KDet: Coordinated Detection of Forwarding Faults in Wireless Community Networks

Ester López
Department of Computer Architecture
Universitat Politècnica de Catalunya
Barcelona, Spain
Email: esterl@ac.upc.edu

Leandro Navarro
Department of Computer Architecture
Universitat Politècnica de Catalunya
Barcelona, Spain
Email: leandro@ac.upc.edu

***Abstract*—Wireless Community Networks (WCN) are grass-root initiatives that leverage the connectivity gap by providing an alternative ownership model for IP-networks, where every piece of equipment is managed and owned in a decentralized fashion by members from the community. However, 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. Here we present KDet, a distributed 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.**

I. 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. Many issues can hinder connectivity; they are generally classified as control-plane faults or data-plane faults. We focus on the data-plane, and more specifically, on finding traffic forwarding faults in the context of Wireless Community Networks (WCN), which are especially vulnerable to forwarding failures because of the following:

- 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.
- Administration is distributed and not always handled by experts, which may result in misconfiguration errors.
- Open to everyone, which implies being also open to people with malicious intentions.

However, the context is more general as we consider crowd-sourced networks: computer networks built by citizens and organisations who pool their resources and coordinate their efforts to build network infrastructures in a rather decentralized manner. Typical scenarios can be long-running networks in remote locations, mass public events with large audiences, or in a disaster area or emergency situation.

In this paper we follow the problem formulation presented by Mizrak et al. [1], characterised by three sub-problems:

Traffic Validation How to recognise a misbehaving path, node or link? Which information is required for it?

Distributed Detection What network areas need to be monitored? How is the monitored information distributed among the network?

Response How does routing take this information into account to improve traffic delivery?

The contribution of this paper is KDet, the first distributed detection protocol that is both collusion resistant and does not rely on the knowledge of a packet's path. By decoupling the knowledge of the path from the detection protocol we give the network complete freedom for using the routing protocol of its choice, even combining several of them within the same network, common in several WCN [2], or using routing optimisations such as load balancing.

Other relevant characteristics of KDet are that it is run in a decentralized way, and it faces and solves false accusation: a monitoring node announcing false reports for another node can never cause that node to be detected as faulty.

The rest of the paper is structured as follows: first we introduce previous work on the topic (section II); then we properly define the problem and system model in section III. Section IV introduces our solution, KDet, and it is validated and analyzed in sections V and VI respectively. Finally, we discuss KDet implementation and limitations in section VII and we conclude in section VIII.

II. Background

Network connectivity is the result of a collaborative effort: a packet reaches its destination if every node in the path properly forwards it to the next hop. However, a node may exhibit faulty behavior for many reasons, e.g. misconfiguration, faulty hardware or tampering, resulting in (partial) losses of connectivity.

We see in the literature two common approaches to increase network robustness against such failures: using resilient forwarding mechanisms or detecting and reacting to faulty routers. Examples of resilient forwarding are robust flooding [3] or ACR [4]. However, because forwarding resilience

is achieved by using several paths to reach a destination, it requires modifications in the network stack (e.g. using source routing or flooding), which does not sound reasonable for many networks, and particularly for WCNs with distributed management and extreme diversity.

Then, on the detection approach we need to consider the two mechanisms involved in routing: learning the proper path to the other nodes and following the routing rules when forwarding a packet. The former relates to the control plane, and many solutions have been proposed to ensure the veracity of the routing update messages. We focus on the second mechanism: once the routing table is established, independently of the routing protocol used, a node should forward packets properly, which is referred to as data-plane detection.

Although many studies have resulted in global solutions to forwarding faults, these can usually be split into three different mechanisms, each dealing with one of the sub-problems introduced previously. Solutions for each are quite interchangeable, allowing development of solutions that combine the different pieces. Here we focus on the distributed detection sub-problem, because it is the focus of the paper, but pointers for traffic validation and response can be found later in the paper.

As explained before, distributed detection is about determining the elements on the network to monitor and how to distribute the collected information to detect faulty elements. A detection protocol has two main characteristics: its **precision** and its **scope**. A detection protocol can be more, or less, precise on the detection, assigning blame of a faulty behavior to a node, link, path, or set of nodes. Then, when an entity running the detection protocol discovers a faulty behavior, it can either keep this information to itself (local scope) or share it with the rest of the network (global scope). In a WCN, we are interested in a global scope, so that when a node is discovered as faulty we can react; however, as we will see later, the global scope is harder to achieve because it needs to face and solve false accusation (a node reporting another as faulty when it is not).

Solutions that keep information local usually focus on monitoring end-to-end paths. Some examples are ODBSR [5], SSS [6], secure traceroute [7] and those presented in [8].

Among the solutions with global scope, one of the first distributed detection protocols was proposed by Bradley et al. [9]. WATCHERS proposes to monitor every neighbor and count the packets to each destination sent through each link. Then, periodically every node floods their traffic summaries and everybody can determine the behavior of every other node. However, as proven by Hughes et al. [10], some issues with WATCHERS allow faulty routers to go undetected. DynaFL [11] and χ [12] propose a similar approach: every node N is monitored by its neighbors, who keep traffic summaries of the traffic sent, $S^{\rightarrow N}$, and received from N, $S^{\leftarrow N}$. These traffic summaries are compared to decide whether or not N is behaving properly. The difference between DynaFL and χ is concerning who performs the actual detection. In DynaFL, traffic summaries are collected by a central trusted authority, who compares and then determines which nodes are faulty. However, this approach presents two problems, because the summaries collected between neighbors about the same link cannot be compared (e.g. in Figure 1 the traffic summary that A collects about the traffic sent to B, $S^{A\rightarrow B}$, does not include traffic destined to B; and the traffic summary collected by B about the traffic received from A, $S^{B\leftarrow A}$, does not include traffic with A as source). The problems that arise are false accusation (the control authority has no means of knowing if there is any discrepancy between traffic summaries for the same link) and collusion (node A could hide the packets dropped by B by simply modifying its report $S^{A\rightarrow B}$, because it does not need to be coherent with $S^{B\leftarrow A}$ and it will not be part of the summaries considered when evaluating A). χ proposes a different approach that solves false accusation, but it is still vulnerable to collusion. In χ every neighbor will send its traffic summaries to the other neighbors through the node N being monitored, which allows N to make sure those are the expected traffic summaries by comparing them with a local version and disconnecting from nodes that are falsely accusing N. Finally, when the neighbors receive all the summaries, the behavior of N is examined by the neighbors themselves. The paper does not go into further details about how that information is later shared with the rest of the network nodes.

Only the work proposed in [1] is capable of facing both false accusation and collusion. Π_{k+2} monitors all the path segments of length 3 to $k+2$ (where k is the maximum number of adjacent faulty routers in the network) and the end points of the path segment detect it as faulty when their traffic summaries are not consistent. By detecting the whole path segment, including the nodes doing the detection, false accusation is solved, because a node accusing a path is always part of the faulty path. In addition, by monitoring every path segment of the given lengths, any set of faulty routers will be covered by one of the monitored paths and therefore detected. The main drawback of this approach is that a node needs to know the next $k+1$ hops a packet will follow to include it in the proper summary. Other works with global scope are [13], [14], and [15] but, for similar reasons, they do not solve collusion and false accusation.

In summary, several solutions have been proposed to detect forwarding faults; however, none of them are capable of solving the problems of false accusation and collusion of several faulty nodes, except Π_{k+2}. Our work presents a solution that covers false accusation and collusion, but in contrast with Π_{k+2} it does not require knowledge of paths and will detect the faulty nodes with more accuracy than Π_{k+2}. Additionally, because KDet is path independent and a data-plane solution, it can be deployed as an independent daemon on the routers without the need for modifying existing network processes.

III. Problem Statement

The goal of the KDet is to provide a secure mechanism to exchange monitoring information between network routers so that faulty routers can be detected. In a network, compromised routers can lie on behalf of each other to avoid detection:

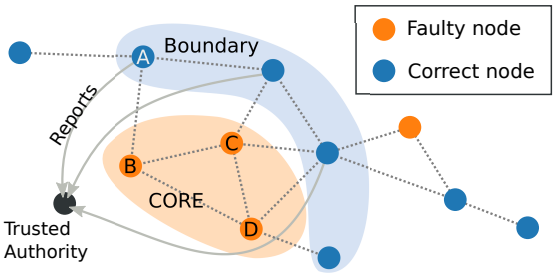

Fig. 1. Example network

e.g. in Figure 1, node A may cover for node B, reporting that it has received all the traffic that node B is dropping. The distributed detection algorithm must make sure that this situation is properly detected, usually by monitoring larger network areas.

Because we are focused on WCN, we must consider their characteristics when defining KDet. For instance, Guifi.net is a large network with over 28,000 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-areas 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, several factors will influence KDet:

- We cannot rely on a single routing protocol running on the network, neither that it will be a link-state routing protocol.
- We cannot rely on overhearing techniques, because a node may have several antennas.
- The detection protocol should be distributed, so that it scales up to thousands of nodes.

Next, we discuss in more detail the system model and the problem specification; and, using a simplified detection mechanism, we determine the main challenges for the distributed detection problem.

A. System Model

In this section we introduce the assumptions and characteristics considered while designing KDet regarding the network, traffic validation mechanism and the adversary. We also introduce some notation and elemental pieces that will be used thorough the paper.

1) Network model:

Because our network assumptions and characteristics should be consistent with those of a WCN, 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, which discard unidirectional links. We also assume that adjacent nodes agree on the traffic that they have exchanged, which can be easily achieved for unicast traffic because 802.11 acknowledges received packets.

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. A lightweight mechanism to achieve this could be SEMTOR [16]. 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 [17]. 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). However, a flooding mechanism for reports with the proper signature could be used, instead, if such entities were not available.

Finally, we make two assumptions about timing: the bound is known 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.

2) Traffic Validation function:

As explained previously, we focus on the distributed detection sub-problem, and therefore we need a valid Traffic Validation mechanism to go with KDet. Essentially, the traffic validation process is defined by a traffic summary function and a validation function.

In this paper, we will use $\boldsymbol{\mathcal{S}}(\cdot)$ to represent that summary function and S to represent the summary itself. Of course, many summaries will be involved, so we will use the notation $S^{\text{link}}_{\text{traffic}}$ for the summary of some traffic going through a specific link. For example, node A may keep a summary for all the traffic sent to B, $S^{A\to B}$; and another for the received traffic, $S^{A\leftarrow B}$. Sometimes, several links may exist between A and B, and then, we may refer to an specific link as i, having, for instance, the summary of the traffic sent with destination n through that link be $S^{\to i}_{\text{to}(n)}$.

A requirement when implementing KDet following the second strategy (section VI) is to have defined $+$ and $-$ operations over summaries. Luckily, many summary functions satisfy it: sketches [18], [8], [11], counters [9], fingerprinting [12] or sampling [8]; so they can be used for KDet.

Regarding the validation mechanism, it will be typically based on the Conservation of the Flow (CoF) principle: *traffic entering a network area should be the same as traffic leaving, except traffic destined to and generated by it* [9]; so the validation function $\boldsymbol{\mathcal{V}}$ is simply evaluating:

$$\boldsymbol{\mathcal{S}}\left(\text{traff}_{\text{in}}\right) - \boldsymbol{\mathcal{S}}\left(\text{traff}_{\text{to}}\right) \stackrel{?}{\approx} \boldsymbol{\mathcal{S}}\left(\text{traff}_{\text{out}}\right) - \boldsymbol{\mathcal{S}}\left(\text{traff}_{\text{from}}\right) \quad (1)$$

Naturally, a network may experience some packet losses without being faulty, but we expect $\boldsymbol{\mathcal{V}}(S_{\text{in}}, S_{\text{out}})$ to allow for

such differences and only return false when the behavior of the network area evaluated is underperforming, and therefore actually faulty.

It is also important to mention that the capability of KDet in detecting a specific faulty behavior is determined by the capabilities of $\boldsymbol{\mathcal{S}}$ and $\boldsymbol{\mathcal{V}}$; e.g. if the summary function is a simple counter, KDet will be able to detect packet droppers, but a faulty node may modify packets without being detected.

3) Threat model:

We use a threat model equivalent to the one proposed by Mizrak et al. [1]:

- A *p-faulty* node does not participate properly in the detection protocol. For instance, it could provide false traffic summaries or simply not participate in the protocol.
- A *t-faulty* node misbehaves in the traffic forwarding process and its behavior can be detected by the traffic validation function.
- A *faulty* node is either *p-faulty*, *t-faulty* or both.

The strength of the adversary can be measured by the biggest set of adjacent faulty nodes, expressed by k. For example, in Figure 1, $k = 3$.

We assume, that the network is sufficiently connected, so that even if the faulty nodes are removed from the routing fabric, the network is still connected and that terminal routers are not faulty with respect to the traffic they originate or consume.

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

B. Specification

KDet's concept is that a set of nodes, the core, is monitored by its boundary with the rest of the network; and it is the responsibility of the boundary to assess the behavior of the core. We define the **core** as a set of directly connected nodes, and the **boundary** of a core is every neighbor of that set of nodes which is not already part of the core (see Figure 1).

KDet can be seen as a failure detector, and as such, its 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. [1]:

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

C. Protocol χ

In the simplest scenario where there are no colluding nodes ($k = 1$), we can implement the protocol χ [12]. There, every node N is monitored by its neighbors (i.e. every node is a core and its neighbors are the boundary). Periodically, those neighbors share the perceived traffic summary of N through N and that way, if N is faulty, its neighbors will detect it and, conversely, if a neighbor sends a false report, N will notice and disconnect from it. For example, in the network in Figure 1, if B is the node being monitored, A, C and D will send to B their traffic summaries periodically; B will compare those summaries with its own perspective and will disconnect from any node sending false summaries. Then, B forwards every summary to A, C and D and they check if the Conservation of the Flow principle is satisfied; B will be detected accordingly.

Based on this idea, in this paper we show how we can move this set of monitoring nodes farther away, so that colluding nodes are kept within their boundary; and therefore we can detect them. The main challenges to solve for it to work are:

- How to make sure that every boundary node has the same set of traffic summaries?
- How to verify the authenticity of the reports and prevent false accusation?
- How to determine which cores to monitor to ensure completeness?

IV. KDet: Coordinated Detection

The main concept behind KDet is to extend the χ protocol to monitor instead of a single router, a group of them, but still maintain its properties: if the core is faulty, the boundary can detect it, and if the boundary falsely accuses the core, the core can also detect it.

This is achieved by designing KDet to satisfy two principles:

- If there are one or more t-faulty nodes in the core whose behavior influences the traffic of the rest of the network, and there are no faulty nodes in the boundary, the core will be detected as faulty.
- If there are no faulty nodes in the core, one or more faulty nodes on the boundary cannot cause the false detection of the core.

The first principle is enforced through the *boundary protocol*, the second by the *core protocol*, and their results are combined in the *coordinated detection* phase.

A. Boundary protocol

The goal of the boundary is to determine the forwarding behavior of the core. This is achieved by monitoring the traffic entering and leaving the core, checking if it satisfies the Conservation of the Flow principle, and suspecting the core if not.

More formally, every node in the boundary monitors its links with the core and keeps a summary for the incoming traffic:

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

and a summary for the outgoing traffic:

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

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, traffic summaries are exchanged with all the boundary nodes by means of robustly flooding them and their signature through the core.

After waiting a reasonable amount of time, each node expects to have the summaries of every other node in the boundary (signatures included). If some of the summaries are missing, or their signature is not valid, the core is suspected; if everything is as expected, the boundary node evaluates the behavior of the core using the validation function:

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

Then, every node in the boundary emits its verdict by sharing $V_{\text{core}}(T)$ and a bitmap that indicates which summaries were received with the corresponding TA.

B. Core protocol

Meanwhile, nodes in the core need to be on the lookout for p-faulty nodes on the boundary and disconnect from them. A correct boundary behavior is characterized by:

1) There is a single and consistent version of a link's summary.
2) $V_{\text{core}}(T)$ is consistent with the exchanged summaries.

To assess the behavior of a boundary node A, every node in the core B, connected to it (via link i) will go through the steps that follow.

First, at the end of the monitoring interval, it expects the traffic summaries from i, $S^{\rightarrow i}_{\text{core}}(T)$ and $S^{\leftarrow i}_{\text{core}}(T)$. In the case in whih there is no traffic summary, it is not properly signed, or there is more than one version, B will consider A p-faulty.

Then, the node will compare $S^{\rightarrow i}_{\text{core}}(T)$ and $S^{\leftarrow i}_{\text{core}}(T)$ with its own local version, and if they are not the same, A is considered p-faulty as well.

Finally, B compares the $V_{core}(T)$ announced by A with the one that results from combining the summaries that have been exchanged (available to any node in the core because of robust flooding). Again, if the results do not match A will be considered p-faulty.

If in any case A is considered p-faulty, then B disconnects from it.

C. Coordinated detection

At the end of each interval, the TA gathers a report from every node in the boundary that consists of a bitmap indicating which summaries were received and $V_{\text{core}}(T)$. If there is any missing summary 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:

$$\begin{aligned}&\text{core} \in \text{suspected}[T_1, T_2] \quad \wedge \\ &\exists T' \in (T_1, T_2] \big| \text{boundary}(\text{core}, T') \subseteq \text{boundary}(\text{core}, T_1) \\ &\qquad \Rightarrow \text{core} \in \text{detected}(T')\end{aligned}$$

Where $\text{core} \in \text{suspected}[T_1, T_2]$ means that the core was suspected on every interval from T_1 to T_2, and $\text{boundary}(\text{core}, T') \subseteq \text{boundary}(\text{core}, T_1)$ implies that for every link existing between the core and the boundary at time T' the same link also exists at time T_1.

V. KDet validation

To prove the correctness of KDet we will first prove that, no matter what faulty nodes do, KDet's principles, defined in section IV, are always satisfied. Then, we will use those principles to prove that by choosing an appropriate set of cores to monitor, KDet becomes k-accurate and k-complete under the assumption of k being the maximum number of directly connected faulty routers.

The first principle, enforced by the boundary protocol, is equivalent to:

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

Because the core is faulty, we know that summary functions as measured by the boundary satisfy:

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

Then, by the definition of KDet, the core cannot avoid being suspected, because if it robustly floods the summaries, it will be suspected and if it does not, because not every summary will be received, it will also be suspected. Modifying the reports is not possible because they are signed.

However, the core may avoid detection by modifying its boundary, ensuring that:

$$\text{boundary}(\text{core}, T') \nsubseteq \text{boundary}(\text{core}, T_0)$$

That is, on each interval, it needs to reduce its boundary 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.*

To continue, the second principle of KDet, enforced by the core protocol, can be expressed as:

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

Therefore, it is sufficient to prove that every time the core is suspected, its boundary changes in the next interval.

By definition, we know that if a core is not faulty, then the real traffic summaries satisfy:

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

And a core can only be suspected if one of the nodes of the boundary claims that:

1) Not every summary was received
2) Summaries were received but $V_{\text{core}}(T) = \text{false}$

Because we use robust flooding through correct nodes, 1) can only occur if not every summary was shared (or there was more than one version or it was not properly signed) or a boundary node (A) misreports the summaries it has received. In the first case, the core will know which summary is missing (S^{A-B}), and then B will disconnect from A, making the boundary change in the next interval. In the second case, every core node connected to A will realise it is misreporting the received summaries and will disconnect from it, and therefore the boundary also changes.

In case 2), if every summary is received but there is a node A that reports $V_{\text{core}}(T) = \text{false}$, that implies that either one of the flooded summaries has been modified ($S^{A\to B}_{\text{core}} \neq S^{B\leftarrow A}_{\text{core}}$) or $V_{core}(T) \neq \boldsymbol{\mathcal{V}}\left(\sum_{\forall i} S^{\to i}_{\text{core}}(T), \sum_{\forall i} S^{\leftarrow i}_{\text{core}}(T)\right)$; i.e. the reported V_{core} is not correct. In the first case, B will realise that the flooded summary, $S^{A\to B}_{\text{core}}$, is not the same as its local version, $S^{B\leftarrow A}_{\text{core}}$, thus disconnecting from A (and therefore changing the core's boundary). In the second case, every node connected to A will disconnect from it (changing the boundary again).

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

Now, let's assume that for a given k we define the set of cores to monitor as the sets of connected nodes that are smaller than or equal to k:

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

Thanks to Lemma 1, 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. And given the definition of C, such core will always have a size below k.

If the set of faulty nodes 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 C and c_i will have a correct boundary, because there are no direct connections with any other f_j. Therefore, applying Lemma 2, every f_i will be detected and since $|\text{c}_i| = |f_i| \leq k$, KDet is k-complete.

VI. KDet analysis

Since we have proven the accuracy and completeness of KDet, in this section we study its cost. This is mainly determined by the memory required to store the state of the protocol and the network overhead required to share traffic summaries.

A. 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 $-$ operation, a node can follow two strategies to save the required summaries:

1) Keep a summary for every core that it monitors (without considering the core's $\text{from}$and to traffic) and for every link that connects to it.

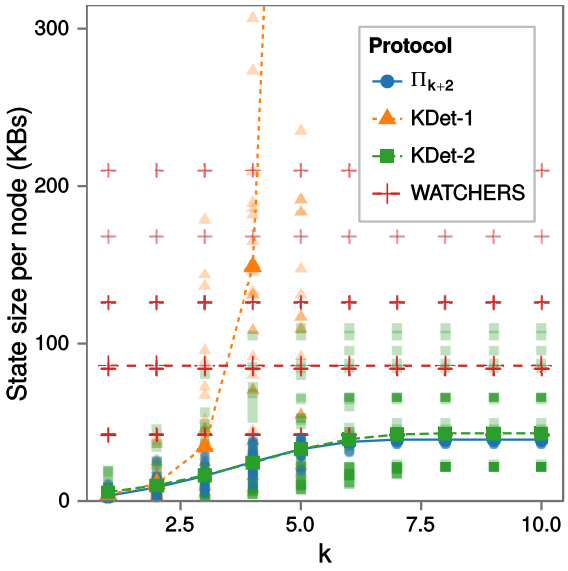

Fig. 2. State size

2) Keep for every link a summary of the incoming and outgoing traffic and additionally, for each link, the traffic from and to every node in k-hops, so that a core summary, $S^{\to i}_{\text{core}}$, can be computed as:

$$S^{\to i}_{\text{core}} = S^{\to i} - \sum_{\forall n \in \text{core}} S^{\to i}_{to(n)} \quad (2)$$

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.

Given k, if N is the number of nodes in the network and R is the maximum node's degree, then the number of cores a node monitors is below $O(R^k)$ and, for the first strategy, since it keeps a summary per link, its cost will be $O(R^{k+1})$.

The second strategy keeps a summary per link and at most another per node and link (there will not be traffic from/to every node on every link), therefore its cost is $O(R * N)$. In [1] Mizrak et al. show that WATCHERS cost is $O(R * N)$ and Π_{k+2}, $O(min(R^{k+1}, N))$. If we choose the optimal strategy among the two here proposed, our solution will have a tendency that is between WATCHERS and Π_{k+2}.

Figure 2 shows the state size stored by each node for the guifi.net Barcelones area [19] assuming summaries of 500B [11], [8], 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 1 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 110KB, which sounds reasonable. Similar results have been found for different network topologies. See [17] for these results and more detailed figures.

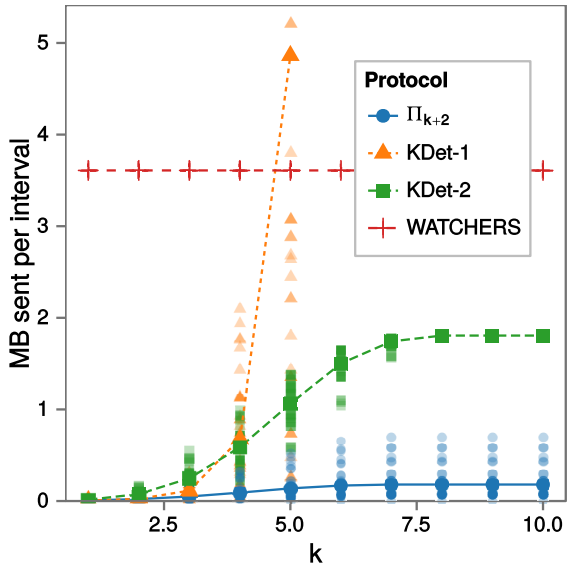

Fig. 3. Network overhead

B. 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 ($S^{\rightarrow i}_{\text{core}}$ and $S^{\leftarrow i}_{\text{core}}$).
2) Robustly flood every summary maintained following the second strategy presented before ($S^{\rightarrow i}$ and $S^{\leftarrow i}$ for every link i and $S^{\rightarrow i}_{to(n)}$ and $S^{\leftarrow i}_{from(n)}$ 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.

Figure 3 shows the data (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 complete path of a packet. Given that at most around 2MB of data needs to be shared per link, a reasonable time interval would be around 1 minute, so that the overhead is kept below 250kbps. Moreover, techniques such as LEDBAT[20] can be used to avoid clogging the network links with the protocol's traffic.

VII. Discussion

In this section we discuss further some of the aspects of KDet, the assumed k and the limitations imposed by the Traffic Validation mechanisms.

A. KDet Implementations

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

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 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.

B. The largest number of adjacent colluding nodes (k)

We have proposed a solution for detecting forwarding faulty nodes in the presence of collusion and we have characterized KDet as a function of k or the largest number of adjacent colluding nodes. But k may seem a concept too abstract, so here we try to give a more concrete example. Figure 4 shows the probability of detection when deploying KDet for a given

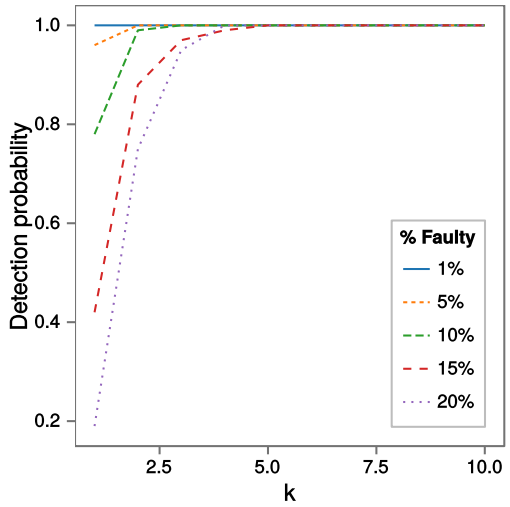

Fig. 4. Detection probability

value of k on the Barcelones network, but instead presents a percentage of randomly selected nodes acting as faulty.

As we can see, even with 20% of faulty nodes, if we deploy KDet assuming $k = 3$, the probability of detection is still 95%. Similarly, on each network, the proper value of k can be chosen by studying the number of nodes that may fail within a protocol's interval in an area of the network (k will be equal to that number).

C. Limitations

KDet relies on a precise traffic validation mechanism; however, some of the proposed mechanisms in the literature, such as sampling and sketches, do not provide an accurate measure, but just an estimation with some probabilistic warranties. In such cases, the completeness and accuracy properties should be modified to accommodate the probabilistic nature of the validation function, $\mathcal{V}$, in a similar fashion to Goldberg et al. [8].

VIII. Conclusions and Future Work

In this paper we have presented KDet, a detection protocol for forwarding faults tailored to the requirements of WCNs,but we believe that it can be used in other types of networks, especially in crowdsourced networks, which share some of the critical characteristics of community networks. We have proven KDet's correctness and studied its performance compared with Π_{k+2} and WATCHERS, 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.

Regarding future work, we want to explore further the effect of non-deterministic traffic validation mechanisms over KDet and also, study methods to randomize the algorithm so that we can further reduce its overhead cost.

Acknowledgement

This work is supported by European Community Framework Programme 7, FIRE Initiative project "Community Networks Testbed for the Future Internet" (CONFINE), FP7-288535 and the Spanish government contract TIN2013-47245-C2-1-R.

References

[1] A. Mizrak, K. Marzullo, and S. Savage, "Fatih: Detecting and Isolating Malicious Routers," in *International Conference on Dependable Systems and Networks*, 2005.

[2] J. Avonts, B. Braem, and C. Blondia, "A questionnaire based examination of community networks," in *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, 2013.

[3] R. Perlman, "Network layer protocols with byzantine robustness," Ph.D. dissertation, Massachusetts Institute of Technology, 1988.

[4] D. Wendlandt, I. Avramopoulos, D. Andersen, and J. Rexford, "Don't secure routing protocols, secure data delivery," in *ACM Workshop on Hot Topics in Networks*, 2006.

[5] B. Awerbuch, R. Curtmola, D. Holmer, C. Nita-Rotaru, and H. Rubens, "ODSBR: An on-demand secure byzantine resilient routing protocol for wireless ad hoc networks," *ACM Transactions on Information and System Security*, 2008.

[6] B. Barak, S. Goldberg, and D. Xiao, "Protocols and Lower Bounds for Failure Localization in the Internet," *Lecture Notes in Computer Science*, 2008.

[7] V. N. Padmanabhan and D. R. Simon, "Secure traceroute to detect faulty or malicious routing," *ACM SIGCOMM Computer Communication Review*, Jan. 2003.

[8] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford, "Path-quality monitoring in the presence of adversaries," *ACM SIGMETRICS Performance Evaluation Review*, Jun. 2008.

[9] K. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. Olsson, "Detecting disruptive routers: a distributed network monitoring approach," *IEEE Network*, 1998.

[10] J. J. R. Hughes, T. Aura, and M. Bishop, "Using conservation of flow as a security mechanism in network protocols," in *IEEE Symposium on Security and Privacy*, 2000.

[11] X. Zhang, C. Lan, and A. Perrig, "Secure and scalable fault localization under dynamic traffic patterns," in *IEEE Symposium on Security and Privacy*, 2012.

[12] A. Mizrak, S. Savage, and K. Marzullo, "Detecting Malicious Packet Losses," *IEEE Transactions on Parallel and Distributed Systems*, Feb. 2009.

[13] F. S. Proto, A. Detti, C. Pisa, and G. Bianchi, "A Framework for Packet-Droppers Mitigation in OLSR Wireless Community Networks," *IEEE International Conference on Communication*, 2011.

[14] R. Matam and S. Tripathy, "AFC: An effective metric for reliable routing in wireless mesh networks," *IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 2013.

[15] V. Desai, S. Natarajan, and T. Wolf, "Packet forwarding misbehavior detection in next-generation networks," *IEEE International Conference on Communications*, 2012.

[16] A. Neumann, B. Braem, L. Cerda-Alabern, P. Escrich, C. Barz, J. Kirchhoff, J. Niewiejska, and H. Rogge, "D4.3 experimental research on testbeds for community networks," CONFINE Project, Tech. Rep., 2014.

[17] "KDet: additional information," http://dsg.ac.upc.edu/esterl/KDet, accessed: 2015-03-31.

[18] F. Rusu and A. Dobra, "Statistical analysis of sketch estimators," in *ACM SIGMOD International Conference on Management of Data*, 2007.

[19] "Guifi.net Barcelones area," http://guifi.net/en/node/2435, accessed: 2015-03-23.

[20] "Low Extra Delay Background Transport (LEDBAT)," http://tools.ietf.org/html/rfc6817, accessed: 2015-03-08.