Local detection of forwarding faults in Wireless Community Networks

Ester López and Leandro Navarro

Department of Computer Architecture
Universitat Politècnica de Catalunya
Barcelona, Spain
{esterl,leandro}@ac.upc.edu

Abstract. In our connected society there is a critical reliance on the continuous operation of computer networks. Localizing compromised, misconfigured or simply faulty routers that break this connectedness is imperative. In this paper we focus on wireless community networks, which are specially vulnerable because of their characteristics, and yet none of the previous work is adapted to work on them. We propose a mechanism for the detection of incorrect routers just based on the validation of local data traffic using sketches, without making any assumption on the routing protocol or knowing a flow's path. This is a purely distributed solution without the need for a central authority nor clock synchrony. An evaluation based on real traffic shows how the proposed method is able to detect forwarding faults while the cost in terms of the network overhead is kept within reasonable bounds.

Keywords: fault-localization, wireless community network, routing robustness

1 Introduction

Wireless Community Networks (WCN) are open IP-based network infrastructures that have grown organically and collaboratively reaching up to thousands of nodes by using inexpensive hardware that is owned and managed by each participant. These networks can use one or a combination of several routing protocols to reach scales beyond 25,000 nodes. The choice of routing protocol varies among networks and environments, typically using mesh routing protocols such as BATMAN Adv, BMX6 or OLSR in more dense areas, and OSPF or BGP for the interconnection of these mesh zones across longer distances or when the network becomes too large for a single mesh routing protocol. With this diversity of routing protocols, even on a single network, monitoring and fault location cannot rely on routing-specific information.

Network monitoring and management is usually done by non-dedicated administrators, that combined with the use of cheap equipment and amateur outdoor setups result in infrastructures more prone to failure than other commercial wireless networks. Therefore, fault localization mechanisms can be extremely

useful, as they will help in automating the detection of anomalies and raise notifications or perform rapid preventive measures to keep the network stable.

Incorrect routers can result from anomalies such as degradation due to software or hardware aging, misconfiguration error, uncooperative users throttling or discriminating traffic, even malicious traffic dropping or alteration. Our approach is to detect these arbitrary, also know as Byzantine, failures in routers based on traffic statistics collected in each router by a monitoring daemon that considers the routers limited computational and communication capabilities.

Our main contribution is a fault localization protocol that relies only in data traffic. More specifically, we propose:

- A traffic validation mechanism based on sketches that does not require clock synchronization.
- An extension of the χ protocol to handle inaccurate traffic validation functions.

The rest of the paper is organized as follows. We start by describing related previous work in this area and the problem in hand. Then, in section 3, we present our solution based on computing summaries as sketches of local data traffic and distributed detection to decide on incorrect nodes. Section 4 presents the validation of the mechanism based on real data traffic, followed by a discussion in section 5 and conclusions in section 6.

2 Background and related work

Connectivity in networks is the result of a cooperative work: to reach a far away node, we rely on the fact that every node in the path will forward the packet to the next hop. However, a node may exhibit a faulty behavior for many reasons: misconfiguration, hardware failures or tampering, resulting on partial losses of connectivity. WCN are specially vulnerable because they operate over cheap equipment that is managed by non-expert users and is open to everyone.

Once the connectivity is lost, it is difficult to find and determine which node is the source of the problem. The goal of this paper is to provide a mechanism that monitors network nodes to help administrators on this task. The problem is not new and has been widely studied previously as we will present next; but a solution for detecting faulty nodes without requiring clock synchronization, control plane information and in a distributed fashion still had to be proposed.

In the literature, the problem of routing robustness is usually faced from a resilience point of view or from a detection perspective. Solutions to achieve resilience usually modify the network stack to use some kind of redundancy to ensure that traffic is delivered. A classical example is *robust flooding* [16], which uses network redundancy and buffers to ensure packets are delivered. Another solution is ACR [21], which proposes a routing protocol that provides multiple paths, so that when a path does not perform as expected, an alternative path can be used. However, given the nature of WCN, which are managed in a distributed manner and are extremely diverse, changing the network stack does

not seem reasonable, and because of this reason, we focus instead on detection-based solutions purely based on the analysis of traffic in the data-plane.

Detection solutions usually focus on one of these three sub-problems [12]:

Traffic Validation How to recognise a misbehaving path, node or link?
Distributed Detection How the information learned through Traffic Validation is distributed among the network?
Response How routing takes this information into account to improve traffic delivery?

On *Traffic Validation*, a simple mechanism to detect a node or area that does not forward traffic properly is **counters**: if the number of packets entering does not match the number of packets leaving (excluding its own traffic), then we can conclude that the node is faulty. This is the approach taken by WATCHERS [5] at node level and AudIt [1] at AS level. Counters can only detect packet dropping, while packet modification and corruption goes undetected.

If we are considering a path, another widely used mechanism is **acknowledgements**. Some solutions are purely end-to-end, where the destination acknowledges every packet [4, 18], or some of the packets [4, 17] and the fault is assigned to the whole path. Others go further and when a failure is detected, a mechanism is triggered to pinpoint the specific faulty link [3, 14]. And others require the involvement of every node in the path [2], to directly detect the faulty link. From a more local perspective, acknowledgements have been also used in 2ACK [9] and NACK [20], where nodes monitor their next hop by requiring acknowledgments from the next-next hop. The main disadvantage of ACK-based solutions is that they require paht information to localize the specific link or node that is failing, but that is not available for distance vector protocols.

In the context of wireless networks, many authors have proposed to make the most of the medium characteristics and determine if a node actually forwards traffic simply by **overhearing** [10, 22]. This solution, though attractive, makes the assumption that nodes do not have several antennas, on different directions and using different channels, a configuration that is quite common in WCN.

Finally, other summary mechanisms more robust than counters have been studied, so that by comparing them a node's behavior can be characterized: **sampling** [7,8], **packet-fingerprinting** [12], packet-reception bitmaps and **homomorphic linear authenticator** [19] and **sketches** [8, 23]. However, most of them require clock synchronization to agree on the time interval to monitor.

On the sub-problem of *distributed detection*, many solutions consider a reliable central entity, which establishes a secure channel with the involved parties and draws conclusions from the given evidence [7, 19, 23] and others flood the evidence through the network [5]; both cases do not scale on a wireless network of thousands of nodes and no central authority. Then, on some solutions only the source uses the learned evidence, such as [2] or [8], but that approach does not fit with the collaborative spirit of a WCN. Finally, Mizrak [12] proposes that each nodes monitors its neighborhood and then maintains network connectivity by disconnecting from faulty nodes.

To conclude, the sub-problem of *Response* is not the main concern of this paper, but there are several solutions that propose to feed the routing protocol with the learnt evidence in the form of metrics [3, 11, 15, 17], whereas others prefer creating a trust or reputation system [10].

Our solution builds upon the solutions proposed on [8, 12, 23]. As Goldberg and Zhang we use sketches for traffic validation, but we propose an alternative way of using sketches so that synchronization is not required. Then, on the distributed detection side, our work is based on the solution proposed by Mizrak et al. [12], since it is better suited for WCN; however it was adapted so that it is capable of adapting to an inaccurate traffic validation mechanism.

3 Local Detection

We propose to perform neighborhood monitoring in each node and locally decide whether that node is behaving properly or not. For traffic validation we use sketches, a data structure that summarizes traffic flows with the properties of providing a fair estimation of the second frequency moment, allowing intersection between traffic flows and linear combinations.Once captured, this information needs to be shared reliably among the nodes involved on the distributed detection making sure that: nodes cannot falsely accuse another node and the reports cannot be modified. In our approach we involve the monitored node to enforce the first requirement and keep the report dissemination local, i.e. within the neighborhood of the monitored node. To avoid report falsification we resort to cryptography: so by signing the reports we ensure they have not been modified.

3.1 Assumptions

For properly locating faulty nodes, the proposed solution assumes that:

- There is a public-key infrastructure. More specifically, there is a mechanism that allows nodes to verify the authenticity of the messages sent by their 2-hop away neighbors. This can be achieved, for instance, using the authentication mechanism of SEMTOR [13].
- We assume that established links are bidirectional and messages are eventually received. This assumption is reasonable since most mesh routing protocols would only consider bidirectional links and links with very low quality are not likely to be used.
- There is a reliable way of discovering the neighborhood of your neighbors. A possible solution is sketched on section 5.
- The traffic validation mechanism is second pre-image resistant, i.e. a faulty node is very unlikely change a packet for another and still produce the same traffic summary. This is further discussed in section 5.
- There is no collusion between faulty nodes, and their behavior is clearly differentiable to that of a proper node. Sybil attacks are also not considered; given that nodes in a WCN usually require to be registered into a database anomalies could easily detected by inspecting it.
- Links are considered stable within the interval of the detection protocol.

3.2 Traffic validation

Traffic validation is the mechanism that determines which traffic information is collected and how that information helps us to determine that a network entity is faulty. Typically, traffic validation is built upon the conservation of the flow principle: traffic going into a network area should be (almost) the same to the traffic leaving it; as long as we do not consider the traffic destined to it and coming from it. We can apply traffic validation with different granularity in terms of the traffic being considered (individual flows or the whole traffic) and in terms of the area being monitored (link, node path, AS...). In our case, we use traffic validation for monitoring the whole traffic going through a node, so our explanations will be focused on this particular case, but a traffic validation function can be applied on any other scenario.

Consider the node being monitored, M, and their neighborhood, Neigh(M), then by conservation of the flow we would expect in ideal conditions:

$$\bigcup_{i \in \text{Neigh}(M)} \mathbb{T}_{i \to M} = \bigcup_{i \in \text{Neigh}(M)} \mathbb{T}_{i \leftarrow M}$$

Where $\mathbb{T}_{i \to M}$ is the traffic from neighbor i to M without considering the traffic destined to M and $\mathbb{T}_{i \leftarrow M}$ is traffic from M to i without considering the traffic coming from M, in both cases from i's perspective. Of course, since networks suffer of packet losses due of congestion, collision, etc. both sides of the equation will not be exactly the same, but approximately.

Sharing all the traffic already sent to study whether a node is behaving properly is terribly expensive, so we require a summary function that still allows us to detect faulty nodes. A simple example of a traffic validation mechanism would be the use of counters: if every node keeps a counter for the traffic coming from and sent to M, those counters can be shared and the sum of the traffic going in should be approximately the traffic leaving M. However, counters can only detect packet drop and packet creation faults, so in our case, we decided to use a more complex summary function: sketches, that allows us to also detect packet modification or corruption faults.

Sketches have been proposed before as a traffic validation mechanism in [8, 23] because they satisfy the following properties: (i) a sketch can be computed online, updating its counters every time a new packet arrives; (ii) in a distributed fashion, where each monitoring node can compute the sketch of the portion of traffic it sees and the global sketch can be computed by combining each local sketch; and finally, (iii) a sketch has relatively low requirements in terms of processing, memory and network overhead. A sketch consists mainly in a matrix of counters that are updated as packets arrive; and as such they can be linearly combined: every neighbor will keep an sketch for the monitored node's incoming and outgoing traffic and then all of them will be combined to obtain the sketch that represents the difference between the global incoming and outgoing traffic:

$$S_{\text{diff}} = \sum_{i \in \text{Neigh}(M)} \mathcal{S}(\mathbb{T}_{i \to M}) - \sum_{i \in \text{Neigh}(M)} \mathcal{S}(\mathbb{T}_{i \leftarrow M})$$

Where $\mathcal{S}(\mathbb{T})$ is the sketch of the traffic flow $\mathbb{T}$. Then, since sketches provide a good estimation for the second frequency moment of the traffic flow it summarizes, we know that:

$$||S_{\text{diff}}||^2 \approx ||\mathbb{T}_{*\to M} - \mathbb{T}_{*\leftarrow M}||^2 \leq ||\mathbb{T}_{*\to M} - \mathbb{T}_{*\leftarrow M}||$$

Ideally, we would like the inequality to be an equality, which happens when the elements of $\mathbb{T}$ are unique. This is the case for IPv4 packets, because of the identification field, but not in IPv6. In the case there are duplicate elements sketched on S_{diff} the second frequency moment will be an overestimate of the difference between incoming and outgoing traffics. We will see in section 4 that this is a rare event, and so sketches can be used effectively for traffic validation.

In any case, there are a couple of considerations in order to use sketches for traffic validation:

- *Input space*: the original space of all possible packets is terribly big ($I = \{2^{8\times 1500}\}$ for Ethernet v2 and bigger with Jumbo Frames). As a result, computing the sketch over this space is very expensive in terms of computation, and therefore we require a space reduction function. In this work we use the last bytes of SHA-256 to reduce the space to a reasonably sized one.
- *Changeless elements*: to be able to compare the sketches produced in two different neighbors of M, the elements sketched need to be the same. That implies that layers below the IP layer must be removed and that the TTL must be adapted by either having the sender reduce it by 2 before sketching the packet or simply setting its value to 0. Also, networks that perform packet fragmentation cannot use sketches for fault localization.

In essence, every node will keep a sketch of the traffic of each neighbor. Whenever a node receives a packet, it will remove its lower layers and adapt the TTL field as required. Then it will generate its SHA-256 hash and keep the last significant bytes as the element to sketch. Finally it will update the sketch of the source neighbor if that packet was not originated from it. Something similar will be done every time a message is sent.

3.3 Traffic Validation without clock synchronization

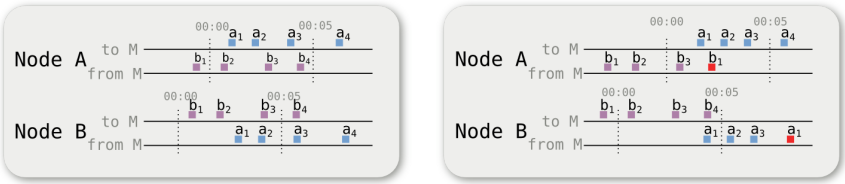
The previously proposed traffic validation mechanism measures the difference between the traffic flows captured by the sketches, but if the neighbors do not agree on the packets that are to be captured, the measured difference will be bigger than expected. Consider, for instance, that node M is connected to node A and B, and the traffic perspective from A and B is as shown in figure 1a. In this case, because of the lack of synchronization, $||S_{\text{diff}}||^2 = |\{a_3, b_1, b_4\}| = 3$, however, M is forwarding packets properly.

To overcome this problem we propose to use the sketches sent on the previous and next interval. Consider $S_{\text{in}}(t)$ the sketch as result of summing every sketch related to the incoming traffic at time interval t and $S_{\text{out}}(t)$ the one related to the outgoing traffic. As we have seen, some packets from $S_{\text{in}}(t)$ may not be on

$S_{\text{out}}(t)$, but on $S_{\text{out}}(t-1)$ or $S_{\text{out}}(t+1)$. Similarly, some of the packets from $S_{\text{out}}(t)$, from a neighbor's perspective may have been sent during interval $t-1$ or $t+1$, so we obtain:

$$\widehat{\text{diff}} = ||S_{\text{in}}(t) - S_{\text{out}}(t)||^2 - S_{\text{in}}(t) \cdot S_{\text{out}}(t-1) - S_{\text{in}}(t) \cdot S_{\text{out}}(t+1) \\ - S_{\text{out}}(t) \cdot S_{\text{in}}(t-1) - S_{\text{out}}(t) \cdot S_{\text{in}}(t+1)$$

Where $S_i \cdot S_j$ refers to sketches i and j inner product.



(a) Correct nodes (b) Faulty node

Fig. 1: Example packets with unsynchronized clocks.

This estimation is reliable as long as M behaves properly, but if it does not, it could replace some packets for others sent previously and avoid detection. Consider for instance the scenario on figure 1b, $\widehat{\text{diff}} = |\{a_2, a_3, b_4, b_3, b_1\}| - |\{b_2\}| - |\{a_1, a_2, a_3\}| - |\{b_2\}| - |\{\}| = 0$. Therefore, we cannot discount every packet in the previous and next sketches, but only the ones that are not duplicated from the current sketch:

$$\begin{aligned}\widehat{\text{diff}} = &||S_{\text{in}}(t) - S_{\text{out}}(t)||^2 \\ &- S_{\text{in}}(t) \cdot S_{\text{out}}(t-1) - S_{\text{in}}(t) \cdot S_{\text{out}}(t+1) \\ &- S_{\text{out}}(t) \cdot S_{\text{in}}(t-1) - S_{\text{out}}(t) \cdot S_{\text{in}}(t+1) \\ &+ S_{\text{in}}(t) \cdot S_{\text{in}}(t-1) + S_{\text{in}}(t) \cdot S_{\text{in}}(t+1) \\ &+ S_{\text{in}}(t) \cdot S_{\text{in}}(t-1) + S_{\text{out}}(t) \cdot S_{\text{out}}(t+1)\end{aligned}$$

In section 4 we will refer to the metric introduced in the previous section as *without intersection* and this one as *with intersection*.

Summary: *We define a traffic validation function based on sketches that provides a reliable estimation of the difference between a node's incoming and outgoing traffic. A sketch computation is reasonably small and the required state on each node is proportional to the number of nodes and the sketch size.*

3.4 Detection protocol

The next piece of the puzzle is determining how to share this information so that it can be compared and the behavior of M can be determined. We propose

to share this information only within M's neighborhood. This strategy allows us to achieve two goals: false accusation can be detected and information is kept local, and, therefore, the protocol's overhead is bounded.

The protocol runs in two rounds. During the first round every neighbor sends M's sketch to M. Then M ensures that the sketches are reasonable and if not, it removes the faulty node from its neighborhood. We call this round *report validation*. On the second round, M forwards all the correct reports to its neighborhood and then the neighbors are capable of determining M's behavior using the sketch second frequency moment. This round is called *node validation*.

Report validation

During the report validation phase, it is M's responsibility to detect every node that is not properly participating in the detection protocol. First, it needs to corroborate that every neighbor participates in the detection protocol, that is, it shares its sketches. And second, it needs to make sure that the sketches sent are consistent.

Periodically, every neighbor i sends to M its traffic report ($S_{i\leftarrow M}$ and $S_{i\rightarrow M}$) signed, so it cannot be falsified. In case that M does not receive a neighbor's report within reasonable time, that neighbor will be detected as faulty. Otherwise, upon the report's reception, M will first verify the report signature, detecting the node as faulty if the message cannot be authenticated. Then it will compare the reported number of packets with ones actually received:

$$\widehat{\text{received}} = ||S_{M\leftarrow i}||^2$$

$$\widehat{\text{sent}} = \widehat{\text{received}} + ||S_{M\leftarrow i} - S_{i\rightarrow M}||^2$$

$$P(\widehat{\text{received}}, \widehat{\text{sent}}, P_{\text{loss}}) \overset{?}{<} \text{threshold}_1$$

To estimate the number of sent packets we use the difference between sketches instead of directly $S_{i\rightarrow M}$ because the sketch could represent a traffic flow with the same number of packets, but different content, and therefore accusing M of packet modification.

To determine if a report is not consistent we will check the probability of the estimated parameters, $P(\widehat{\text{received}}, \widehat{\text{sent}}, P_{\text{loss}})$, considering a binomial distribution: how likely is M to have received $\widehat{\text{received}}$ packets or less if neighbor i has sent $\widehat{\text{sent}}$ packets and the loss probability is P_{loss}? P_{loss} considers not only the link loss probability, but also the probability of losing a packet due to other reasons like congestion or corruption. We use a binomial distribution instead of directly comparing the estimated proportion of packet loss with a threshold because the traffic is not constant, and it is not the same to lose a packet when a single packet was sent than losing thousands when thousands were sent.

Since links are assumed to eventually deliver messages, it is assured that M will eventually receive a neighbor's report if tried enough. If a node does not send the report, then it is properly detected as faulty. On the other hand, a neighbor could send a false report declaring packets that were not present in the original traffic flow. Given that M compares its local perspective of the traffic

flow with its neighbors perspective, the corruption of the report is kept within reasonable bounds as will be shown in section 4.

Every time M detects a node N as faulty, this node is removed from M's neighborhood during W_1 periods, $W_1 > W$, i.e. M will disconnect from N This is needed, to ensure that M does not simply accuse its neighbors of being faulty and uses its own sketch versions to avoid detection.

Summary: *The report validation phase ensures that every neighbor participates properly on the detection protocol or it will be detected as protocol faulty with high probability.*

Node validation

On the node validation phase, M will share with its neighbors the reports collected from non-faulty nodes, its own reports from those that were faulty and the categorization of its neighbors as faulty or not. Once received the traffic reports, each neighbor N will check the following:

- There is a traffic report for every neighbor of M.
- Traffic reports from non-faulty neighbors are properly signed.
- There isn't a traffic report for a neighbor that was accused as faulty in the last W_1 intervals.

If any of these points is not satisfied, N will detect M as faulty, and will act as the response mechanism dictates. Otherwise, N estimate the packet loss ratio as follows:

$$\widehat{\text{loss}(t)} = \frac{\widehat{diff(t)}}{\text{num\_pkts}(t)}$$

$$\widehat{\text{loss}_W} = \sum_{t=0}^{-W} \widehat{\text{loss}(t)} \cdot \text{num\_pkts}(t) / \sum_{t=0}^{-W} \text{num\_pkts}(t) \stackrel{?}{<} \text{threshold}_2$$

Where num_pkts is the estimated number of packets for each interval, num_pkts = $max(||S_{in}||^2, ||S_{out}||^2, 1)$. To determine if the node is behaving properly, $loss_W$ is compared with threshold_2, calculated using the bounds of misbehavior of non-faulty nodes (α) and faulty nodes (β) [8]: $\text{threshold}_2 = \frac{2 \cdot \alpha\beta}{\alpha+\beta}$.

Our proposal for the metric averages the node's packet loss estimation during W intervals because traffic in WCN is not stable (some intervals will have a large amount of traffic while others will not). In section 4 we will show the effect of averaging or not the packet loss estimation.

Summary: *The node validation phase ensures the detection of nodes that do not properly forward traffic with high probability.*

4 Evaluation

Our evaluation is based on a traffic capture from a wireless node member of the qMp Sants WCN [6]. Its main characteristics are summarized in table 4. At the

moment of the capture, the node (M) was connected to three other nodes, two of them, A and B, had really good link quality (loss below 10^{-6}) and another, node C, with bad link quality (50% packet loss measured with MTR). Because qMp Sants is an ad-hoc network, the routing protocols will rarely use the link with low quality (less than 0.1% of the traffic is routed through C). The traffic is quite variable in terms of packets per second. The α of this traffic capture is 0.058 (given by the periods with most traffic from C); therefore, for a β of 0.1, threshold_2 is 0.73. On each experiment, we replay the traffic capture, simulating the loss on each link and compute the expected sketches on each of the nodes with the given experiment parameters.

Number of packets	50000
Duration	842.2 seconds
Average packets per second	59 p/s
Average bytes per second	45746.6 B/s
Duplicated packets	60

4.1 Experiment 1: Report Validation

The goal of the first experiment is to study the effect of false accusations. On every interval, node M will receive a report from every neighbor and will compare it to its own. Based on threshold_1 it will determine whether that report is coherent or not and the neighbor will be considered protocol-faulty (or not) accordingly. Choosing the proper threshold is a matter of balancing two different interests: minimize the number of false positives on the report validation; and minimize the effect of malicious reports, i.e. minimize the false positives on the node validation phase. This experiment proves that if the proper threshold is chosen, we can achieve reasonably good results on both interests.

In this experiment we use sketches of 32 by 32 counters, and an interval duration of 5 seconds. Every neighbor computes its sketch and then corrupts it by adding random packets to the point just before being detected. Then, with all the corrupted sketches loss_W is computed and checked whether it would be accused as faulty or not based on a threshold of 0.073 and a window size of 10.

Figure 2 shows the percentage of false positives on both phases depending on the value of threshold_1 with 200 simulations:

- M shows the percentage of time it is detected as faulty on the node validation phase (even though it is behaving properly) because of false traffic reports from its neighbors that have not been detected. As expected, as threshold_1 is higher, reports cannot be modified as much without detection and therefore M is less likely to be falsely accused.

– *Neighbors* shows the percentage of time a neighbor of M is accused of report falsification when it is not. In contrast, now a higher threshold will cause more false positives.
– C shows the specific case of neighbor C being detected as protocol faulty. Because C is more likely to loose packets, so it is its probability of being accused of falsely reporting traffic.

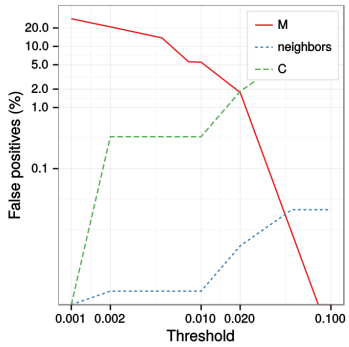


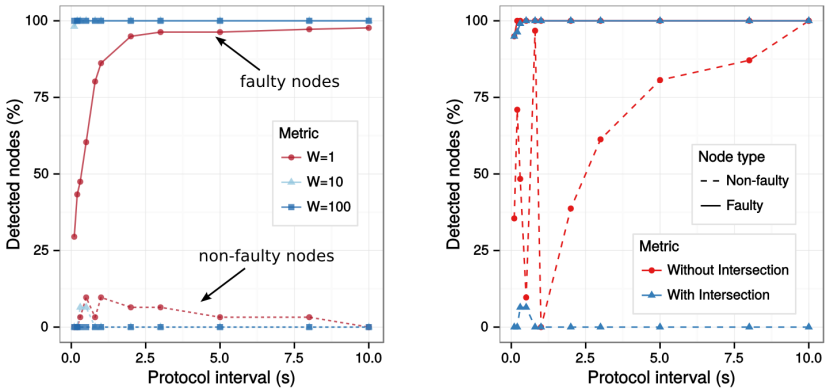
Fig. 2: False positives depending on the threshold

For this network, a reasonable threshold could be 0.02 that will produce less than 2% false positives in both cases.

4.2 Experiment 2: Node Validation

Our second experiment validates sketches as mechanism for traffic validation. Each scenario replays the traffic capture simulating the measured link properties (link quality and delay), but this time the monitored node will either behave properly or drop 10% of the packets. The detection protocol is then simulated using different interval durations and the node being monitored is classified as faulty or non-faulty on every interval considering a threshold of 0.073.

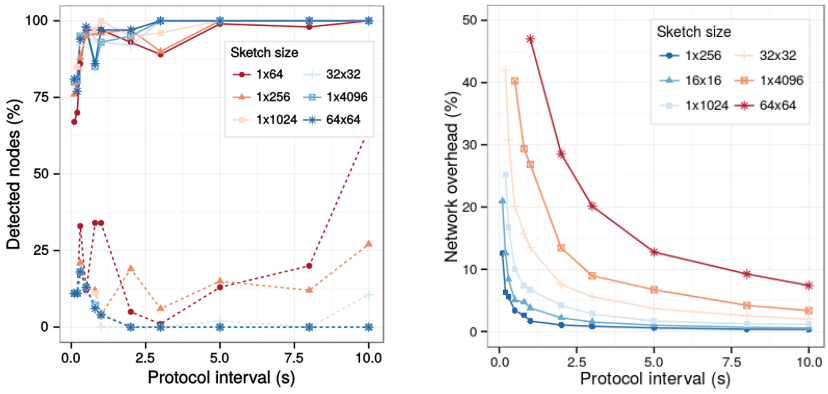
Synchronized scenario First we will consider that neighbors have synchronized clocks, and so the sketches they share are perfectly aligned. In figure 3a we see the proportion of times faulty and non-faulty behaviors are classified as faulty. For very small intervals, it is difficult to decide whether a node is faulty or not, because there may be no packets or very few to determine the node's behavior. As the interval becomes larger, detection becomes more accurate. Longer intervals will imply lower overhead and longer detection times. Another option is to average the estimation during W intervals. On this scenario we have found no relevant difference on the results when using sketches of different sizes.



(a) With different window lengths (b) Using or not intersection

Fig. 3: Proportion of detected nodes

Not synchronized scenario Then, to measure the effect of clock synchronization, we have advanced the clock of $A$and delayed the clock of B a proportion of the interval. Figure 3b shows the difference between the metric *with intersection* and *without* when the skew is 0.1 times the interval, sketch size is 64 by 64 and a window size of 10. However, because we are now using the sketches for predicting larger numbers (the difference between 2 sketches is always expected to be 0, whereas the intersection between 2 misaligned sketches is not), the size of the sketch matters on the precision of the prediction (see figure 4a): it has to be large enough to avoid false positives.



(a) Proportion of detected nodes (b) Network overhead

Fig. 4: Effects of the sketch size

Network overhead Our last concern was to measure the network overhead consumed by sharing the sketches. To measure the overhead, we have to consider that every neighbor will first send 2 sketches to M. Later M will send back to each neighbor $(N-1)\cdot 2$ sketches (where N is the number of neighbors) if we consider that they are shared via unicast. In our scenario that would be 18 sketches shared on every interval. Figure 4b shows the relative overhead as compared with the traffic going through M. We can see that the cost is unreasonable for smaller intervals, but for intervals higher than 5 seconds the overhead is below the 5% when sketches smaller than 32 by 32 are used.

5 Discussion

In this section we address the main concerns introduced along the paper.

5.1 Reliable neighbor's neighborhood discovery

One of the assumptions is a reliable mechanism to discover a neighbor's neighborhood. To prove its necessity let's go through an example: imagine nodes A, B and C connected and monitoring M. To make the example simpler, let's assume that only A sends traffic towards M which is routed through either B or C. On the process of discovering M's neighbors, M could tell A and B that its neighbors are $\{A, B\}$ and to C that they are $\{A, C\}$. Then, M is free to duplicate every packet from A through both B and C violating the conservation of the flow principle without being detected, because A, B and C do not agree on M neighborhood. A simple solution could be to have M announce its neighborhood to a reliable node (e.g. in Guifi.net that could be its graph server) and then every neighbor will know it from the reliable node, obtaining a unified view.

5.2 Second pre-image attacks

Along the paper we have assumed that sketches are second pre-image resistant, i.e. is very unlikely that M finds an alternative packet that produces the same effect on the sketch; otherwise, M could forward that alternative packet instead of the original without being detected. The likeliness of finding an alternative packet is determined by the number of bits used from the SHA256 function and the size of the sketch ($n \times m$). In the case of taking b bits from the SHA256 function, we can obtain up to 2^b different values. Then, this obtained number will update by incrementing by 1 the m counters selected, giving up to n^m possible combinations.

For example, in a 32 by 32 sketch using the last 32 bits of the packet's hash, we are limited by the 32 bits that gives us around $4.29 * 10^9$ possible combinations. If the smallest size of a packet is 20 Bytes (length of the IPv4 header), we will require 85 GBs to store all the possible combinations, 8.5 GBs to have a 10% probability of finding an alternative packet. If instead of using storage, the node tries to compute an alternative, given that a Core 2 processor typically

computes 111 MioB/s [1], it will need 25 minutes to compute as many SHA256 values, but given an interval of only 5 seconds, it could only go through 0.3% of the possibilities. If we consider that a faulty node has the same capabilities of typical community wireless devices; then, its memory space is in the order of MBs and the computing capabilities are below the mentioned ones[2], and therefore a sketch of such characteristics would be second pre-image resistant.

5.3 Limitations

Because of the characteristics of the Traffic Validation mechanism, the proposed solution does not consider neither multicast traffic nor traffic that is fragmented on M. In any case, none of these situations were present in the traffic capture since both situations are very unlikely in the context of WCN. Another limitation is that it is not capable of detecting misrouting faults; however, without any additional information about the traffic path, a node cannot determine which should be M's next hop.

6 Conclusions

This paper proposes a distributed mechanism based on sketches to detect forwarding faults in WCN that works even under the assumption of neighbors without synchronized clocks and control plane information. The proposed solution keeps the information within the monitored node neighborhood, achieving a two-fold benefit from this: detecting false accusations and reducing the network overhead. We have also evaluated the accuracy of the solution and shown that it performs remarkably with a reasonable cost in terms of network overhead. Finally, we have also discussed its limitations and assumptions, as well as provided some guidance on how to choose its parameters properly.

In the future we plan to study in more detail the requirements in terms of CPU and memory of the proposed protocol and study its performance in a testbed.

Acknowledgement

This work is supported by European FP7 programme, FIRE Initiative projects "Community Networks Testbed for the Future Internet" (CONFINE), FP7-288535 and the Spanish government contract TIN2013-47245-C2-1-R.

References

1. Argyraki, K., Maniatis, P., Irzak, O., Ashish, S., Shenker, S.: Loss and delay accountability for the internet. In: IEEE ICNP (Oct 2007)

[1] http://www.cryptopp.com/benchmarks.html
[2] http://wiki.openwrt.org/inbox/benchmark.openssl

2. Avramopoulos, l., Kobayashi, H., Wang, R., Krishnamurthy, A.: Highly secure and efficient routing. In: IEEE INFOCOM (2004)
3. Awerbuch, B., Curtmola, R., Holmer, D., Nita-Rotaru, C., Rubens, H.: ODSBR: An on-demand secure Byzantine resilient routing protocol for wireless ad hoc networks. ACM Transactions on Information and System Security 10 (2008)
4. Barak, B., Goldberg, S., Xiao, D.: Protocols and Lower Bounds for Failure Localization in the Internet. Lecture Notes in Computer Science (2008)
5. Bradley, K., Cheung, S., Puketza, N., Mukherjee, B., Olsson, R.: Detecting disruptive routers: a distributed network monitoring approach. IEEE Network 12 (1998)
6. Cerdà-Alabern, L., Neumann, A., Escrich, P.: Experimental evaluation of a wireless community mesh network. In: ACM international conference on Modeling, analysis & simulation of wireless and mobile systems (2013)
7. Desai, V., Natarajan, S., Wolf, T.: Packet forwarding misbehavior detection in next-generation networks. IEEE ICC (2012)
8. Goldberg, S., Xiao, D., Tromer, E., Barak, B., Rexford, J.: Path-quality monitoring in the presence of adversaries. ACM SIGMETRICS Performance Evaluation Review (Jun 2008)
9. Liu, K., Deng, J., Varshney, P., Balakrishnan, K.: An Acknowledgment-Based Approach for the Detection of Routing Misbehavior in MANETs. IEEE Transactions on Mobile Computing (2007)
10. Marti, S., Giuli, T.J., Lai, K., Baker, M.: Mitigating routing misbehavior in mobile ad hoc networks. In: MobiCom. ACM Press, New York, New York, USA (2000)
11. Matam, R., Tripathy, S.: AFC: An Effective Metric for Reliable Routing in Wireless Mesh Networks. IEEE TrustCom (2013)
12. Mizrak, A., Savage, S., Marzullo, K.: Detecting Malicious Packet Losses. IEEE Transactions on Parallel and Distributed Systems (Feb 2009)
13. Neumann, A., Braem, B., Cerda-Alabern, L., Escrich, P., Barz, C., Kirchhoff, J., Niewiejska, J., Rogge, H.: D4.3 experimental research on testbeds for community networks (2014)
14. Padmanabhan, V.N., Simon, D.R.: Secure traceroute to detect faulty or malicious routing. ACM SIGCOMM Computer Communication Review (2003)
15. Paris, S., Nita-Rotaru, C., Martignon, F., Capone, A.: Efw: A cross-layer metric for reliable routing in wireless mesh networks with selfish participants. In: IEEE INFOCOM (2011)
16. Perlman, R.: Network layer protocols with byzantine robustness. Ph.D. thesis, Massachusetts Institute of Technology (1988)
17. Proto, F.S., Detti, A., Pisa, C., Bianchi, G.: A Framework for Packet-Droppers Mitigation in OLSR Wireless Community Networks. In: IEEE ICC (2011)
18. Rogers, M., Bhatti, S.: A lightweight mechanism for dependable communication in untrusted networks. In: IEEE DSN (2007)
19. Shu, T., Krunz, M.: Detection of malicious packet dropping in wireless ad hoc networks based on privacy-preserving public auditing. ACM WISEC (2012)
20. Sun, H.M., Chen, C.H., Ku, Y.F.: A novel acknowledgment-based approach against collude attacks in MANET. Expert Systems with Applications (2012)
21. Wendlandt, D., Avramopoulos, I., Andersen, D., Rexford, J.: Don't secure routing protocols, secure data delivery. In: ACM Hotnets (2006)
22. Yang, S., Vasudevan, S., Kurose, J.: Witness-based detection of forwarding misbehaviors in wireless networks. In: IEEE WiMesh (2010)
23. Zhang, X., Lan, C., Perrig, A.: Secure and scalable fault localization under dynamic traffic patterns. In: IEEE Symposium on Security and Privacy (2012)