Coordinated Detection of Forwarding Faults in Wireless Community Networks

Ester López[a], Leandro Navarro[a]

[a] *Universitat Politècnica de Catalunya*

Abstract

Wireless Community Networks (WCN) are crowdsourced networks where equipment is contributed and managed by members from a community. WCN have three intrinsic characteristics that make forwarding faults more likely: inexpensive equipment, non-expert administration and openness. These characteristics hinder the robustness of network connectivity. We present KDet, a decentralized protocol for the detection of forwarding faults by establishing overlapping logical boundaries that monitor the behavior of the routers within them. KDet is designed to be collusion resistant, ensuring that compromised routers cannot cover for others to avoid detection. Another important characteristic of KDet is that it does not rely on path information: monitoring nodes do not have to know the complete path a packet follows, just the previous and next hop. As a result, KDet can be deployed as an independent daemon without imposing any change in the network, and it will bring improved network robustness. Results from theoretical analysis and simulation show the correctness of the algorithm, its accuracy in detecting forwarding faults, and a comparison in terms of cost and advantages over previous work, that confirms its practical feasibility in WCN.

Keywords: Wireless Community Networks, Decentralized detection, Collusion, False accusation

1. Introduction

Internet protocols were not designed with resilience and security in mind but, as connectivity becomes an essential part of our everyday life, the fragility of networks has become a key challenge.

Wireless Community Networks (WCN) or crowdsourced networks are grassroots initiatives addressing connectivity gaps by providing an alternative development model for IP-networks, where equipment is owned, pooled and managed by members from the community [1, 2]. Typical scenarios can be long-running networks in remote or underserved locations [3, 4, 5], mass public events with large audiences, disaster areas or emergency situations [6]. Representative examples[1] are Guifi.net in Spain, Freifunk (FF) in Germany, Sarantaporo.gr in Greece, Ninux.org in Italy, FunkFeuer (0xFF) in Austria, NYC mesh in USA, and Zenzeleni Networks in South Africa. All of them with thousands of links, mostly wireless but integrating optical fibre. In these networks, participants can expand the network with new routers and links, but the incorrect behaviour of any can cause forwarding faults.

The goal of this paper is to provide a solution to detect traffic forwarding faults in such networks as they are especially vulnerable to disrupted connectivity due to: a) Low-end equipment, which may result for instance in routing tables that do not fit in memory, or experimental software, more likely affected by memory leaks [7]. b) Distributed administration are not always handled by dedicated experts, which may result in misconfiguration errors [8]. c) Open to everyone, which implies being also open to not careful people or with malicious intentions [9].

As an example, in Guifi.net, users experience from time to time what is known as a "pollastre de rutes" (from Catalan *a mess of routes*), which means that the routing tables become corrupted and connectivity is broken [10, 11]. The typical cause is a faulty routing table due to memory or hardware issues, software bugs, misconfiguration, etc. In such occasions, usually an experienced user logs into the involved nodes hopping from one

Email addresses: esterl@ac.upc.edu (Ester López), leandro@ac.upc.edu (Leandro Navarro)

[1] Guifi.net: http://guifi.net, FF: http://freifunk.net, Sarantaporo: http://sarantaporo.gr, 0xFF: http://www.funkfeuer.at, NYC mesh: https://nycmesh.net, Zenzeleni: http://zenzeleni.net

Please cite this article as: E. López, L. Navarro, Coordinated Detection of Forwarding Faults in Wireless Community Networks, Journal of Network and Computer Applications (2018).

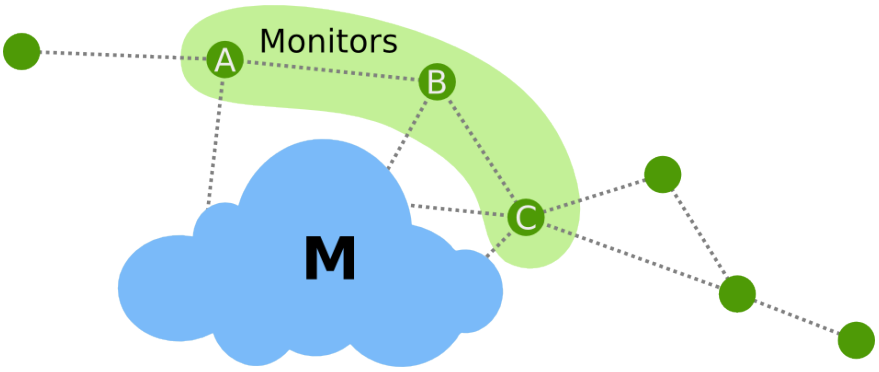


Figure 1: Example network

node to the next using a layer 2 version of ssh, and checking that everything is working properly, until the failure is localized and fixed. Evidently, such mechanism is slow and requires somebody with quite a lot of experience to be able to find and fix the fault. Monitoring tools that automate the process of detecting failures and localizing them would be extremely practical for WCN's users, simplifying the process of repairing connectivity, and as such, impacting considerably their quality of service.

Consider the network in Figure 1, where several monitors (A, B and C) observe a network area, M, to determine if it is properly forwarding traffic. We can divide the problem of forwarding fault detection into four sub-problems:

Traffic Summarization First of all, the monitors need to keep track of the traffic sent and received to M, so they keep a summary with the relevant information about it.

Summary Dissemination and Detection Then, the traffic summaries need to be shared between the monitors, so that at least one network entity has all the relevant summaries and can evaluate the behavior of M.

Traffic Validation is the mechanism that determines the behavior of M using the traffic summaries previously captured. Usually, traffic validation will be based on the conservation of the flow principle: traffic entering a network area must be approximately equal to the traffic leaving that network area, not considering the traffic that is destined or originated from that network area [12].

Response Finally, if we detect any faulty behavior, we can react so that faulty nodes do not disrupt the network connectivity. This reaction should be adapted to the specific network, as proposed in [13].

This paper proposes a solution for the second sub-problem, summary dissemination and detection: KDet (k-Detection), capable of handling false accusation and collusion of up to k nodes, and that doesn't require full path knowledge, so that it is compatible with both distance-vector and link-state routing protocols. We propose two different ways of implementing KDet and compare the memory and network overhead of each implementation.

The rest of the paper is structured as follows: Section 2 introduces the problem of forwarding fault detection and the state of the art solutions to each of its sub-problems; Section 3 describes the system model, with its requirements and assumptions. Section 4 introduces our solution, KDet, that is validated and analyzed in Sections 5 and 6 respectively. We discuss the implementation and limitations of KDet in Section 8 and how it works in some practical scenarios and finally, we conclude in Section 9.

2. Background and related work

As explained in the introduction, forwarding fault detection can be divided into four sub-problems. The first focuses on summarizing the traffic that is being monitored. Possible traffic summary functions are counters [12, 14], packet fingerprinting [15], sampling [16] or sketches [17].

Then comes summary dissemination and detection, the focus of this paper. We can consider two different approaches, global or local, depending on whether the detection results are shared or not (only a subset of those in

the traffic path learn about the forwarding misbehavior, typically just the source). Examples of local detection are Stealth probing [18], Secure sketch PQM [16], Symmetric and Asymmetric Secure Sampling [16] and Faultprints [19]. However, on the problem at hand we want to know globally that a node fails, so that it can be fixed for everyone, like in AudIt [14], WATCHERS [12], DynaFL [20], χ [15], Π_2 and Π_{k+2} [21] and Chao et al.'s work [22]. Among those, AudIt keeps a summary for each traffic flow (pairs of source and destination), whereas the others aggregate the traffic (by neighbor for instance) to reduce the number of summaries kept and, therefore, the memory and network overhead requirements, as in the first case the number of summaries grows with $O(n^2)$. KDet keeps aggregated summaries to keep memory and network overhead reasonable, but because it considers aggregated traffic and global detection it has to face two challenges assuming that nodes can be malicious: **false accusation**, i.e. if a node emits a false report about a neighbor, the neighbor should not be detected as faulty; and **collusion**, that is, a node should not be able to avoid detection by colluding with one of its neighbors[2]. However, of the solutions mentioned before only Π_2 and Π_{k+2} considers and faces solution, but it does so using the packet's path information, which is not available in WCNs.

Most solutions proposed in the literature for solving the traffic validation sub-problem are based on the conservation of the flow principle [12, 20, 23]. But there are other solutions based on more complex statistical models: Mizrak et al. [15] propose a validation function based on the probability of losing a packet based on the network congestion, as estimated with the traffic rate and buffer sizeñ Shu and Krunz [24] use the correlation between packet losses to determine if the loss pattern corresponds to normal operation or the router is faulty instead and Ning et al. [25] estimate the likelihood of packet losses based on the network parameters and previous link history.

Finally, possible response mechanisms are just to simply announce the discovered discrepancies [14], remove the faulty link or node from the routing fabric [12], or incorporate the obtained knowledge into the routing protocol as part of its metric [26]. In the case of WCN, we believe that a simple mechanism as notifying the owner of the faulty equipment and some network administrator is enough.

3. System Model

In this section we introduce some concepts that will be used through the rest of the paper, the assumptions that have been made and why they are reasonable, and we conclude with a list of assumptions.

3.1. Detection and threat model

The goal of KDet is to solve the problem of forwarding fault detection under the assumption of a reliable traffic validation function. As a failure detector, KDet's correctness can be expressed in terms of accuracy and completeness. The definitions of accuracy and completeness we use are based on those presented by Mizrak et al. [27]:

- *a–Accuracy*: A failure detector is *a-accurate* if whenever a correct router detects a set of routers as faulty (S), then, there is at least one router (r) in S that is faulty, and $|S| \leq a$.
- *a-Completeness*: A failure detector is *a-complete* if, whenever a router r is faulty, then, eventually all correct routers will suspect a set of routers (S) such that $r \in S$ and $|S| \leq a$.

We will later prove that KDet is k-accurate and k-complete.

In our threat model we consider two different type of faulty nodes: t-faulty (t as in traffic) and p-faulty nodes (p as in protocol). A *t-faulty* node misbehaves in the traffic forwarding process, e.g., it may drop or corrupt packets instead of forwarding them. A *p-faulty* node does not participate properly in the detection protocol. No assumptions are made regarding how p-faulty nodes participate in the detection protocol. For instance, they may send false traffic summaries to avoid the detection of a colluding node, or corrupt KDet reports or drop them instead of flooding them. A *faulty* node is either p-faulty, t-faulty or both; and a set of nodes is p-faulty, t-faulty or faulty when at least one of its nodes is p-faulty, t-faulty or faulty, respectively.

As Mizrak et al. [27], we measure the strength of the adversary as the largest set of connected faulty nodes, expressed by k. For example, in Figure 2, if faulty nodes are the ones in orange, $k=2$, as there are only two of them connected, even if in total there are 3 faulty nodes.

Because the failure detector is defined in terms of intervals of time, we assume that the misbehavior of faulty routers lasts long enough to be detected.

[2] `https://goo.gl/McHYDe` explains collusion through an example and how other solutions in the literature fail to solve it.

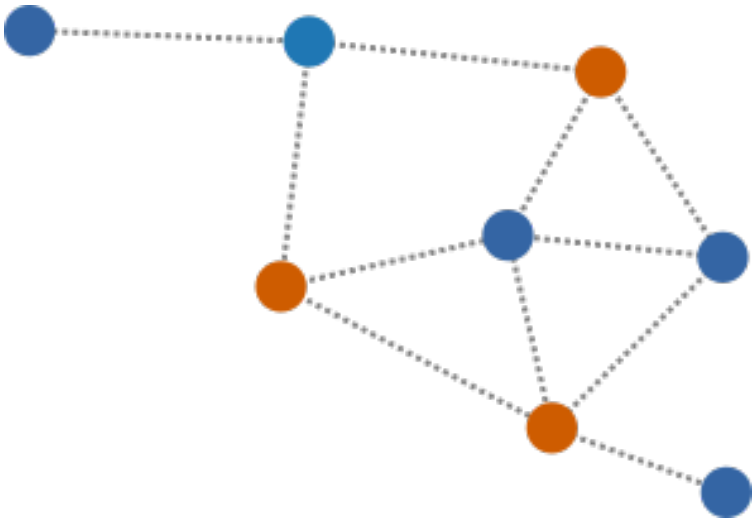


Figure 2: Example network with 3 faulty nodes and $k=2$.

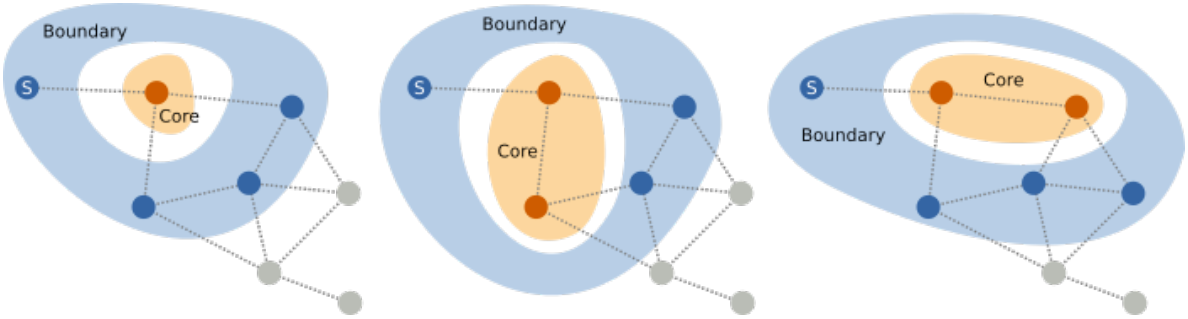


Figure 3: Cores to monitor for node S

3.2. Network model

Additionally, the goal of KDet is to provide such solution in the context of WCN, therefore the network model characteristics should match those of a WCN. For instance, Guifi.net is a large network with over 30,000 operational nodes, born in the Osona area and currently extending through Spain and beyond. Guifi.net uses BGP for inter-areas routing and a mix of several routing protocols (OSPF, BMX6, and OLSR, among others) for intra-area routing. Another example is FunkFeuer, a smaller WCN with around 600 nodes deployed in seven regions of Austria. There, the mesh backbone relies solely on OLSR for routing. Both of them commonly feature the use of supernodes: nodes that group together several devices and antennas to have a broader reach.

Therefore no assumption is made regarding the network routing protocol, and nodes may have one or several antennas. However, we assume that only bidirectional links are considered, which is already implicit for the vast majority of routing protocols in WCN (OLSR, BMX6, BATMAN, etc.) [28], which discard unidirectional links. We also assume that adjacent nodes agree on the traffic they have exchanged, which is reasonable given that 802.11 acknowledges received packets for unicast traffic. To avoid having to deploy additional nodes for monitoring, in KDet every router takes both the role of monitor, making sure its k-hop neighbors behave properly, and suspect, being monitored by its neighbors. Monitoring works by defining a set of connected routers that are being monitored (the *core*) by a group of monitors (the *boundary*). In KDet, the *boundary* of a *core* is the set of nodes that the core is directly connected. Figure 3 shows for the same network some of cores monitored by node S and their boundaries. That results in a set of cores that correspond to every possible (and overlapping) set of connected nodes of size k or smaller.

We also assume the existence of three supporting services provided by the network in a reliable way. First, we assume the presence of a public-key infrastructure, more specifically a mechanism to verify the authenticity of

KDet's messages. This is easily achieved in a WCN, as they could obtain a certificate when they join the network or, alternatively, they can use a lightweight mechanism like SEMTOR [29, 13], part of the experimental BMX7 routing protocol, developed as an evolution of BMX6, and used in several WCN. A reliable neighborhood discovery mechanism must exist, something like the connectivity maps used on several WCN [2]. As a side note, keeping an updated vision of the neighborhood is easier than maintaining updated path information [30]. Finally, for the sake of simplicity, we also assume the existence of a set of **trusted authorities (TA)** responsible for collecting KDet's reports and making them publicly available. This role could be assigned to network monitoring servers (e.g. graph servers in Guifi.net) and we assume that every node in the network is capable of communicating securely with the TA that is responsible for monitoring its behavior (i.e. each network node is paired with a TA) and the TAs that are assigned to its neighbors. Moreover, the information about which TA is responsible for each node is publicly available. Possible ways to implement the communication mechanism with the TA are the intrusion-tolerant overlay by Obenshain et al. [31] or using the spanning tree proposed by Zhang et al. in DynaFL [32].

Finally, we make two assumptions about timing: there is a known bound for the packet delay between two network nodes, and network changes (e.g. a node joins the network) happen on a larger time scale than the protocol's convergence, which is reasonable because membership changes in a mesh network are infrequent.

3.3. Traffic Validation function

KDet is a decentralized detection algorithm independent of the mechanism used to summarize and validate traffic. Still, to achieve detection we need a traffic summary function and a validation mechanism for network areas. We will use $\boldsymbol{\mathcal{S}}(\cdot)$ to represent that summary function and S to represent the summary itself. If there is more than one summary involved, we will use $S^{\leftarrow \mathrm{l}}_{\mathrm{traffic}}$ for the summary of some traffic being sent through link l (and $S^{\rightarrow \mathrm{l}}_{\mathrm{traffic}}$ would be the summary of the received traffic). Depending on the implementation strategy (Section 6), it should be possible to add and subtract traffic summaries. Luckily many summary functions have defined $+$ and $-$ operations, such as sketches [33, 16, 32], counters [12], fingerprinting [15] or sampling [16]; so they can be used for KDet. Moreover, we assume that traffic summary functions are second-preimage resistant, i.e. faulty nodes cannot modify the packets in such a way that they would produce the same resulting summary.

For traffic validation we assume that there is a validation function that validates forwarding behaviour of a core given the summaries of the traffic entering and leaving that area: $\boldsymbol{\mathcal{V}}(S_{\mathrm{in}},S_{\mathrm{out}})$. $\boldsymbol{\mathcal{V}}$ equals *true* when the behavior of the core is as expected (i.e. it is valid) and *false* when it is under-performing (i.e. it is not valid). We assume that whenever there is a t-faulty node in the evaluated core the validation function will classify it as faulty (i.e. $\boldsymbol{\mathcal{V}} = \mathit{false}$).

3.4. Summary of assumptions

In summary (and for future reference), the assumptions made are:

1. Bidirectional links where endpoints can agree on the traffic exchanged.
2. Public key infrastructure.
3. Neighbor discovery mechanism.
4. Trusted Authorities reliably connected to monitored routers.
5. Known bound for packet delay.
6. Misbehaving behavior is long enough to be detected.
7. Summary function is second pre-image resistant.
8. There is a validation function that properly detects t-faulty behavior.

4. The KDet detection protocol

KDet implements a detection protocol where a group of boundary nodes monitor the traffic on its k-hop neighborhood. KDet defines the different overlapping cores that need to be monitored by the nodes that act as boundary with the rest of the network. KDet's main goals are that if a core is faulty, its boundary can detect it, and if a boundary falsely accuses a core, the core can detect that. This is achieved by designing KDet to satisfy two principles:

Detection If a core is t-faulty and there is no p-faulty node in its boundary, the core is detected as faulty or disconnected from the network.

No false accusation If a core is not faulty, then no TA declares it as faulty.

The first principle is enforced through the *boundary protocol* and the second principle is attained by detecting false accusation through the *core protocol* and delegating detection to the TAs in the *coordinated detection* phase.

4.1. Boundary protocol

The goal of the boundary protocol is to determine the forwarding behavior of the core. This is achieved by, first, monitoring the traffic entering and leaving the core and, later, using the traffic validation function to test the core's behavior. More formally, every node in the boundary monitors each link (i) with the core and keeps a summary of the incoming traffic.

$$S_{\text{core}}^{\rightarrow i}(T) = \boldsymbol{\mathcal{S}}(\text{traffic}_{\text{in}}(T) - \text{traffic}_{\text{to}}(T))$$

and a summary for the outgoing traffic:

$$S_{\text{core}}^{\leftarrow i}(T) = \boldsymbol{\mathcal{S}}(\text{traffic}_{\text{out}}(T) - \text{traffic}_{\text{from}}(T))$$

Where $\text{traffic}_{\text{in}}$ is the traffic sent through link i during period T; $\text{traffic}_{\text{to}}$ is the part of that traffic destined to nodes in the core. Similarly, $\text{traffic}_{\text{out}}$ is the received traffic from link i, and $\text{traffic}_{\text{to}}$ is the part of that traffic whose source is a node in the core. At the end of the period, each boundary node creates a report with the traffic summaries and a nonce, signs it, and shares it with the rest of the boundary nodes by robustly flooding [34] it through the core:

$$R(T) = \text{signed}(S_{\text{core}}^{\rightarrow i}(T) \forall i \in \text{links}, S_{\text{core}}^{\leftarrow i}(T) \forall i \in \text{links}, T)$$

After waiting a bounded amount of time, each node expects to have the report of every other node in the boundary. The timeout can be easily computed as there is a known bound for the network delays (assumption 5). After that timeout, if any report is missing, or its signature is not valid, the core is suspected. Otherwise, each boundary node evaluates the behavior of the core using the validation function [35]:

$$V_{\text{core}}(T) = \boldsymbol{\mathcal{V}}\left(\sum_{\forall i} S_{\text{core}}^{\rightarrow i}(T), \sum_{\forall i} S_{\text{core}}^{\leftarrow i}(T)\right)$$

Then, the core the node shares a verdict ($\mathcal{V}(T)$) with every TA assigned to a core node it is connected to. $\mathcal{V}(T)$ consists of $V_{\text{core}}(T)$; a bitmap indicating which reports have been received (received_reports); the list of core nodes the report refers to (i.e. those assigned to the given TA), $[n_i]$; and a nonce:

$$\mathcal{V}(T) = \text{signed}(V_{\text{core}}(T), \text{received\_reports}, [n_i], T)$$

At this point, the TA will accept the report if it acknowledges the links between the reporter and all the nodes in $[n_i]$, or discard it and informs the reporter about which link is not valid (see 4.3). If a link is not valid, the reporter will disconnect from that node.

4.2. Core protocol

In parallel, nodes in the core look for p-faulty nodes on the boundary by checking the reports and core evaluation, and disconnect from nodes detected as p-faulty. A correct boundary node behavior is characterized by:

1. There is a single and consistent report for each interval $R(T)$.
2. $\mathcal{V}(T)$ is consistent with the exchanged reports.

To assess the behavior of a boundary node B, every node in the core C, connected to it (via link i) keeps local summaries of the traffic on link i and checks the following:

1. At the end of the monitoring interval, there is a single report from B with a valid signature being robustly flooded on the core.
2. The traffic summaries in the report related to link i are the same as those kept locally.
3. $\mathcal{V}(T)$ (obtained from C's TA) is consistent with the reports that have been robustly flooded, i.e. all the reports shared are marked as received on received_reports and $V_{core}(T)$ is the result of applying $\boldsymbol{\mathcal{V}}$ over them.

If any of those checks fail, C considers B p-faulty, disconnects from it, and informs the TA that the link is no longer valid.

4.3. Coordinated detection

At every moment the TA keeps track of the nodes it is responsible for and the links that they have declared as not valid. At the end of each interval, the TA receives the verdicts from the boundary nodes for those nodes that it is responsible. For every $\mathcal{V}(T)$ received from node B the TA does the following checks:

1. It is responsible for every node in $[n_i]$.
2. No node in $[n_i]$ has reported the link with B as not valid.

If any of the checks is not valid, the TA will send a signed message to B rejecting the verdict and indicating why (i.e. which node should not be in $[n_i]$). Then, with the remaining verdicts the TA checks whether there is any missing report (as marked on received_reports) or $V_{\text{core}}(T) = \text{false}$ the TA will add the core to the list of suspected cores ($\text{core} \in \text{suspected}(T)$). Then, if in the next intervals the core is still suspected and every core node is still connected to the same nodes in the boundary, the core is detected as faulty ($\text{core} \in \text{detected}(T')$). Formally:

$$\begin{aligned} &\text{core} \in \text{suspected}(T_1) \quad \wedge \text{core} \in \text{suspected}(T_2) \quad \wedge \\ &T_1 < T_2 \quad \wedge \text{boundary}(\text{core}, T_1) \subseteq \text{boundary}(\text{core}, T_2) \\ &\quad \Rightarrow \text{core} \in \text{detected}(T') \end{aligned}$$

Where $\text{boundary}(\text{core},T_1) \subseteq \text{boundary}(\text{core},T_2)$ implies that for every link existing between the core and the boundary at time T_1 the same link also exists at time T_2, i.e. the number of links from the core to the rest of the network has not decreased.

4.4. Core and boundary selection

Finally, to ensure that KDet is k-accurate and k-complete, we need to properly define the cores (and its boundaries) that will be monitored. This is achieved by monitoring every possible set of connected nodes of size $\leq k$.

$$\mathbf{C} = \{c \mid |c| \leq k \wedge \text{connected}(c)\}$$

For example, Figure 3 shows all the cores that node S should monitor for that network if $k = 2$.

5. Validation

To prove the correctness of KDet we will first prove that, no matter what faulty nodes do, KDet's principles are always satisfied or the faulty nodes are disconnected from the network. Then, we will use those principles to prove that KDet is k-accurate and k-complete under the assumption of k being the maximum number of directly connected faulty routers.

We can re-state the first principle as:

$$\begin{aligned} &\text{core} \in \text{faulty} \ \wedge \ \text{boundary} \in \text{correct} \\ &\quad \Rightarrow \exists t \mid \text{core} \in \text{detected}(t) \ \vee \ \text{core} \in \text{disconnected}(t) \end{aligned}$$

Because the core is faulty, if every traffic summary is available, the traffic validation function will detect the core (assumption 8):

$$V_{\text{core}}(T) = \boldsymbol{\mathcal{V}}\left(\sum_{\forall i} S^{\rightarrow i}_{\text{core}}(T), \sum_{\forall i} S^{\leftarrow i}_{\text{core}}(T)\right) = \text{false}$$

A core node could consider dropping some of the traffic summaries within a report, but because the reports are sent signed, that would cause the signature to be invalid and the report would be marked as not received. Therefore, core nodes can either robustly flood the boundary reports as expected and become suspected because $V_{\text{core}}(T) = \text{false}$ or not flood them and become suspected because received_reports will indicate that some reports were not received. In any case, the core cannot avoid being suspected; and once the core is suspected, its only way of avoiding detection is by modifying its boundary, so that:

$$\text{boundary}(\text{core}, T_1) \subsetneq \text{boundary}(\text{core}, T_2)$$

That is, on each interval, it needs to reduce its boundary by, at least, one link. However, because the number of links are finite, eventually the core will be disconnected from the rest of the network, or detected.

Lemma 1: *Eventually, every faulty core is either detected or it stops being part of the network.*

Similarly, the second principle is equivalent to:

$$\text{core} \notin \text{faulty} \Rightarrow \text{core} \notin \text{detected}$$

A core can only be detected if it is first suspected and, moreover, its boundary does not change; therefore, we only need to prove that whenever a correct core is suspected, its boundary changes.

A core is suspected if there is a boundary node that claims either that not every report was received or that reports were received but $V_{\text{core}}(T) = \text{false}$. In the first case, if boundary node B claims that it has not received the report from boundary node B', since the core is not faulty and we use robust flooding, it can either be because B' has not sent the report or because B is lying about not receiving the report. In any case, core nodes will know which one of the two nodes is p-faulty (checks 1 and 3 of the core protocol) and disconnect from it and report it to their TA, changing the boundary as expected. In the second case, because the core is not faulty we know that the traffic summaries satisfy:

$$V_{\text{core}}(T) = \boldsymbol{\mathcal{V}}\left(\sum_{\forall i} S^{\rightarrow i}_{\text{core}}(T), \sum_{\forall i} S^{\leftarrow i}_{\text{core}}(T)\right) = \text{true}$$

Thus, if boundary node B announces $V_{\text{core}}(T) = \text{false}$, either one of the traffic summaries has been modified or B is lying about V_{core}. If a node is misreporting the traffic summaries, the node in the core at the other end of the link will notice it and disconnect from it (check 2 of the core protocol). If B is lying about V_{core}, every neighbor in the core will detect the inconsistency and disconnect from B (check 3 of the core protocol). Therefore, in any case the boundary changes when the core is suspected and so a correct core is never detected as faulty.

Lemma 2: *A correct core is never detected as faulty.*

Now, let's consider the set of monitored cores as defined in section 4.4. Thanks to Lemma 2, we can prove that KDet is k-accurate, since every time a correct router detects a core as faulty ($\text{core} \in \text{detected}$) it has to be a faulty core or it will contradict Lemma 2. Furthermore, given the definition of $\mathbf{C}$, such core will always have a size below or equal to k.

If the faulty nodes set of sets is $F = \{f_1 \cup f_2 \cup \cdots \cup f_N\}$ such that for every f_i, $|f_i| \leq k$ and there is no link between f_i and f_j if $i \neq j$ (because we assume k). Then, there will be a core c_i for every f_i, such that $\text{c}_i = f_i$ by the construction of $\mathbf{C}$ and c_i will have a correct boundary, because there are no direct connections with any other f_j. Therefore, applying Lemma 1, every f_i will be detected and since $|\text{c}_i| = |f_i| \leq k$, KDet is k-complete.

6. Analysis

The previous section proves KDet's accuracy and completeness; here we will study what is the cost to achieve it. KDet's cost is mainly determined by the memory required to store the state of the protocol and the network overhead required to share traffic summaries.

6.1. State size

The size of the state is determined by the number of summaries a node needs to keep. Given a summary function that supports the + and − operations, a node can follow two strategies to save the required summaries:

1. For every monitored core and every link, i, that connects to that core, keep a summary, S^i_{core}, of the traffic traversing the core.

2. Keep a summary of the incoming and outgoing traffic of every link, $S^i = S^{\rightarrow i} - S^{\leftarrow i}$; and additionally, for each link, the traffic related to every node k hops away, $S^i_n = S^{\rightarrow i}_{\text{to}(n)} - S^{\leftarrow i}_{\text{from}(n)}$, so that the traffic summary for that core and link i can be computed as:

$$S^i_{\text{core}} = S^i - \sum_{\forall n \in \text{core}} S^i_n$$

For small values of k, the number of cores to monitor will be small, and therefore, option 1 will be preferred. Conversely, as k grows bigger, the number of cores grows exponentially, and 2 becomes a better choice. In further detail, if N is the number of nodes in the network, and R is the maximum number of links per node, then the number of cores a node monitors is bounded by $O(R^k)$ and the number of summaries $O(R^{k+1})$, since it can have up to R links to a core. In contrast, the second strategy keeps a summary per link and at most another per node, that is, the number of summaries will be bounded by $O(\min(R^{k+1}),RN)$, since for every link there will be at most R^k or N nodes at k hops. Compared with other strategies, Mizrak et al. [27] showed that WATCHERS cost is $O(R*N)$ and Π_{k+2}, $O(\min(R^{k+1},N))$; therefore, if we choose the optimal strategy based on our network, our solution will have a tendency that is between WATCHERS and Π_{k+2}.

6.2. Network overhead

In the decision on how to propagate the traffic summaries, a node could take two similar options:

1. Robustly flood within every monitored core their traffic summaries for each link connected to it (S^i_{core}).

2. Robustly flood every summary maintained following the second strategy presented before (S^i for every link i and S^i_n for every node n in its k-hop neighborhood) with TTL$=k+1$, so that they will reach every node that is also a part of the boundary of a monitored core.

Again, we expect the same behavior: a smaller k implies less cores to monitor and then strategy 1 causes less network overhead; whereas a larger k implies an exponential growth in the number of cores and therefore strategy 2 is more convenient. If we assume messages are flooded using broadcast, the first strategy will cause, that every summary stored will be sent by its owner and the k nodes in the core it relates to (overhead$\in O(kNR^{k+1})$). Instead, the second strategy will broadcast every summary as many times as nodes k hops away plus one, i.e. overhead$\in O(\min(N^2R^{2k+1},N^3R))$. Equivalently, for Π_{k+2} every summary is sent through the monitored path, which is at most $k+1$ hops long, so its overhead will be bounded by $O(\min(kNR^{k+1},kN^2))$; and for WATCHERS, every summary is flooded, so it will be sent by every network node, $O(R*N^3)$. So as before, the solution proposed, if chosen to optimized the overhead will be within the bounds of Π_{k+2} and WATCHERS.

6.3. A realistic example

However, the given trends are only an upper bound to the actual space and overhead consumed by the protocols. Figures 5a and 5b show which would be the real cost of each detection protocol for the guifi.net Barcelones area network (Figure 4) [36], a network with 113 nodes and 134 links; assuming there is traffic between every pair of nodes.

Figure 5a shows the state size stored by each node for the guifi.net Barcelones area [36] assuming summaries of 500B [32, 16], the lines show us the average cost for each value of k. As expected, the cost of the first strategy is exponential, and for any k bigger than 2 the second strategy already outperforms the first one. The second strategy has a state cost that is just slightly worse than Π_{k+2}, and in any case always below 1MB, which sounds reasonable. Similar results have been found for different network topologies [37].

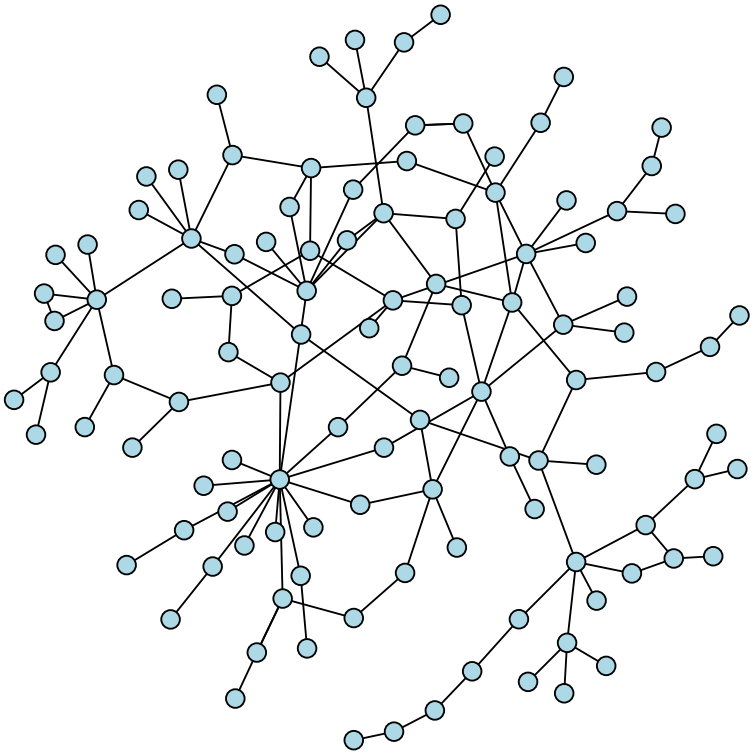
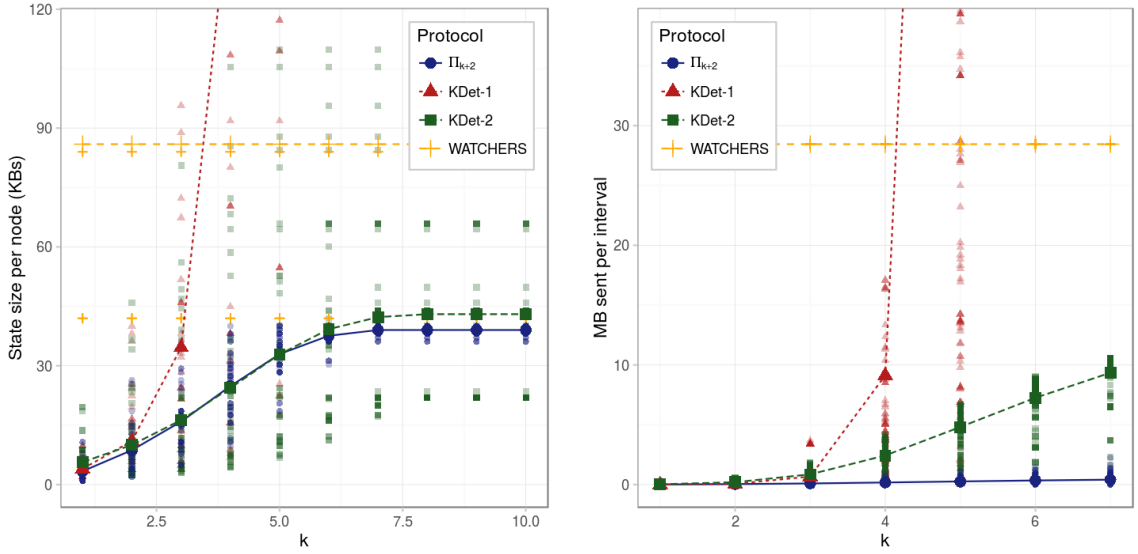


Figure 4: Barcelones network



(a) State size

(b) Network overhead

Figure 5: Protocol cost

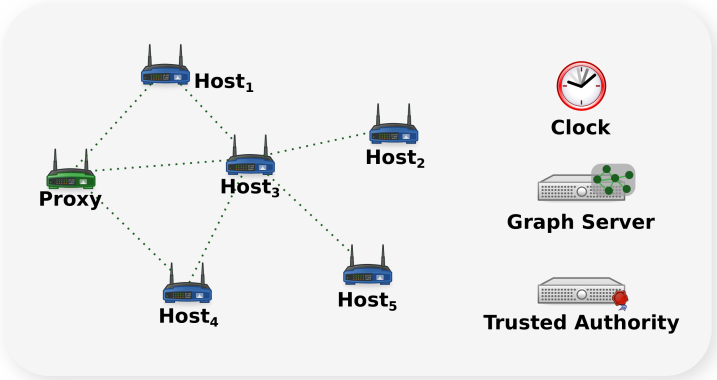


Figure 6: Network model for simulation

Figure 5b shows the data in MB to be sent through each link due to traffic summaries for every protocol period. As we can see, here the strategy 1 outperforms 2 whenever k is smaller than 4, because the second strategy floods traffic summaries indiscriminately, whereas KDet-1 only floods it within the respective cores. We observe also that now the difference between KDet-2 and Π_{k+2} is considerable because KDet requires flooding within cores, but Π_{k+2} does not, since its monitored units are paths. This difference in cost is caused by the fact that KDet behaves independently from the routing protocol and therefore cannot consider the path a packet follows. For this case, we can see that at most around 10MB of data needs to be shared per link, so for instance if we want to keep the overhead below 250kbps, the interval should be longer than 6 minutes. In any case, in a WCN, there will not be traffic between every pair of nodes, but mostly between every node and a selected gateway node. In those cases, as we will see in the following section, the network overhead is kept well below the limits that we have shown. Moreover, to alleviate the effect of such overhead, techniques such as LEDBAT [38] can be used to avoid clogging the network links with the protocol's traffic.

7. Evaluation by simulation

On the previous sections we have studied the KDet protocol from a theoretical and analytic point of view and set bounds on its performance. However, the actual performance and precision of KDet will depend on the network traffic, the traffic summary function and validation mechanism chosen to work together with the detection protocol.

To achieve a more realistic view of the behavior of the whole detection mechanism we have implemented a simplified but complete version of KDet [39] with sketches [17] as summary function and conservation of the flow [12] as validation mechanism in OMNeT++ [40] and studied its detection accuracy and network overhead.

7.1. Simulation model

Our simulated network consists of a set of network nodes, a clock, a graph server and a trusted authority (see Figure 6). The **clock** ticks every interval, so that every other module knows when an interval is over and does the required actions. The **graph server** keeps track of the network graph and replies queries related to it. It also keeps the nodes updated on which are the cores that need to monitor.

Lastly, **network nodes** are implemented by extending the AdHocHost module from the INET framework. Network nodes are divided into gateway nodes and hosts: hosts send and receive traffic from the gateways; and they are randomly placed so that the network is a single connected component. A WCN node extends the AdHocHost by including the following modules: a traffic generator, a packet dropper, a topology announcer and the KDet protocol module. The **traffic generator** uses IPvXTrafGen to generate network traffic. The **packet dropper** is responsible for implementing the malicious behavior by dropping a preconfigured percentage of the packets being forwarded. Then, the **topology announcer**, as expected, is responsible of monitoring the node's neighborhood and updating the graph server when there is a change.

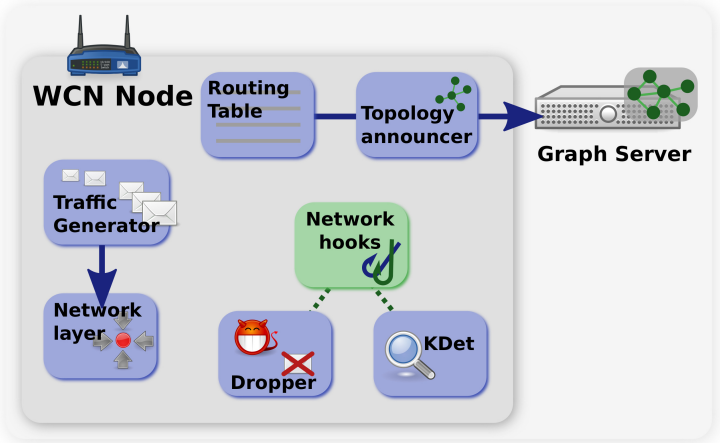


Figure 7: Simulation of a WCN Node

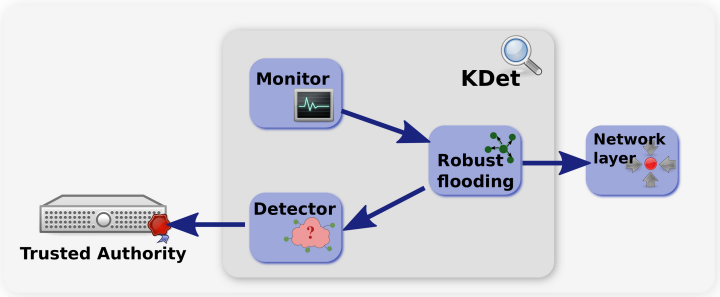


Figure 8: Simulation of the KDet module

Finally, the KDet protocol is composed by several submodules: the monitor, the robust flooding module and the detector. Both the monitor and the detector have two different implementations, each implementing one of the two strategies explained before in Section 4. The **monitor** uses network hooks to monitor the traffic sent to and received from a node's neighbor, and updates the required traffic summaries depending on the strategy it implements. Then, periodically, the monitor creates the required reports and sends them to the flooder so that they are shared with the corresponding nodes. Moreover, if the node is faulty, the report is marked as bogus, so that it cannot be taken into account to determine the behavior of the core it relates to. The **flooder** is an application-layer implementation of the robust flooding protocol proposed by Perlman [34]. To conclude, the **detector** gathers every received report and evaluates each monitored core by using conservation of the flow. Then, the detector sends an evaluation to the **Trusted Authority** with its findings: which reports were received and which were not, and the estimated core's drop probability.

7.2. Experiments

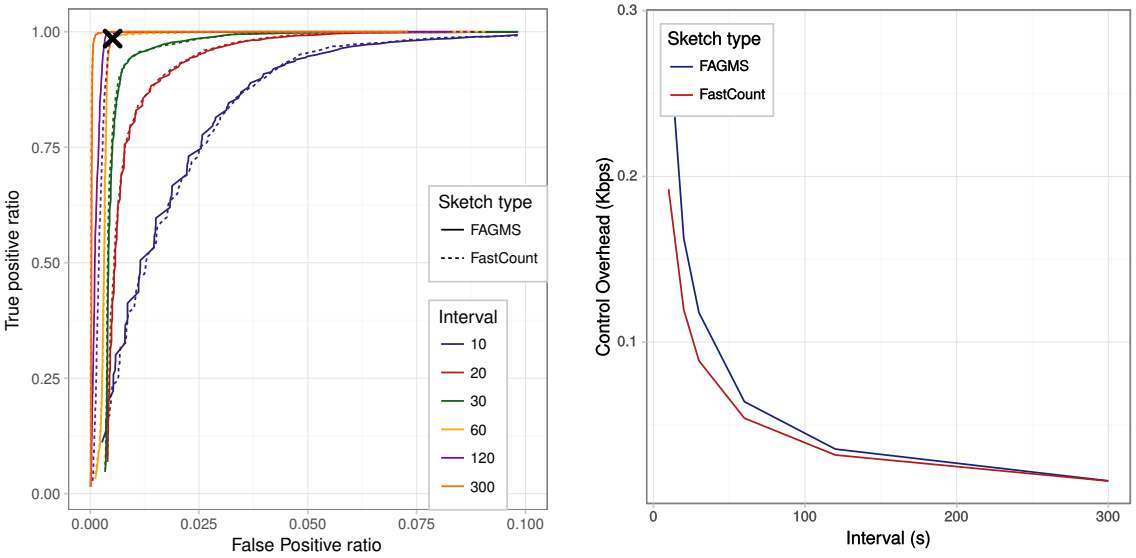
In our first experiments we will measure the accuracy of the detector in terms of false positive and negative ratio, i.e. how likely is a faulty node to avoid detection and for a non-faulty node be detected as faulty, depending on the sketch size and time interval. In previous work [17], we have measured the precision of sketches, so this paper focuses only on the cost of KDet when using a sketch of 64 columns, but a more intensive study of the effects of changing the sketch characteristics can be found in our web report [37].

Protocol interval.

Our first experiment simulated a network of 50 nodes and 3 gateway nodes. Each node would connect to one of the gateway nodes and exchange traffic with it with an average of 1 packet per second. Of these 50 nodes, 5 were faulty and dropped 10% of the traffic. The KDet protocol was run with k=1, i.e. assuming no collusion and using a sketch of 64 columns and 1 row, but varying the time intervals to study its effect.

As we can see in Figure 9a, longer intervals lead to better accuracy, because rare isolated events (like a packet lost because of collision) are averaged with normal events, so that the behavior of a node can be better measured. In general, detection will be pretty accurate if we use an interval longer than 60 seconds; for example, if the detection threshold is 0.04, KDet gives a false positive ratio of 0.5% and detect 96% of faulty routers when using a sketch of 64 columns. There is almost no difference in accuracy when using Fast-AGMS or FastCount sketches.

In terms of overhead, as expected, the longer the interval, the less control traffic. Furthermore, Fast-AGMS requires more bits because its ± 1 function implies that we need a bit to code the sign of the counter values and with the given parameters and the traffic on the network, there will not be many packets per counter, so the likelihood of a Fast-AGMS counter of reaching a value as high as those in the FastCount sketch is high (there are little combinations and 64 counters), and one additional bit when the number is so small represents a great increase in the number of required bits.



(a) ROC curve

(b) Traffic overhead

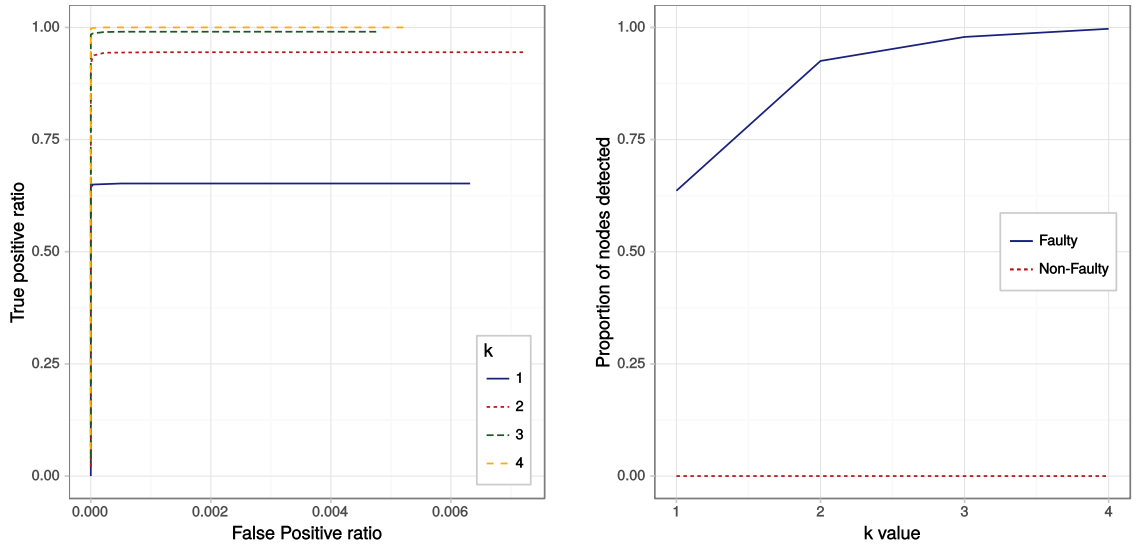
Figure 9: Different time intervals

Value of k.

Our second experiment considers collusion, and to detect it, we run KDet with a variable value for the parameter k. Figure 10a shows the ROC curve for different values of k. We define the true positive ratio as the proportion of faulty nodes that have been detected in at least one of the cores they belong to. Furthermore, we define the false positive ratio as the proportion of cores that are detected as faulty when they are not. In this case, we have a network of 50 nodes, with 10% of them being faulty; which makes the maximum possible number of faulty nodes connected as 5. As we can see in Figure 10b, if we use $k=1$ or $k=2$, there are many faulty nodes that are not detected, because the likelihood of two or three faulty nodes being connected is not negligible. But for $k=4$, KDet is capable of finding every faulty node.

Finally we study the cost of each of the two implementations presented. Figure 11a shows the network overhead for the first strategy, that is based on *core* monitoring; and the second, based on *link* monitoring. As our analysis showed, the cost of the second implementation approach scales way better as k increases, but in any case, for

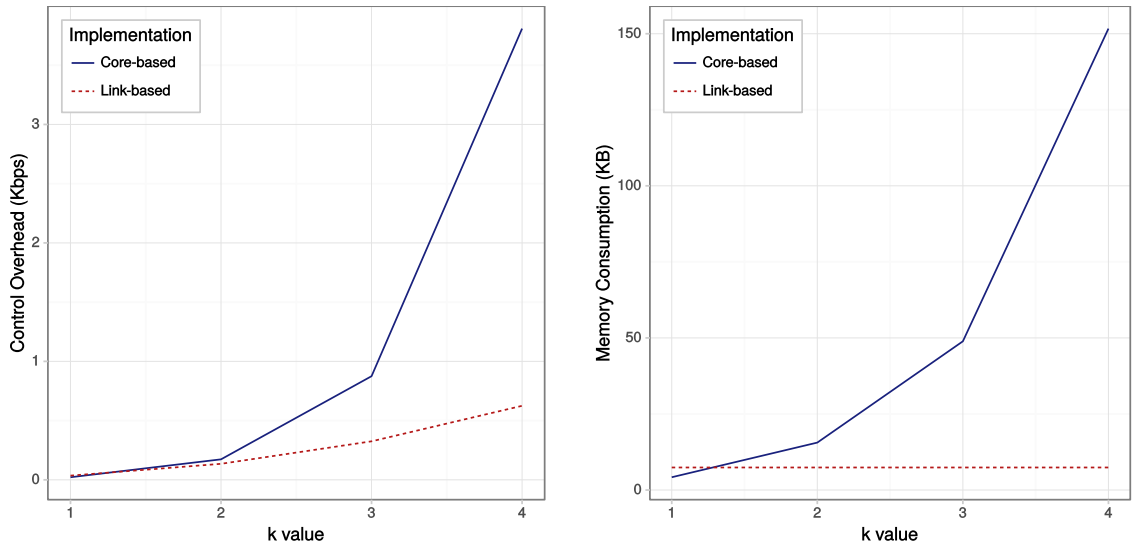
the given k values, the overhead is reasonable (always below 5 Kbps per node). Furthermore, for the memory (Figure 11b), we have a constant cost for the second implementation and again an exponential one for the first. This is due to how the simulation model was implemented, which saved a sketch for every source and destination per link, even though in some cases it was not required, because such nodes were more than k hops away. Again, the cost seems reasonable, as each node is required to store sketches taking at most 150 KBs.



(a) ROC curve

(b) Proportion of detected nodes

Figure 10: Different k values



(a) Traffic overhead

(b) Memory consumption

Figure 11: KDet cost for different k values

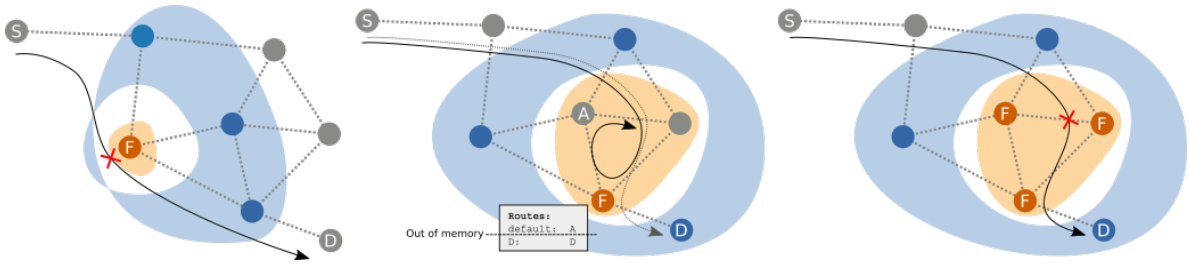


Figure 12: Three failure scenarios

8. Discussion

In this section we discuss further some of the aspects of the proposed implementations for KDet, how KDet fares in 3 practical scenarios, the limitations imposed by the Traffic Validation mechanisms and a simple way of reducing KDet's network overhead.

8.1. KDet Implementations

We have presented two possible implementation strategies for KDet in section 6. However the implications of using one instead of the other go beyond the costs in terms of state size and overhead presented in sections 6 and 7.

To recap, in terms of local storage, the first strategy keeps track of every core and keeps at every moment its summaries updated, while the second strategy keeps several summaries for each link: one for the total traffic going through it and then one for each traffic stream from (or to) the set of nodes that belong to a monitored core. When we follow the first strategy, whenever a packet is sent (or received) through a link, it will cause the update of several summaries, because it will relate to several cores (whenever k is bigger than 1), and, as we have seen, the number of summaries grows exponentially with k, making the process of updating the summaries a computation hog. However, if we were to follow the second strategy, each packet will only update two summaries (the one related to the link and the one related to the source -or destination) and, therefore, will be able to keep up even when k is high and the traffic rate is high. In consequence, the second strategy should be preferred.

Then, in the flooding process we face again the same decision: if we used the second strategy, we can first obtain the core summary as in equation 2 and then flood that summary through the core (strategy 1) or simply flood every summary with TTL$=k+1$ and compose the core traffic summary when it reaches the other boundary nodes. Now, in terms of computation, the second strategy will be more costly, because not only one node needs to compute the traffic summary, but every node in the boundary, even the sender itself, must do so. Nevertheless, for big values of k, the second strategy is less costly in terms of overhead, as we have seen, and it is also simpler in terms of flooding because it only needs to consider the TTL and not to which core it belongs. Thus, there is not a strategy that is clearly better, but it will depend on the situation (value of k) and priorities (CPU vs. overhead).

Regarding the cores, they can be easily computed locally. Every node looks up on the connectivity map its neighbors up to k hops starting from itself. Without considering the node itself, each of these paths is a core that needs to be monitored, and every neighbor of the core's node that is not already part of the core is the boundary that monitors it.

8.2. Practical scenarios

This section describes 3 different scenarios and how KDet reacts in each case.

8.2.1. Single faulty router

For the first scenario, let's imagine one single router has started failing and it is not properly forwarding traffic (Figure 12-1). This could be due to a hardware failure or a misconfiguration. In this case, KDet will easily find which is the failing router as there are no p-faulty routers, so its neighbors would figure out that it is failing, and its TA will report that router as faulty.

8.2.2. Routing loop

The second scenario is one of the typical causes of the "pollastre de rutes" mentioned before, it can be caused by a node that starts wrongly announcing addresses or the routing table running out of space (see Figure 12-2). In this case, KDet will be able to detect the faulty router as long as the length of the loop is equal or below k, as the neighbors will see that packets go into the loop, but not leaving. However, in this scenario KDet is not able to accurately pinpoint the faulty router, but blames the whole loop, and additional investigation would be required to detect which one of the nodes in the loop is the actual faulty one.

8.2.3. Compromised subnetwork

Finally, because of the network openness, and adversary could take control of a whole subnetwork (Figure 12-3), dropping or corrupting all the traffic that goes through it, and try to avoid detection using collusion. In this scenario, KDet will be able to detect the faulty subnetwork as long as the number of corrupted nodes is equal or below k, so that it exists a boundary of non-faulty nodes that can notice their behavior.

8.3. Limitations

KDet assumes that there is a 100% accurate traffic validation function ($\mathcal{V}$), however all networks are likely to lose some traffic even without faulty routers, specially wireless networks, hence the comparison of traffic summaries should allow for some difference as long as it is within a reasonable margin. The consequence is that accuracy and completeness should be expressed in probabilistic terms (as suggested by Goldberg et al. [16]).

Moreover, most validation functions assume that all traffic is unicast, so they will not work for multicast traffic. Luckily, as mentioned by Mizrak et al. [27], multicast traffic is not common. In fact, we have studied traffic captures from Guifi.net and they show that the only multicast traffic is the routing protocol control packets and, accordingly, with TTL$=1$ and should not be forwarded.

Another limitation of KDet is that as k increases so does its cost in terms of memory and network bandwidth. In networks where collusion is not expected, KDet can be heavily simplified (no need of signatures and checks) and use $k=1$ to keep its cost to a minimum.

8.4. Fish-eye KDet

We have seen that the main drawback of KDet is that, as k increases, it requires more and more network bandwidth, a resource that is quite scarce in wireless networks. To overcome that drawback, we have proposed using techniques as LEDBAT, so that traffic summaries are exchanged with low priority compared with the normal data-traffic. Another possibility is to use a technique similar to that used in OLSR to reduce its control overhead [41], fish-eye relies on the fact that only nodes close by need to know that there has been a network change, whereas nodes far away, only need an update less frequently, so TC messages are propagated with different TTLs, so most of the time only reach nearby nodes, but from time to time, they flood thorough all the network.

In our case, one could consider that a single node failure is more likely than two or more nodes colluding to avoid detection, so we could share and evaluate frequently the cores of size 1, and for the rest of the nodes use a longer interval. For the first implementation, since the traffic summaries are already divided by core, no additional memory is required. But for the second implementation, traffic summaries should keep two copies: one for the smaller interval and another for the long interval; otherwise, colluding nodes could avoid detection by only dropping packets during the time that there is no traffic summary exchanged.

9. Conclusions

This paper proposes a solution to the summary dissemination and detection sub-problem of forwarding fault detection, in the context of Wireless Community Networks. k-Detection (KDet) is a decentralized protocol that doesn't require path knowledge and keeps aggregated traffic summaries, but still resilient to false accusation and collusion, unlike previous solutions. We have proven KDet's correctness and studied its performance compared with Π_{k+2} and WATCHERS from an analytical point of view, which showed that by choosing the proper value of k and detection interval, the cost of KDet is reasonable for a WCN. KDet, even though it has a higher cost

than Π_{k+2}, comes with two advantages over Π_{k+2}: it can be deployed as an independent daemon on the routers, without the need of a link-state routing protocol, and it gives a more accurate prediction of the failing areas.

We have also measured KDet's accuracy, memory and bandwidth consumption by simulation, using OMNeT++. Our results show that the cost of KDet is kept within reasonable bounds (less than 5 Kbps of network bandwidth and at most 150 KBs of memory) and it detects perfectly faulty cores when k is chosen properly. In case the network bandwidth needs to be reduced any further, we propose an approach based on the OLSR fish-eye mechanism that shares summaries with different frequencies based on the core size, so that faulty nodes that do not collude (the most likely scenario) are detected fast; but still collusion is detected, though it may take a little longer.

To sum it up we have looked at how Wireless Community Networks can be monitored to automatically detect forwarding faults. We consider the open and self-managed nature of these networks and the effect of incorrect routers or groups of them, that can perform false accusation to correct routers and collusion among incorrect ones. And because none of the previous solutions tackled both false accusation and collusion when the traffic paths are unknown, we have proposed KDet, a detection protocol that does so. The cost analysis for a real WCN, the evaluation of the protocol and its components by simulation, the consideration of implementation alternatives, and ways to address the limitations, confirm the practical feasibility of KDet in WCN.

Acknowledgement

This work was carried out with the support of the European Commission through the 7th ICT Framework Programme in the CONFINE project ("Community Networks Testbed for the Future Internet", 288535), the EU Horizon 2020 Framework Program project netCommons (H2020-688768), and by the Spanish government under contracts TIN2013-47245-C2-1-R and TIN2016-77836-C2-2-R.

References

[1] R. Baig, R. Roca, F. Freitag, L. Navarro, guifi.net, a crowdsourced network infrastructure held in common, Computer Networks 90 (2015) 150–165. doi:10.1016/j.comnet.2015.07.009.

[2] J. Avonts, B. Braem, C. Blondia, A questionnaire based examination of community networks, in: 2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2013, pp. 8–15. doi:10.1109/WiMOB.2013.6673333.

[3] L. Maccari, R. L. Cigno, A week in the life of three large wireless community networks, Ad Hoc Networks 24 (2015) 175–190.

[4] C. Rey-Moreno, J. Miliza, F. Mweetwa, G. van Stam, D. Johnson, Community networks in the african context: Opportunities and barriers, in: Proceedings of the First African Conference on Human Computer Interaction, ACM, 2016, pp. 237–241.

[5] L. Belli (Ed.), Community Connectivity: Building the Internet from Scratch. Annual Report of the UN IGF Dynamic Coalition on Community Connectivity, FGV, 2016.

[6] F. Chiti, R. Fantacci, L. Maccari, D. Marabissi, D. Tarchi, A broadband wireless communications system for emergency management, IEEE Wireless Communications 15 (3).

[7] D. Vega, R. Baig, L. Cerdà-Alabern, E. Medina, R. Meseguer, L. Navarro, A technological overview of the guifi.net community network, Computer Networks 93, Part 2 (2015) 260 – 278, community Networks. doi:10.1016/j.comnet.2015.09.023.

[8] P. Kriz, F. Maly, Troubleshooting assistance services in community wireless networks, Journal of Computer Networks and Communications 2012 (2012) 1–6. doi:10.1155/2012/621983.

[9] N. Samian, Z. A. Zukarnain, W. K. Seah, A. Abdullah, Z. M. Hanapi, Cooperation stimulation mechanisms for wireless multihop networks: A survey, Journal of Network and Computer Applications 54 (2015) 88 – 106. doi:10.1016/j.jnca.2015.04.012.

[10] Pollastre de rutes I, https://guifi.net/node/27450, accessed: 2017-10-16.

[11] Pollastre de rutes II, https://lists.guifi.net/pipermail/guifi-rdes/2012-June/024384.html, accessed: 2017-10-16.

[12] K. Bradley, S. Cheung, N. Puketza, B. Mukherjee, R. Olsson, Detecting disruptive routers: a distributed network monitoring approach, IEEE Networkdoi:10.1109/65.730751.

[13] A. Neumann, Cooperation in open, decentralized, and heterogeneous computer networks, Ph.D. thesis, Universitat Politècnica de Catalunya (2017).

[14] K. Argyraki, P. Maniatis, O. Irzak, S. Ashish, S. Shenker, Loss and delay accountability for the internet, in: Proceedings - International Conference on Network Protocols, ICNP, IEEE, 2007, pp. 194–205.

[15] A. T. Mizrak, S. Savage, K. Marzullo, Detecting Malicious Packet Losses, IEEE Transactions on Parallel and Distributed Systemsdoi:10.1109/TPDS.2008.70.

[16] S. Goldberg, D. Xiao, E. Tromer, B. Barak, J. Rexford, Path-quality monitoring in the presence of adversaries, ACM SIGMETRICS Performance Evaluation Reviewdoi:10.1145/1384529.1375480.

[17] E. López, L. Navarro, Tight bounds for sketches in traffic validation, in: 2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC), 2017, pp. 210–215. doi:10.1109/ICNSC.2017.8000093.

[18] I. Avramopoulos, J. Rexford, Stealth Probing : Efficient Data-Plane Security for IP Routing, in: Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference, USENIX Association, 2006, pp. 25—–25.

[19] C. Basescu, Y. H. Lin, H. Zhang, A. Perrig, High-speed inter-domain fault localization, in: 2016 IEEE Symposium on Security and Privacy (SP), 2016, pp. 859–877. doi:10.1109/SP.2016.56.
[20] X. Zhang, C. Lan, A. Perrig, Secure and scalable fault localization under dynamic traffic patterns, in: Proceedings - IEEE Symposium on Security and Privacy, 2012, pp. 317–331.
[21] A. T. Mizrak, Y.-C. C. Cheng, K. Marzullo, S. Savage, Detecting and isolating malicious routers, IEEE Transactions on Dependable and Secure Computing 3 (3) (2006) 230–244.
[22] T. W. Chao, Y. M. Ke, B. H. Chen, J. L. Chen, C. J. Hsieh, S. C. Lee, H. C. Hsiao, Securing data planes in software-defined networks, in: 2016 IEEE NetSoft Conference and Workshops (NetSoft), 2016, pp. 465–470. doi:10.1109/NETSOFT.2016.7502486.
[23] S. Goldberg, D. Xiao, E. Tromer, B. Barak, J. Rexford, Path-Quality Monitoring in the Presence of Adversaries: The Secure Sketch Protocols, IEEE/ACM Transactions on Networking PP (2014) 1–13.
[24] T. Shu, M. Krunz, Detection of malicious packet dropping in wireless ad hoc networks based on privacy-preserving public auditing, Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks - WISEC '12 (2012) 87.
[25] J. Ning, S. Singh, K. Pelechrinis, B. Liu, S. V. Krishnamurthy, R. Govindan, Forensic analysis of packet losses in wireless networks, IEEE/ACM Transactions on Networking 24 (4) (2016) 1975–1988. doi:10.1109/TNET.2015.2448550.
[26] B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru, H. Rubens, ODSBR: An on-demand secure Byzantine resilient routing protocol for wireless ad hoc networks, ACM Transactions on Information and System Security 10 (4).
[27] A. T. Mizrak, Y. C. Cheng, K. Marzullo, S. Savage, Fatih: detecting and isolating malicious routers, in: 2005 International Conference on Dependable Systems and Networks (DSN'05), 2005, pp. 538–547. doi:10.1109/DSN.2005.49.
[28] J. Avonts, B. Braem, C. Blondia, A questionnaire based examination of community networks, in: Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on, 2013, pp. 8–15. doi:10.1109/WiMOB.2013.6673333.
[29] A. Neumann, B. Braem, L. Cerda-Alabern, P. Escrich, C. Barz, J. Kirchhoff, J. Niewiejska, H. Rogge, D4.3 experimental research on testbeds for community networks, Tech. rep., CONFINE Project (2014).
[30] E. Lopez, KDet: additional information, http://dsg.ac.upc.edu/esterl/KDet, accessed: 2015-03-31.
[31] D. Obenshain, T. Tantillo, A. Babay, J. Schultz, A. Newell, E. Hoque, Y. Amir, C. Nita-rotaru, Practical Intrusion-Tolerant Networks, Tech. rep., Distributed Systems and Networks Lab (2016).
[32] X. Zhang, C. Lan, A. Perrig, Secure and scalable fault localization under dynamic traffic patterns, in: 2012 IEEE Symposium on Security and Privacy, 2012, pp. 317–331. doi:10.1109/SP.2012.27.
[33] F. Rusu, A. Dobra, Statistical analysis of sketch estimators, in: ACM SIGMOD International Conference on Management of Data, 2007.
[34] R. Perlman, Network layer protocols with byzantine robustness, Ph.D. thesis, Massachusetts Institute of Technology (1988).
[35] E. Lopez, L. Navarro, Local Detection of Forwarding Faults in Wireless Community Networks, in: XXIII Jornadas de Concurrencia y Sistemas Distribuidos, 2015.
[36] Guifi.net, Guifi.net Barcelones area, http://guifi.net/en/node/2435, accessed: 2015-08-11.
[37] E. Lopez, KDet: additional information, http://dsg.ac.upc.edu/esterl/KDet, accessed: 2015-03-31.
[38] IETF, Low Extra Delay Background Transport (LEDBAT), http://tools.ietf.org/html/rfc6817, accessed: 2015-03-08.
[39] E. Lopez, Scripts and code for KDet evaluation using OMNeT++ (2016).
URL https://github.com/esterl/kdet-omnetpp
[40] A. Varga, The omnet++ discrete event simulation system 9.
[41] G. Pei, M. Gerla, T.-W. Chen, Fisheye state routing: a routing scheme for ad hoc wireless networks, in: Communications, 2000. ICC 2000. 2000 IEEE International Conference on, Vol. 1, 2000, pp. 70–74 vol.1. doi:10.1109/ICC.2000.853066.