Enforcing Data Integrity in Very Large Ad Hoc Networks

Daniela Gavidia
Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081HV
Amsterdam, The Netherlands
daniela@cs.vu.nl

Maarten van Steen
Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081HV
Amsterdam, The Netherlands
steen@cs.vu.nl

Abstract

Ad hoc networks rely on nodes forwarding each other's packets, making trust and cooperation key issues for ensuring network performance. As long as all nodes in the network belong to the same organization and share the same goal (in military scenarios, for example), it can generally be expected that all nodes can be trusted. However, as wireless technology becomes more commonplace, we can foresee the appearance of very large, heterogeneous networks where the intentions of neighboring nodes are unknown. Without any security measures in place, any node is capable of compromising the integrity of the data it forwards. Our goal in this paper is to ensure the integrity of the data being disseminated without resorting to complex and expensive solutions. We achive this by discouraging malicious behavior in two ways: a) enforcing integrity checks close to the source and b) refusing to communicate with obviously malicious nodes. We find that by having nodes sample their traffic for corrupted messages, malicious nodes can be identified with high accuracy, in effect transforming our collection of nodes into a self-policing network.

1 Introduction

Given the dynamic nature, often unreliable links and lack of a central authority that characterize ad hoc networks, giving any hard guarantees regarding their performance is a difficult task. The situation becomes even more complex if we envision very large networks of heterogeneous nodes. In this scenario, not only would we have to deal with the issue of scale, but also with the fact that we cannot be certain of the willlingness of all nodes to cooperate towards a common goal. The lack of a central authority to oversee the good behavior of nodes is a clear disadvantage.

A common assumption is that nodes adhere to executing the chosen communication protocol. Under this condition, content can be disseminated through the network in a reliable way. However, when some nodes decide not to play by the rules, the characteristics of the dissemination as well as the reliability of the content being forwarded might change. We refer to any kind of message placed in the network as a result of malicious behavior as *spam*, as these unsolicited messages serve only the interest of the malicious node(s) and waste the already limited resources in the network. We use the term malicious node and spammer interchangeably.

This paper studies the effect of having misbehaving nodes in the network that compromise the integrity of the data being disseminated and the measures that can be taken to counter act such malicious behavior. Since we are dealing with large-scale networks, our intention is to develop effective solutions that scale easily. Therefore, we favor simplicity and the use of local interactions and decisions only.

The Cost of Guaranteeing Data Integrity The conventional approach to ensuring the integrity of a message is to require that the message be signed by its creator. By verifying the digital signature on the message, the receiver can be assured of its integrity. However, this procedure is computationally expensive. In a wireless ad hoc network, where nodes act themselves as routers, a message may have travelled several hops before reaching its destination. Due to the lack of a trusted infrastructure for routing, the message might have become corrupted along the way. If that is the case, and the receiver verifies this with an integrity check, the cost incurred due to the corrupted message is not just limited to the verification of the signature, but it also includes the cost of routing. This situation could be avoided by executing integrity checks at every hop. As a result, data integrity would be guaranteed and malicious nodes could be easily detected.

The downside of this approach is the heavy computational load inflicted on the nodes, as each node would have to check every message it forwards. Therefore,

even at times when no malicious nodes are present, the nodes in the network would be wasting resources checking valid messages.

Contribution Our work strives to find a middle ground with regard to the workload imposed on nodes to guarantee the integrity of data in the network. First, we present a probabilistic data verification scheme, which dynamically adapts the workload of each individual node according to the threat of malicious nodes in its surroundings, in essence reducing the amount of work required by nodes that are not in the vicinity of malicious nodes. As a result, the overall workload in the network is kept low and it concentrates around the malicious nodes. Second, we take a proactive approach to enforcing data integrity in the network by having the nodes constantly monitor the good behavior of their neighbors. In addition, we show that the immediate neighbors of malicious nodes are able to detect their suspicious behavior with high accuracy, enabling them to take measures to prevent further corruption of data.

Nodes make their own decisions to regulate traffic according to perceived adherence to good behavior by their neighbors. As a consequence, suspicious behavior is penalized and the malicious nodes are faced with the decision of adhering to the rules or be isolated.

Related Work In our work, discovery of malicious nodes is made possible by statistical analysis of incoming messages. Unlike reputation-based systems [2] where nodes rely on second-hand reputation reports (which could be false) to determine if a neighbor is misbehaving, our approach avoids issues of trust by relying only on first-hand observations to asses the behavior of a neighbor. In that sense, our work lies closer to [1], which avoids issues of trust by relying only on first-hand observations to build the reputation of a node. Other efforts to alleviate the problem of malicious behavior by enforcing cooperation include payment systems [3] which assume that a node can be swayed away from his selfish behavior through economic incentives.

From the data integrity point of view, our work is closely related to the efforts to counteract content pollution in peer-to-peer networks [8]. A related problem due to malicious behavior is index poisoning [6], which could lead to effective DDOS attacks [7].

Roadmap The reminder of this paper is structured as follows. In the next section, we describe the system model for our data integrity enforcement mechanism, specifying the assumptions we make and describing the platform used for testing. Section 3 details the type of attack expected from malicious nodes, as well as its effect on well-behaved nodes and the clues that could help a node identify a misbehaved neighbor. In Section 4, we analyze (through simulations) the damage caused by malicious nodes in terms of the amount of corrupted content they are able to place in the network. Section 5 describes the method we propose to identify malicious nodes. An experimental evaluation based on simulations is presented. We conclude in Section 6.

2 System Model

We focus on store-and-forward systems where nodes devote a limited amount of space to store messages. We refer to this space as the node's *cache*. The communication medium is wireless and, therefore, messages are disseminated through the network in a multi-hop fashion. We assume that nodes forward a batch of messages at a time.

Concerns regarding authentication and integrity are addressed through conventional security measures. Nodes that publish information are required to digitally sign their messages. Consequently, all messages in the network have a digital signature and are subject to integrity checks. We assume that executing an integrity check for every message received is prohibitively expensive for a node.

Wireless gossip networks fall under these assumptions. The goal of these networks is to achieve reliable, robust and scalable data dissemination. To this end, nodes are required to engage in communication with their neighbors on a regular basis. Executing integrity checks under these conditions would be too expensive and undesirable. With this in mind, we propose a probabilistic solution for integrity verification. Using wireless gossip networks as an example platform, we evaluate the effectiveness of our proposed data integrity enforcement solution.

2.1 Example: Gossip-based News

The Gossip-based News Service, as introduced in [5], serves as the experimental platform to evaluate the data integrity enforcement measures described in this paper. The service is provided by a mesh backbone made up of a large number of wireless routers that communicate through gossiping. Owners of the routers running the service are able to publish *news items* of interest to mobile users, which are gossiped through the mesh backbone. The users carry portable mobile devices capable of connecting to the mesh backbone to retrieve news items. By informing a nearby router of their preferences, users are able to receive only relevant news items in their portable devices. A detailed evaluation of the service can be found in [5].

At the router level, the news service uses a gossip protocol to disseminate news items. News items are propagated through the network in the form of *entries*.

While an item is a piece of information, an entry is the representation of the item in the network and for each item several entries may exist, since entries are replicated during gossiping. Figure 1 shows the skeleton of this push-pull epidemic protocol. Nodes gossip periodically, initiating an exchange (active thread) once every *round* (fixed gossiping interval). Three methods, `selectPeer()`, `selectItemsToSend()` and `selectItemsToKeep()` represent the core of the protocol:

- `selectPeer()`: Select a neighbor randomly
- `selectItemsToSend()`: Randomly select s entries from the local cache and send a copy of those entries (`buff_send`) to the selected peer.
- `selectItemsToKeep()`: **Probabilistically verify data integrity**. Add received entries (`buff_recv`) to the local cache and remove repeated entries. If the number of entries exceeds c, remove entries among the ones that were previously sent (unless they were also in `buff_recv`) until the cache contains c entries.

In bold we highlight the added measure to enforce data integrity as nodes disseminate data through the network. In the remainder of the paper we examine the effect of adding these measures to the protocol for a wireless gossiping network where a number of malicious nodes compromise the integrity of messages.

2.2 Probabilistic Verification

Upon receiving a set of messages from a neighbor, nodes execute integrity checks by verifying a digital signature on each received message with probability P_{check}. If the message is valid, then it is marked as `checked` and stored in the node's cache. Otherwise, the message is discarded. We call this process *probabilistic verification*. As introduced in [4], probabilistic verification proves to be an effective method for reducing the amount of corrupted content and restricting its spread.

While [4] proved that probabilistic integrity checks could reduce the impact of spam in the network, the nodes did not have a role in determining the strength of the measures taken against malicious nodes. The probability of checking an entry P_{check} was a fixed network-wide parameter, chosen at the beginning of each experiment. As a result, nodes had to do the same amount of work regardless of the conditions in their surroundings (being flooded with corrupted messages or not).

In this paper, nodes are given the autonomy to apply probabilistic verification on an individual basis, in effect transforming our collection of nodes into a self-policing network. By making nodes self-aware with respect to malicious behavior in the network (in particular, corrupt messages being disseminated), small aedjustments to local behavior can be made. The result of these adjustments is that security measures are applied where needed, while nodes in safe areas keep their work to a minimum (but always maintaining a watchful eye).

Discouraging malicious behavior in the network involves two steps: a) executing probabilistic verification and b) updating the checking probability P_{check} for the next round. P_{check} is a local parameter and its value is updated according to the observations made during the probabilistic verification stage. The dynamic nature of ad hoc networks makes it necessary to continuously adjust the value of P_{check}.

2.3 Verifying the Integrity of Data in a Dynamic Environment

We define the level of *pollution* in a collection of messages as the fraction of corrupted messages. Nodes get an insight into the pollution levels in their neighborhood during the probabilistic verification phase. When participating in a gossip exchange, a node P receives s entries from a neighbor Q. These entries are subject to checking their integrity with a probability P_{check}, resulting in a fraction of the s entries being checked. Since the neighbor selected the s entries at random from its cache, this sample gives node P an estimate of the level of pollution in the neighbor's cache. Node P can then use this information to update P_{check} for the next round in the following manner:

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

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

P' is the level of pollution calculated after checking *numChecked* items in the probabilistic verification phase. *numRemoved* is the number of items that did not pass the integrity test and were removed. The value of P_{check} for the next round ($P_{check_{t+1}}$) is updated as a weighted sum of its previous value (P_{check_t}) and the level of pollution P'. The parameter α determines the sensitivity of parameter P_{check} to changes in the pollution levels in the neighborhood.

In general, nodes are bound to have more than one neighbor. For this reason, each node should maintain a different P_{check} for each neighbor. In essence, for each neighbor i node P maintains a $P_{check}[i]$.

2.4 Experimental Setup

All nodes in the network start gossiping with no knowledge of their environment besides the identity of their immediate neighbors. Since nodes have no preconceptions

```
/*** Active thread ***/
// Runs periodically every T time units
Q = selectPeer()
buff_send = selectItemsToSend()
send buff_send to Q
receive buff_recv from Q
cache = selectItemsToKeep()
```

```
/*** Passive thread ***/
// Runs when contacted by another node
receive buff_recv from any P
buff_send = selectItemsToSend()
send buff_send to P
cache = selectItemsToKeep()
```

Figure 1. Skeleton of the gossip protocol for a News service.

about their neighbors, they start gossiping with little caution. This means that they apply a low level of checking at $t = 0$. For the experiments presented in the upcoming sections, $P_{check_{t=0}}[i] = P_{check_{min}} = 0.05$ for all nodes and all neighbors i. $P_{check_{min}}$ is also the lower bound for P_{check}. A lower bound for P_{check} is necessary, since some checking is needed to monitor any changes in the behavior of neighbors. The implications of this are that there is a minimum workload imposed on the network, even in the absence of malicious nodes, and that there is a reaction time upon appearance of malicious nodes during which P_{check} does not match the amount of spam being received.

In our experiment, nodes are arranged in a square grid topology, with 50 nodes on each side, over an area of 50×50 units. Each of the 2500 nodes has a range of 1 unit, making communication possible with its immediate neighhbors to the North, South, East and West. From this collection of nodes, 250 are selected to be malicious at the beginning of each experiment. The selection is random, resulting in malicious nodes being placed at random locations in the grid. For all experiments, nodes have a cache size of $c = 100$ and during each gossip exchange they exchange $s = 50$ entries.

3 Enforcing Data Integrity by Discouraging Malicious Behavior

We define malicious behavior as the execution of a variation of the gossip protocol with the intent of gaining an unfair advantage in the use of a shared resource. In our system, the shared resource is storage space. By deviating from the data exchange rules defined in the shuffle protocol that the majority of nodes are executing, a malicious node can increase its share of storage space. The method used by a malicious node to place large quantities of its own content in the network is compromising the data integrity of the messages it forwards.

3.1 Attack Model

In order to test the effectiveness of the proposed method for enforcing data integrity, we assume that a relatively small number of nodes in the network are malicious. These malicious nodes, which we also call *spammers*, are randomly placed in the network and execute a slightly different version of the shuffle protocol. Their basic attack model is to corrupt entries before forwarding them to the nodes they communicate with. A spammer may deviate from the normal behavior of the general population in the following ways:

- Corrupt outgoing entries (in `selectItemsToSend()`) with a probability P_{spam} (also referred to as *spamming rate*).
- Fail to execute any integrity checks (in `selectItemsToKeep()`).

Although the most effective attack appears to be spamming with $P_{spam} = 1$, we will show later that a high spamming rate is actually counter productive. In Section 4, we illustrate through simulation results that the only way for a malicious node to place more spam in the network is by spamming less (i.e. "behaving better").

3.2 Can Malicious Nodes Be Identified?

This section describes the expected composition of the cache of a node that properly applies our spam removal algorithm and how that could help differentiate well-behaved nodes from malicious ones.

3.2.1 Cache Contents of a Well-behaved Node

The probability of having a corrupted entry after the probabilistic verification phase, $P_{spam\_in\_cache}$, is directly related to the probability with which a node's neighbor forwards spam to the node. For a node Q with a neighbor i that forwards a corrupted entry with a probability P_i, the probability of a corrupted entry making its way into Q's cache can be expressed in terms of P_i and the probability of Q checking an entry received from a neighbor i, $P_{check}[i]$:

$$P_{spam\_in\_cache}[i] = P_i\,(1 - P_{check}[i]) \qquad (1)$$

In a similar way, we can determine the probability of Q marking an entry as `checked` and placing it into its cache, $P_{checked\_in\_cache}$, by calculating the probability that an entry

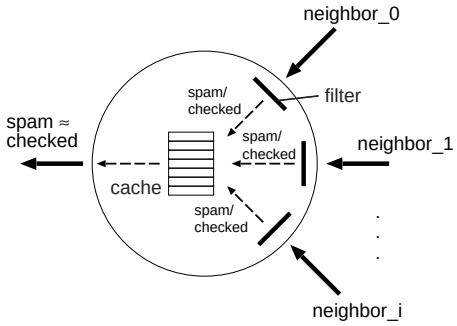

Figure 2. If a node applies probabilistic verification properly, the amount of spam and checked items in its cache should be similar.

received by Q is selected to be checked and is not corrupted. The probability of this occuring is:

$$P_{checked\_in\_cache}[i] = P_{check}[i]\,(1 - P_i) \qquad (2)$$

Assuming that node Q is executing the probabilistic verification properly, $P_{check}[i]$ should approximate P_i, the probability that a received entry is corrupted. Therefore, we can expect that when dealing with neighbor i the percentage of spam sent by i that makes it into Q's cache roughly approximates the percentage of entries marked as `checked` by Q and placed in Q's cache. This comes as a result of (1) and (2) being approximately the same when $P_{check}[i] \approx P_i$.

As a general case, node Q has many neighbors and each neighbor may forward a different amount of spam. For example, neighbor A may be malicious and send many corrupted messages, while B may pass along a few corrupted messages sporadically. Nevertheless, the interaction with each neighbor should result in similar amounts of spam and checked entries arriving into Q's cache. Therefore, neighbor A may be responsible for a large number of corrupted and checked entries in Q's cache, while neighbor B, which rarely forwards spam, is only responsible for a few corrupted and checked entries. As a result, if node Q is properly filtering the content received from its neighbors, Q's cache should have similar amounts of corrupted and checked entries. Figure 2 illustrates this scenario.

3.2.2 Expected Incoming Traffic

When exchanging entries with a malicious node, spam may come in two forms: as checked entries or as unchecked entries. Marking a corrupted entry as `checked` is a big risk as it effectively proves that the neighbor is indeed misbehaving. If a node runs an integrity check on an entry marked as `checked` by a neighbor and the test fails, the neighbor identifies itself as a spammer. Therefore, it is unlikely that a malicious node would send spam marked as `checked`. We can expect corrupted entries to arrive as unchecked.

The shuffle protocol specifies that entries to shuffle are selected at random from a node's cache. Therefore, given that a well-behaved node has a similar amount of spam and checked entries in its cache, we can reasonably expect to receive a similar amount of corrupted entries and checked entries when engaging in a gossip exchange with a well-behaved neighbor. A significant difference between the two should raise concerns. the spammer detection mechanism (introduced later on) is built on this principle.

4 Effect of Spammers on the Network

This section shows the extent of the damage caused by spammers in terms of the amount of spam they can place in the network.

4.1 The Effect of Alpha

The parameter α, introduced in Section 2.3, determines the sensitivity of the integrity enforcement mechanism to the current levels of pollution observed. In essence, $\alpha \in [0,1]$ controls the speed with which $P_{check}[i]$ adjusts to the pollution levels observed in the link corresponding to neighbor i.

Figure 3 shows the effect of α on the proliferation of corrupted entries through the network. For this experiment, 250 malicious nodes corrupt entries with a probability $P_{spam} = 0.50$. It is important to point out that the parameter α has no effect on the final amount of spam in the network. It only affects the speed at which the level of spam stabilizes. The level to which spam converges is dictated by the number of malicious nodes and the rate at which they place spam in the network.

4.2 Varying Spamming Rates

Whenever a node exchanges entries with a malicious node, the malicious node has the opportunity to send spam. The amount of spam included in the collection of entries sent by the malicious node is regulated by the parameter P_{spam}, which is the probability that an entry sent by a spammer is corrupted.

Figure 4 shows the amount of corrupted content in the network over time, after the appearance of spammers. The results of six independent experiments, each with a different spamming rate, are shown. In each experiment, 250 spammers (10% of the network) at random positions start generating spam at round 50. Prior to that moment,

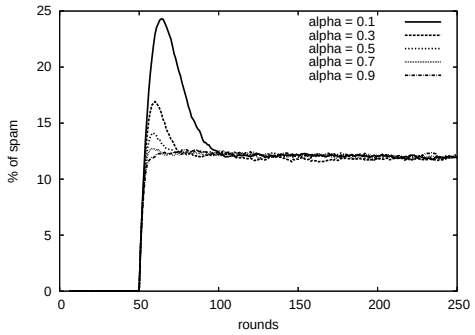

Figure 3. Percentage of corrupted entries in the network over time for different values of α.

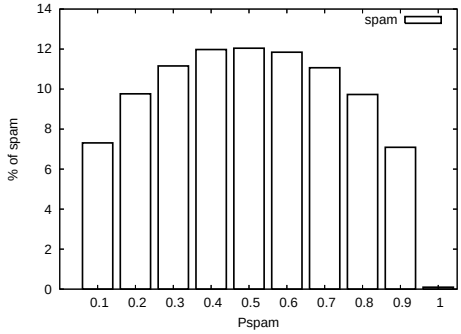

Figure 5. Average number of corrupted entries in the network for different spamming rates.

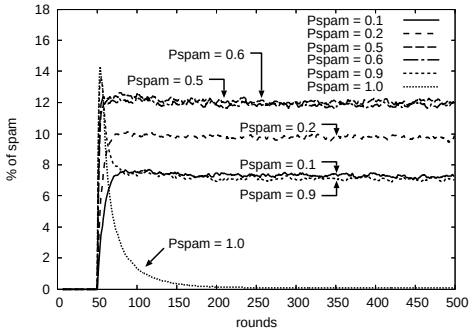

Figure 4. Amount of corrupted entries in the network over time, for different spamming rates.

all nodes in the network are checking the traffic in each of their links at the minimum level of $P_{check_{min}} = 0.05$. The appearance of spammers is followed by a fast increase in the amount of spam in the network. However, in all cases the amount of spam stabilizes after the initial period of growth. An important observation is that the value to which the amount of spam converges is not proportional to the spamming rate of the malicious nodes. In fact, a closer look reveals that spamming at a high rate is actually detrimental to the spammer's ability to place corrupted entries in the network. By measuring the value to which the amount of spam converges (by averaging the last 200 rounds), we can observe its relationship to the spamming rate P_{spam} more clearly. Starting from a low spamming rate, we observe that initial increases of P_{spam} yield positive results for the malicious nodes. However, after reaching the mid-point of $P_{spam} = 0.5$, increasing the spamming rate turns counter productive.

After an initial period of adjustment following the apperance of spammers, well-behaved nodes tune their filters to the actual amounts of spam observed through each of the links to their neighbors. Consequently, neighbors of spammers create a barrier that filters out corrupt messages produced by malicious nodes. The strength of the filtering is proportional to the amount of corrupted entries observed. This explains the inability of spammers to disseminate corrupt data when using a very high spamming rate. In essence, the large amounts of spam produced are being filtered out after a hop or two.

In contrast, a somewhat lower spamming rate results in neighbors lowering their checking, allowing more spam into the network. The key to understanding this behavior lies in equation (1). In the stable state, where nodes have adjusted their filters ($P_{check}[i] = P_i$), the percentage of spam in a well-behaved node's cache is dictated by the value of $P_{check}[i]$, peaking when $P_{check}[i] = 0.5$ and reaching 0 at the extremes ($P_{check}[i] = 0$ and $P_{check}[i] = 1$).

4.3 Workload Caused By Spammers

The previous sections have established that nodes regulate the amount of checks they perform according to the amount of corrupted content they observe. Since the nodes' wireless transceivers have a limited range, nodes that are in physical proximity of spammers are more likely to receive corrupt entries and, therefore, execute more integrity checks. This causes some nodes to have a higher workload than others with regard to data integrity enforcement.

The amount of checking done by a node is dictated by the values of P_{check} on its incoming links. Using the average value of P_{check} per node as a metric, Figure 6 shows the imbalance in the workload placed on the nodes in the network. While the majority of nodes has a low average P_{check} (with a high number doing only the minimum amount of checking, $P_{check_{min}} = 0.05$), a considerable

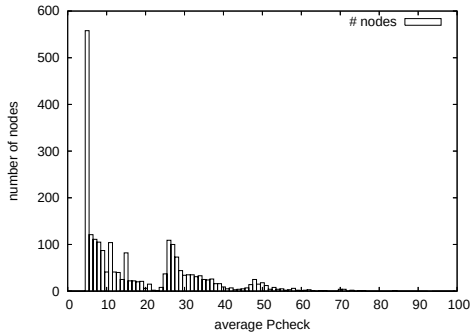

Figure 6. Histogram of average values of P_{check} for every node in the network (malicious nodes excluded). 250 malicious nodes insert corrupted entries with $P_{spam} = 0.90$.

number checks more than a quarter of their incoming traffic. Even a few nodes check up to 50% or more of the entries they receive. These nodes are directly affected by having malicious nodes as neighbors. Even though their work prevents malicious nodes from disseminating corrupted entries, the malicious nodes succeed at disrupting the network by wearing out their neighbors by increasing their workload. For this reason, it is not enough to monitor and reduce the amount of corrupted entries, but also to take active measures towards isolating misbehaving nodes.

5 Detecting Malicious Behavior

Identifying a neighbor as a spammer boils down to being able to differentiate between a node that actively corrupts entries and one that simply forwards the corrupted content received from somebody else. Since nodes do not check 100% of the entries they receive from a neighbor, they can not assume that receiving spam from their neighbor makes the neighbor a spammer. In fact, forwarding some spam is a perfectly valid situation in our network. In a similar way, the number of checked entries by itself is not enough to determine if a neighbor is a spammer or not. In fact, nodes doing minimal checking will forward very few checked entries.

5.1 Detection Mechanism

The key to detecting a spammer lies not in the amount of spam or the number of checked entries received, but in the relationship between these two values. As explained previously in Section 3.2, if a node is behaving properly, there should be a balance between the amount of spam and the number of checked entries it sends. With this in mind, a node can monitor the behavior of its neighbors by individually tracking the difference between the amount of spam received and the number of checked entries for each neighbor.

In a similar way as keeping track of the value of P_{check} for every neighbor, a node Q can keep track of the number of checked entries it receives from each neighbor i. Node Q does not do any additional checking, it just counts the number of entries flagged as `checked`. After every exchange with neighbor i, node Q updates its estimate of the number of checked items received from i:

$$checked[i]_{t+1} = (1-\alpha)checked[i]_t + \alpha fractionChecked$$

The fraction of checked entries for the next round ($checked[i]_{t+1}$) is updated as a weighted sum of its previous value ($checked[i]_t$) and the percentage of checked entries found ($fractionChecked$).

The spammer detection mechanism is based on monitoring the difference between $P_{check}[i]$ (which is an approximation of the amount of spam received from neighbor i) and $checked[i]$ (which reflects the fraction of checked entries expected from neighbor i). An acceptable difference is defined as the parameter δ. In every round, node Q:

- Calculates $diff = |P_{check}[i] - checked[i]|$
- If $diff \leq \delta$, neighbor i is behaving properly. Otherwise, i is a suspected spammer.

Figure 7 shows the result of applying the proposed detection method for different values of δ, with P_{spam} set to 0.5 (the value that allows for the most spam to be placed in the network, as shown in Figure 5). The graphs depict the number of spammers detected, as well as the number of false negatives (spammers that avoid detection) and positives (well-behaved nodes that are confused as spammers). The results are counted per link, since nodes do separate analyses for each of their neighbors. The threshold δ affects the detection of spammers in the following ways:

- A small δ results in spammers being detected quickly after their appearance, but could also lead to well-behaved nodes being mistakenly identified as spammers, i.e., false positives [see Figure 7(a)].
- With larger values of δ, spammers can operate for a longer period of time before being identified. However, a larger δ prevents well-behaved nodes from being confused with spammers, i.e., false negatives.

The initial period after the appearance of spammers is characterized by well-behaved nodes struggling to adjust their values of P_{check} for every neighbor to the appropriate level. As a result, good nodes may store and forward more corrupted entries than expected and could easily be confused with spammers if the threshold δ is too restrictive.

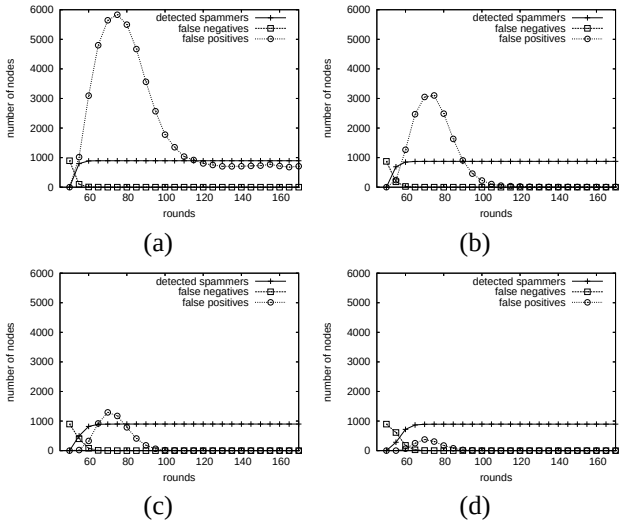

Figure 7. Results of applying the detection algorithm over time for different values of δ: a) δ = 0.05, b) δ = 0.10, c) δ = 0.15 and d) δ = 0.20.

5.2 Flying under the Radar

We say that a malicious node is "flying under the radar" if the system consistently detects it as a false negative. As seen previously, after a short initial period where nodes adjust to the presence of spammers, spammers can be accurately discovered given an appropriate value of δ. In order for a malicious node to fly under the radar, it needs to modify its behavior enough to be confused with a well-behaved node. We identify two possible strategies for a malicious node to achieve this:

- Checking entries just as a normal node would do and spamming with probability P_{spam} in the unchecked entries.
- Reducing P_{spam} in the hopes of reducing *diff* enough so that $diff \leq \delta$.

Figure 8 shows the results of the first approach. The experiment records statistical information for *diff* (average value and standard deviation) when the neighbor is a spammer and when the neighbor is a normal node. By executing integrity checks just like a well-behaved node (now `selectItemsToKeep()` is the same for spammers and normal nodes), spammers can lower the value of *diff* [see Figure 8(a)] to the point were some nodes avoid detection [Figure 8(b)]. However, only a very small number of nodes are not discovered as being malicious. Moreover, this does not happen consistently.

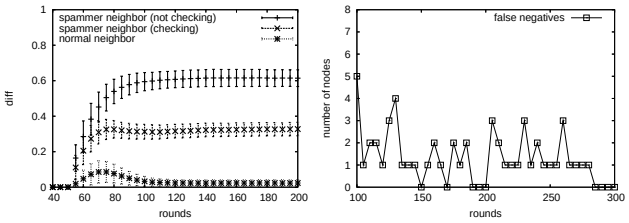

Figure 8. Spammers try to prevent being discovered by executing integrity checks: a) average value of $diff \pm \sigma$ for spammers and normal nodes as neighbors ($P_{spam} = 0.5$, $\delta = 0.20$), b) spammers that avoid detection.

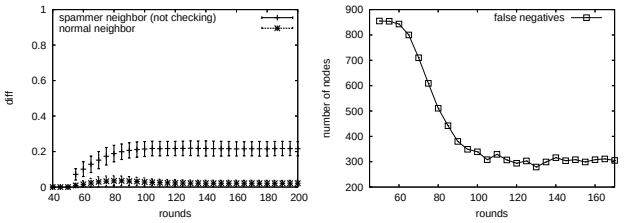

Figure 9. Spammers try to avoid detection by lowering their spamming rate to 0.1.

The second approach is more effective in terms of avoiding discovery. With a spamming rate of 0.1, the value of *diff* for a considerable number of spammers falls below the threshold $\delta = 0.20$, as can be seen in Figure 9(a). As a consequence, many spammers are not discovered as such [see Figure 9(b)]. However, at $P_{spam} = 0.1$ the amount of spam they can place in the network is low. And after discovered spammers are removed, the amount of spam will decrease even more. In addition, the spammers that are not discovered are not always the same. Therefore, a well-behaved node could identify spammers by taking as a policy to stop communication with another node if the node qualifies as a spammer x number of times over a period of time.

6 Conclusions

In this paper, we explored the feasibility of ensuring data integrity in very large ad hoc networks by means of a simple and inexpensive solution based on probabilistic integrity checks and traffic analysis. Our approach has proven to be effective in containing the spread of corrupted content without imposing a burden for all nodes in the network. In fact, the workload placed on nodes is proportional to

the amount of corrupted content they receive, affecting mostly nodes in the neighborhood of spammers. By keeping track of their incoming traffic, nodes affected by malicious neighbors can discover and isolate those misbehaving nodes. We also explored to what extent a malicious node can avoid detection. Although possible, avoiding detection implies a considerable decrease in the level of malicious behavior, lessening the impact of spammers in the network.

References

[1] S. Bansal and M. Baker. Observation-based cooperation enforcement in ad hoc networks. Technical Report, 2003.

[2] S. Buchegger and J.-Y. Le Boudec. Self-policing mobile ad hoc networks by reputation systems. *IEEE Communications Magazine*, 43(7):101–107, July 2005.

[3] L. Buttyan and J.-P. Hubaux. Enforcing service availability in mobile ad-hoc wans. In *MobiHoc '00: Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, pages 87–96, Piscataway, NJ, USA, 2000. IEEE Press.

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

[5] D. Gavidia, S. Voulgaris, and M. van Steen. A gossip-based distributed news service for wireless mesh networks. In *Proceedings 3rd IEEE Conference on Wireless On demand Network Systems and Services (WONS)*, Les Menuires, France, January 2006.

[6] J. Liang, N. Naoumov, and K. Ross. The index poisoning attack in p2p file-sharing systems. In *Proceedings of IEEE Infocom 2006*, Barcelona, Spain, 2006.

[7] N. Naoumov and K. Ross. Exploiting p2p systems for ddos attacks. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 47, New York, NY, USA, 2006. ACM Press.

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