity checks (and there is no isolation mechanism) with the number of corrupted entries when spammers use the approach introduced in this section (their average spamming rate is equal 0.50). In the latter situation, even though the isolation mechanism is active, the fraction of corrupted entries present in the network is higher than in the former one by more than 2%.

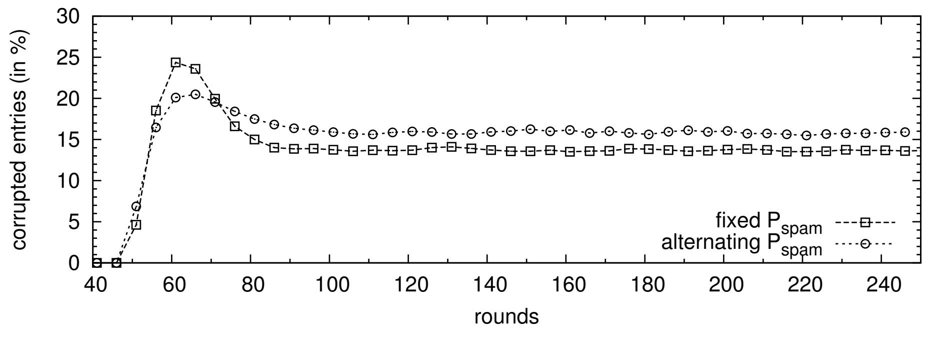


Figure 5.5: Fraction of corrupted entries in the network of 2500 nodes when 250 malicious nodes spam with constant spamming rate 0.50 (fixed P_{spam}) or change their spamming rate every two gossip exchanges (alternating P_{spam}).

The weak point of our isolation mechanism that is responsible for the problems with detecting and isolating this type of a spammer seems to be the function for updating P_{check} that doesn't converge to the average level of received spam. Using the function based on the weighted sum $P_{check_{t+1}} = (1 - \alpha)P_{check_t} + \alpha P'$ with $\alpha = 0.10$, for example, would yield much better results (compare Figures 5.3 and 5.6). When this equation is used, P_{check} would fluctuate in the range between ~ 0.47 and ~ 0.53 and obviously the spammer would be isolated. But we must recall from Section 4.2 the motives behind the decision to use equation 3.4 instead of equation 3.1 that uses weighted sum. Especially for small values of α like 0.10 the time needed by a node to adjust P_{check} while using equation 3.1 may be much longer than while the node is using equation 3.4. Thus, when detection mechanism is active, percentage of false positives is visibly higher (for example, around 55% instead of 39% when $P_{spam} = 0.90$) and more time is needed for the level of false positives to stabilize at less than 1% (~ 55 rounds instead of ~ 35). As a consequence, applying isolation mechanism may take a heavier toll of false positives when equation 3.1 with $\alpha = 0.10$ is used instead of equation 3.4.

Therefore a much better solution could be introducing the isolation decision based on a sliding window, when a node would take into the account the average number of gossip exchanges when the neighbour misbehaved, rather than the counter, that is reset on the basis of a single exchange.

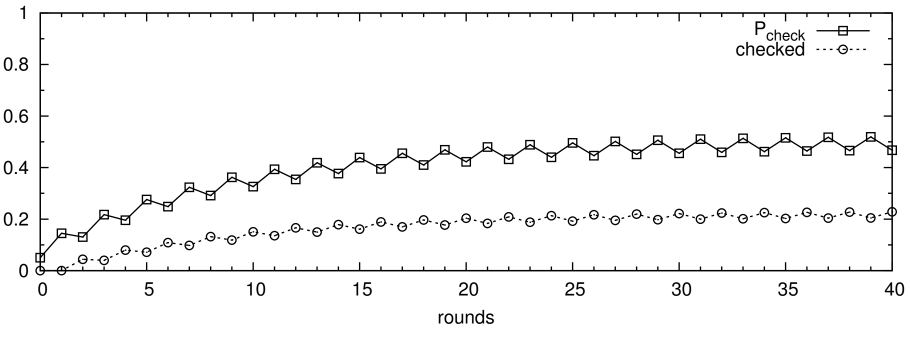


Figure 5.6: Fluctuations of P_{check} and *checked* values when a spammer changes its *spamming rate* and checked entries fraction every two gossip exchanges (equation 3.1 with $\alpha = 0.10$ is used for updating P_{check}).

Chapter 6

Conclusions and Future Work

6.1. Conclusions

The problem discussed in this thesis is a good example of how adding a simple constraint to the assumptions of an algorithm can change an easy task into something much more complicated. If nodes in the network could verify digital signatures of every entry they receive, guaranteeing the integrity of data in the network and isolating malicious nodes would be trivial. A corrupted entry would be discarded by the very first node that receives it and the sender would be isolated immediately.

However, when we introduce a constraint that the costs of verifying all entries at every hop are prohibitively high, the problem increases in complexity. The direct result of this restriction is that it can no longer be assumed that a node is malicious only because it has forwarded some spam. On the contrary, it becomes a normal situation for a node in the network to relay some number of corrupted entries. As a consequence, detection and isolation of malicious nodes cannot be based solely on the discovery that the neighbour has sent a corrupted entry to a node.

I have proposed an algorithm that addresses the problems of assuring data integrity and isolation of malicious nodes. The algorithm can be divided into two parts: *probabilistic verification* and *node isolation*. The probabilistic verification phase is based on the works of Gavidia at al. ([10], [9]) and was modified by me by introducing a new equation for updating P_{check} values. The isolation phase is aimed at isolating the malicious nodes and at the same time preventing isolation of well-behaved nodes that could be mistakenly suspected. This is achieved by differentiating the isolation threshold on the basis of the level of harmfulness of a node (nodes with higher P_{check} are isolated faster). The strength of our solution stems from the fact that the decision on the isolation of a neighbour is taken solely on the grounds of a node's individual observations of the incoming traffic (no reputation information is exchanged).

The experiments proved that by using this algorithm, malicious nodes that spam with a constant spamming rate can be isolated without difficulties. The results show that even spammers with spamming rate as low as $P_{spam} = 0.10$ are detected and isolated.

Unfortunately, when more elaborate spammer attacks were considered, the weak points of the presented algorithm were revealed. In general, when we assume that a spammer knows all about the algorithm that is used in the network, it can adapt its own behaviour in a way which allows him to avoid detection and at the same time its outgoing traffic can be comprised in 50% (or even more) of corrupted entries.

The vulnerability of our algorithm to this kind of spammers is induced by the deterministic nature of the protocol. The spammer can easily predict how it will be perceived by its neighbours. Thus, it can deceive nearby nodes by manipulating the numbers of corrupted and checked entries it sends.

One of the possible readjustments that could improve our solution is to introduce randomness into our protocol regarding isolation threshold but more importantly, the functions for updating P_{check} and *checked*. This modifications would hinder spammers from predicting whether their behaviour is perceived as suspicious or not.

6.2. Future Work

In the future, it seems crucial to explore how to detect and isolate malicious nodes that deploy more elaborate schemes of attack (for example, use variable P_{spam}). One of possible directions in which the research could go is introducing variable isolation threshold and more randomness of functions for updating P_{check} and *checked*. As far as the isolation threshold is concerned, we can analyse approaches where the threshold is chosen at the beginning individually for every node or per connection or even changes in time. Randomness in P_{check} and *checked* update function can be understood as using equation $P_{check_{t+1}} = (1-\alpha)P_{checkt} + \alpha P'$ (3.1) with different (randomly chosen) values of α but we can also imagine having a set of completely different functions like a set containing equation 3.4, equations 3.1 with different values of α etc. from which a node randomly chooses the equation to use. This kind of solution should make it much more difficult for malicious nodes to find a way of avoiding isolation, because spammers would not be able to predict how their neighbours perceive their behaviour and if they are already suspected or not.

Another issue, which was beyond the scope of this thesis but which seems worth inquiring, is whether the P_{check} value can be used to adjust the frequency of communication with particular neighbours. In the current solution the neighbour for the next gossip exchange is selected randomly. If a node observes that its neighbour is behaving maliciously it can only completely refuse to communicate with it. Another solution could allow nodes to adjust the probabilities with which a node can contact each of its neighbours. In result, the node would be able to contact more frequently with the neighbours that it perceives send less spam and less frequent with those sending more spam. We already can foresee that in this suggested solution the measures against the spammers that would like to manipulate P_{check} values are even more important.

Bibliography

[1] S. Bansal, M. Baker, *Observation-based Cooperation Enforcement in Ad Hoc Networks.* Tech. Rep., Stanfor University, USA, July 2003.

[2] R. Bobba, L. Eschenauer, V. Gligor, W. Arbaugh, *Bootstrapping security associations for routing in mobile ad-hoc networks.* May 2002.

[3] S. Buchegger, J.-Y. Le Boudec, *Performance Analysis of the CONFIDANT Protocol; Cooperation of Nodes: Fairness in Dynamic Ad Hoc NeTworks.* In Proc. of IEEE/ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC), Lausanne, Switzerland, June 2002.

[4] S. Buchegger, J.-Y. Le Boudec, *The Effect of Rumor Spreading in Reputation Systems for Mobile Ad Hoc Networks.* In WiOpt'03: Modelling and Optimization in Mobile, Ad Hoc and Wireless Networks, March, 2003.

[5] S. Buchegger, J.-Y. Le Boudec, *Coping with False Accusations in Misbehaviour Reputation Systems for Mobile Ad Hoc Networks.* EPFL Technical Report Number IC/2003/31, 2003.

[6] L. Buttyan, J. P. Hubaux, *Enforcing Service Availability in Mobile Ad Hoc WANs.* In Proc. of IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC), Lausanne, Switzerland, June 2002.

[7] L. Buttyan, J. P. Hubaux, *Stimulating Co-Operation in Self Organizing Mobile Ad Hoc Networks.* Mobile Networks and Applications, vol. 8 no. 5, pp. 579-592, October 2003.

[8] V. Coscun, E. Cayirci, A. Levi, S. Sandak, *Quarantine Region Scheme to Mitigate Spam Attacks in Wireless Sensor Networks.* In IEEE Transactions on Mobile Computing, vol. 5, no. 8, pp. 1074-1086, August 2006.

[9] D. Gavidia, M. van Steen, *Enforcing Data Integrity in Very Large Ad Hoc Networks.* In Proc. 8th IEEE Int'l Conf. on Mobile Data Management (MDM'07), Mannheim, Germany, May 2007.

[10] D. Gavidia, G. P. Jesi, C. Gamage, M. van Steen, *Canning Spam in Wireless Gossip Networks.* In Proc. Fourth Annual Conference on Wireless On demand Network Systems and Services (WONS), Obergurgl, Austria, January 2006.

[11] D. Gavidia, S. Voulgaris, M. van Steen, *A Gossip-based Distributed News Service for Wireless Mesh Networks.* In Proc. 3rd IEEE Conference on Wireless On demand Network Systems and Services (WONS), Les Nenuires, France, January 2006.

[12] Q. He, D. Wu, P. Khosla, *SORI: A secure and Objective Reputation-based Incentive Scheme for Ad hoc Networks.* In Proc. of IEEE Wireless Communications and Networking Conference (WCNC2004), Atlanta, USA, March 2004.

[13] Y.-C. Hu, D. B. Johnson, A. Perrig, *Sead: Secure efficient distance vector routing for mobile wireless ad hoc networks.* In Proc. of the Fourth IEEE Workshop on Mobile Computing Systems and Applications, IEEE Computer Society, 2002.

[14] Y.-C. Hu, A. Perrig, D. B. Johnson, *Ariadne: a secure on-demand routing protocol for ad hoc networks.* In Proc. of the 8th annual international conference on Mobile computing and networking, ACM Press, 2002.

[15] D. B. Johnson, D. A. Maltz, *Dynamic source routing in ad hoc wireless networks.* In Imielinski and Korth, editors, Mobile Computing, volume 353. Kluwer Academic Publishers, 1996.

[16] P. Papadimitratos, Z. Haas, *Secure routing for mobile ad hoc networks.* January 2002.

[17] V. D. Park, M. S. Corson, *A highly adaptive distributed routing algorithm for mobile wireless networks.* In INFOCOM (3), pp. 1405-1413, 1997.

[18] A. Patwardhan, J. Parker, A. Joshi, M. Iorga, T. Karygiannis, *Secure Routing and Intrusion Detection in Ad Hoc Networks.* In Third IEEE International Conference on Pervasive Computing and Communications (PerCom'05), 2005.

[19] C. Perkins, E. Belding-Royer, S. Das, *Ad hoc On-Demand Distance Vector (AODV) Routing.* July 2003.

[20] C. Perkins, P. Bhagwat, *Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers.* In ACM SIGCOMM.94 Conference on Communications Architectures, Protocols and Applications, pp. 234-244, 1994.

[21] M. T. Refaei, V. Srivastava, L. DaSilva, M. Eltoweissy, *A Reputation-based Mechanism for Isolating Selfish Nodes in Ad Hoc Networks.* In Proc. IEEE Second Annual International Conference on Mobile and Ubiquitous Systems (MOBIQUITOUS 2005), San Diego, USA, July 2005.

[22] K. Walsh, E. G. Sirer, *Fighting peer-to-peer spam and decoys with object reputation.* In Proc. of P2PECON Workshop, Philadelphia, USA, August 2005.

[23] Y.-C. Tseng, J.-R. Jiang, J.-H. Lee, *Secure bootstrapping and routing in an ipv6-based ad hoc network.* In ICPP Workshop on Wireless Security and Privacy, 2003.

[24] M. G. Zapata, *Secure Ad hoc On-Demand Distance Vector (SAODV) Routing.* In Internet Draft, 2002.

[25] S. Zhong, J. Chen, Y. R. Yang, *Sprite: A Simple, Cheat-Proof, Credit-Based System for Mobile Ad Hoc Networks.* In Proc. of IEEE Infocom 2003, pp. 1087-1997, San Francisco, USA, April 2003.