UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Arquitectura de Computadors

Forwarding fault detection in Wireless Community Networks

Ester López Berga

PhD Thesis in Computer Architecture
by the Universitat Politècnica de Catalunya

Advisor: Leandro Navarro Moldes

Barcelona, Spain 2017

Abstract

The most important service a Wireless community network (WCN) offers is connectivity. However, WCN's connectivity is fragile because of their intrinsic characteristics: inexpensive hardware that can be easily accessed; decentralized management, sometimes by non-experts, and open to everyone; making it prone to hardware failures, misconfigurations and malicious attacks. The goal of this thesis is to increase routing robustness in WCN using forwarding fault detection, so that we can find and fix problematic routers. Forwarding fault detection can be explained as a four-step process: first, nodes monitor and summarize the traffic on the network; then, those traffic summaries are shared among peers, so that, by analyzing the relevant ones, we can asses the behavior of a given node. Finally, when we find a faulty router, a response mechanism is triggered to solve the issue.

On this thesis we focus on the first three subproblems. First, on monitoring, we study and characterize the distribution of the error of sketches, a traffic summary function that is resilient to packet dropping, modification and creation and that has better statistical guarantees than sampling. Second, we propose KDet, a traffic summary dissemination and detection protocol that, unlike previous solutions, is resilient to collusion and false accusation without the need of knowing a packet's path. KDet is a solution adapted to WCN, because it can be deployed without the need of modifying its current network stack. Finally, we consider the case of nodes with unsynchronized clocks, and we propose a traffic validation mechanism based on sketches that is capable of discerning between faulty and non-faulty nodes even when the traffic summaries are misaligned, i.e. they refer to slightly different intervals of time.

Resumen

El servicio más importante ofrecido por una red comunitaria es la conectividad. Sin embargo, las redes comunitarias son especialmente vulnerables a errores en la retransmisión de paquetes de red, puesto que están formadas por equipos de gama baja, de fácil acceso; están gestionados de manera distribuida y no siempre por expertos, y además están abiertas a todo el mundo; con lo que de manera habitual presentan errores de hardware o configuración y son sensibles a ataques maliciosos. Para mejorar la robustez en el enrutamiento en estas redes, proponemos el uso de un mecanismo de detección de routers defectuosos, para así poder corregir el problema. La detección de fallos de enrutamiento se puede explicar como un proceso de cuatro pasos: el primero es monitorizar el tráfico existente, manteniendo desde cada punto de observación un resumen sobre el tráfico observado; después, estos resumenes se comparten entre los diferentes nodos, para que podamos llevar a cabo el siguiente paso: la evaluación del comportamiento de cada nodo. Finalmente, una vez detectados los nodos maliciosos o que fallan, debemos actuar con un mecanismo de respuesta que corrija el problema. Esta tesis se concentra en los tres primeros pasos, ya que podemos utilizar simplemente una notificación para el mecanismo de respuesta. Respecto a los resumenes de tráfico, presentamos un estudio y caracterización de la distribución de error de los sketches, una estructura de datos que es capaz de resumir el tráfico y es resistente a la pérdida, manipulación y creación de paquetes; además, tiene mejor resolución que el muestreo. Para cada tipo de sketch, definimos una función de distribución que caracteriza el error cometido, de esta manera somos capaces de determinar con más precisión el tamaño del sketch requerido bajo unos requisitos de falsos positivos y negativos. Después proponemos KDet, un protocolo de diseminación de resumenes de tráfico y detección de nodos erróneos que, a diferencia de protocolos propuestos anteriormente, no require conocer el camino de cada paquete y es resistente a la confabulación de nodos maliciosos. KDet puede ser desplegado sin modificar los sistemas y protocolos de red ya existentes, algo imprescindible en el contexto de las redes comunitarias. Por último, consideramos el caso de nodos con relojes desincronizados, y proponemos un mecanismo de detección basado en sketches, capaz de discernir entre los nodos erróneos y correctos, aún a pesar del desalineamiento de los sketches (es decir, a pesar de que se refieran a intervalos de tiempo ligeramente diferentes).

Contents

List of Publications

[P1] Ester Lopez and Leandro Navarro. Local Detection of Forwarding Faults in Wireless Community Networks. In *XXIII Jornadas de Concurrencia y Sistemas Distribuidos*, pages 1–15, 2015. (page 27, 34, 50)

[P2] Ester Lopez and Leandro Navarro. Tight bounds for Sketches in Traffic Validation. *14th IEEE International Conference on Networking, Sensing and Control*, 2017. (page 23)

[P3] Ester Lopez and Leandro Navarro. Coordinated Detection of Forwarding Faults in Wireless Community Networks. *Journal of Network and Computer Applications*, pages 1-–20, 2017 (under review). (page 26)

[P4] Ester López and Leandro Navarro. KDet: Coordinated detection of forwarding faults in wireless community networks. In *Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA - Volume 01*, TRUSTCOM '15, pages 734–741, Washington, DC, USA, 2015. IEEE Computer Society. (page 26)

[P1] Ester Lopez and Leandro Navarro. Byzantine Failure Detection in Wireless Mesh Routing. In *XXII Jornadas de Concurrencia y Sistemas Distribuidos*, pages 1–15, 2014. (page)

Other Publications

The following are other publications written during the duration of the thesis that relate to wireless community networks but not to the forwarding fault detection problem:

[P5] Axel Neumann, Ester López, and Leandro Navarro. Evaluation of mesh routing protocols for wireless community networks. *Elsevier Computer Networks*, 93(P2):308–323, December 2015.

[P6] Axel Neumann, Ester Lopez, and Leandro Navarro. An evaluation of bmx6 for community wireless networks. In *Proceedings of the 2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, WIMOB '12, pages 651–658, Washington, DC, USA, 2012. IEEE Computer Society.

[P7] Axel Neumann, Ivan Vilata, Xavier Leon, Pau Escrich Garcia, Leandro Navarro, and Ester Lopez. Community-lab: Architecture of a community networking testbed for the future internet. In *Proceedings of the 2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, WIMOB '12, pages 620–627, Washington, DC, USA, 2012. IEEE Computer Society.

Chapter 1

Introduction

Wireless community networks (WCN) are experiencing an exponential growth thanks to the popularization of non-expensive WiFi devices, open software solutions for mesh networks and because Internet access has become a key element for individual and collective participation in society. WCN are IP-based network infrastructures built by citizens that support and provide services to the community behind it. They grow organically, as the needs of the community arise; most of the time, there is no network planning involved, but when a new user decides to join the network, a new link is set up, connecting that new user with the closest existing one. To date, these networks have presence all over the world, some of them with thousands of nodes[56].

But as one can imagine, keeping these networks up and running is not a mere trifle; WCN are more vulnerable to failures than commercial networks because of their intrinsic characteristics:

- They are built using inexpensive hardware installed outdoors, which can result in hardware failures because of the rough conditions and, because of their location, they can be physically tampered, allowing an adversary to launch an attack from inside the network.
- Each node belongs and is usually managed by the user that registered it. As a consequence, the network nodes are managed in a decentralized way and sometimes by inexperienced people, making nodes misconfiguration more likely.

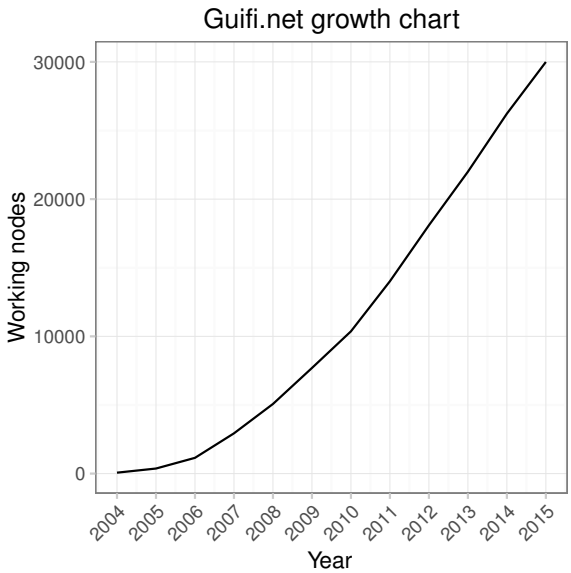


Figure 1.1: Guifi.net growth.

- Finally, the network is open to everyone, so, again, a malicious user could run faulty routing protocols to compromise the network.

As an example, in Guifi.net, users experience from time to time what is known as a "pollo de rutas" (from Spanish *a mess of routes*), which means that the routing tables become corrupted and connectivity is broken. The cause may be a faulty route announcement, routing table overflow because of low memory, hardware issues, software bugs, etc. In such occasions, usually an experienced user logs into the involved nodes hopping from one 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.

As we can see, developing monitoring tools that automate the process of detecting failures and localizing them would be extremely practical for WCN's users, reducing and easing the process of finding faults, and as such, it would have a considerable impact on their quality of service. However, increasing

the routing robustness on such networks is specially challenging because of the following:

- WCN are incredibly heterogeneous in terms of hardware and software.
- They run different routing protocols [6], not only from one network to another, but even within the same network is common to find several routing protocols interacting at different levels.
- The nodes have limited capacity, so the proposed solutions should have low cost in terms of computation and memory;
- Since we are talking about wireless networks, the solutions should be also optimized in terms of network bandwidth.
- Finally, because of their decentralized management, the best solution should be one that can be installed as an independent daemon, that can give useful feedback, even when it is not running in all the nodes of the network. This decoupling is also know as the unbundled management architectural principle [43].

Hence, the goal of this thesis is to find ways to improve the routing robustness of WCN by studying in depth the different parts involved in the location of forwarding faults, providing solutions that match the requirements of WCN.

1.1 Problem statement

End-to-end connectivity is achieved by properly routing network packets. WCN networks use proactive routing protocols, and therefore, routing can be divided in two different steps; first, the routing protocol must learn the path between every pair of network nodes within the network; then, when there is a packet that needs to be forwarded, nodes must properly choose the next hop, using the information learnt by the routing protocol.

In normal operation, none of these steps is a challenge as there are many different mesh routing protocols like OLSR, BMX6 or B.A.T.M.A.N that are able to find the best path between nodes and build the routing table that will be used for packet forwarding afterwards; however, connectivity may

be impaired in both steps, either by participating incorrectly in the routing protocol (causing the routes learnt to be incorrect) or by not following the rules imposed by the routing table. Solutions that focus on the first part are "control-plane solutions", as they ensure that the information shared through control messages is truthful; and solutions that focus on the second part are "data-plane solutions", as they rely on the data traffic instead. And, although both steps need to be secured against failures, because control-plane solutions depend on the specific routing protocol being used, it is out of the scope of this thesis; however, there are already some solutions that work on ensuring the veracity of the path learned for the different routing protocols, like SEMTOR [37] (for BMX6), secure BGP [26] (for BGP) or S-RIP [57] (for RIP).

An alternative approach, also from the control-plan perspective, is to focus on increasing the network resilience, by means of sending packets with some redundancy such as robust flooding [41, 42] or intrusion-tolerant overlays [38].

Regardless, for the case of WCN, focusing on the problem from the data-plane perspective is more suitable, because it allows us to withdraw any assumption regarding the routing protocols and deploy our protocol as an independent daemon, without any need of modifying the network stack. Moreover, if there is any problem on the routes learned through the routing protocol, the forwarding fault protocol will still be able to detect that some packets are not being delivered. Another reason to focus on detection rather than in resilience, is that in WCNs, network bandwidth is a scarce commodity, and resilient solutions require redundancy, for instance, by sending packets through several paths or repeatedly.

Thorough this thesis, we divide the problem of forwarding fault detection into four different steps, inspired in Mizrak's division [36]. Consider, without any loss of generality, the network on Figure 1.2; we have several monitors (A, B and C) that monitor a network area, M, to determine if its forwarding behavior is faulty or not using only data-plane information, i.e. the actual data traffic being forwarded. The four steps that any fault detection protocol will go through are:

Traffic Summarization First of all, the monitors will need to keep track of the traffic sent and received to M. Because keeping all the packets seen

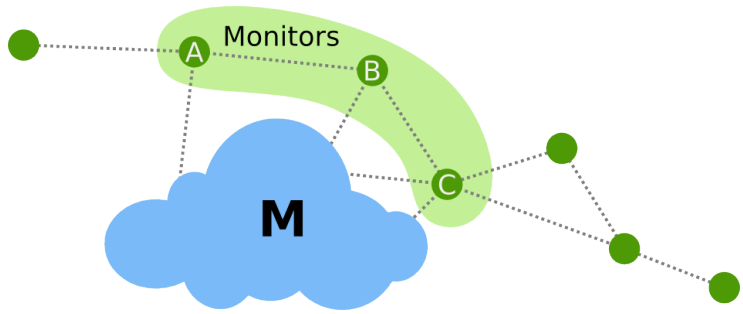


Figure 1.2: Example network

is very expensive, monitors must use a summary function that keeps the relevant information about the data streams but requires less space.

Summary dissemination 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. Once collected all the traffic summaries, we need to determine whether M is faulty or not, by examining the information available. 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 [10].

Response Finally, if we detect any faulty behavior, we must react, so that faulty nodes do not disrupt the network connectivity.

We address each of these sub-problems on a chapter of this thesis, except for the case of *response*, which is briefly addressed on Chapter 2, as it is considered out of the scope of this thesis.

1.2 Methodology

Thorough this thesis we use mathematical derivations, experimental evaluations, OMNeT++ simulations and theoretical proofs to analyze, compare and

evaluate different solutions proposed for each sub-problem of forwarding fault location. On each case, we choose the technique that is more suitable for the problem at hand; but what remains constant on every step of this thesis is that we strive for making all our results reproducible. We do so by sharing the necessary code to reproduce our experiments, the scripts to reproduce the figures and, whenever reasonable, the *csv* files obtained from our experiments. Besides, when possible, we use data from the real world.

1.3 Thesis organization

The rest of this thesis is organized as follows:

- Chapter 2 lists and describes previous solutions that have been proposed by the literature, and breaks them down into the different sub-problems that we have introduced.
- Chapter 3 summarizes the thesis contributions.
- Chapter 4 focuses on the traffic summary functions. More specifically, we focus on sampling and sketches and we describe the accuracy of both solutions.
- Chapter 5 presents a summary dissemination protocol for networks without a link-state routing protocol resistant to collusion and false accusation.
- Chapter 6 discusses traffic validation mechanisms and describes a traffic validation mechanism that does not require time synchronization.
- Chapter 7 concludes and summarizes the findings of this thesis.

Chapter 2

State of the art

As we know, network connectivity is the result of a cooperative effort: it takes the collaboration of every router in the path for a packet to reach its destination. As a consequence, if one of those routers fails to properly forward traffic, network connectivity will be hindered. Because network connectivity depends on the cooperation of every node in the network, it is fragile and mechanisms to protect it should be designed and deployed.

The first effort in increasing routing robustness was proposed by Perlman on her thesis [41]: robust flooding. Robust flooding relies on buffering and flooding packets to ensure that resources are fairly allocated and that if there is a correct path between two nodes, then they can communicate. Her solution increases connectivity resilience by means of redundancy (packets are sent repetitively and through multiple paths); however, her proposal requires changes in the network stack that we do not see feasible in WCN, because they require the modification of hundreds of nodes that are not managed by a single entity. Obenshain et al. solution [38] also focuses on resiliency, but it avoids modifying the network stack by using an overlay for critical traffic. The intrusion-tolerant overlay that they propose uses redundancy at the network level, by using several ISPs, and at the overlay level, by forwarding the overlay messages either through several paths or using constrained flooding. Still, this overlay cannot be used to forward all traffic, but just priority traffic, as it is too costly in terms of network overhead and it is more oriented to traditional networks, than WCN.

If we consider the problem from a bottom-up perspective and the current way networks operate, we can see that network connectivity resilience comes from securing the two mechanisms involved in packet delivery: first, routers need to discover the proper path to a packet's destination; and, second, they have to forward the packet as dictated by the routing protocol. Ensuring correct path discovery is usually called control-plane solutions, as they focus on the control plane, and always depend on the routing protocol. Examples of solutions proposed in the literature are JiNao [59] for OSPF, Whisper [52] and S-BGP [26] for BGP, LT-OLSR [23] for OLSR and SEAD [24] based on DSDV.

On the other hand, we have data-plane solutions, that guarantee that the forwarding process is working properly. As Perlman, we can use redundancy to ensure that packets reach their destination, or, to keep the network bandwidth consumption low, we can focus on the problem from the detection perspective: nodes are monitored, evaluated and a response mechanism is triggered when a problem is detected. This is the approach taken by this thesis, and we divide the forwarding fault detection problem into several pieces, that will be addressed one by one on its different chapters.

The first piece of the puzzle is **traffic monitoring and summarization**. To detect traffic anomalies, traffic must be monitored at different points of the network and compared to determine the network behavior. Monitoring can be done by special nodes strategically placed on the network, but for convenience it may be the network nodes themselves that monitor incident paths or neighboring nodes. The unit being monitored can be a link, a path, a node or a set of nodes, and for every unit a summary of the traffic going through it will be kept. Because not all the traffic information is kept, the capabilities of the detection mechanism will be highly influenced by the traffic summary function used. We review the different traffic summary functions and their characteristics in section 2.1.

Then we need a mechanism to compare the different traffic summaries and determine whether the monitored unit is behaving as expected or not. This is what we call the **Traffic Validation** (TV) mechanism, since it validates the traffic for that unit. We discuss TV functions in section 2.3.

But because monitored information is local, we need to share it so that the TV function can be applied. How this information is shared and who validates the traffic is determined by the **distributed detection** protocol (section 2.2).

Finally, we need a **response** mechanism to react when a failure is detected. Different response mechanisms are presented in section 2.4.

2.1 Traffic Summary functions

Traffic summary functions are used by traffic monitors to retain relevant information about the traffic at a certain network point without needing to keep a copy of the whole traffic. In our case, because the goal is to detect forwarding faults, the relevant information will be the one that allows us to detect such failures. Mizrak et al. [35] divide these failures into 5 different threats:

- Packet loss: the faulty network area drops a subset of the packets that should be forwarded.
- Packet fabrication: the faulty network area creates new packets with an IP it does not own.
- Packet modification: the faulty network area modifies or corrupts the packets that should forward. A specific example of packet modification is reducing more than expected the TTL field, which could lead to falsely detecting some other network areas as faulty.
- Packet reordering: the faulty network area may send the packets it forwards in a different order than received.
- Time behavior: the faulty network area may introduce different delays to a subset of the packets it should forward.

The strength of a traffic summary function relates to the capability of detecting each of these failures and its accuracy relates to how accurately it allows to measure such failures; and because traffic summary functions keep just some information about the traffic, they compromise their strength and/or their accuracy, as we will see in the examples below.

The simplest summary function is counting the number of packets going through the monitored point. Solutions like WATCHERS [10] or AudIt [5] keep counters for flows of different granularity to detect packet loss and packet fabrication. Of course, because counters keep very little information about the traffic being examined, they allow us to detect only packet loss or packet fabrication and only when they do not happen at the same time, but they are very precise. On the other side of the spectrum we have packet fingerprinting, used in many solutions like Mizrak et al. [36], but also all the solutions based on acknowledging every packet [8, 17, 15, 19], which consists on keeping a small value per packet (its digest), e.g. the result of applying to a hash function to it. In this case, as long as the hash function is second pre-image resistant, we would be capable of detecting any type of attack if the fingerprints are exchanged in a timely fashion, and all but time behavior anomalies if they are exchanged grouped together at the end of an interval.

In some cases, packet fingerprinting is too costly because the summary is not small enough, so the network overhead when sharing the digests is too high. In those cases, we can keep only a random sample of those fingerprints as traffic summary. Sampling has been proposed by Goldberg et al. [20], Desai et al. [18] and Zhang et al. [60] among others. The most important point on a sampling summary mechanism is the strategy that selects the sampled packets. The sampling strategy must make sure that the monitored area cannot predict which packets will be sampled or it could modify its behavior based on whether a packet will be sampled. Goldberg et al. [20] sampling strategy is to sample based on a secret key that is not known by the monitored area. As a consequence, it will have to keep a copy of all the fingerprints if it wants to be able to detect false accusation (more on false accusation in section 2.2), and tight time synchronization is needed since the key changes between intervals. The synchronization required will depend on the interval and the needed accuracy [21]. Desai et al. [18] propose a hash-based delay sampling technique, which keeps all the fingerprints into a buffer, until a packet with a hash below a threshold is received. This packet is called the initiator, and its hash will be used to determine which packets in the buffer will be shared. In this case, every node involved needs to keep the fingerprints of every packet for a given interval, but we only require loose synchronization (e.g. using NTP is enough). But hash-based delay sampling will only work to monitor network paths, because every node involved needs to receive the same set of

packets, to be sure that the initiator packet is received by everyone. Sampling techniques will allow us to detect packet drop, modification and fabrication, and depending on the sampling probability, packet reordering. But because the sampling probability is likely to be small, so will be the probability of detecting packet reordering.

Sampling provides an estimation of the packet drop, modification or fabrication percentage based on a subset of the data traffic, so the traffic summary function now compromises the accuracy to reduce the memory and network bandwidth required. Similarly, we can use other data structures that give us also a probabilistic estimation about the traffic, but using all the packets as input instead of just some: sketches. Sketches are a matrix of counters which are updated every time a new packet arrives. Benefits of sketches compared with sampling is that they are more accurate [20] and their size is bounded; the drawback is that they cost slightly more to compute in terms of CPU, though the cost of updating them is still reasonable. Sketches have been proposed as traffic summary functions for traffic validation by Goldberg et al. [20] and Zhang et al. [61]. Small sketches are vulnerable to second pre-image attacks, and therefore should be based on a secret key. As before, this will imply that we require tight time synchronization; and, in case of considering false accusation, the monitored network area should keep the fingerprints of all the packets forwarded during an interval. However, when the sketch is big enough a key is no longer necessary and neither time synchronization nor storing all fingerprints (just the sketch instead). Another difference between sketching and sampling is that in some cases, we can share the packet fingerprint of the sampled packets in a timely fashion, so time misbehavior can be detected; whereas for sketches, they will be always shared after a period of time, so time misbehavior cannot be detected.

Shu and Krunz [51] propose sharing a bitmap of the received packets signed using homomorphic linear authentication (HLA) to ensure its truthfulness. This approach is only valid for path monitoring and, moreover, it requires the modification of data traffic, since packets should include an HLA signature.

Finally, very little has been proposed to study anomalies regarding time behavior. AudIt [5] and DynaFL [61] propose to keep the average arrival time to be able to determine if too much delay has been introduced for a given flow.

In ideal conditions, overhearing techniques are able to detect any type of attack in wireless networks. Overhearing consists on nodes listenint in promiscuous mode to ensure that neighbors forward the packets sent to them, and because every node has complete information on the node it monitors, there is no need of sharing traffic summaries. However, overhearing techniques are vulnerable to stealthy attacks [27], where the faulty node could, for instance, forward the packet to a non-existent neighbor or without sufficient power. Moreover, they assume that nodes have a single antenna, which is not always the case in WCNs.

Table 2.1 summarizes each traffic summary function discussed in terms of its Standard Error (SE) and the type of attacks it can detect: packet dropping, fabrication, modification, reordering and delay. In some cases, as we have discussed, some failures may be detected depending on other factors of the detection protocol (like when the summaries are shared), those are marked with a "∼".

	SE	**Drop**	**Fab.**	**Mod.**	**Order**	**Delay**
counters	0	✓	✓	✗	✗	✗
fingerprinting	0	✓	✓	✓	✓	∼
keyed sampling	$\sqrt{\frac{p \cdot (1-p)}{\#\text{pkts} \cdot P_{\text{samp}}}}$	✓	✓	✓	∼	∼
delayed sampling	$\sqrt{\frac{p \cdot (1-p)}{\#\text{pkts} \cdot P_{\text{samp}}}}$	✓	✓	✓	∼	✗
sketches	$\frac{P_{\text{drop}}}{\sqrt{2 * \text{size}}}$	✓	✓	✓	✗	✗
bitmap + HLA signatures	–	✓	✓	✓	∼	✗
average arrival time	–	✗	✗	✗	∼	✓

Table 2.1: Precision and strengths of each summary function

On the other hand, Table 2.2 summarizes how costly the summary functions are in terms of processing cost, network bandwidth; which types of network

areas they can monitor; whether they require or not time synchronization and modifying the network stack.

	CPU	Bandwidth	Area	Sync	Network stack
counters	very low	constant (low)	any	loose	unchanged
fingerprinting	very low	proportional (high)	any	none	unchanged
keyed sampling	very low	proportional (low)	any	tight	unchanged
delayed sampling	very low	proportional (low)	path	loose	unchanged
sketches	low	constant (low)	any	none	unchanged
bitmap + HLA signatures	high	proportional (high)	path	none	sign packets + id
average arrival time	very low	constant (low)	path	tight	unchanged

Table 2.2: Cost summary for each summary function

2.2 Summary dissemination and distributed detection

As we have mentioned in the previous section, in the case of WCNs we cannot use overhearing techniques, because they miss the traffic from directional antennas or antennas using a different frequency. Therefore, we need a protocol that determines how the traffic summaries are shared and who validates the traffic for each node.

In the easiest scenario, let's consider a single traffic flow from a source node to a destination and assume that none of them is faulty regarding that traffic flow. In this case, if we only involve the source and the destination in the detection process, we don't have to worry about faulty nodes interfering in the detection other than by dropping the protocol messages. But, at the same time, we cannot locate the fault with precision, just notice that the path is misbehaving

and assign the blame to the whole path. Solutions in the literature that take this approach are:

- **Stealth probing** [7] uses a secure tunnel between the source and destination to transmit data packets and either active or passive probes. Then, the drop probability of the path is estimated by measuring how many probes are acknowledged. Thanks to the tunnel, intermediate nodes cannot discern which packets are probes and which ones are normal data packets, so that the drop estimation can be reliably estimated.
- In **Secure sketch PQM** [20], the destination sends to the source an sketch representing the traffic received at the end of every interval. Using this sketch, the source is able to estimate the number of packets dropped, modified and fabricated by the path and use it to determine if the path is faulty or not.
- **Symmetric Secure Sampling** [20] uses sampling and the destination acknowledges the reception of the sampled packets by sending a MAC to the destination. By counting the missing ACKs, the source is able to estimate the loss probability of the path and by comparing the timestamps of when the packet was sent and the ACK received, the path delay.
- **Asymmetric secure sampling** [20] proposes an alternative to symmetric secure sampling, for the case of a sever and several clients. In such case, the server does not need to have a different key for each client, but releases the salt at the end of the interval and it is the client's responsibility to monitor the quality of the path.

The two main difficulties of these approaches are how to ensure that nodes in the path cannot avoid detection and cannot fake the summaries. For the case of probing and sampling, the former means that nodes cannot discern which packets are going to trigger an ACK on the destination, and that can be achieved by using secure tunnels or a keyed sampling function. For the case of sketches, nodes should not be able to find an alternative packet that will alter the sketch the same way, which is achieved by using keyed hash functions on the sketch. To avoid fake acknowledgments or sketches, they are computed using a secret key only known by the source and destination.

In any case, these solutions are not very precise in pinpointing the source of the fault. To be able to determine which node or link is the faulty one, the detection protocol needs to involve and obtain the traffic summaries of the rest of the nodes in the path. This approach is the one followed by the following solutions:

- **ODSBR** [8] is an on-demand source routing that detects Byzantine failures by receiving acknowledgments from the destination. When the source does not receive enough ACKs, it triggers a probing mechanism to detect the malicious link. It works as follows: the source sends a probe to the node in the middle of the path, trying to figure out which half is not working properly. This is done in an iterative fashion, until the faulty link is found.

- In **FL-PQM** [9], every node in the path receives an acknowledgement from the destination with certain probability, and acknowledgments for different nodes are sent using MACs in an onion fashion. At the end of each interval, the source will request each node a report with the estimation of the drop probability between itself and the destination, which is sent back to the source using again a set of MAC'd onion reports. With these reports, the source will determine that a link is faulty if the difference between the drop probability between two nodes is higher than a threshold, or the reports are not received properly.

- In **NACK** [53] nodes in the path keep track of its next-hop node by receiving an acknowledgment from the next-next-hop. In parallel, because NACK is based in DSR, the source receives acknowledgments from the destination through a path that does not involve any of the nodes in the normal data path. When the source fails to receive enough acknowledgments, it requests each node in the path its 2-hops acknowledgments to determine which node in the path is being faulty.

- Finally, **PAAI-1** [60] uses delayed sampling, so that the source, after the packet has supposedly arrived to the destination, sends a request for the packet acknowledgment; then, every node in the path sends the ACK using an onion report. To avoid nodes that forward packets only when the probe is received, each packet has a timestamp, and correct nodes only forward those packets with a recent timestamp.

Some of the solutions proposed, like ODSBR [8], PVM [3] or Distributed probing [25], propose to have first a detection algorithm and when a fault is detected, trigger the localization algorithm to find the faulty path. The detection algorithm is a simple end-to-end exchange of ACKs, when more ACKs that are expected are lost, the localization algorithm is triggered. But as Just et al. [25] mentions, the probing technique cannot go from source to destination, as it will alert the faulty node that a localization mechanism has been triggered, and it may change its behavior to avoid detection. To avoid that issue, the last three described protocols have every node run the monitoring process in parallel, so that there is no way for the faulty node to avoid detection.

In any case, all of these solutions keep the information local, i.e. only the source knows that there was a failure and which one was the faulty link, but in general, it would be in the network's benefit if the rest of nodes can learn about the failures from others' experience. Solutions that have been proposed to share the failure localization globally are:

- **Hash-based delayed sampling** [18] relies on a central authority which collects the sampling summaries from each node in the path and determines which node is faulty by comparing them.
- Similarly, **HLA-based detection** [51], has a central authority that collects traffic summaries (reception bitmaps and an HLA-based challenge) from every node in a path when the source detects that there is a failure.
- **Byzantine tomography** [7] also relies on a central authority that collects the results from Stealth probing and looks for the minimal set of nodes that could explain such results, including the possibility that the source and destination are providing a false report.

The main difficulty when dealing with global solutions is how false accusation is solved. For instance, in Hash-based delayed sampling, because blame is assigned to a node, its neighbors could send false reports, so that it ends up being accused as faulty when it is not. HLA-based detection solves this by assigning blame to a link and ensuring that the reports cannot be faked. Finally, Byzantine tomography faces false accusation by also considering the

fact that the reports could be false, but no guarantees are made, so a non-faulty node could end up being detected as faulty.

Solutions based on traffic flows can grow up to $O(n^2)$, being n the number of nodes in the network, if the failure localization is to be shared with the rest of the network, so if there is a central authority, $O(n^2)$ traffic summaries need to be exchanged through the network. In that case, when a WCN has too many nodes, the network bandwidth required would be too high. Additionally, it assumes that paths are known and stable, two things that are not necessarily true in WCNs, as not always a link state routing protocol is used and wireless links are not that stable, so a path may change within the same traffic flow.

The next set of solutions that we present are based on aggregated traffic statistics, that using conservation of the flow, make sure that the traffic received and sent by a network node is consistent.

- In **WATCHERS** [10] every node keeps a set of counters of the packets sent and received, as well as for the packets destined to and originated from them. These counters are flooded through the network so that everyone can check if a node is faulty. When a node receives a set of counters from its neighbor that is not consistent with its own, disconnects from that neighbor. In a second iteration, every node evaluates its neighbors using the counters from their neighborhood and determines whether the node is behaving properly or not based on the conservation of the flow principle. But, because WATCHERS uses flooding to share its counters, it can become too costly in wireless mesh networks.

- **DAMON** [50] proposes a hierarchical architecture that divides the network into clusters, so that traffic summaries do not need to be flooded: a monitoring agent on each cluster will collect the summaries related to that cluster, pre-process it and forward it to to the sink agent, which is ultimately responsible of analyzing the traffic summaries and detecting the faulty nodes.

- **DynaFL** [61] relies in sketches instead of counters and shares its traffic summaries with a central authority using a spanning tree and onion reports. The central authority is then responsible of analyzing the reports and finding the faulty neighborhoods. To avoid detecting a node as

faulty when it is actually its neighbor sending a false report, the central authority does not accuse a node, but a node's neighborhood.

- **Catch** [32] takes a more distributed approach by having nodes directly disconnect from its faulty neighbors. Catch relies on anonymous packets for estimating the link quality between one node and its neighbors; then, using overhearing it estimates the forwarding probability for the normal data-traffic. When these two values are too different, the neighbor will be considered faulty and this discovery will be shared to the rest of the faulty node's neighbors by using another mechanism based on anonymous messages.

- $\boldsymbol{\chi}$ [36] is also distributed, but works slightly different. First, every node in M's neighborhood will keep a fingerprint of every packet sent to M. Once the interval is over, it will forward the fingerprints to M as a signed report. If M sees that one of the reports is not truthful, it disconnects from that node, signaling that it is protocol-faulty. Then it broadcasts every other report to the rest of the neighbors, replacing the faulty reports with its own. Finally, M's neighbors will validate M's behavior by first comparing the traffic reports and, secondly, ensuring that it has disconnected from neighbors that were accused as protocol-faulty in previous iterations.

- Finally, $\boldsymbol{\Pi_2}$ **and** $\boldsymbol{\Pi_{k+2}}$ [35] do not monitor a single node, but a set of overlapping path-segments. Later, by comparing the traffic summaries between either all the nodes in the path (Π_2) or between the end-points (Π_{k+2}), they are able to find which path-segment is faulty.

Because we are now dealing with aggregated traffic, two or more consecutive nodes may collude to avoid detection. Consider the network of Figure 2.1, node M could drop some packets and have later B report that it has received them. Because traffic is aggregated, B could report that those packets are part of the packets destined to itself, so that neither M nor B are detected as faulty. Only Π_2 and Π_{k+2} are resilient to collusion (up to k consecutive colluding nodes), because they compare traffic summaries not only considering a node and its neighborhood, but also for longer paths.

The main challenge that we will address in this thesis is to find a solution that provides global fault localization, does not require path knowledge and

is resilient to false accusation and collusion, a combination of characteristics that is not available in the previous solutions proposed in the literature.

Table 2.3 summarizes the solutions proposed in the literature. For every solution, it notes how accurate the algorithm pinpoints the failure, whether or not it requires network stack changes and knowledge about the path a packet follows, if the traffic summaries kept relate to a single flow from source to destination (S-D) or aggregates all the traffic. It also notes in the cases where collusion or false accusation can happen whether the solution is resilient to them; and finally whether the findings are kept locally, shared with the rest of the network or the faulty node isolated by disconnecting from it.

2.3 Traffic Validation mechanism

Once the summaries have been shared, to decide whether a node, link or path is faulty by comparing the information collected. Most of the mechanisms for traffic validation are based in the conservation of the flow principle, which states that the traffic flow entering a network area should be approximately the same as the traffic flow leaving it, without considering the traffic destined to or originated from that network area (see Figure 2.2). Some solutions based on conservation of the flow are WATCHERS [10] or DynaFL [61], which take all the counters (or sketches) related to a single node and determine if it is dropping packets based on the difference between the incoming traffic and outgoing traffic. Of course, because there are some reasons under which it is licit for a router to lose packets, e.g. queue overflow or physic medium losses; in such cases we expect conservation of the flow techniques to accommodate some margin on the detection so that such losses are not detected as faulty.

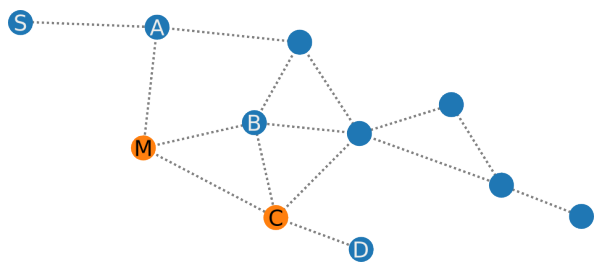


Figure 2.1: Example of a network

	Accuracy	Changes	Summary	Collusion	Accusation	Path	Scope
Stealth probing	Path	✗	S-D	–	–	✓	Local
Secure Sketch PQM	Path	✓	S-D	–	–	✓	Local
Secure sampling	Path	✓	S-D	–	–	✓	Local
ODSBR	Link	✓	S-D	–	✓	✗	Local
FL-PQM	Link	✓	S-D	–	✓	✗	Local
NACK	Node	✓	S-D	–	✗	✗	Local
PAAI-1	Link	✓	S-D	–	✓	✗	Local
Hash based delayed sampling	Node	✓	S-D	–	✗	✗	Global
HLA	Link	✗	S-D	–	✓	✗	Global
Byzantine tomography	Link	✗	S-D	–	✗	✗	Global
WATCHERS	Node	✓	Aggregated	✗	✓	✓	Global
DAMON	Node	✓	Aggregated	✗	✓	✓	Global
DynaFL	Neighborhood	✓	Aggregated	✗	✓	✓	Global
Catch	Node	✓	–	✗	✗	✓	Local disconnect
χ	Node	✓	Aggregated	✗	✓	✓	Local disconnect
Π_2	Link	✓	Aggregated	✓	✓	✗	Global

Table 2.3: Characteristics of dissemination solutions

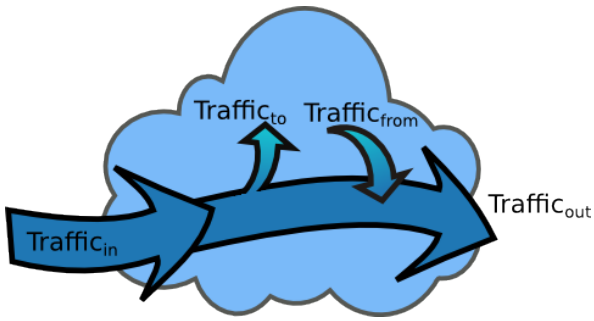


Figure 2.2: Conservation of the flow principle

In some cases, because not all the packets have the same relevance, the faulty routers may drop selectively traffic so that the drop percentage is within the expected natural loss rate, while maximizing the damage. To solve this issue, more complex statistical methods have been proposed. Mizrak et al. [36] propose a model to predict congestion based on traffic rates and buffer sizes, which allows estimating the probability of loosing each packet. Shu and Krunz [51], because they have information about which packets were lost, use the correlation between those losses to determine if the loss pattern corresponds to normal operation or the router is faulty instead.

Though seemingly different, acknowledgement based solutions, like ODBSR [8] or NACK [53], apply the same principle, just that instead of sharing the monitored information once every interval, they share it continuously as a response of the data traffic. But in the end, they estimate the percentage of packet losses using a fingerprint (the acknowledgment) and expect it to be below a threshold. For instance, ODBSR monitors end-to-end packet losses and when delivery ratio goes below the acceptable rate (measured by counting the acknowledgments received), it will start a detection mechanism that splits the monitored path into halves to detect the failing link and assign blame to a single link. But the principle is the same: most of the packets sent by the source are supposed to be received by the destination.

2.4 Response

The final piece of the puzzle is how to react when a faulty node or link is detected.

Some solutions, like AudIt [5], simply announce the discovered discrepancies, so that they are further investigated by a human. Many other solutions propose a more drastic solution: to remove the faulty link or node from the routing fabric, like WATCHERS [10], RSR [17], DISA [28], Catch [32] or NACK [53]. Others, incorporate the measurements to the routing protocol as a part of its metric: ODSBR [8], CASTOR [19], AFC [33], EFW [39] and EigenTrust-OLSR [44]. Finally, CONFIDANT [11] and DAMON [50] build a reputation system with the measurements of the forwarding fault detector that later interacts with the routing protocol to increase the routing robustness.

In the case of WCN, we believe that a simple mechanism as notifying the owner of the faulty equipment and some network administrator should be sufficient, and, therefore, we will not focus our attention in this topic in the rest of the thesis.

2.5 Filling the gaps

In the rest of the thesis we will focus on each of the first three sub-problems. For the traffic summary functions, we will study in depth the accuracy of sketches compared with sampling, so that the network bandwidth consumption can be minimized for some given constrains in terms of false positives and false negatives. Then, for the part of traffic dissemination and distributed detection we will propose a solution that can be deployed as an independent daemon and does not require changes in the network stack. Moreover, it will make no assumptions on the number of antennas used by the node, nor the routing protocol used in the network. At the same time, it must be a solution that is cheap in terms of CPU, memory and network bandwidth required, but resistant to collusion and false accusation. Finally, we will propose a mechanism that does not require time synchronization for traffic validation, as many WCN nodes do not have a reliable clock.

Chapter 3

Contributions

This chapter presents the main contributions of this thesis. We discuss how each contribution shows new light on the problem of detecting forwarding faults and how they fill the gaps of existing solutions in the context of WCNs. The contributions are later presented in detail in chapters 4, 5 and 6.

3.1 Tight bounds for sketches in Traffic Validation

Sketches have been proposed before [20, 61] as traffic summary function for Traffic Validation because they are resilient to packet dropping, modification and creation; and compared with sampling, they achieve more accurate predictions for the same network bandwidth. However, none of the previous solutions took into account that the digest of data packets, as a stream that is going to be sketched has a very specific distribution: it is uniformly distributed, and that with high probability the stream elements will be unique. So, the first question that we asked ourselves was: *Can we use this information to find better bounds on the error of sketches when predicting the number of different packets between two traffic flows?* On Chapter 4 and paper [P2] we propose three different random process for the three studied sketches, AGMS [4], Fast-AGMS [16] and FastCount [55] that accurately describe the error made by the sketches when estimating the number of different packets between two traffic flows. The benefit of proposing a random process to describe the estimation error is that it allows us to provide tighter bounds to the error, for instance, using the bounds previously computed by Goldberg et al. [20], if the faulty nodes drop 1% of the traffic and the non-faulty nodes 0.5% of the traffic

at worst and 10^7 packets are exchanged per interval, a Fast-AGMS sketch needs at least 300 counters to guarantee that there will be less than 1% false positives and negatives. However, using the random process that we propose, we can optimize the size of the sketch to 128 counters, which implies that the memory and the control overhead required to share the traffic summaries can be reduced by more than half. In a wireless network, network bandwidth is a scarce commodity, and therefore, having tools that allow us to optimize the use of this network bandwidth is imperative.

Next, we focus on the implementation details: the sketch accuracy is usually measured by the number of counters it has (i.e. its size) and the number of packets that are being sketched, but there are a couple of implementation parameters that may also influence its accuracy: (i) the size of the packet's digest and (ii) the implementation of the pseudo-random functions. So our second question was: *How does the digest size influence a sketch's accuracy? Do the different implementations for the pseudo-random functions affect the sketch's accuracy? And then, which implications do they have in the processing time?*

Regarding the digest size, from the processing time standpoint, the best would be to keep it as small as possible, since using larger digests causes the pseudo-random function to be more costly in terms of computation; specially if we require operations of numbers bigger than the processor word size. But on the other hand, as Chapter 4 shows, using a digest too small will cause too many collisions, causing a bias that is proportional to N^2. Our recommendation respect the digest size is to use at least 32 bits in general, and if the number of packets being estimated goes over one million, use 64 bits. In any case, because the number of packets being estimated is the number of packets dropped, it will rarely reach that limit.

On the other hand, we found that every tested pseudo-random function provides the same error distribution, and therefore, even though FastCount proposes to use a 4-way hash function, a 2-way hash function is enough in the context of Traffic Validation. Same for the ξ functions, using $BCH3$ is enough. This simplifies the implementation of the pseudo-random functions and also provides better processing time, since 2-universal random functions are faster to compute than 4-universal ones.

Our final concern regarding traffic summary functions, is that all the work done related to measuring their accuracy was based under the assumption that the number of packets per interval is fixed. However, if the network area being monitored is not a path, there is no way of ensuring the number of packets that there will be on each interval, because nodes do not have a unique view of the traffic going through the network area. In such situations, the interval will be determined by time and not number of packets. In such situation, because we are not interested in the total number of packets dropped (or modified, or created), but rather the proportion of packets that have been dropped, there are two possible approaches that can be taken: (i) first, we can estimate both the number of packets being dropped and the total of number transmitted using the sketches (we will name this approach **proportion**); or (ii) second, we can only estimate the number of packets being dropped, and share the total number of packets being transmitted using an additional counter (**dropped**). So the natural question that arises is: *Which approach will provide more accurate results? And how does the standard error of* ***proportion*** *change for different sketch sizes and estimated values?*

First, we find that the standard error of the prediction using the **proportion** decreases with the $\sqrt{size}$, as for the **dropped** approach, but its relation with the number of dropped packets is not lineal as for **dropped**, but proportional to $p*\sqrt{(1-p)}$, because there is a correlation between the error on the estimation of the dropped packets and the transmitted packets. The consequence is that **dropped** will be a more reliable estimation when the percentage of dropped packets is below 50%, and viceversa. And because, the number of dropped packets for non-faulty nodes will be well below 50%, on the cases that the detector will have more trouble detecting faulty routers, will be when their drop probability is close to that of the non-faulty ones, and therefore, using the **dropped** approach will be preferred.

3.2 KDet: a distributed detection protocol

Our next contribution relates to the traffic summary dissemination and distributed detection protocol. As we have seen in Chapter 2, in the case of WCNs, our goal is to provide a distributed solution that:

- Detection results are available globally, not only to the source and destination of a traffic flow.
- Does not rely in overhearing, because nodes may have several antennas.
- Does not require a specific routing protocol, in specific, it does not require path knowledge.
- Is resilient to false accusation and collusion.

In Chapter 5, conference paper [P4] and journal paper [P3] we propose KDet, a detection protocol that satisfies all the previous requirements. KDet is a detection protocol based on boundaries, i.e. a set of nodes that act as a boundary for the area being monitored (the core) and that assess their behavior as faulty or not. It works in a distributed fashion, so every node belongs to several boundaries and cores, so it is at the same time monitoring and being monitored.

The main challenges while designing KDet were:

No false negatives How can we ensure that a faulty core cannot avoid being detected as faulty?

No false positives How can we prevent false accusations? That is, how can we make sure of the authenticity of the boundary reports and avoid that non-faulty nodes are detected as faulty?

Completeness How do we determine which cores need to be monitored to ensure that the detection protocol is complete? That is, if there is a set of faulty nodes F, how can we ensure that they will be monitored and detected?

To ensure that a faulty core will be detected as such, KDets relies on robust flooding [41] within the core to share the reports of its boundary nodes which are signed to avoid modification. At the end of the interval, if a boundary node does not have all the traffic reports from its fellow boundary nodes, or the traffic reports are not consistent (e.g. they do not satisfy the conservation of the flow), their verdict will be to suspect the core. If during the next intervals the boundary of that given core does not change, the core will be

detected as faulty. Because a faulty core has no way of modifying its boundary reports, it does not matter what it does, that it will be eventually detected, or disconnected from the network.

Then, to prevent false accusation, KDet involves the nodes in the core in the detection process. While the core nodes are robustly flooding the reports that relate to themselves, they compare these reports with its local traffic summaries. If there is a discrepancy between the report and the local traffic summaries, the core node will disconnect from the boundary node sending false reports, effectively removing it from its boundary, and therefore, removing that faulty node from the detection process.

Finally, KDet measures the strength of the adversary as the largest set of connected faulty nodes, expressed by k. To ensure that any set of faulty nodes will be detected even in the case of collusion, KDet requires that every set of connected nodes of size k or smaller should be monitored. That way, we can ensure that if there is a set of faulty nodes, there will be at least one boundary composed of non-faulty nodes that will detect them as faulty.

3.3 Traffic Validation for misaligned summaries

Our final contribution relates to the fact that tight clock synchronization in WCNs is difficult, because they rely on affordable equipment (so GPS or atomic clocks are out of question), NTP cannot provide the desired precision in this wireless environment and sometimes clocks are not properly configured. As we have seen in the previous chapter, all the traffic validation mechanisms proposed before, are either based on end-to-end measurements or require clock synchronization, so that the traffic summaries shared are about the same packets.

In Chapter 6 and conference paper [P1] we propose a traffic validation mechanism based on sketches, for the case of unsynchronized clocks: Misaligned Traffic Validation (MTV). MTV relies on the fact that sketches can be used not only to compute the second frequency moment of a data stream, but also to estimate the intersection size between two different data streams. Consider the simpler case, where we have a sketch that represents the incoming traffic and outgoing traffic of a single node. If the period of time during which those sketches are computed does not perfectly match, when comparing them, the

validation mechanism will assume that the node has dropped some packets (those that were in the incoming sketch but not in the outgoing sketch) and created some other packets (viceversa). MTV avoids making such assumption, by comparing the incoming and outgoing sketch of each interval, with those of the previous and next intervals using the intersection: to describe the incoming traffic, we do not only consider the packets that are sketched on the current interval incoming sketch, but also those that are present on the next and previous intervals' incoming sketches and that are in the current outgoing sketch (obtained using the intersection). Similarly, we consider for the outgoing traffic, not only the packets sketched on the outgoing sketch for the current interval, but also those that intersect with the incoming sketch on the previous and next interval outgoing sketches. By using this approach, we can obtain a more reliable validation mechanism when the clocks are not synchronized at the cost of using larger sketches, so that the intersection size can be computed accurately.

Chapter 4

Traffic Summary Functions

This chapter focuses on the study of different traffic summary functions. As we have previously established, the network monitors need to summarize the traffic seen towards and from the monitored nodes so that later summaries can be shared to assess the behavior of the network area being monitored. Depending on the way we summarize the traffic, the accuracy and strength of the detection protocol will vary. For instance, if we simply count the number of packets that enter and leave the network and later compare such counters [10], we will have a very accurate detection protocol, but it won't be very strong, since a faulty network can fool the detection protocol by simply corrupting the packets. A stronger summary function is to select a sample of the packets and keep its digest, later we can compare the set of digests to determine the behavior of the network [20, 60]. Because we keep a digest for the packets, now the detection protocol is robust against packet modification, but because the digest is only kept for a proportion of the packets it will not be as accurate. As we can see, characterizing the accuracy achieved by the traffic summary function is fundamental, as it will determine the guarantees given by a detection protocol, i.e. on one hand it will make it robust against one type of failures and ineffective against others; on the other hand, it will determine its accuracy in terms of false positive and negative ratios.

We have briefly mentioned in Chapter 2 how each of the proposed traffic summary functions affect the strength of the detection protocol, so on this chapter we will focus on studying the accuracy of two of the non-deterministic traffic summary functions: sampling and sketching. In both cases, we will consider that there may be two measurements of interest in the case of traffic

validation; either estimating the total number of different packets between two traffic streams or estimating the proportion of such packets (e.g. the number of different packets over the total number of packets sent). First, we briefly describe how sampling works for traffic validation and characterize its accuracy. Because sampling has been studied deeply before [20], we do not go into much detail. Then, we study three different types of sketches, namely AGMS, Fast-AGMS and FastCount [48], and how they can be characterized as a random process when used to summarize traffic packets. Next, both solutions are compared and studied in detail experimentally, and every factor that influences its accuracy is listed and analyzed. To obtain the whole picture, we include some numbers related to their cost, mainly in terms of CPU, memory and network bandwidth consumption.

Overall, the goal of this chapter is to help in the process of designing, implementing and deploying a detection protocol by giving all the necessary information regarding the traffic summary functions that can be implemented. Moreover, every piece of code used in this chapter is available online [30], and can be used as starting point for a custom solution.

4.1 Sampling

Sampling is one of the simplest mechanisms for summarizing traffic. It consists on keeping just a proportion of the traffic packets as a representation of the whole traffic stream. This is usually achieved by obtaining a digest of the packet, $d = H(\text{packet})$, and a function, $\text{Probe}(d)$, determines whether the packet should be sampled or not. A simple Probe function could be to compare the digest with a threshold and store the digest if it is below the threshold, drop it otherwise. However, because in such case, every node in the network would know which packets are to be sampled and which are not, a faulty node may behave differently depending on that fact. Therefore, the Probe function should not only depend on the digest, but also on a secret key, unknown by the monitored nodes, but known by the monitors, as they have to agree on the same key so that the summaries are consistent. More details on how to implement secure sampling and its accuracy bounds can be found in Goldberg et al.'s paper [20].

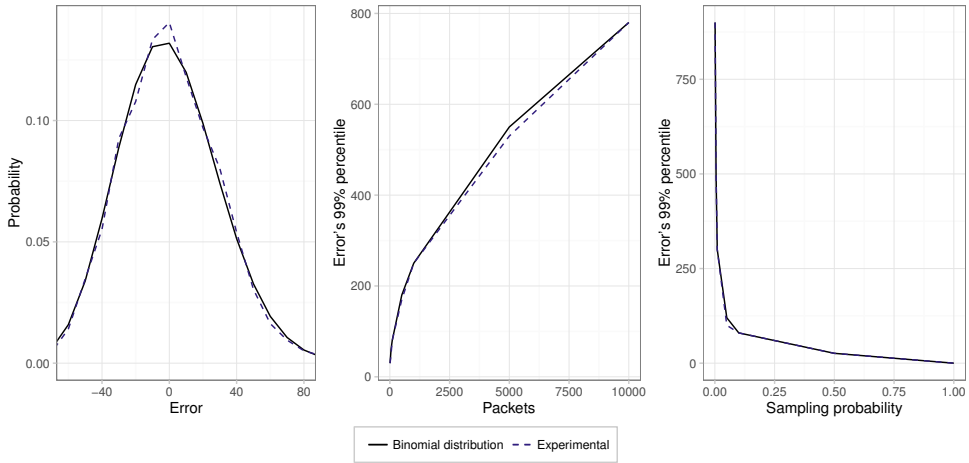


Figure 4.1: Sampling error when estimating the total number of dropped packets.

4.1.1 Mathematical approximation

When estimating the number of packets of a data stream, the distribution of the estimation and the error follow a binomial distribution with the support being $x = \{1 \dots N\} * 1/p$ and $x = \{1 \dots N\} * 1/p - N$ respectively; where N is the number of packets and p is the sampling probability. On the other hand, if we are estimating the proportion of packets dropped (or modified), and the value being estimated is big enough, the error can be approximated by a Gaussian variable centered in 0 and with deviation $\sigma \approx \sqrt{\widehat{p} * (1 - \widehat{p})}/\sqrt{n}$, where $\widehat{p}$ is the proportion being estimated and n is the number of total sampled packets.

Figure 4.1 shows the PMF of the error when estimating the total number of packets different between two traffic streams and how the error declines as the number of packets in the time interval increases (N) or the sampling probability increases. Figure 4.2 shows instead the equivalent figures for the case of estimating the ratio of dropped packets.

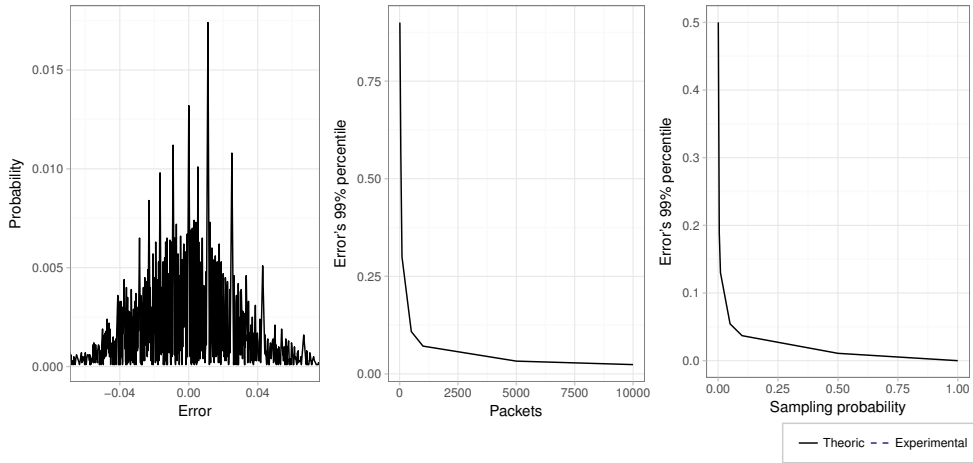


Figure 4.2: Sampling error when estimating the ratio of dropped packets.

4.2 Sketches characterization

Sketches first appear in the context of relational databases as a solution to estimate the size of joins in limited storage. A sketch is capable of summarizing a data stream while still being able to provide accurate estimations of its second frequency moment and the second frequency moment of the intersection of two different streams. As data structure, a sketch can be seen as a matrix of counters, with each row being a basic estimator, whose result is later averaged with the other rows to improve its precision.

In the case of traffic validation, thanks to the fact that packets are mostly unique [20], we can use sketches estimation of its second frequency moment to measure the difference between the traffic entering and leaving a network area, since the second and first frequency moment will be equivalent. Other characteristics that make sketches ideal to the context of traffic summarization are that they can be updated online, that is, every time a new packet arrives, and that they support linear combinations (so they can be added and subtracted). In a simplified example, using Figure 4.3 as reference, sketches for traffic validation would work as follows: let's assume A, C and D are monitoring a single node, B, by keeping an sketch that summarizes all the traffic that has been sent to B and received by B. At the end of the interval,

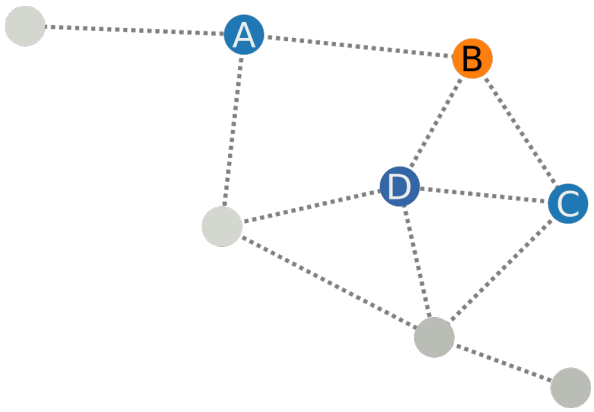


Figure 4.3: Example network

A receives a copy of all the other sketches, and A sums all the sketches for the incoming traffic and subtracts to the result all the sketches for the outgoing traffic. Ideally, the resulting sketch should represent the empty sketch, i.e., its second frequency moment should be 0. Otherwise, its second frequency moment will tell us how many packets are different from B's incoming and outgoing traffic. If we are interested in the ratio of different packets instead, we also need to compute the second frequency moment of the incoming traffic, so that we will have the number of sent packets and divide both estimations. Alternatively, if we are monitoring a path from source S to destination D, the source will sketch every packet sent to D, and the destination every packet received from S. Then those two sketches will be compared as before. Finally, if we are monitoring a network area, we will keep sketches at every neighboring node, and only consider the traffic that is not destined or sent from the nodes in that network area.

In the following subsections, we start by describing each sketch and proposing a random process that has the same distribution for the estimation when the stream elements being sketched are traffic packets (i.e. unique). We will validate the proposed random processes on the following section and describe the accuracy of an sketch as a function of its size and the value to be estimated.

As the reader has probably noticed, we use the same names for each sketch type as Rusu and Alin [48], and we will do the same for the different pseudo-random

functions [47]. Other recurring parameters on the next sections will be the number of packets being estimated, N, the number of columns, nc, and the number of rows, nr.

4.2.1 AGMS Sketch

The AGMS sketch was proposed by Alon et al. [4] and it is inspired in the tug-of-war game, i.e. each counter represents the result of a tug-of-war game, and a ± 1 4-wise independent variable (ξ)[1]determines on which team each packet "plays". When a new packet arrives, each counter is updated, simulating the rope displacement on each assigned team:

$c_i = c_i + \xi_i(e)$

To estimate the second frequency moment we use:

$\widehat{F_2} = \frac{1}{\mathrm{nc}} \sum_{i=1}^{\mathrm{nc}} x[i]^2$

Which gives us an unbiased estimation with variance:

$Var[\widehat{F_2}] = 2 \times \frac{(F_2^2 - F_4)}{\mathrm{nc}}$

To reduce the variance, several estimations are combined using the median, reducing the variance of the error, as given by a combination of the Chernoff and Chebyshev bounds [48]:

$$Prob(|\widehat{F_2} - F_2| \leq \frac{4}{\sqrt{\mathrm{nc}}} \times F_2) \geq 1 - 2^{-\mathrm{nr}/2} \tag{4.1}$$

Luckily, our data stream has very unique characteristics that will allow us to provide an approximation to the probability mass function (PMF) of the estimation, and therefore, find a better approximation for a given percentile than the previous bounds. As we know, traffic packets are unique with high probability and if we use a 2-way hash function with a sufficiently big output space, so will be the digest generated [P1]. Considering that assumption, after sketching N packets, the probability of obtaining k '+1' from the ξ

[1] A ± 1 function can be consider a special case of a hash function, where the output is either $+1$ or -1 with the same probability. A k-wise hash or ± 1 function means that it guarantees that the result for any k keys is independent from each other [58]

function follows a binomial distribution with 0.5 success probability and N trials. Therefore, the distribution of the counter value will also follow the shape of a binomial distribution, but its support will be $2k - N$, $k \in [0, N]$. When we square the value of the counter, negative values will overlap with their corresponding positive value, doubling its probability, so the distribution of the square value of the counter has the shape of half a binomial distribution, with double probability, except for 0 (in case N is even), whose probability does not double. The formula of its distribution is as follows:

$$f(c^2; N) = \begin{cases} 0.5^N & \text{if } c = 0 \text{ and } N \text{ even} \\ 2 \times \binom{N}{(c+N)/2} \times 0.5^N & \text{if } \exists k = \frac{c+N}{2} \text{ such that } k \in \{0, 1, \ldots, N\} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Finally, the PMF of the estimation will be the convolution of nc of such distributions:

$$f_{\text{AGMS}}(\widehat{N}; N, \text{nc}) = \sum_{\overrightarrow{c} \in \mathcal{R}_c} \prod_{i=1}^{\text{nc}} f(c_i^2; N) \quad : \quad \mathcal{R}_c = \left\{ \overrightarrow{c} \;\middle|\; \frac{1}{\text{nc}} \sum_{i=1}^{\text{nc}} c_i^2 = \widehat{N} \right\} \tag{4.3}$$

The PMF of the error will be the same, but with a different support: error $= \widehat{N} - N$. The error ranges from $-N$ to $N^2 - N$ and the discrete step between possible error values is $4/\text{nc}$.

4.2.2 Fast-AGMS Sketch

The main drawback of the AGMS sketch is that when a new packet arrives, every counter needs to be updated, which may slow down the sketch update process for big sketches. To solve this problem, Cormode and Garofalakis proposed the Fast-AGMS sketch [16] (which works as [14]). The Fast-AGMS sketch improves the update time by updating a single counter on each row of the sketch, chosen by a 2-way hash function, h; i.e., when a new packet, p, arrives, the sketch is updated as follows:

$c_{h(p)} += \xi(p)$

For the Fast-AGMS sketch, the second frequency moment is estimated as:

$\widehat{F_2} = \sum_{i=1}^{\mathrm{nc}} x_k[i]^2$

Giving us again an unbiased estimator that has the same variance as the AGMS sketch, and therefore, also the same theoretical bounds (formula 4.2.1) if we combine several of them and average using the median.

To estimate the PMF of the Fast-AGMS's estimation, we have to consider that in this case, the number of packets affecting each counter is different, let's say n_i. In this case, the PMF of the value of each counter has the same shape as before, but now, it depends on the number of packets that were hashed into that counter, n_i, instead of the total number of packets. n_i follows a multinomial distribution, with each counter having the same probability of being chosen, 1/nc. Putting all together, the distribution of the basic estimator is:

$$f_{\mathrm{FAGMS}}(\widehat{N}; N, \mathrm{nc}) = \sum_{\substack{\overrightarrow{c} \in \mathcal{R}_c \\ \overrightarrow{n} \in \mathcal{R}_n}} \prod_{i=1}^{\mathrm{nc}} f(c_i^2; n_i) \tag{4.4}$$

$$\mathcal{R}_c = \left\{ \overrightarrow{c} \,\middle|\, \sum_{i=1}^{\mathrm{nc}} c_i^2 = \widehat{N} \right\} \quad , \quad \mathcal{R}_n = \left\{ \overrightarrow{n} \,\middle|\, \sum_{i=1}^{\mathrm{nc}} n_i = N \right\}$$

As before, the error ranges from $-N$ and $N^2 - N$, but now the discrete step between possible error values is 2.

4.2.3 FastCount Sketch

The last sketch studied, FastCount sketch, was proposed by Thorup and Zhang [55]. The FastCount sketch proposes to increase by one a counter on each row, using this time a 4-way hash function (h) instead of a 2-way hash function to choose which one:

$c_{h(p)} = c_{h(p)} + 1.$

In this case, if we were to estimate F_2 as before, we would end up with a biased estimator so we need a more complex estimator:

$$\widehat{F_2} = \frac{\text{nc}}{\text{nc}-1} \sum_{i \in [\text{nc}]} c_i^2 - \frac{1}{\text{nc}-1} (\sum_{i \in [\text{nc}]} c_i)^2$$

The variance in this case is slightly higher:

$Var[\widehat{F_2}] = 2 \times \frac{(F_2^2 - F_4)}{\text{nc}-1}$

But for large values of nc the difference is negligible.

As for the Fast-AGMS, the PMF for the FastCount sketch will be intrinsically related to the multinomial variable that determines the number of packets hashed to each counter. But the challenge here is to properly determine the support of the PMF. As shown in [54], the second moment estimation (or, equivalently, $\widehat{N}$ in our case) can be expressed as:

$$\widehat{N} = N + \sum_{a \neq b} v_a v_b X_{a,b} \quad \text{such that} \quad X_{a,b} = \begin{cases} 1 & \text{if a} \sim \text{b} \\ -\frac{1}{\text{nc}-1} & \text{otherwise} \end{cases}$$

And where, $a \sim b$ means that $h(a) = h(b)$. In our case, because we assume network packets to be unique, we can simplify the equation further:

$$\widehat{N} = N + \sum_{a \neq b} X_{a,b}$$

The second term of the equation, the error, is determined by the number of collisions, and the contribution of a packet, p, that colludes with other C packets, is:

$$\text{error}_p = 1 \cdot C - (N - 1 - C) \cdot \frac{1}{\text{nc} - 1}$$

If the multinomial that determines the number of packets on each counter is $\overrightarrow{n}$, then we have n_i packets colliding with other $n_i - 1$ packets, and therefore, the estimation in that case can be expressed as:

$$\begin{aligned}\widehat{N} &= N + \sum_{i=1}^{\mathrm{nc}} n_i \times \left(1 \cdot (n_i - 1) - (N - n_i) \cdot \frac{1}{\mathrm{nc} - 1}\right) \\ &= \frac{\mathrm{nc}}{\mathrm{nc} - 1} \sum_{i=1}^{\mathrm{nc}} X_i^2 - \frac{N^2}{\mathrm{nc} - 1}\end{aligned}$$

Then, the PMF of the FastCount sketch is:

$$f_{\mathrm{FastCount}}(\widehat{N}; N, \mathrm{nc}) = \sum_{\overrightarrow{n} \in R_n} \frac{N!}{n_1! n_2! \cdots n_{\mathrm{nc}}!} \frac{1}{\mathrm{nc}^N} \tag{4.5}$$

$$R_n = \left\{ \overrightarrow{n} : \sum_{i=1}^{\mathrm{nc}} n_i = N, \sum_{i=1}^{\mathrm{nc}} n_i^2 = \frac{\mathrm{nc} - 1}{\mathrm{nc}} \widehat{N} + \frac{N^2}{\mathrm{nc}} \right\}$$

As for the other sketches, the error ranges between $-N$ and $N^2 - N$ and, in this case, the smallest difference between two possible values of the error is $2\mathrm{nc}/(\mathrm{nc} - 1)$.

4.3 Empirical Evaluation

Our next step is to characterize the error distribution of each of the three sketches described and see if they match with the proposed PMFs. To do so, on one hand we have implemented several functions to reproduce the PMFs proposed in section 4.2; and, in the other hand, we have extended the code by Rusu [46] to test the sketches with traffic capture (pcap) files. All the code required to reproduce the results on this chapter can be found online [30].

Each experiment computes both the expected distribution and empirical distribution of the estimation. The expected distribution is computed using the proposed PMF when N was not too large and using Monte-Carlo techniques, as otherwise the computing time in such cases was prohibitive. And the empirical distribution is computed by initializing 100 sketches randomly and making

at least 100 predictions simulating the traffic summary process using a pcap file. Because the used pcap files contain only the first bytes of each packet, the missing bytes were generated randomly, but in any case, the important information obtained by the pcap file is the moment a packet is transmitted and its size, as the packet will be hashed, giving us a pseudo-random distribution after that.

In our experimentation, we consider two different scenarios, a wired Internet network, with high network bandwidth and pretty constant utilization; and a Wireless Community Network, which has a low and variable network load. The first scenario is represented by pcaps from the CAIDA dataset [12], whereas the second uses captures from Guifi.net, one from a mesh router in qMp Sants [45], and the other from one of Guifi.net's proxies. These differences can be clearly seen on Table 4.1.

	CAIDA	**Sagunt**	**Proxy**
Duration	10.59 s	842 s	58.99 s
Packet rate	∼350 Kpackets/s	59 packets/s	168 packets/s
Bitrate	2120 Mbps	0.366 Mbps	1.52 Mbps
Mean arrival time	0.0028 ms	16.84 ms	5.95 ms
σ **arrival time**	0.0029 ms	97.73 ms	11.3 ms

Table 4.1: Characteristics of the traffic capture files

As mentioned before, using sketches for traffic validation allows us to measure the difference between two different traffic flows, which using the Conservation of the Flow principle [10], will allow us to determine the behavior of the monitored entity. But we can make two different estimations: (i) the absolute number of different packets between the incoming and outgoing traffic or (ii) the proportion of different packets. We will address each case on the following subsections, and, as we will see, there is actually a difference in the accuracy between predicting one or the other.

4.3.1 Estimating the number of packets

In this subsection, we will focus on characterizing the error on the estimation of the absolute number of different packets, i.e. between the incoming and outgoing streams there will be a difference of N packets, which will be estimated

by computing the second frequency moment of the difference between the sketches that represent each traffic stream. Given that sketches support linear combinations, for this scenario we do not need to emulate the incoming and outgoing traffic streams, but just update the sketch with the different packets between the traffic streams. The number of such packets will be referred through this section as number of sketched packets or N.

When studying the accuracy of a sketch there are many variables that influence it. First of all, the packets will be reduced to a digest of B bits. A digest too small will cause collision between the sketched packets, and therefore overestimating the number of packets. A digest too big will require pseudo-random functions that accept a bigger input space and, therefore, will be slower, i.e. more costly in terms of CPU. Then we have the pseudo-random functions themselves. AGMS and Fast-AGMS require a 4-wise independent ± 1 function (ξ) and Fast-AGMS and FastCount require a 2 and 4-way hash respectively. In this section, we will study the impact on accuracy of choosing an implementation or another for those functions, while later, in section 4.3.3, we will see the impact they have on the CPU cost. Next, if we are averaging the result of several estimators, using the mean or the median will affect on the quality of the prediction. Which will be the best for the case of traffic validation? And finally, the two evident variables that influence the accuracy of the result, as predicted by their expected PMF, are the sketch size (number of columns and rows) and the value being estimated.

Digest size

To study the effect of the digest size we have considered a sketch of 256 columns and 1 row and measured the error on the estimation when the expected number of different packets is 100. Figure 4.4 shows the experimental and expected PMF of the error as a histogram with bins of similar size, so that the results are comparable. As we can see, if we use digests of size 16 bits or more the error is centered in 0 and the distribution can be properly estimated using our proposed mathematical approximation. However, if we only take 8 bits, the distribution is no longer centered in 0 since there are more collisions than expected. A collision between 2 packets when creating their digest will cause, from the sketch's perspective, the same effect of having two instances of the same element. When sketched elements are not unique anymore, we cannot

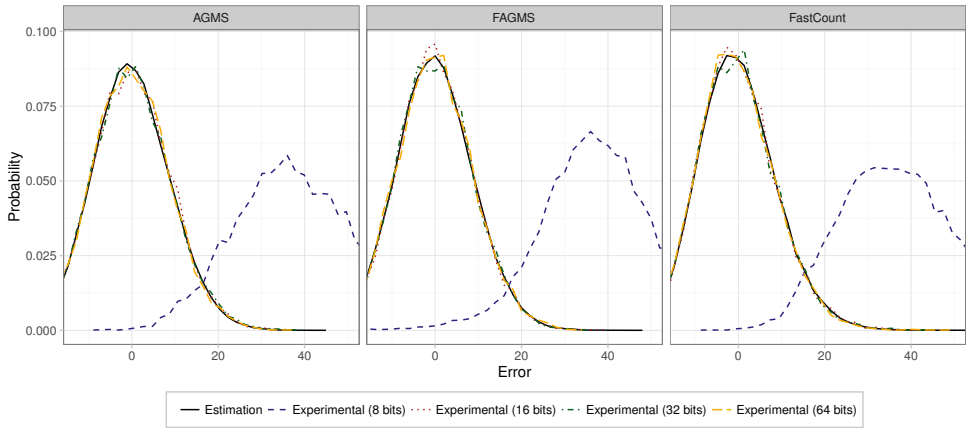


Figure 4.4: PMF of the error for different digest sizes.

rely on the fact that $F_2 = F_1$, being F_2 an overestimation of F_1, as we are using the square of the number of instances.

This is further investigated in Figure 4.5, that shows the bias of the estimator as the number of sketched packets increases, for a sketch of the same characteristics as before and digests of different sizes. As we can see, for digests of 8 and 16 bits, the tendency soon becomes proportional to N^2, whereas for 32 and 64 bits the bias is proportional to N up to $N = 10^5$. We recommend using digests with as many bits as necessary so that the output space of the digests is at least three orders of magnitude bigger than the expected number of dropped or corrupted packets in the benign case, so that the bias is kept within the linear region. If the number of different packets is overestimated on the faulty case, it is alright, because it will only further differentiate the benign and faulty nodes.

Pseudo-random function

Next, we study the effect on the sketch accuracy of the different possible implementations for the hash and ξ functions. For the ξ functions we consider those proposed by Rusu and Dobra [47]: 3 and 5-wise independent BCH schemes (bch3 and bch5 respectively), 3-wise independent extended hamming (eh3) and 2 and 4-wise independent hash functions based on the Carter-Wegman trick (cw2 and cw4). On the other hand, for the hash functions

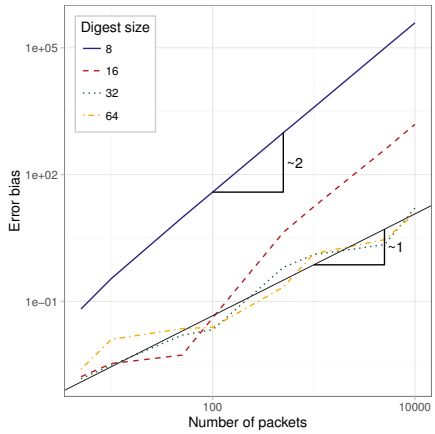


Figure 4.5: Bias of the estimation as the number of packets increases.

we will also consider cw2 and cw4 and tabulated hashing [55]. Figure 4.6 shows the PMF of the error using each of the proposed implementations for the ξ function, for AGMS and Fast-AGMS sketches of size 16 by 16 and a digest size of 32 bits, when predicting $N = 10000$. As we can see, every implementation provides equivalent results in both of the sketches, so choosing the best pseudo-random function will be a matter of their computational and memory cost. For the hash functions (Figure 4.7), we can see similar results: every hash tested follows the expected PMF.

Number of packets

Our following experiment studies how wide is the distribution of the error of the sketch's estimation as a function of the number of sketched packets. Here we show how the 99 percentile of the absolute error increases with the number of packet, but similar results are found for different percentiles [31]. In Figure 4.8 we see the 99 percentile for sketches of 1 row and 256 columns and a digest size of 32 bits when predicting different values experimentally and as given by the proposed model, as well as the Chebyshev bounds and the bounds by Goldberg et al. [20] for the Fast-AGMS. As predicted by the bounds, the relationship is approximately linear, but we can predict with greater precision the 99 percentile using the random process described in the previous section than the known bounds.

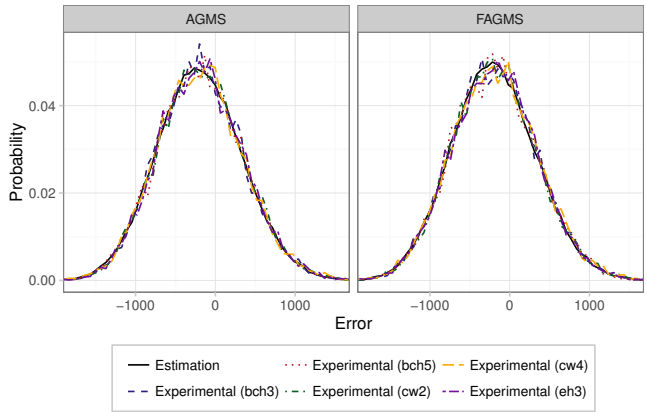


Figure 4.6: PMF of the error for different ξ implementations.

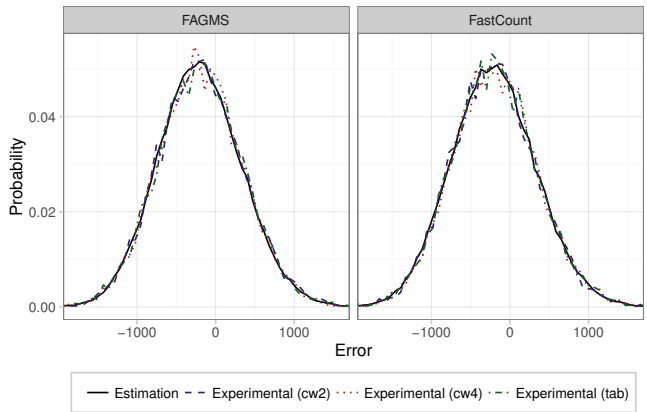


Figure 4.7: PMF of the error for different hash implementations.

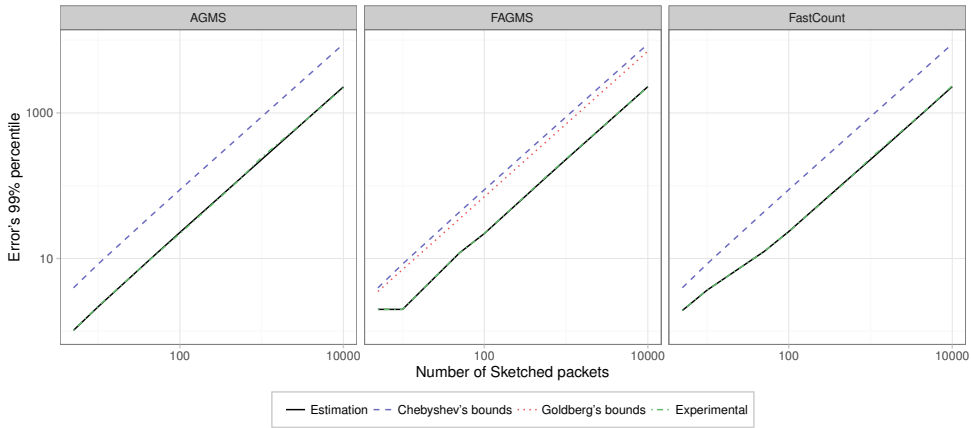


Figure 4.8: 99 percentile of the error vs. the number of packets

Number of columns

Similarly, Figure 4.9 shows the 99 percentile when varying the number of columns. In this case, the relationship is approximately proportional to $1/\sqrt{\mathrm{nc}}$ ($1/\sqrt{\mathrm{nc}-1}$ for FastCount), i.e. we need four times the number of columns to reduce by two the 99 percentile of the error. Again, our proposed random process predicts with great accuracy the 99 percentile of the error. For example, if we use the scenario proposed by Goldberg et al. [20] ($\alpha = \beta/2$; $\tau = (\beta - \alpha)/(\beta + \alpha)$), our solution predicts 128 columns are good enough, instead of more than 1024 that the other bounds give us.

Average function

Before studying the effect of increasing the number of rows, we need to consider whether is better to use the mean or the median to average each row estimation. Figure 4.10 shows the PMF of the error for sketches of size 32 by 32 and an estimation of 10,000 packets. As we can see, the median produces a biased estimation with a higher deviation, and therefore either the mean or a trimmed version of the mean should be preferred. The results were consistent for different sketch sizes and, for every sketch the mean provided the best results in terms of bias and deviation [31].

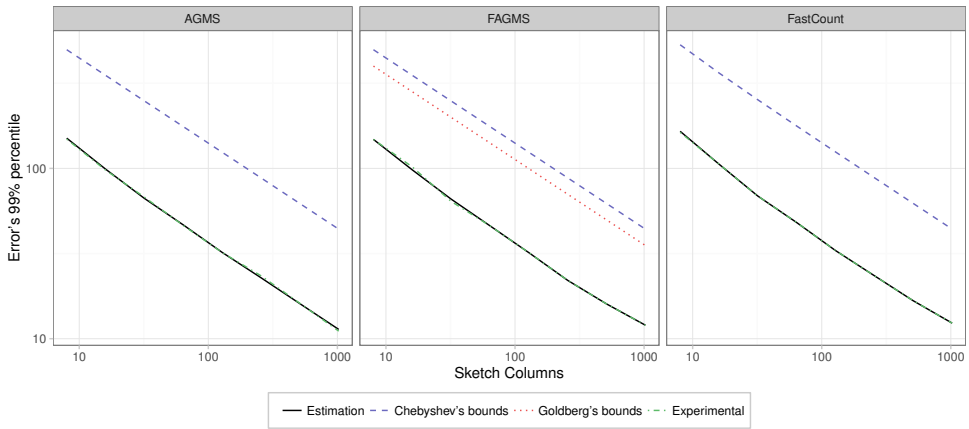


Figure 4.9: 99 percentile of the error vs. the number of columns

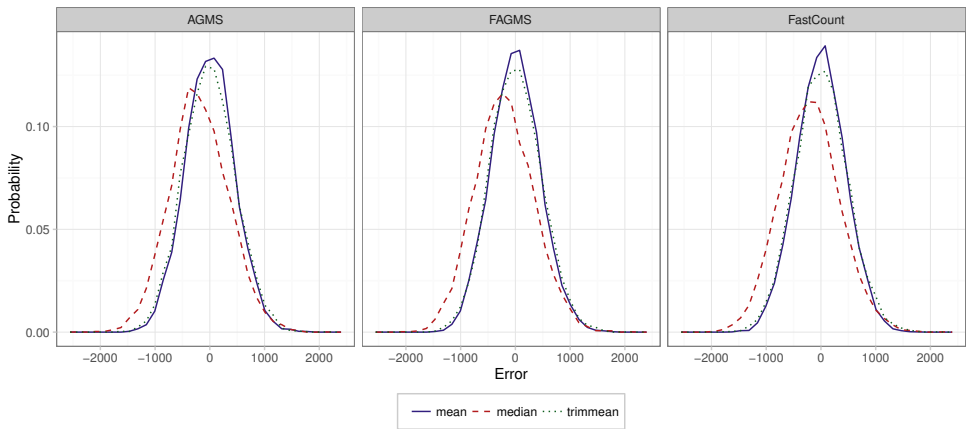


Figure 4.10: PMF of each sketch for a 32 by 32 sketch using different average functions.

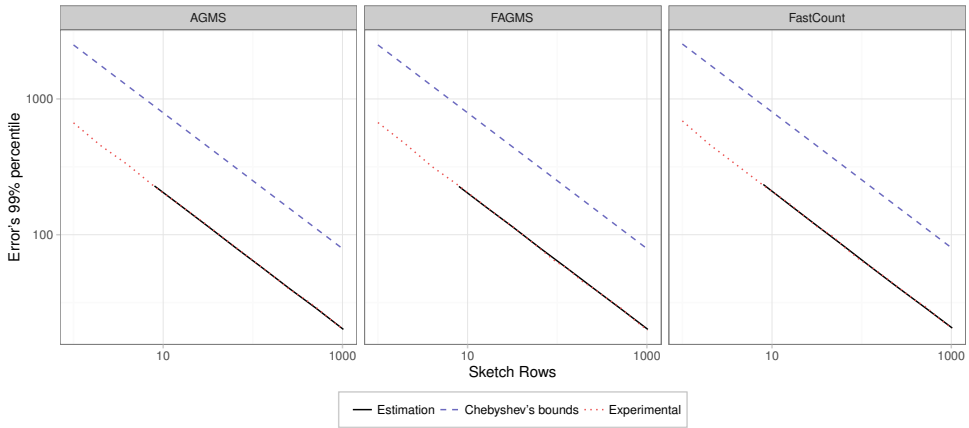


Figure 4.11: 99 percentile of the error vs. the number of rows

Number of rows

Figure 4.11 shows how the 99 percentile of a sketch of 32 columns is reduced when the number of rows, i.e. the number of basic estimators, is increased. As for the number of columns, the percentile decreases proportionally to $1/\sqrt{(}\mathrm{nr})$.

We used linear regression, to estimate the relation between different percentiles and the standard deviation of each of the sketches ($N/\sqrt{2\mathrm{size}}$ for AGMS and Fast-AGMS, $N/\sqrt{2\mathrm{nr}(\mathrm{nc}-1)}$ for FastCount) and found that they can be approximated with great precision using a Gaussian distribution with the given deviation and mean 0 [31], as long as the number of packets is relatively high. Therefore, the AGMS and Fast-AGMS provide slightly better approximation than the FastCount sketch when estimating the number of packets dropped by a network area.

Sketch's aspect ratio

Increasing the number of rows increases the sketch's accuracy; however, it also increases the cost in terms of memory and computational cost. Besides, compared with increasing the number of columns, for FastCount and Fast-AGMS and sketches of the same size, the one with more rows will be more costly in terms of CPU, since it will have to update more counters (one per

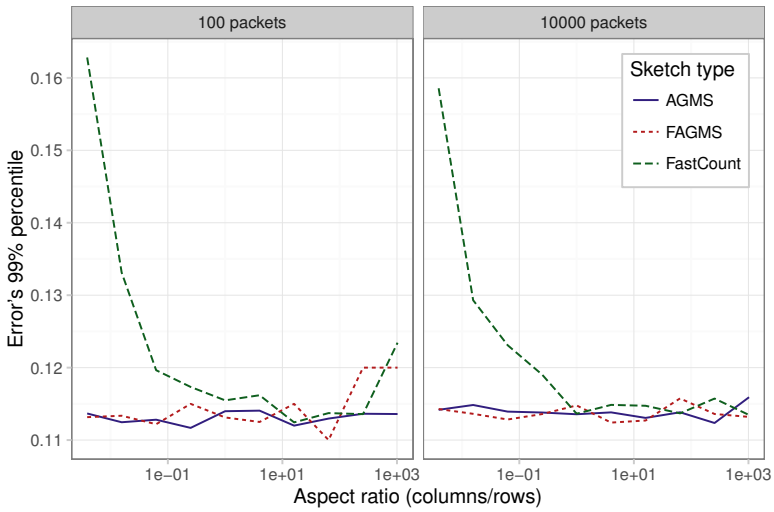


Figure 4.12: 99 percentile of the error vs. the sketch aspect ratio

row); and in terms of network bandwidth, because, on average, counters will reach a higher maximum value. Therefore, the question that arises is: what is the ideal aspect ratio of the sketch?

Figure 4.12 shows the 99 percentile of the relative error of a sketch of size 1024 as its aspect ratio varies. As we can see in the figure, AGMS and Fast-AGMS are not influenced much by the aspect ratio, but for the FastCount, we need at least as many columns as rows to obtain better results. After studying the results in more detail [31], we conclude that for the case of many packets, a single row sketch provides the best compromise, since it has the same accuracy than other aspect ratios, but better processing time and less network bandwidth requirement. On the other hand, because when there is a single row and just a few packets, the support for the error is just a small set of values, so the percentile values will highly depend on the percentile itself, so if we have some specific requirements, in some situations the single row will perform better and in others worse.

4.3.2 Estimating the ratio of dropped packets

Usually, we will be more interested in the probability that a packet will not be successfully transmitted through a network area, rather than the absolute number of packets that differ between the incoming and outgoing flow. When the network area monitored is a link or a path, monitors can exchange the sketches when they reach a certain number of sent packets, making both estimations equivalent. However, in the general case, a single node will not know the total number of packets, and therefore, after sharing the sketches, both the number of dropped (or modified) packets and the total number of sent packets will need to be estimated. As we will see, under this second scenario, the accuracy of the prediction is slightly different from the previous case, since now we have to make two estimations and then divide them to obtain the proportion of dropped packets. First, we study the effect on the accuracy of the actual ratio of dropped packets and the total number of sent packets. However, as mentioned previously, in the general case, monitors are more likely to share their sketches based on a time bound, rather than on a given number of sketched packets, since none of the monitors has a global view of the total traffic for a given network area; thus, our last experiment will compare the accuracy of each sketch for different time intervals and for each of the given pcaps.

Ratio of dropped packets

In this scenario, we assumed that 10^4 packets were sent per interval and a varying proportion of packets were dropped. Figure 4.13a shows the 99 percentile of the error of a 32 by 32 sketch. We compare two ways of estimating the proportion of dropped packets, *proportion* and *dropped*. *Proportion* estimates two different numbers: first the number of different packets ($\widehat{F_2}(\text{input}) - \widehat{F_2}(\text{ouput})$) and then the total number of sent packets ($\widehat{F_2}(\text{input})$) and then divides them. *Dropped* assumes that the total number of packets is known, and, therefore, it only needs to estimate the number of different packets. This is equivalent to the problem in the previous section. For the case of the FastCount, the number of total packets could be obtained directly from summing all the counters in a row for the sketch of the input's flow. For the AGMS and Fast-AGMS, an additional counter could be shared.

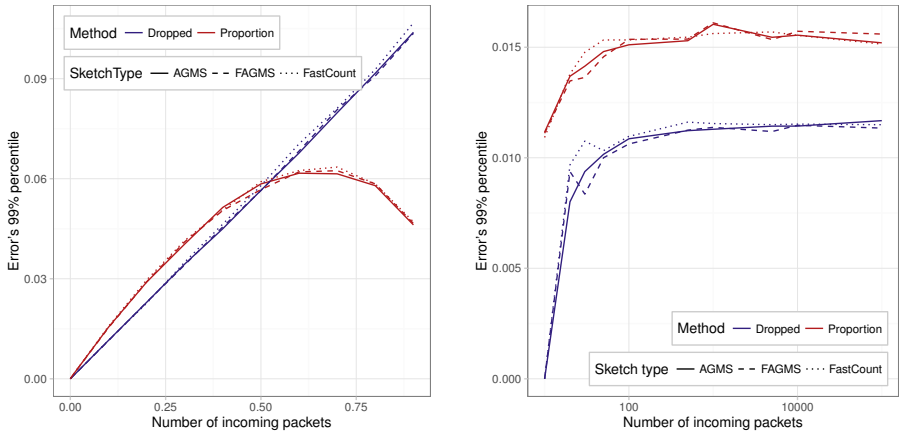
As we see in Figure 4.13a, the second method behaves as we showed on the previous section: as the ratio increases, so does the error and the relation is linear. However, if we must estimate both quantities, the error increases at first, but then, as the rate is bigger than 60%, the error decreases. This is due to the fact that the estimation of the number of different packets and total packets becomes correlated as the proportion of dropped packets increases, and therefore, the estimation is more precise; e.g. if the estimation overestimated the total number of packets, it is more likely to overestimate also the estimation for the number of dropped packets. Hence, if we are considering drop probabilities below 50%, using the second method will provide more accurate results; in contrast, if the drop probability is above 50%, the first will provide more accurate estimations. In any case, the differences between sketches is almost indistinguishable, though in almost every case the FastCount sketch performed slightly worse than the others.

Number of total packets

Now we assume that the percentage of dropped packets is fixed, 10%, but we increase the total number of packets sent. Figure 4.13b shows the 99 percentile of the error when predicting the drop probability, for a sketch of 32 by 32, and using the two strategies presented before. As expected, because the drop probability is below 50%, *dropped* obtains better results. In both cases, we see that the 99 percentile of the error grows until it stabilizes once the number of packets is big enough. As before, the difference between the different types of sketches is almost indistinguishable.

Time interval

But, as we said before, monitors will usually exchange traffic summaries after some elapsed time, rather than after an specific number of sent packets. For this case, as Figure 4.14 shows, we have studied the accuracy of each sketch type on three different networks using the three mentioned pcaps assuming a dropping probability of 10% and sketches of size 16 by 16. As we can see, longer intervals lead to higher precision, since for short intervals, the actual drop probability varies highly from one interval to the next one, so intervals with higher drop probability will cause the higher deviation on the



(a) vs. ratio of dropped packets (b) vs. the total packets

Figure 4.13: 99 percentile of the error

error (Figure 4.13a). This can be alleviated by averaging the estimated drop probability through several intervals, though, as we showed in [P1]. The figure is annotated with the average number of packets per interval, so to reach the minimum value for the 99 percentile, we need at least as many packets so that we will drop around 30 packets (e.g. in this case, around 300 sent packets). The time interval will then vary for each network accordingly to their network load.

4.3.3 Cost analysis

In this last section, we will study the trade-off between accuracy and cost. As we have seen, larger sketches will provide more reliable estimations at the price of higher requirements in terms of memory, network overhead and CPU consumption.

Memory and network overhead

To study the cost in terms of memory and network overhead, we have considered the same scenario as in the previous experiments. We assume that the memory required for a sketch will be proportional to its size, and each counter will take 64 bits. On the other hand, when transmitting the sketch over the network,

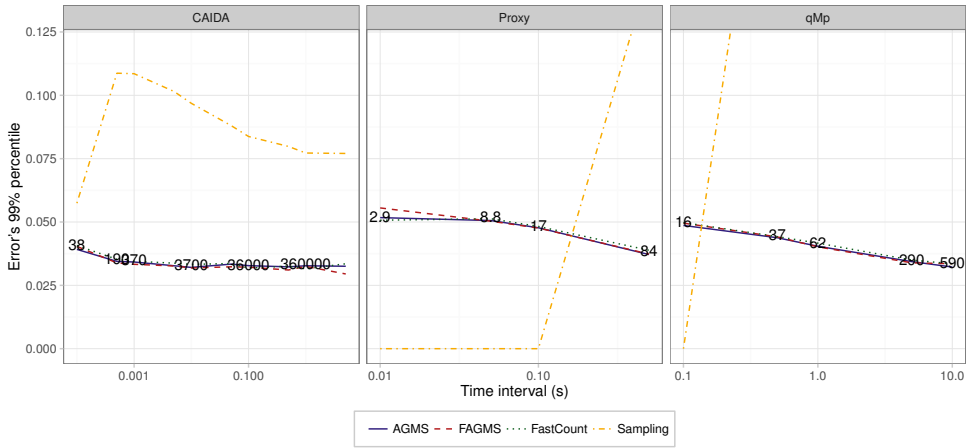


Figure 4.14: 99 percentile of the error vs. time interval

we assume that it will be compressed so that every counter uses only the necessary bits to code the largest value in the sketch.

Additionally, as baseline, we also measure the cost in memory and network overhead that a sampling mechanism would have, depending on its sampling probability. For sampling, we assume that the sampling digest has 32 bits [36].

Figure 4.15a shows the 99 percentile of the error for sketches of different sizes and sampling with different probabilities. We used the CAIDA pcap, a time interval of 5 ms and packets were randomly dropped with probability 0.1. As before, we considered estimating the drop probability by estimating both the number of dropped packets and the total number of packets (*ratio*) or only the dropped packets (*drop*). In this case, the latter will provide better estimations since the drop probability is below 50%. As the figure shows, the difference in the accuracy between sketches of the same size is negligible, and compared with sampling, sketches provide a better compromise when their size is smaller than 16 by 16 or the sampling probability is below 50%. In any case, the memory costs are always reasonably small, so, probably, it will not be the critical variable when choosing the optimal size of the sketches for a given network.

In comparison, Figure 4.15b, shows the 99 percentile of the error as a function of the network overhead. And in this case, the difference between sketches is bigger: for smaller sketches FastCount causes a higher network overhead, as its expected biggest value will always be bigger than the one for Fast-AGMS and AGMS, whose counters are not always increased by one, but randomly increased or decreased by each packet. However, as the size of the sketch increases, AGMS causes a higher overhead, because AGMS requires that every counter is updated, not only one for row, and therefore, the largest value will tend to be higher for AGMS.

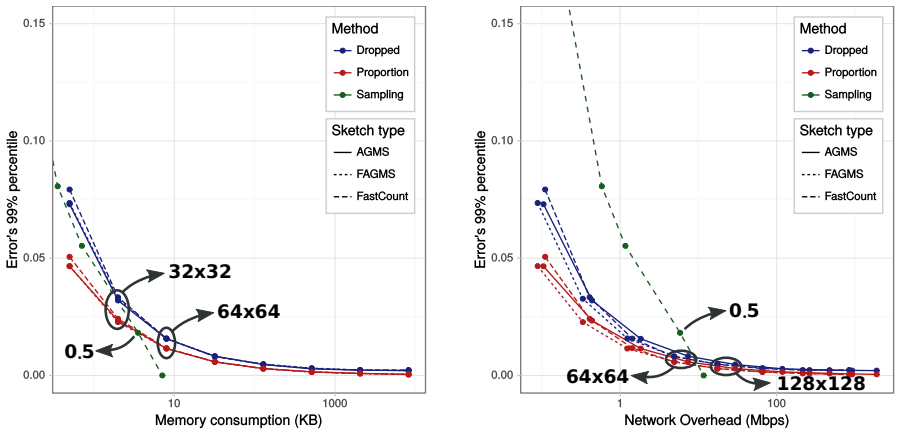
Compared with sampling, in this case, sketches provide a better trade-off than before thanks to its compression into less bits, and only when the sketches are higher than 64 by 64, sampling provides a better compromise. As we can see, the network overhead is considerable and in most of the cases will be the decisive factor that will determine the size of the sketches (or sampling probability if using sampling). To reduce the network overhead, we could reduce the size of the sketches, obtaining less accurate predictions or increase the time interval, delaying the detection. The best approach will depend on the network.

The effect of the time interval is different for sketches and sampling. For sketches, a longer time interval reduces the network overhead, because we only need to send one sketch each interval; however, for sampling, a longer interval does not reduce the network overhead, since it is determined by the average network load and the sampling probability; but a longer time interval produces better estimations than shorter intervals.

Figure 4.14, shows also the accuracy of sampling when adapting its sampling probability so that it has a network overhead similar to the sketches' one. As we can see, for very short time intervals, sampling should be preferred, as even using sampling with 100% sampling probability would cause less overhead than sketches, but as the interval increases, sketches provide better results, than sampling.

Processing time

The last cost that we have to consider is the cost in terms of CPU: the router should be able to process the packets faster than they arrive. There are



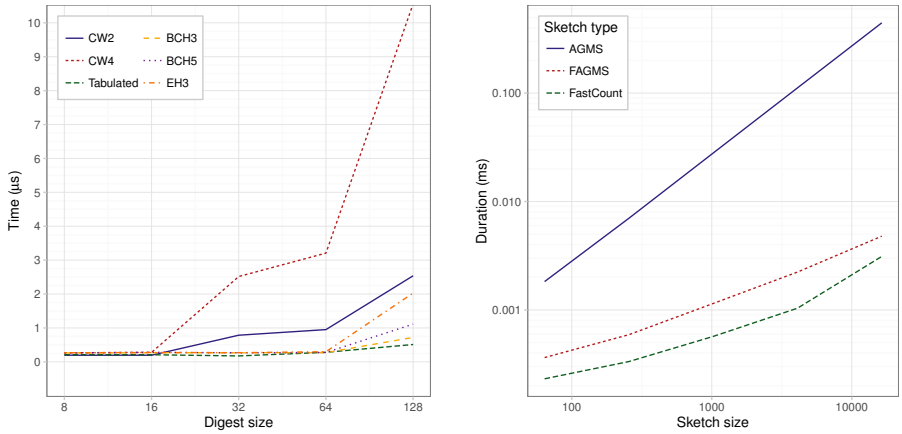
(a) vs. memory requirements (b) vs. network overhead

Figure 4.15: Sketch accuracy

three main factors that influence the CPU cost: the digest size, the pseudo-random function used as ξ and hash functions and the size of the sketch. The experiments on this subsection measured the time it takes to execute each operation on an Intel i5-3470 @ 3.20GHz, and assume that SHA256 is used to obtain the packet's digest.

In Figure 4.16a we can see the cost of executing the pseudo-random function, depending on the digest size. As the digest size increases, because we can't perform arithmetic operations with a single machine instruction anymore, the cost increases significantly. For hashes based on the Carter and Wegman trick, because their need to be over a bigger prime, the cost increases from digests of 32 bits, whereas for the other ξ functions, the increase begins for 128 bits. Tabulated hashing is capable of obtaining really fast results for any digest size. Results between the different ξ function, are quite similar for small digest sizes, and only for bigger digest sizes, we see how bch3 outperforms the other two. For hash functions, tabulated hashing produces the fastest results but it requires more memory [55].

Figure 4.16b shows the cost of updating each sketch without considering the computation of the packet's digest. We assume a digest's size of 32 bits and use tabulated hashing as hash function and bch3 as ξ function. As expected, the



(a) Pseudo-random functions (b) Sketch update

Figure 4.16: Measurements of the processing time

cost increases proportionally to the sketch size for AGMS and proportionally to the number of rows for the FastCount and Fast-AGMS sketches.

Finally, Figure 4.17 shows the cost of updating a sketch of 16 by 16. For Fast-AGMS, we assume that cw2 was used as hash function; however, tabulated hashing could be used instead, and it would show the same cost for computing the hash as in FastCount. In terms of CPU requirements, the best option is FastCount using tabulated hashing, and its cost is comparable to that of Fast-AGMS if we also use tabulated hashing. However, tabulated hashing requires additional memory, e.g. for 32 bits hashing, each independent hash required 512 KB; and for 64 bits, 3.5 MB. In the case were such additional memory cost is not reasonable, the size of the sketch becomes relevant, and AGMS with bch3, bch5 or eh3 can have a cost close to that of Fast-AGMS and FastCount.

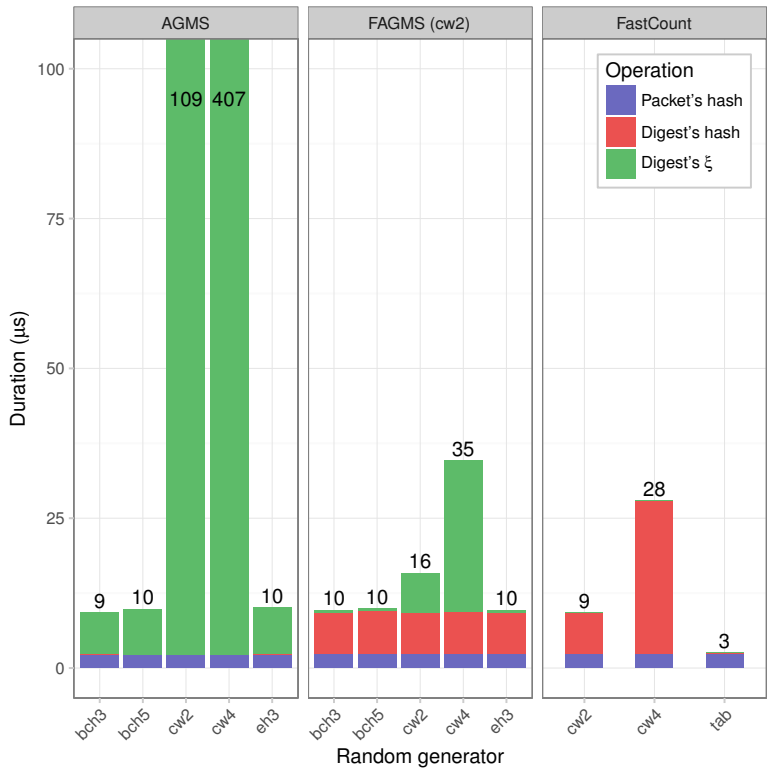


Figure 4.17: Time execution of each sketch update operation

4.4 Discussion

4.4.1 To key or not to key

A sketch is updated by first computing the packet's digest using a hash function, H, and then applying a set of hash and ξ functions over the packet's digest. It is imperative that the network area being monitored is not capable of predicting an alternative packet that alters the sketch in the same way, i.e. the sketch update process must be second pre-image resistant, otherwise, it can change the packet with its alternative, effectively disturbing the traffic, but avoiding detection.

There are two approaches to make sketching robust to second pre-image attacks: we can use a keyed hash function and keep the key secret from the monitored network area; or we can make sure that the cost of obtaining an alternative packet is too high in terms of CPU and memory. The first approach has been proposed by Goldberg et al. [21] and its benefits are twofold: on one hand, we can use faster implementations for the hash function that computes the packets digest, H_k, which as we have seen, is the bottleneck when using tabulated hashing. On the other hand, we can also use smaller sketches, since the adversary has no way of knowing beforehand how the sketch will look like. The main drawbacks of this approach are that, first, monitors for a network area need to synchronize and design a mechanism for obtaining the keys for each period; and, second, that if we want the entities being monitored be able to detect false reports, (i.e. be able to study the resulting sketches and determine if they are truthful), they need to keep a copy of all the traffic for an interval, and then, after the key is released, process all that traffic and compute the resulting sketch to compare it with the reported sketch. This requirement will be typically unreasonable in terms of memory, so if false report detection is required, this approach is not viable.

For the second approach, we first need that H is second pre-image resistant, since two packets with the same digest will update the sketch in the same way. Using an unkeyed cryptographic hash function ensures that the hash is second pre-image resistant, but if we limit the number of bits we take, we may still be vulnerable to brute force attacks, e.g. if we only use 32 bits, we can fit packets that produce any possible digest in 80 GB (assuming they are 20 B). Next, this digest is going to be used to update the sketch, and if the sketch is too

small, the possible ways in which the sketch can be updated may be lesser, and, therefore, ease finding a collision. For example, if we have a FastCount sketch of 256 columns and one single row, simply knowing 256 precomputed packets that hash each to one column will allow us to change any given packet for the precomputed one that falls in the same column. In more detail, the number of different ways a sketch can be updated is nc^{nr} for FastCount, $(2nc)^{nr}$ for Fast-AGMS and 2^{nc+nr} for AGMS. So, the weakest point of the sketching process will be the minimum between number of possible digests (2^{bits}) and the possible ways of updating an sketch; therefore, the bits of the digest, and columns and rows of the sketch, should be chosen in such a way that the cost of a brute force attack is disproportionate for the considered scenario. In contrast, this approach allows us to have loose synchronization between monitors as we will see later in chapter 6 and to detect false accusation easily by having every node compute the sketch online. However, the sketch update is slower because of the use of a cryptographic hash, longer digests and bigger sketches; and more memory and network bandwidth is consumed because the sketches are bigger.

Finally, we can apply a hybrid solution, using a key for the process after obtaining the packet's digest, for example by xor'ing the digest with a key or using a keyed hash function [21], but still using a second pre-image resistant hash. This will alleviate the memory constrains for the false report detection, since now the monitored entity only needs to keep the packets' digests, instead of the whole packets and will also allow the use of smaller sketches, reducing the network overhead. Still, the CPU cost of the initial hash function cannot be avoided.

4.4.2 Choosing the proper sketch

As we have seen, there are many factors that influence the precision of the sketches, on this section we will go through them and present a summary of the results obtained in Section 4.3.

- **Sketch type**: in most of the cases, Fast-AGMS is the sketch to go for, since it provides slightly better precision than FastCount and better network bandwidth requirements. However, if the processing time is the most critical factor, FastCount may be chosen instead.

- **Sketch size**: the size of the sketch will be determined by the detection requirements and network characteristics, i.e. what is the expected behavior of non-faulty routers? and how many false positives and false negatives are we willing to tolerate? Using that information we can then determine for a given percentile which is the max error that should the estimation give us.

- **Digest size**: choosing the proper size is especially important when we are estimating the proportion of packets, because in such case, the estimation for faulty behavior is underestimated [31]. In any case, in most of the cases, using a 32 bits digest size should be enough.

- **Aspect ratio**: the aspect ratio will depend mainly on whether we are using keyed or not keyed functions. If we are using keys, a single row sketch will give us a better trade-off between cost and precision; its PMF will not fit as well the Gaussian approximation, but we can use the proposed random processes to compute its PMF and give accurate predictions about the error for a given percentile. On the other hand, if we are not using keys, using square sketches will provide a more secure sketching process, and a reasonably good sketch accuracy and processing time cost.

- **Pseudo-random functions**: as we have seen, all the analyzed pseudo-random functions provide the same accuracy. In terms of processing time, tabulated hashing gives the best results, but it requires way more memory. Depending on which is more important, memory or processing time, cw2 or tabulated hashing should be chosen as hash function. For ξ, bch3 provides the better results.

- **Time interval**: finally, choosing the proper time interval will allow to fine tune the trade-off between network bandwidth requirement and responsiveness. Moreover, the time interval should be small enough, so that packet repetition is kept at bay.

As a final remark, between estimating the proportion of packets or the total number of packets, we have seen that in the expected case (packet drop should certainly be below 50%), the latter provides a better estimation, so it would be always better to provide the additional counter with the total number of

packets per sketch; and the estimation method could be chosen using the estimation itself, i.e. when the estimation is below 50%, estimate only the total number of packets different and use the counter with the total number of packets, on the other hand, when the estimation is above 50%, estimate both numbers using the sketch.

4.5 Conclusions

This chapter describes AGMS, Fast-AGMS and FastCount as random processes for the case of traffic validation. Thanks to this representation as a random process we are able to provide tighter bounds on their error, allowing us a dramatic reduction in the memory requirements for given specifications in false positives and negatives. Furthermore, we have measured the effects of each of the parameters involved on the sketch's accuracy and its cost in terms of CPU, memory and network overhead. One remarkable difference between sketches for traffic validation and for other applications is that, because of the characteristics of data traffic, we do not require pseudo-random functions as strong as on the other cases, e.g. bch3 and cw2 perform just as well as bch5 and cw4. Another important remark is that when estimating a proportion below 50%, knowing the total number of packets increases the prediction accuracy. Also, we should notice, that even though for sketches of the same size, AGMS tends to produce better results, when comparing the network bandwidth required, Fast-AGMS outperforms AGMS, and so does FastCount for bigger sketches.

We believe an important point to further study is finding a better function that reduces a variable length packet into a fixed size digest. As we have discussed in section 4.4, we require a hash function that is second pre-image resistant, but using traditional cryptographic hash functions can be too costly in terms of CPU for networks with high packet-rate, because they were specifically designed to be sufficiently slow. Other hash functions should be studied, to see which fits better for the problem at hand.

Chapter 5

Distributed Detection

In the previous chapter, we have described how monitors can summarize the traffic that enters and leaves a network area and characterized their accuracy and precision. In this chapter, we will address the next step to locate forwarding faults: how the traffic information is shared among peers to be able to detect those network areas that are faulty, as a single monitor does not have complete information.

If we consider the example in Figure 5.1 it is clear that a single monitor does not have complete information. Imagine that we want to determine the behavior of the network area M. A, B, C and In the only case that monitors could determine M's behavior would be in the context of wireless networks, if M was a single node and its neighbors used overhearing techniques. However, in the ase of WCNs, if we have supernodes, that is no longer the case, because the use of several antennas, with different frequencies or orientation, as the monitor will not be able to overhear all of M's traffic. Moreover, there are some attacks to which overhearing is not resilient [28].

As a consequence, we need a dissemination mechanism that will share the traffic summaries from each monitor, so that at least one entity has the global view of the forwarding behavior of M, and can determine M's behavior.

As we will see in the following sections, the main challenges that a traffic summary dissemination protocol needs to face are false accusation and collusion. False accusation refers to the fact that some monitors may lie, and alter the traffic summaries for M, causing M to be falsely detected as faulty. Collusion

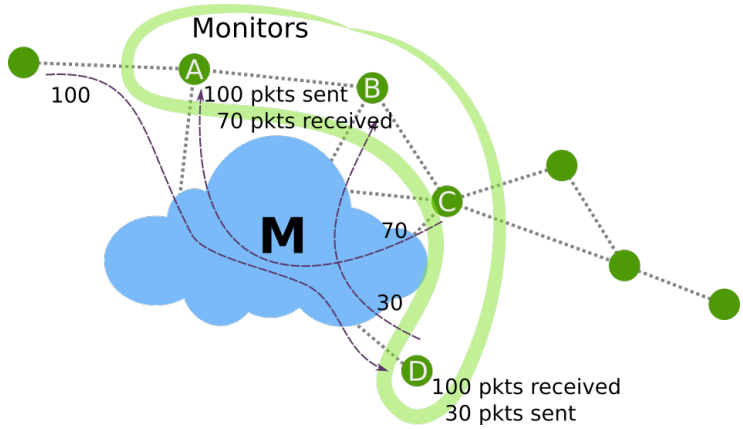


Figure 5.1: Example network

is when M and some of its monitors (A for instance) collude, so that A will send false traffic summaries, allowing M not to be detected despite of its faulty behavior.

On this chapter, we propose a traffic summary dissemination and distributed detection protocol, KDet, capable of handling false accusation and collusion and that doesn't require knowing the path a packet follows, 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, and then propose a fish-eye approach to reduce the network overhead of KDet, at the cost of delayed detection.

5.1 Background

As mentioned in the introduction, and following the example in Figure 5.1, if nodes A, B, C and D have only their partial view of the traffic going through M, they will have to collaborate by sharing their traffic summaries in order to determine whether M is behaving properly or not. The way they share their traffic summaries is what we call the dissemination and distributed detection sub-problem in forwarding fault detection, and it is the focus of this chapter.

Because distributed detection is not easy, some solutions propose to skip this part of the problem altogether by means of using overhearing techniques [32, 39, 49]. Overhearing tries to make the most of the radio medium by actively listening to the monitored nodes transmissions, making sure they retransmit the packets sent to them. However, there are some cases when overhearing is not well suited, e.g. if a node has several directional antennas, its neighbors won't be able to listen to all the traffic it is forwarding as they will only receive the traffic sent by the antenna pointed towards them; the same would happen if the node has two antennas, one operating at 2.4 GHz and the other at 5.0 GHz. Besides, there are several attacks that can be implemented to avoid detection when using overhearing techniques [28], like sending packets to a non-existent neighbor or without enough power. Considering these caveats, forwarding fault detection in WCNs should not rely in overhearing techniques.

As we have seen in Chapter 2, the solutions proposed in the literature can be classified in those that keep a summary per flow and those that aggregate the traffic (by neighbor for instance) to reduce the number of summaries. Because the number of summaries grows with $O(n^2)$, flow-based solutions tend to keep their findings local, i.e. only the source of the traffic flow knows which node or link is failing. Moreover, they usually rely on a routing protocol that allows them to know the path between the source and destination, which is not the case of WCNs, as they sometimes use distance-vector protocols. Finally, flow-based solutions tend to assume that traffic flows have a stable path, whereas in wireless networks, paths may fluctuate as the link quality fluctuates.

There are two main challenges that need to be addressed when we consider aggregated traffic and a global scope. First, the detection protocol needs to be resilient to false accusation, i.e. if a node emits a false report about a neighbor, the neighbor should not be detected as faulty. Second, the detection protocol should be resilient to collusion, that is, a neighbor node should not be able to claim that the packets dropped by the monitored node were received and targeted for itself, so forwarding to the next hop is not expected.

The only approach presented in the literature that considers aggregated traffic summaries and faces collusion are Π_2 and Π_{k+2} [35]. But understanding why the other are vulnerable to collusion attacks will help us to better understand

the solution proposed in this chapter, so follows an example that discusses showcases them.

WATCHERS [10] has every node (N) monitor each of its neighbors (M) by keeping a set of counters. Concretely, for every destination, N keeps track of how many packets it has sent and received from M and how many of those packets originate from M. Periodically, those counters are exchanged by flooding them through the network. Once N receives all the counters from M and its neighbors, it can start the detection process. First, it will do a preliminary check by making sure that it has received M's counters and just a single version of them, if M fails this preliminary checking, N will detect M as faulty and as response, it will disconnect from M. Then, it will validate each of M's links by comparing the counters announced by M and the node at the other end. If a link fails this validation, it will be added to the check-set; and N will expect M to disconnect from the other end-point of the link on the next iteration, or it will detect M as faulty. The final check consists on comparing the whole set of M's neighbors counters, using conservation of the flow, i.e. for every destination M is expected to have received the same number of packets it sends to others if we do not consider those that originate from M. The main problem with WATCHERS is that because it only distinguishes packets by destination, a faulty node may drop as many packets towards a destination as it sends itself to that destination if it has a colluding neighbor. Consider the network in Figure 5.2. Let's assume that M and C are two faulty colluding nodes and that there is a traffic flow from S to D of 100 packets and another one from M to D of 50 packets. Node A will forward D's traffic towards M, but M instead of forwarding S's packets, drops half of them. During the reporting period, M will announce that through its link to C it has sent 100 packets destined to D, and none of them were originated from itself, C, because it is also malicious will corroborate its story, saying it has indeed received 100 packets from C destined to D; and neither of them will be found guilty, because none of the neighbors of C will keep enough information to remember that part of those packets iare from M instead of from S. WATCHERS is also vulnerable to other attacks described by Hughes et al. [22]. On the other hand, WATCHERS is not vulnerable to false accusation because counters are flooded through the network and if a node announces counters that are false, the accused node will disconnect from it.

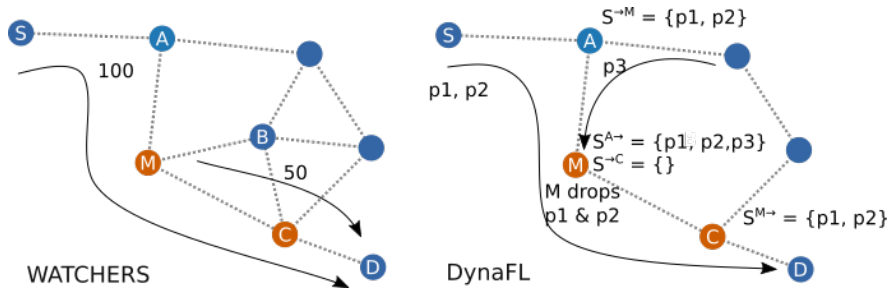


Figure 5.2: Example networks to showcase collusion

In a different manner, DynaFL [61] and χ [36] are also vulnerable to collusion. Both of them propose similar approaches: every node M is monitored by its neighbors, who keep traffic summaries of the traffic sent, $S^{\rightarrow M}$, and received from M, $S^{M\rightarrow}$; in the case of DynaFL, a sketch, and a set of fingerprints for χ. The most important fact about those traffic summaries is that they do not include the traffic information about the packets destined to or originated from M, so that they can be compared afterwards to decide whether M is behaving properly or not. So let's we consider the network in Figure 5.2, where node S wants to send two packets to D (p1 and p2), but M drops them and there is an additional packet from another side of the network destined to M (p3). In this situation, the traffic summary in A regarding the traffic sent to M will only consist of p1 and p2, because p3 is destined to M. If C is a colluding node, it could report that it has actually received p1 and p2 "pretending" that they were destined to itself (remember that the summaries are a sketch or hashes of the packet, so no information about the destination is included, so C cannot be found out), effectively helping M to avoid collusion. At the same time, C is not found faulty because M doesn't report p1 and p2 in $S^{\rightarrow C}$, as if those packets were destined to C. The fact that traffic summaries have aggregated information instead of per flow information is what allows M and C to collude to avoid detection.

The difference between DynaFL and χ is 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. Because $S^{\rightarrow M}$ and $S^{A\rightarrow}$ are not comparable as they refer to different subsets of traffics (one includes p3 and the other doesn't), one could see how a malicious node could

lie, and the central authority wouldn't be able to discern whether a node is falsely accused or it is faulty. DynaFL solves false accusation by accusing the whole neighborhood (i.e. including the reporter and reportee) instead of only the node being monitored, so that if A sends a fake traffic summary about B, it will also be part of the nodes detected as faulty).

χ proposes a different approach that solves false accusation without the need of losing precision on its detection. In χ every neighbor will send its traffic summaries to the other neighbors through the node M being monitored, which allows M to make sure those are the expected traffic summaries by comparing them with a local version and disconnecting from nodes that are falsely accusing M. Finally, when the neighbors receive all the summaries, the behavior of M 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, but in any case, as long as the neighbors are able to detect the faulty nodes, they could disconnect from them and prevent them from disrupting the network.

Going back to Π_{k+2} and Π_2, they propose to monitor all monitors all the path segments of length 3 to $k+2$ (where k is the maximum number of adjacent faulty routers in the network), either just by the end-points (Π_{k+2}) or by every node in the path (Π_2). Π_{k+2}, detects the whole path segment as faulty when their traffic summaries are not consistent, and by including the reporters themselves as part of the path segment, false accusation is avoided. In Π_2, false accusation is solved by using consensus among all the nodes in the 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.

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} and Π_2. Our work presents a solution that covers false accusation and collusion, but in contrast with Π_{k+2} it does not require path knowledge. 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.

5.2 Problem statement

KDet's goal 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. [34]:

- *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 $|S| \leq a$.

Additionally, the goal of KDet is to provide such solution in the context of WCNs, and, therefore, their characteristics must be considered when defining KDet. 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-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, KDet:

- Cannot rely on a given routing protocol, i.e. should be based on data traffic, not control traffic.

- Must consider nodes with several antennas.

- Must scale up to thousands of nodes.

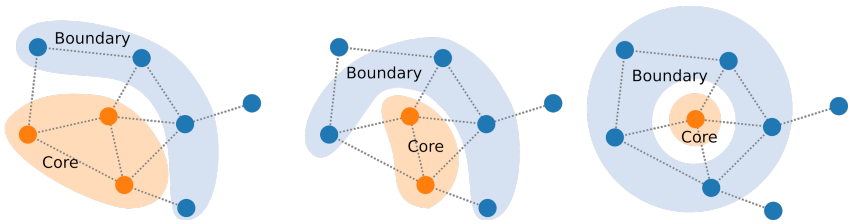


Figure 5.3: Examples of cores and boundaries

5.3 System Model

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

5.3.1 Network model

We assume that the networks on which KDet runs are similar to WCNs, so 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, etc.), 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. In KDet every router takes both the role of monitor and suspect. 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 5.3 shows for the same network some cores and boundaries as example.

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, a lightweight mechanism could be SEMTOR [37], which is part of the experimental BMX7 routing protocol, developed as an evolution of BMX6, and used in several WCNs. A reliable

neighborhood discovery mechanism must exist, something like the connectivity maps used on several WCNs [6]. As a side note, keeping an updated vision of the neighborhood is easier than maintaining updated path information [29]. 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 those TA that are responsible for monitoring its behavior. Possible ways to implement the communication mechanism with the TA are the intrusion-tolerant overlay by Obenshain et al. [38] or using the spanning tree proposed by Zhang et al. in DynaFL [61].

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.

5.3.2 Threat model

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; for instance, it could provide false traffic summaries or simply not participate in the protocol. A *faulty* node is either p-faulty, t-faulty or both; and a core or boundary is p-faulty, t-faulty or faulty when at least one of its nodes is p-faulty, t-faulty or faulty, respectively.

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 the reports instead of flooding them.

As Mizrak et al. [34], we measure the strength of the adversary as the largest set of connected faulty nodes, expressed by k. For example, in Figure 5.2, $k = 2$.

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.

5.3.3 Traffic Validation function

KDet is a distributed 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 5.6), it should be possible to add and subtract traffic summaries. Luckily many summary functions have defined + and − operations, such as sketches [48, 20, 61], counters [10], fingerprinting [36] or sampling [20]; 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, $\boldsymbol{\mathcal{V}}(S_{\mathrm{in}}, S_{\mathrm{out}})$, that evaluates a core given the summaries of the traffic entering and leaving that area. $\boldsymbol{\mathcal{V}}$ equals *true* when the behavior of the core is as expected and *false* when it is under-performing. We assume that whenever there is a t-faulty node in the evaluated core the validation function will classify it as faulty (i.e. $\boldsymbol{\mathcal{V}} =$ *false*). Most solutions proposed in the literature are based on the conservation of the flow principle, i.e. traffic flow entering a network area should be approximately the same as the traffic flow leaving it, without considering the traffic destined to or originated from that network area [10, 61, 21]. Mizrak et al. [36] proposes a validation function based on the probability of losing a packet based on the node congestion, as estimated with the traffic rate and buffer size. Shu and Krunz [51] use the correlation between packet losses to determine if the loss pattern corresponds to normal operation or the router is faulty instead.

5.3.4 Summary of assumptions

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

1. Bidirectional links that agree on the traffic exchanged.
2. Public key infrastructure.
3. Neighbor discovery mechanism.
4. Trusted Authorities reliably connected with monitor 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.

5.4 The KDet detection protocol

KDet implements a detection protocol that monitors a set of nodes from its boundary with the rest of the network. KDet's main goals are that if the core is faulty, the boundary can detect it, and if the boundary falsely accuses the core, the core can detect that. This is achieved by designing KDet to satisfy two principles:

Detection If the core is t-faulty and there is no p-faulty node in the boundary, the core is detected as faulty.

No false accusation If the core is not faulty, then it is never detected 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 *coordinated detection* phase.

5.4.1 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.

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

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, 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 [41] 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 reasonable 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 (5). After that timeout, if any report is missing, or its signature is not valid, the core is suspected. Otherwise, the boundary node evaluates the behavior of the core using the validation function:

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

Then, every node emits its verdict by sharing with the corresponding TA a signed core evaluation, which consists of $V_{\text{core}}(T)$, a bitmap indicating which reports have been received (received_reports) and a nonce:

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

5.4.2 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.
2. $V_{\text{core}}(T)$ 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) will go through the steps that follow:

1. At the end of the monitoring interval, C expects a report from B. In case there is no report, its signature is not valid, or there is more than one version, C will consider B p-faulty.
2. C will compare the traffic summaries in the report related to the link i, $S^{\rightarrow i}_{\text{core}}(T)$ and $S^{\leftarrow i}_{\text{core}}(T)$, with its own local version, and if they are not the same, B is considered p-faulty as well.
3. C compares the $V_{core}(T)$ announced by B with the one that results from combining the reports that have been exchanged (available to any node in the core because of robust flooding) and that received_reports announces the summaries that C has sent to B as received (because of robust flooding, C knows about all the reports that have been sent to B). Again, if the results do not match, B will be considered p-faulty.
4. If in any case B is considered p-faulty, then C disconnects from it.

5.4.3 Coordinated detection

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

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

Where boundary$(\text{core}, T_2) \subseteq$ boundary(core, T_1) implies that for every link existing between the core and the boundary at time T_2 the same link also exists at time T_1.

5.4.4 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 $\leq k$.

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

5.5 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:

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

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

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

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:

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

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 (steps 1 and 3 of the core protocol) and disconnect from it, changing the boundary as expected. In the second case, because the core is not faulty we know that the traffic summaries satisfy:

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

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 (step 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 (step 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.

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

Now, let's consider the set of monitored cores as defined in section 5.4.4. Thanks to Lemma 1, we can prove that KDet is k-accurate, since every time a correct router detects a core as faulty (core $\in$ detected) it has to be a faulty core or it will contradict Lemma 2. Furthermore, given the definition of C, such core will always have a size below 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 $\mathrm{c}_i = f_i$ 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 $|\mathrm{c}_i| = |f_i| \leq k$, KDet is k-complete.

5.6 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.

5.6.1 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 a core's traffic summary can be computed as:

$$S^i_{\text{core}} = \sum_{\forall n \in \text{core}} S^i_n \tag{5.1}$$

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. [34] 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}.

5.6.2 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.

5.6.3 Example

However, the given trends are only an upper bound to the actual space and overhead consumed by the protocols. Figures 5.5a and 5.5b show which would be the real cost of each detection protocol for the guifi.net Barcelones area network (Figure 5.4) [1], a network with 113 nodes and 134 links; assuming there is traffic between every pair of nodes.

Figure 5.5a shows the state size stored by each node for the guifi.net Barcelones area [1] assuming summaries of 500B [61, 20], 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 [29].

Figure 5.5b 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. For this case, we can see that at most around 10MB of data needs to be shared per link, so for

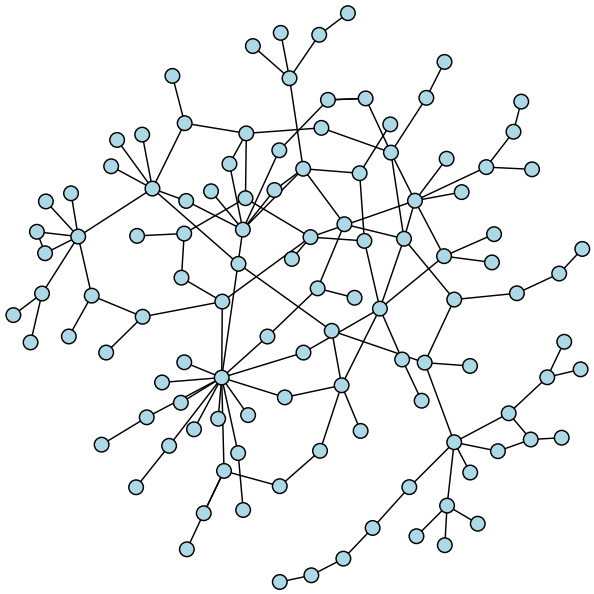
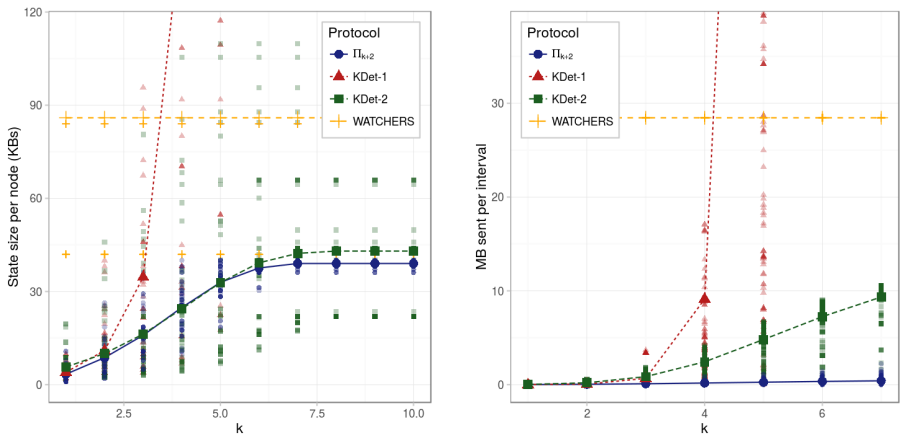


Figure 5.4: State size



(a) State size

(b) Network overhead

Figure 5.5: Protocols' cost

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 proxy. In those cases, as we will see in the following section, network overhead is kept well below the limits that we have shown. Moreover, to alleviate the effect of such overhead, techniques such as LEDBAT [2] can be used to avoid clogging the network links with the protocol's traffic.

5.7 Simulation

In the previous sections, we have studied the KDet protocol from a theoretical and analytical 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 version of KDet with sketches as summary function and conservation of the flow as validation mechanism in OMNeT++ and studied its detection accuracy and network overhead.

5.7.1 Simulation model

Our simulated network consists of a set of network nodes, a clock, a graph server and a trusted authority (see Figure 5.6). 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 proxies and hosts: hosts send and receive traffic from the proxies; 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 can use either IPvXTrafGen to generate traffic or a text file,

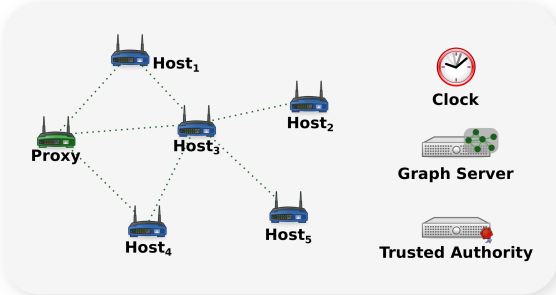


Figure 5.6: Simulation network model

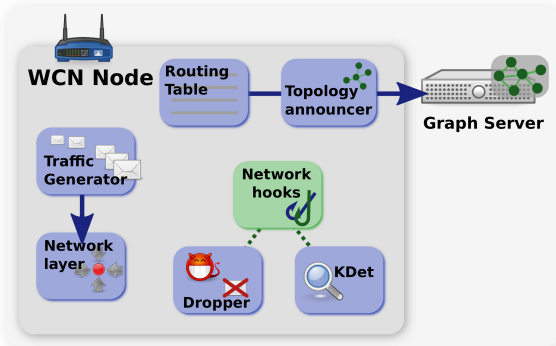


Figure 5.7: Simulation WCN Node

which is useful because we had some traffic captures from a Guifi.net proxy that we have used for the experiments of shorter duration.

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.

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 on section 5.4. The monitor uses network hooks to

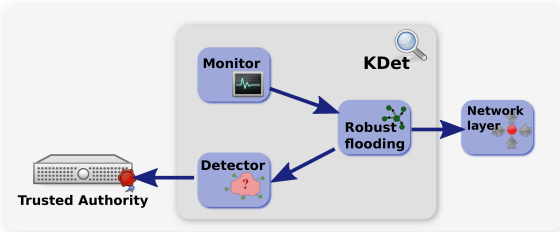


Figure 5.8: Simulation KDet module

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 they 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 [41]. 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.

5.7.2 Experiments

In our first experiments, we will measure the precision 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. Even though we haven measured the precision of sketches in the previous chapter, because we do not know a priori which will be the drop probability of non-faulty nodes (α), we cannot estimate the probability of a false positive or a false negative.

Sketch Size

Our first experiment simulated a network of 50 nodes and 3 proxies. Each node would connect to one of the 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 with a time interval of 60 seconds.

Figure 5.9 shows KDet's ROC curve, showing that even though bigger sketches produce more accurate predictions, as soon as the sketch is bigger than 1 by 32, the benefits of increasing the sketch size are not noticeable. With a sketch of size 32 columns or bigger, we achieve more than 99% detection rate and 1% of false positives.

On the other hand, as one can expect, bigger sketches imply more overhead and memory requirements, as can be seen in figures 5.10 and 5.11, respectively. We assume that sketches are represented as a matrix of numbers, represented by the same number of bits, which is determine by the value of the highest value in the matrix. Evidently, the traffic overhead grows for bigger sketches, since they must share bigger matrices; and those with the same size, but more columns and less rows, cost less because statistically they will require less bits. For the needed memory, because we store the data in a data type of fixed length (a double), the ratio between columns and rows does not affect the memory required. In any case, using the smaller sketch that satisfies our requirements is the best option. For instance, if we choose a sketch of 32 columns, we would require less than 0.05 kbps of control traffic and 4 KB memory per node, both well below the bounds found in the previous section.

Protocol interval

Then we measured the precision using different time intervals, using a sketch of 64 columns and 1 row; and the rest of the simulation parameters as before. As we can see in Figure 5.12, 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. Comparing Fast-AGMS and FastCount, there is almost no difference between both of them These results are in line with the findings presented in Chapter 4.

In terms of overhead, as expected, the longer the interval, the less control traffic. And Fast-AGMS requires more bits because its ± 1 function implies

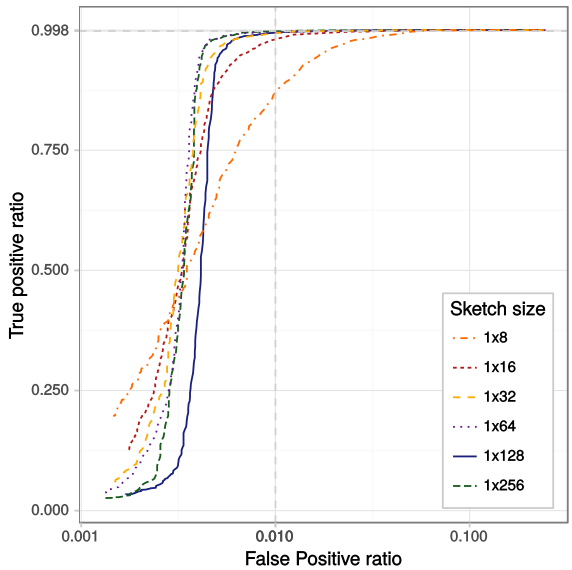


Figure 5.9: ROC curve for different sketch sizes

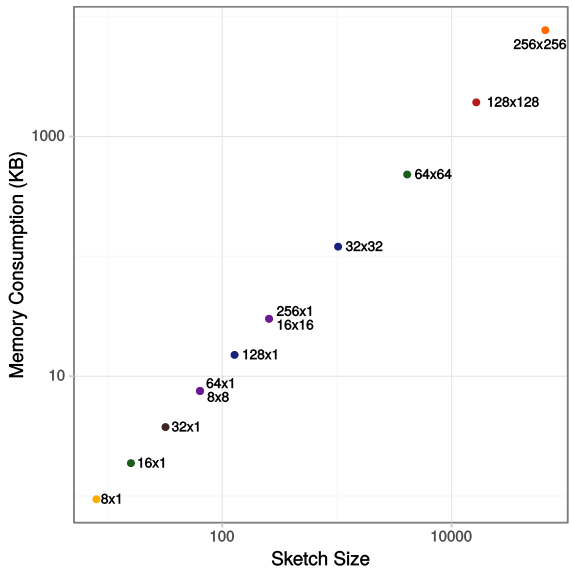


Figure 5.10: Traffic overhead for different sketch sizes

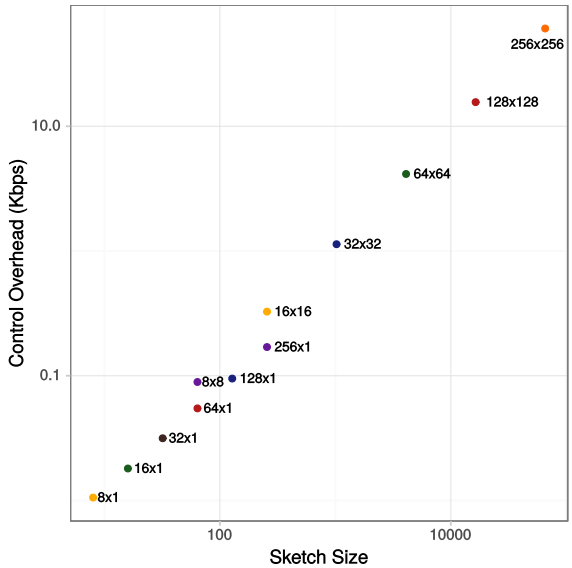


Figure 5.11: Memory requirements for different sketch sizes

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.

Value of k

Our next experiment considers collusion, and to detect it, we run KDet with a variable value for the parameter k. Figure 5.14 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. On the other hand, 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 5. As we can see, if we

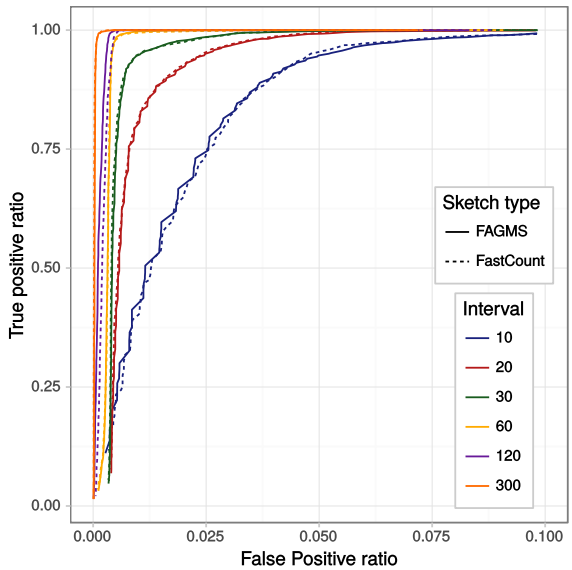


Figure 5.12: ROC curve for different intervals

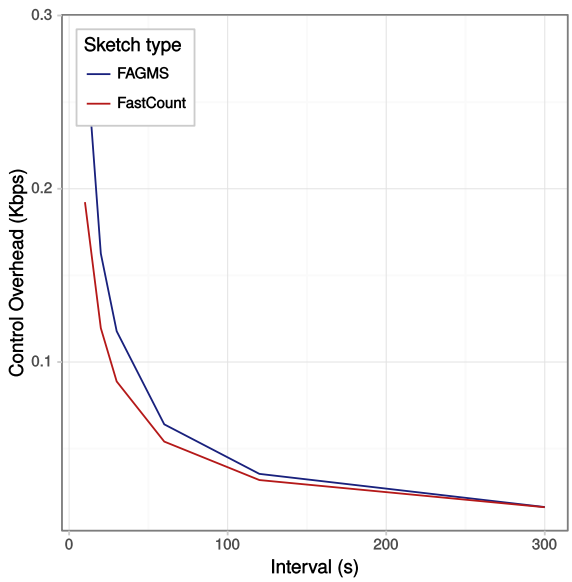


Figure 5.13: Traffic overhead for different intervals

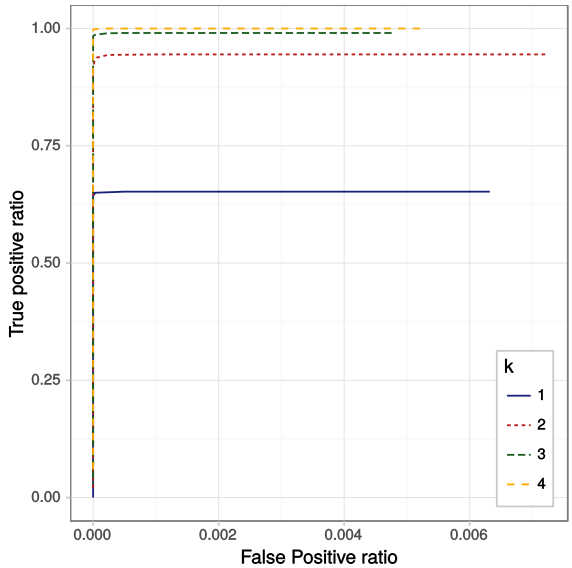


Figure 5.14: ROC curve for different k values

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.

We study the cost of each of the two implementations presented. Figure 5.16 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). On the other hand, for the memory (Figure 5.17), 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 where more than k hops away. Again, the cost seems reasonable, as each node is required to save at most 150 KBs.

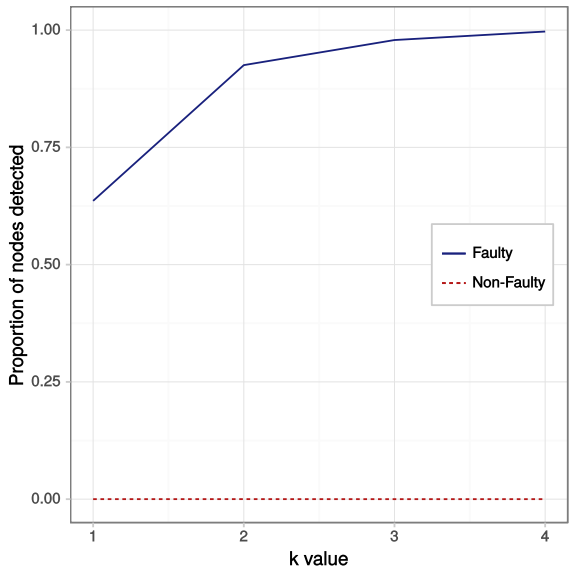


Figure 5.15: Proportion of nodes detected for different k values

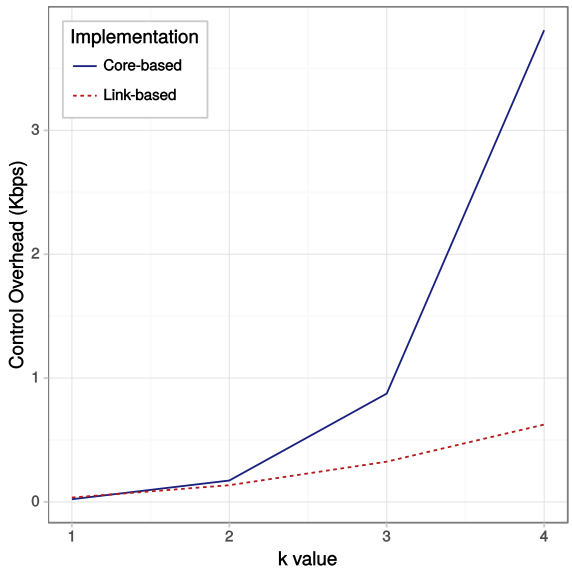


Figure 5.16: Traffic overhead for different k values

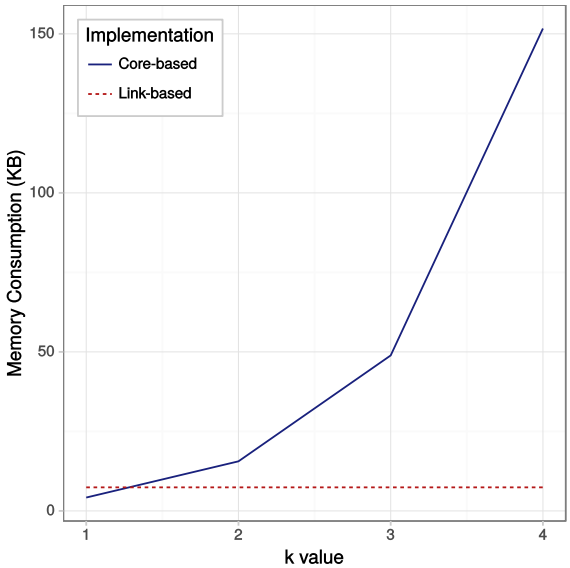


Figure 5.17: Memory for different k values

Network size

Finally, our last experiment shows how the network traffic is bounded if k is kept fixed, even though the network size increases. Figures 5.18 and 5.19 show the reports and kbps each node transmits as the network size increases, when the protocol's interval is 10 minutes, $k = 2$ and the sketch size is 1 row and 64 columns. As we can see, as the network size increases, both the number of reports and kbps stabilize, since the number of hosts reachable with $k + 1$ hops remains constant. A surprising result is that for the given data traffic (1 packet/s each node), the kbps of the Core-based implementation is lower than for the Link-based. This is because most of the cores won't have almost any traffic, and therefore, it's sketch can be represented with very little bits. On the other hand, the number of reports is higher for the Core-based implementation.

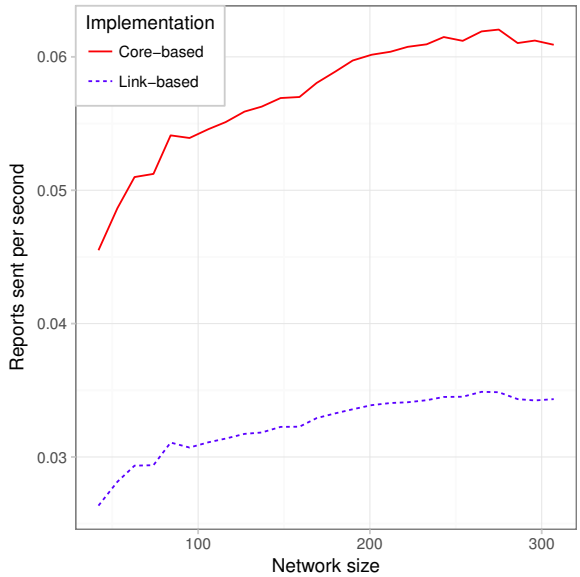


Figure 5.18: Number of reports exchanged for different network sizes

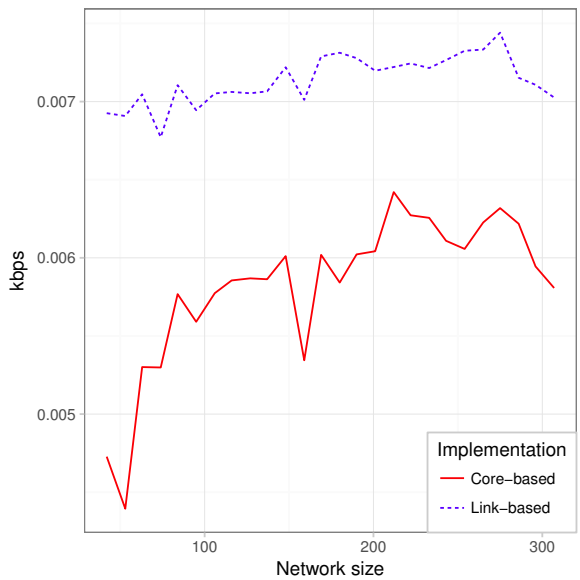


Figure 5.19: Network overhead per node for different network sizes

5.8 Discussion

In this section, we discuss further some of the aspects of the proposed implementations for KDet, the limitations imposed by the Traffic Validation mechanisms and a simple way of reducing its network overhead.

5.8.1 KDet Implementations

We have hinted two possible implementation strategies for KDet in section 5.6. 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 in equation 5.1 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.

5.8.2 Limitations

KDet is based on the conservation of the flow principle, and as such, when the principle is not satisfied by the network itself, it will not work properly.

An example of such situation is multicast traffic. However, as mentioned by Mizrak et al. [34], multicast traffic is not common. In fact, we have studied some traffic captures from Guifi.net and seen that the only multicast traffic is from the routing protocols and, accordingly, with TTL = 1 and should not be forwarded.

Another example is that networks, especially wireless networks, are likely to lose some traffic even without faulty routers, 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. [20]).

5.8.3 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 [40], 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.

5.9 Conclusions

In this chapter, we have focused in the summary dissemination and distributed detection sub-problem of forwarding fault detection. 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. KDet is tailored to the requirement 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 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.

Because the analytic results are based on traffic between every pair on a network, which will not be the case for WCN, we have also measured KDet's accuracy, memory and bandwidth consumption by simulation, using OMNeT++. Our results showed 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.

Chapter 6

Traffic Validation Mechanisms

Finally, we focus on the Traffic Validation sub-problem, i.e. the mechanism that using the different traffic summaries decides whether a node or area is forwarding traffic properly or not. The traffic validation mechanism is based on the simple principle that although a faulty node may behave in many different ways, a correct router will have the expected behavior: packets are correctly forwarded, with low corruption and drop probabilities. Based on this principle Bradley et al. described the Conservation of the Flow principle that correct nodes (or network areas) are assumed to follow: *traffic entering a network area should be (approximately) the same as traffic leaving, except traffic destined to and generated by it* [10]. We have seen in Chapter 2 that most of the solutions that have been proposed simply measure the number or proportion of packets that differ between the traffic entering and leaving a network area and they classify the behavior of the network area by using a threshold. Of course, how we measure the number of different packets between the traffic entering and leaving depends on the way traffic is summarized. But as we have seen, any solution based on monitoring something else than a path (e.g. a router or a set of nodes) requires loose synchronization so that the traffic summaries relate to the same traffic packets. On this chapter we propose a traffic validation mechanism using sketches that does not require clock synchronization: Misaligned Traffic Validation (MTV).

6.1 Misaligned Traffic Validation

Many of the proposed solutions rely on GPS clocks to keep the network nodes loosely synchronized, however, because equipment costs in WCN should be kept low, using GPS clocks in WCN is not feasible. In this context, we study how we can make the most of sketches' properties to avoid requiring clock synchronization, but still be able to detect when the difference between the incoming and outgoing traffic of a network node is too different to be due to natural causes.

To better understand how MTV works, first we will describe a simple mechanism for traffic validation assuming synchronized clocks, and then we will build over that solution for when there is no clock synchronization; also, for the sake of simplicity we will assume that the area being monitored is a single node (M), though MTV can be applied also to network areas with several nodes.

6.1.1 Sketches for Traffic Validation

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

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

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

In any case, because keeping a copy of all the traffic exchanged through M and later sharing it is really expensive, we need a traffic summary function to represent $\mathbb{T}$, and in our case we propose to use sketches because they satisfy the following properties: (i) a sketch can be computed online, updating its counters every time a new packet arrives; (ii) in a distributed fashion, where each monitoring node can compute the sketch of the portion of traffic it sees and the global sketch can be computed by combining each local sketch; and

finally, (iii) a sketch has relatively low requirements in terms of processing, memory and network overhead. A sketch consists mainly in a matrix of counters that are updated as packets arrive; and as such they can be linearly combined: every neighbor will keep a sketch for the monitored node's incoming and outgoing traffic and then all of them will be combined to obtain the sketch that represents the difference between the global incoming and outgoing traffic:

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

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

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

Ideally, we would like the inequality to be an equality, which happens when the elements of $\mathbb{T}$ are unique. This is typically the case for network packets, as the IP protocol has an *id* field [20]. In the case there are duplicate elements sketched on S_{diff} the second frequency moment will be an overestimate of the difference between incoming and outgoing traffics. We will see in section 4.3 that this is a rare event, and so sketches can be used effectively for traffic validation.

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

- *Input space*: the original space of all possible packets is terribly big ($I = \{2^{8 \times 1500}\}$ for Ethernet v2 and bigger with Jumbo Frames). As a result, computing the sketch over this space is very expensive in terms of computation, and therefore we require a space reduction function. In this work we use the last bytes of SHA-256 to reduce the space to a reasonably sized one.

- *Changeless elements*: to be able to compare the sketches produced in two different neighbors of M, the elements sketched need to be the same. That implies that layers below the IP layer must be removed and that the

TTL must be adapted by either having the sender reduce it by 2 before sketching the packet or simply setting its value to 0. Also, networks that perform packet fragmentation cannot use sketches for fault location.

In essence, every node will keep a sketch of the traffic of each neighbor. Whenever a node receives a packet, it will remove its lower layers and adapt the TTL field as required. Then it will generate digest of fixed size using a cryptographically secure hash function and use the digest as the element to sketch (a more detailed description of this process is available in Chapter 4). Finally it will update the sketch of the source neighbor if that packet was not originated from it. Something similar will be done every time a message is sent.

Then, using a dissemination protocol (Chapter 5), those traffic summaries will be shared, so that there is an entity capable of evaluating M. A simple traffic validation function uses $||S_{\text{diff}}||^2$ as an estimation of the number of dropped packets, and determines that M is faulty if it is above a threshold:

$$TV(M,t) = \begin{cases} \text{OK} & \text{if } \text{diff}(t) = ||S_{\text{diff}}(t)||^2 < \text{threshold} \\ \text{Faulty} & \text{otherwise} \end{cases}$$

However, in most of the cases, we are interested in the proportion of dropped packets, not the absolute number. And to improve the precision of the estimation when the interval is too short, we will average the last W measurements:

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

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

6.1.2 Traffic Validation without clock synchronization

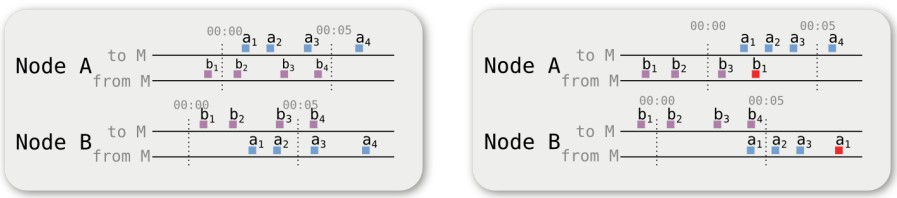
The previously proposed traffic validation mechanism measures the difference between the traffic flows captured by the sketches, but if the neighbors do not agree on the packets that are to be captured, the measured difference will be

bigger than expected. Consider, for instance, that node M is connected to node A and B, and the traffic perspective from A and B is as shown in figure 6.1a. In this case, because of the lack of synchronization, $||S_{\text{diff}}||^2 = |\{a_3, b_1, b_4\}| = 3$, however, M is forwarding packets properly.

To overcome this problem, we propose to use the sketches sent on the previous and next interval. Consider $S_{\text{in}}(t)$ the sketch as result of summing every sketch related to the incoming traffic at time interval t and $S_{\text{out}}(t)$ the one related to the outgoing traffic. As we have seen, some packets from $S_{\text{in}}(t)$ may not be on $S_{\text{out}}(t)$, but on $S_{\text{out}}(t-1)$ or $S_{\text{out}}(t+1)$. Similarly, some of the packets from $S_{\text{out}}(t)$, from a neighbor's perspective may have been sent during interval $t-1$ or $t+1$, so we obtain:

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

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



(a) Correct nodes

(b) Faulty node

Figure 6.1: Example packets with unsynchronized clocks.

This estimation is reliable as long as M behaves properly, but if it does not, it could replace some packets for others sent previously and avoid detection. Consider for instance the scenario on figure 6.1b, $\widehat{\text{diff}} = |\{a_2, a_3, b_4, b_3, b_1\}| - |\{b_2\}| - |\{a_1, a_2, a_3\}| - |\{b_2\}| - |\{\}| = 0$. Therefore, we cannot discount every packet in the previous and next sketches, but only the ones that are not duplicated from the current sketch:

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

6.1.3 Evaluation

Our evaluation is based on a traffic capture from a wireless node member of the qMp Sants WCN [13]. Its main characteristics are summarized in table 6.1.3. At the moment of the capture, the node (M) was connected to three other nodes, two of them, A and B, had really good link quality (loss below 10^{-6}) and another, node C, with bad link quality (50% packet loss measured with MTR). Because qMp Sants is an ad-hoc network, the routing protocols will rarely use the link with low quality (less than 0.1% of the traffic is routed through C). The traffic is quite variable in terms of packets per second.

Using Goldberg et al.'s definition [20], we use α is an upper bound on the percentage of packets lost by a non-faulty node, whereas β is the percentage of dropped packets that if exceeded by a router, it will be considered faulty. The α of this traffic capture is 0.058 (given by the periods with most traffic from C); therefore, for a β of 0.1, threshold should be 0.73 [20].

On each experiment, we replay the traffic capture, simulating the loss on each link and compute the expected sketches on each of the nodes with the given experiment parameters. Then we combine them using traffic validation with and without intersection to see how well each mechanism fares on the detection of faulty routers.

6.1.3.1 Experiment 1: Comparing each TV function

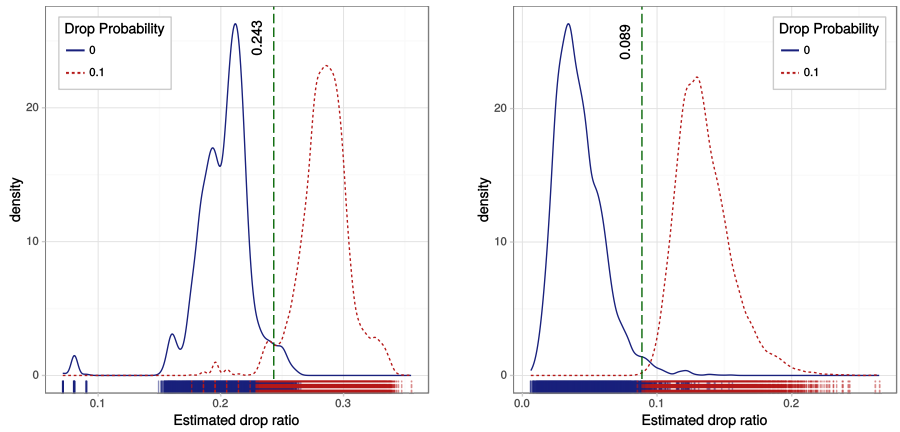
Our first scenario compares how traditional TV and TV using the intersection perform when the clocks of the nodes are not synchronized. To do so, we have advanced the clock of $A$and delayed the clock of B a proportion of the interval (10%). Because our traffic capture is not too long, the intervals are

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

relatively short (10 seconds at most), so we average the measurements for 10 intervals ($W = 10$). Regarding the sketch, this scenario considers a FastCount sketch [55] with 1024 columns and a single row.

In Figure 6.2 we can see the probability density function of the estimated drop probability for faulty and non-faulty nodes and which should be the detection threshold computed using logistic regression. As we can see, when we use the normal TV mechanism, because intervals are misaligned by 10%, nodes that drop no traffic tend to have an estimation of 20% of different packets between incoming and outgoing flows, because there is a 10% of the incoming packets that are not in the outgoing sketch and an additional 10% of packets on the outgoing sketch that is not in the incoming sketch. If we use the intersection, the number of different packets is still overestimated, but not as much as for the normal case: the optimal threshold is below the real drop probability of faulty nodes.

So, if we were to use the optimal threshold, faulty and non-faulty nodes could still be distinguished using any of the TV mechanisms, as shown in Figure 6.3a; though TV using the intersection still produces better results that without using it. However, in real life, we cannot expect to know which is the skew between the different neighbors, or even if there are many neighbors, each with its own clock skew, depending on the interval and the traffic going through the network, the optimal threshold will keep changing; so we cannot rely on a threshold optimized to a given clock skew. So in Figure 6.3b we show the proportion of detected nodes using the threshold computed using Goldberg's approach (0.073). As we can see, now, using normal TV will cause every non-faulty node as faulty, especially as the interval grows, but if we use MTV, we can predict the behavior of the node more accurately.



(a) Using normal TV (b) Using misaligned TV

Figure 6.2: Estimated probability density function of $\widehat{\text{loss}}_W$

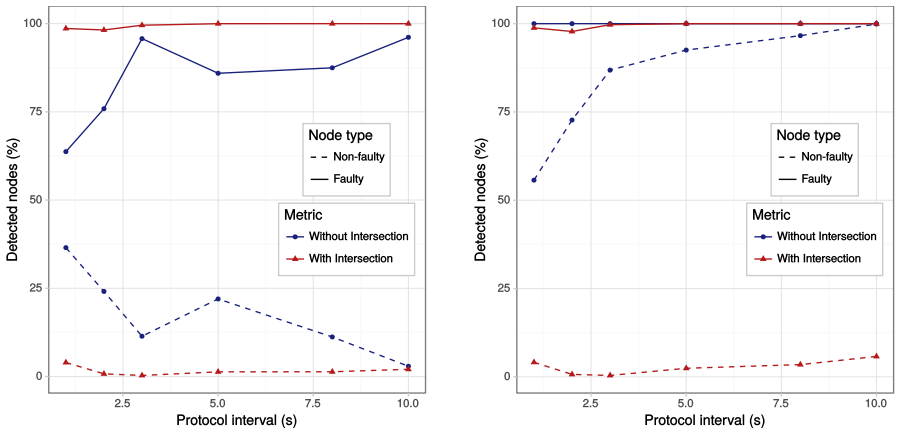
Finally, on Figure 6.4 we can see that in the case that clocks are synchronized, both mechanisms to estimate the proportion of dropped packets give accurate results, so there is no detriment of using MTV when the clocks are synchronized.

6.1.3.2 Experiment 2: Measuring the cost of MTV

In this second experiment, we will analyze the effect of the sketch size on the quality of the prediction, as well as its overhead. As we know from the previous chapters, using larger sketches will cause a greater network overhead when sharing them; however, in the case of MTV, to provide accurate predictions we need greater sketches, as we can see in Figure 6.5a, because MTV still tends to overestimate the proportion of dropped packets, we need sketches bigger than for the case of synchronized clocks. And the overestimation worsens as the interval grows larger, so there is a compromise between the prediction accuracy and the traffic overhead as seen in Figure 6.5b.

6.2 Conclusions

In this chapter, we have proposed a mechanism that allows us to compare sketches for traffic validation even though they are misaligned. However, when



(a) With optimized thresholds (b) With Goldberg's threshold

Figure 6.3: Proportion of nodes detected

clocks are not synchronized, estimating the proportion of dropped packets is more costly than in the case of synchronized clocks, as we require bigger sketches because the computation of the intersection is less accurate.

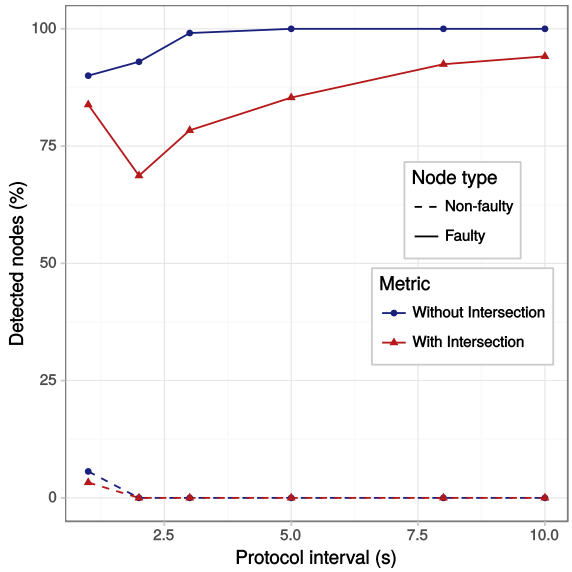
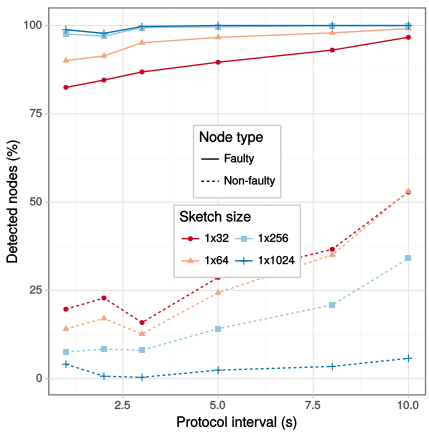
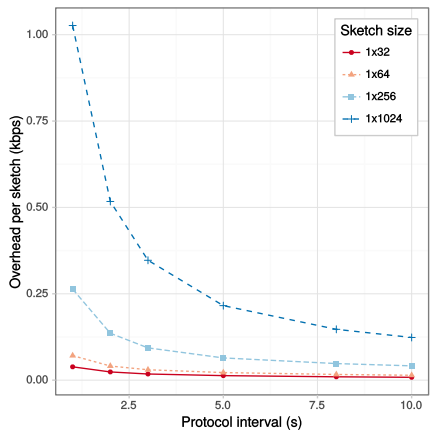


Figure 6.4: Proportion of nodes detected with synchronized clocks



(a) Proportion of nodes detected



(b) Traffic overhead per sketch

Figure 6.5: Effect of the sketch sizes

Chapter 7

Conclusions

This thesis has presented how the problem of forwarding fault detection could be implemented and deployed in a WCN by focusing on each of its sub-problems and proposing solutions for each of them that are suitable for WCN.

Our first challenge was to model the error of sketches for traffic validation, which we faced by considering the specific characteristics of the input space (the digest of traffic packets). Thanks to these characteristics we were able to describe the estimation of the sketch as a simpler random process and, therefore, be able to provide tighter bounds on the sketch accuracy. Using these bounds, we can determine the size of the sketch in a much more precise way, saving memory, processing power and network bandwidth, scarce resources in a WCN.

Our second challenge was to design a traffic summary dissemination and distributed detection mechanism adapted to WCN. There are 2 main considerations that need to be taken into account: (i) in WCN it is usual for a node to have several antennas, in many cases directional, so we cannot rely on techniques based on overhearing; (ii) there is not a single routing protocol that is used for WCN, and some of them even combine two or more; the implication is that we cannot rely on a single routing protocol and more specifically, we cannot rely on a link-state routing protocol, so the path from node to node is not known. Additionally, it is desirable that the distributed detection protocol can be deployed as an independent daemon, but that their results are shared globally, so that every node in the network can benefit from the detection process. Considering these concerns, we proposed KDet, a

distributed detection protocol based on logical boundaries between the nodes monitoring and being monitored, so that the monitors need to compare the traffic entering and leaving the boundary. To avoid false accusation, KDet involves the nodes being monitored in the detection process, so that they can disconnect from those nodes that are creating false reports. And to be able to detect faulty nodes even under the assumption of collusion, KDet defines different sets of boundaries with different sizes, so that there will be always a boundary surrounding the faulty nodes that is being monitored by a set of non-faulty nodes, so there is no way they can avoid detection.

Finally, our last challenge was to study whether we could propose a traffic validation mechanism that did not require the nodes to have synchronized clocks. Thanks to the fact that we can compute the intersection between two sketches, we have proposed misaligned traffic validation, which considers not only the traffic summary from the current time interval, but also the previous and the next, so that traffic validation is possible even when the sketches are not perfectly synchronized.

In conclusion, we have shown that it is possible to deploy a forwarding fault detection protocol adapted to Wireless Community Networks, which would increase the routing robustness of the network and ease its administration. But the results found on this thesis are not only applicable on the context of WCN. Our results on chapter 4 regarding the distribution of the error in sketches can be applied on any network monitoring solution regardless of the network type. KDet (chapter 5) is also suitable for any kind of network; and in the case of a network of networks, if the desired detection granularity was an administration domain or ISP, we can simplify it by setting the cores appropiately, so that each of them covers the subnetworks of interest. Lastly, MTV (chapter 6) is only useful to context of WCNs, as wired network's won't usually have issues to sync their clocks with enough precision.

Future directions

Considering the contributions proposed in this thesis and with all the insight gained during the process, we believe that the following directions can be of interest:

- Regarding sketches for traffic validation, we have seen that the computation of the digest is one of the steps that costs more in terms of processing power for the whole process. Given that the only requirement of the hash is that it is resistant to second pre-image attacks, it would be interesting to find and propose alternatives that have better processing cost.

- Development and fine tuning an implementation in a Community Network to provide accurate and precise indications of anomalies that result in more resilient networks. Evaluation of the trade-offs of this mechanism in realistic conditions. Also, consider some kind of interaction between the routing protocol and the detection protocol, so that the detection thresholds can be adapted with the routing information.

- Consider the second implementation mechanism of KDet and the possibility of computing the intersection of different sketches. We envision that they can be used to maintain a map of the traffic flows going through the network, so that anomalies, such as traffic hijacking attacks can be detected.

Bibliography

[1] Guifi.net Barcelones area. `http://guifi.net/en/node/2435`. Accessed: 2015-08-11. (page 78)

[2] Low Extra Delay Background Transport (LEDBAT). `http://tools.ietf.org/html/rfc6817`. Accessed: 2015-03-08. (page 80)

[3] A. M. Abdalla, I. a. Saroit, A. Kotb, and A. H. Afsari. Misbehavior nodes detection and isolation for MANETs OLSR protocol. *Procedia Computer Science*, 3:115–121, jan 2011. (page 16)

[4] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS '99*, pages 10–20, New York, New York, USA, 1999. ACM Press. (page 23, 34)

[5] K. Argyraki, P. Maniatis, O. Irzak, S. Ashish, and S. Shenker. Loss and delay accountability for the internet. In *Proceedings - International Conference on Network Protocols, ICNP*, pages 194–205. IEEE, oct 2007. (page 10, 11, 22)

[6] 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. (page 3, 69)

[7] I. Avramopoulos and J. Rexford. Stealth Probing : Efficient Data-Plane Security for IP Routing. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference*, pages 25—-25. USENIX Association, 2006. (page 14, 16)

[8] 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*, 10(4), 2008. (page 10, 15, 16, 21, 22)

[9] B. Barak, S. Goldberg, and D. Xiao. Protocols and Lower Bounds for Failure Localization in the Internet. *Lecture Notes in Computer Science*, 2008. (page 15)

[10] K. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. Olsson. Detecting disruptive routers: a distributed network monitoring approach. *IEEE Network*, 1998. (page 5, 10, 17, 19, 22, 29, 39, 64, 70, 95)

[11] S. Buchegger and J.-Y. Le Boudec. Performance analysis of the CONFIDANT protocol. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing - MobiHoc '02*, page 226, New York, New York, USA, 2002. ACM Press. (page 22)

[12] The CAIDA UCSD anonymized internet traces 2013 - [29-05-2013]. `http://www.caida.org/data/passive/passive_2013_dataset.xml`. (page 39)

[13] L. Cerdà-Alabern, A. Neumann, and P. Escrich. Experimental evaluation of a wireless community mesh network. In *Proceedings of the 16th ACM international conference on Modeling, analysis & simulation of wireless and mobile systems*. ACM, 2013. (page 100)

[14] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3 – 15, 2004. (page 35)

[15] M. Conti, E. Gregori, and G. Maselli. Towards reliable forwarding for ad hoc networks. In *PERSONAL WIRELESS COMMUNICATIONS, PROCEEDINGS*, volume 2775, pages 790–804, 2003. (page 10)

[16] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. *Proceeding VLDB '05 Proceedings of the 31st international conference on Very large data bases*, pages 13–24, 2005. (page 23, 35)

[17] C. Crepeau, C. R. Davis, and M. Maheswaran. A Secure MANET Routing Protocol with Resilience against Byzantine Behaviours of Malicious or Selfish Nodes. In *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, pages 19–26. IEEE, 2007. (page 10, 22)

[18] V. Desai, S. Natarajan, and T. Wolf. Packet forwarding misbehavior detection in next-generation networks. *IEEE International Conference on Communications*, 2012. (page 10, 16)

[19] W. Galuba, P. Papadimitratos, M. Poturalski, K. Aberer, Z. Despotovic, and W. Kellerer. Castor: Scalable Secure Routing for Ad Hoc Networks. *2010 Proceedings IEEE INFOCOM*, pages 1–9, mar 2010. (page 10, 22)

[20] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford. Path-quality monitoring in the presence of adversaries. *ACM SIGMETRICS Performance Evaluation Review*, June 2008. (page 10, 11, 14, 23, 29, 30, 32, 42, 44, 70, 78, 92, 97, 100)

[21] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford. Path-Quality Monitoring in the Presence of Adversaries: The Secure Sketch Protocols. *IEEE/ACM Transactions on Networking*, PP:1–13, 2014. (page 10, 56, 57, 70)

[22] 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. (page 64)

[23] Y. Jeon, T.-H. Kim, Y. Kim, and J. Kim. LT-OLSR: Attack-tolerant OLSR against link spoofing. *37th Annual IEEE Conference on Local Computer Networks*, pages 216–219, oct 2012. (page 8)

[24] D. Johnson and a. Perrig. SEAD: secure efficient distance vector routing for mobile wireless ad hoc networks. *Proceedings Fourth IEEE Workshop on Mobile Computing Systems and Applications*, pages 3–13, 2002. (page 8)

[25] M. Just, E. Kranakis, and T. Wan. Resisting malicious packet dropping in wireless ad hoc networks. *Ad-Hoc, Mobile, and Wireless Networks*, pages 151–163, 2003. (page 16)

[26] S. Kent, C. Lynn, and K. Seo. Secure Border Gateway Protocol (S-BGP). *IEEE Journal on Selected Areas in Communications*, 18(4):582–592, apr 2000. (page 4, 8)

[27] I. Khalil and S. Bagchi. Stealthy Attacks in Wireless Ad Hoc Networks: Detection and Countermeasure. *IEEE Transactions on Mobile Computing*, 10(8):1096–1112, aug 2011. (page 12)

[28] I. Khalil, S. Bagchi, N. AbuAli, and M. Hayajneh. Disa: Detection and isolation of sneaky attackers in locally monitored multi-hop wireless networks. *Security and Communication Networks*, 6(12), 2013. (page 22, 61, 63)

[29] E. López. KDet: additional information. `http://dsg.ac.upc.edu/esterl/KDet`. Accessed: 2015-03-31. (page 69, 78)

[30] E. López. Scripts and code for sketches for Traffic Validation. `http://esterl.github.io/sketches-evaluation/scripts`, 2015. (page 30, 38)

[31] E. López. Sketches Evaluation. `http://esterl.github.io/sketches-evaluation/reports`, 2015. (page 42, 44, 46, 47, 58)

[32] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Sustaining Cooperation in Multi-Hop Wireless Networks. In *Proceedings of NSDI*, pages 231–244, 2005. (page 18, 22, 63)

[33] 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. (page 22)

[34] A. Mizrak, K. Marzullo, and S. Savage. Fatih: Detecting and Isolating Malicious Routers. In *International Conference on Dependable Systems and Networks*, 2005. (page 67, 69, 77, 92)

[35] A. T. Mizrak, Y.-C. C. Cheng, K. Marzullo, and S. Savage. Detecting and isolating malicious routers. *IEEE Transactions on Dependable and Secure Computing*, 3(3):230–244, jul 2006. (page 9, 18, 63)

[36] A. T. Mizrak, S. Savage, and K. Marzullo. Detecting Malicious Packet Losses. *IEEE Transactions on Parallel and Distributed Systems*, Feb. 2009. (page 4, 10, 18, 21, 51, 65, 70)

[37] 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. Technical report, CONFINE Project. (page 4, 68)

[38] D. Obenshain, T. Tantillo, A. Babay, J. Schultz, A. Newell, E. Hoque, Y. Amir, and C. Nita-rotaru. Practical Intrusion-Tolerant Networks. Technical report, Distributed Systems and Networks Lab, 2016. (page 4, 7, 69)

[39] S. Paris, C. Nita-Rotaru, F. Martignon, and A. Capone. Efw: A cross-layer metric for reliable routing in wireless mesh networks with selfish participants. In *INFOCOM, 2011 Proceedings IEEE*, April 2011. (page 22, 63)

[40] G. Pei, M. Gerla, and T.-W. Chen. Fisheye state routing: a routing scheme for ad hoc wireless networks. In *Communications, 2000. ICC 2000. 2000 IEEE International Conference on*, volume 1, pages 70–74 vol.1, 2000. (page 92)

[41] R. Perlman. *Network layer protocols with byzantine robustness*. PhD thesis, Massachusetts Institute of Technology, 1988. (page 4, 7, 26, 72, 82)

[42] R. Perlman. Routing with Byzantine Robustness. Technical report, Sun Microsystems, Inc., 2005. (page 4)

[43] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, Jan. 2003. (page 3)

[44] 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. (page 22)

[45] qMp Sants. `http://dsg.ac.upc.edu/qmpsu`, 2015. (page 39)

[46] F. Rusu. Sketches for Size of Join Estimation. `http://faculty.ucmerced.edu/frusu/Projects/Sketches/sketches.html`, 2014. Accessed: 2014-04-30. (page 38)

[47] F. Rusu and A. Dobra. Pseudo-random number generation for sketch-based estimations. *ACM Transactions on Database Systems*, 32(2):11–es, jun 2007. (page 34, 41)

[48] F. Rusu and A. Dobra. Statistical analysis of sketch estimators. *Proceedings of the 2007 ACM SIGMOD international conference on Management of data - SIGMOD '07*, page 187, 2007. (page 30, 33, 34, 70)

[49] L. Sánchez-Casado, G. Maciá-Fernández, and P. García-Teodoro. An Efficient Cross-Layer Approach for Malicious Packet Dropping Detection in MANETs. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 231–238. IEEE, jun 2012. (page 63)

[50] N. Saxena, M. Denko, and D. Banerji. A hierarchical architecture for detecting selfish behaviour in community wireless mesh networks. *Computer Communications*, 34(4):548–555, apr 2011. (page 17, 22)

[51] T. Shu and 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*, page 87, 2012. (page 11, 16, 21, 70)

[52] L. Subramanian, V. Roth, and I. Stoica. Listen and whisper: Security mechanisms for BGP. *Proc. First Symposium on Networked Systems Design and Implementation*, 2004. (page 8)

[53] H.-M. Sun, C.-H. Chen, and Y.-F. Ku. A novel acknowledgment-based approach against collude attacks in MANET. *Expert Systems with Applications*, 39(9):7968–7975, jul 2012. (page 15, 21, 22)

[54] M. Thorup and Y. Zhang. Appendix for "tabulation based 4-universal hashing with applications to second moment m. thorup and y. zhang. appendix for "tabulation based 4-universal hashing with applications to second moment estimation", 2003. (page 37)

[55] M. Thorup and Y. Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 615–624, 2004. (page 23, 36, 42, 53, 101)

[56] D. Vega, R. Baig, L. Cerdà-Alabern, E. Medina, R. Meseguer, and L. Navarro. A technological overview of the guifi.net community network. *Computer Networks*, 93, Part 2:260 – 278, 2015. Community Networks. (page 1)

[57] T. Wan, E. Kranakis, and P. C. Oorschot. *Applied Cryptography and Network Security: Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004. Proceedings*, chapter S-RIP: A Secure Distance Vector Routing Protocol, pages 103–119. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. (page 4)

[58] Wikipedia. K-independent hashing — Wikipedia, the free encyclopedia, 2017. [Online; accessed 21-May-2017]. (page 34)

[59] S. Wu, H. Chang, F. Jou, and F. Wang. JiNao: Design and implementation of a scalable intrusion detection system for the OSPF routing protocol. *ACM Transactiom on Computer Systems, VoL*, pages 0–23, 1999. (page 8)

[60] X. Zhang, A. Jain, and A. Perrig. Packet-dropping adversary identification for data plane security. In *Proceedings of 2008 ACM CoNEXT Conference - 4th International Conference on Emerging Networking EXperiments and Technologies, CoNEXT '08*, 2008. (page 10, 15, 29)

[61] X. Zhang, C. Lan, and A. Perrig. Secure and scalable fault localization under dynamic traffic patterns. In *Proceedings - IEEE Symposium on Security and Privacy*, pages 317–331, 2012. (page 11, 17, 19, 23, 65, 69, 70, 78)

Appendix A

KDet algorithms

This appendix includes the pseudo-code of the boundary and core protocols run by KDet.

Algorithm 1 Boundary protocol

```
1: procedure BOUNDARY MONITORING(packet)
2:     l = get_link(packet)
3:     if packet.next_hop ∈ core.nodes ∧ packet.destination ∉ core.nodes then
4:         update(S^{→l}_core(T), packet)
5:     if packet.previous_hop ∈ core.nodes ∧ packet.source ∉ core.nodes then
6:         update(S^{←l}_core(T), packet)
7: procedure BOUNDARY REPORTING(period, core)
8:     T = now() - (now() % period)          ▷Beginning of the current period
9:     while True do
10:        if now() < T + period then
11:            reports =
12:            for all link ∈ get_links(core, self) do      ▷For all links to core
13:                reports = reports ∪ get_report(link)
14:            robustly_flood(sign(reports, T))
15:            clear_reports(core)
16:            T = T + period
17: procedure BOUNDARY EVALUATION(T)
18:    for all received report do
19:        if signature(report)==OK ∧ report.node ∈ boundary then
20:            reports ← reports ∪ {report}
21:    for all node ∈ boundary do
22:        if ∃ report ∈ reports | report.node==node then
23:            received[node] ← true
24:        else
25:            received[node] ← false
26:    send TA sign(evaluation(V_core(reports), received), T)
27:    reports.clear()
```

Algorithm 2 Core protocol

```
1:  procedure CORE PROTOCOL(core, boundary)
                                   ▷Receive and process reports from the boundary
2:      for all received report do
3:          if signature(report)!=OK ∨ report.node ∉ boundary then
4:              drop report
5:          else
6:              if ∃r_2 ∈ boundary-reports | report.node == r_2.node then
7:                  if report != r_2 then
8:                      p-faulty ← p-faulty ∪ report.node
9:              else
10:                 robustly-flood(report)
                                   ▷Compare reports from neighbors with local versions
11:     for all node ∈ neighborhood ∧ boundary do
12:         if ∃ report ∈ boundary-reports | report.node == node then
13:             if report != local-reports[report.node] then
14:                 p-faulty ← p-faulty ∪ {node}
15:         else
16:             p-faulty ← p-faulty ∪ {node}
                                   ▷At the end of the interval, retrieve evaluations
17:     evaluations = TA.getEvaluations(core)
18:     for all evaluation ∈ evaluation do
19:         if evaluation.node ∈ neighborhood then
20:             if !is-consistent(evaluation.bitmap, boundary-reports) then
21:                 p-faulty ← p-faulty ∪ {evaluation.node}
22:             if evaluation.validation != V_core(boundary-reports) then
23:                 p-faulty ← p-faulty ∪ {evaluation.node}
                                   ▷Disconnect from p-faulty nodes
24:     for all node ∈ p-faulty do
25:         disconnect from node
```