Friginal, J.; Andrés Martínez, DD.; Ruiz García, JC.; Martínez Raga, M. (2015). REFRAHN: A Resilience Evaluation Framework for Ad Hoc Routing Protocols. Computer Networks. 82:114-134. doi:10.1016/j.comnet.2015.02.032.

REFRAHN: A Resilience Evaluation Framework for Ad Hoc Routing Protocols

Jesús Friginal[a,b], David de Andrés[c], Juan-Carlos Ruiz[c], Miquel Martínez[c]

jesus.friginal@scassi.com, {ddandres, jcruizg, mimarra2}@disca.upv.es

[a]*SCASSI, Bâtiment AGORA 1, 209 Rue Jean Bart, 31670 Labège, France*
[b]*CNRS, LAAS, 7 avenue du colonel Roche, 31400 Toulouse, France*
[c]*STF-ITACA Universitat Politècnica de València, Campus de Vera s/n, 46022, Spain*

Abstract

Routing protocols are key elements for ad hoc networks. They are in charge of establishing routes between network nodes efficiently. Despite the interest shown by the scientific community and industry in converting the first specifications of ad hoc routing protocols in functional prototypes, aspects such as the resilience of these protocols remain generally unaddressed in practice. Tackling this issue becomes critical given the increasingly variety of accidental and malicious faults (attacks) that may impact the behavior exhibited by ad hoc routing protocols. The main objective of this paper is to deepen the methodological aspects concerning fault injection in routing protocols. As a result, we will design and implement a framework based on the injection of accidental and malicious faults to quantitatively evaluate their impact on routing protocols. This framework, called REFRAHN (Resilience Evaluation FRamework for Ad Hoc Networks), can be used to (i) reduce the uncertainty about the sources of perturbations in the deployment of ad hoc routing protocols, (ii) design fault tolerance mechanisms that address and minimize such problems, and (iii) compare and select which is the routing protocol that optimizes the performance and robustness of the network.

Keywords: ad hoc networks, fault and attack injection, resilience, experimental evaluation, testbeds.

1. Introduction

Ad hoc networks are multi-hop wireless networks where all nodes cooperate to maintain the network connectivity without centralised administration. The use of this emerging technology ranges from military to civilian applications. Ad hoc networks are expected to be the basis to interconnect future ecosystems of devices based on the notion of Internet of Things [1].

Yet, to fully unleash the potentials of such ad hoc networking paradigms, new challenges must be addressed and solved. In particular, effective routing solutions must be integrated within the lower network layers for a better perception of the physical environment, thus taking more accurate routing decisions [2]. Another challenge concerns the provision of adaptive fault tolerance and secure technologies. In evolving dynamic scenarios, more effective procedures are required to protect paths with minimal effect on the perceived quality [3]. In addition, the battery lifetime of mobile devices defies our ability to define secure and efficient low energy consumption routing protocols [4]. Despite the

importance of providing solutions to face these issues, such efforts will remain questionable in practice while suitable techniques to guarantee persistent routing in spite of changes (also referred to as resilience) in their implementations remain unavailable [5].

The occurrence of faults in ad hoc networks, and more concretely in ad hoc routing protocols, which are critical elements for such networks, represents a major source of changes in the behavior of the system. Unavoidably, they may dramatically degrade the service provided to the upper layers of the network. In order to mitigate the effect of potential accidental faults and attacks on default versions of routing protocols, some authors have proposed secure and fault-tolerant alternatives. Some examples are SEAD [6], SOLSR [7], SAODV [8] or CONFIDENT [9]. Traditionally, simulation has been the platform chosen to evaluate these secure and fault-tolerant protocols. However, there is a growing awareness of the fact that current simulators make several simplifying assumptions to model many essential characteristics of real systems, thus limiting the credibility of their results [10]. The gap between simulated and experimental results may lead to differences between the behavior of the simulated network and that of the real one. This gap becomes extremely important when addressing real systems that may rely on routing protocols, especially when human lives might be at risk, large economic losses may derive from their malfunctioning or even when the reputation of service providers may be dramatically affected, even in non-critical domains. So, it is of prime importance to validate the theoretical design and analysis of routing protocols with sound experiments that are representative of their final deployment. Thus, the experimental resilience evaluation of ad hoc routing protocols plays an essential role to determine the confident use of ad hoc routing protocols.

The resilience evaluation of ad hoc routing protocols is essential to study how the features of ad hoc networks evolve in time. Designing systems, and specially ad hoc networks with resilience in mind [5] is no more a choice but a requirement. So, the need for frameworks to evaluate and justify their resilience is, without doubt, one of the major challenges faced by ad hoc networks [11].

This paper presents a unified framework called Resilience Evaluation FRamework for Ad Hoc Networks (REFRAHN) that exhaustively addresses a problem that previous studies have rarely considered: resilience evaluation in ad hoc routing protocols. Consequently, REFRAHN is not another framework for wireless ad hoc routing protocols. Instead, it is a framework that follows a detailed (but pragmatic) methodology to conceive evaluation from a very specific viewpoint: resilience. This implies addressing the definition of specific measures, the injection of pertinent faults and attacks, and the interpretation of complex and heterogeneous results. So, on one hand, REFRAHN defines an experimental methodology to evaluate the resilience of real ad hoc routing protocols executed in real devices and, on the other hand, it implements a tool that exploits the benefits of emulation to support the process of resilience evaluation. The REFRAHN methodology ensures an adequate level of controllability, repeatability and observability of experiments through the definition of three basic stages. First, it is mandatory to understand which parameters bound the variability of results in the evaluation of ad hoc routing protocols. Then, it is necessary to address the experimental procedure to carry out the evaluation of the ad hoc routing protocols in the presence of faults. Finally, it is essential to define how to transform obtained measurements into resilience metrics that can be analyzed and interpreted by final users.

The rest of this paper is structured as follows. Section 2 analyzes present gaps in the evaluation of ad hoc routing protocols. Section 3 introduces how our fault injection methodology can complement traditional evaluation. The first implementation of REFRAHN is introduced in Section 4 whereas Section 5 shows its feasibility through a case study. Finally Section 6 concludes the paper.

2. Evaluation of ad hoc routing protocols

The practical experimental evaluation of ad hoc routing protocols is essential to determine whether the risks to which ad hoc networks are exposed are acceptable or not. To properly evaluate ad hoc routing protocols, it is necessary to pay attention to three basic aspects that influence the quality of the results: (i) the type of platform used to carry out the evaluation; (ii) the recreation of the conditions affecting ad hoc routing protocols; and (iii) the properness of the set of metrics selected to represent the system behavior.

2.1. Critical overview of experimental evaluation approaches

This section briefly introduces most of current evaluation platforms in the domain of ad hoc networks. Let us present them according to the three major strategies to address the evaluation of ad hoc routing protocols: simulation, prototyping or emulation.

Simulation platforms are in general a good option to check design proposals and discard them before they become too costly to modify in next stages of their life cycle. However, although a simulation platform has a much lower cost compared to an actual testbed, simulations typically simplify several factors from reality, leading to less accurate results. Most well-known approaches today are Network Simulator (NS) 2 and 3 [12], Opnet [13], or Glomosim [14]. Alternatively, there is a module for NS, called click, that enables evaluators to run protocols first in a simulator and later on real hardware[1].

Conversely to simulation, real-world prototyping, that refers to the execution of experiments in real scenarios performed by technically skilled persons or target audience [15], provides more representative results to evaluate ad hoc networks. Real-world prototyping is a good option to evaluate mature developments in their expected environment. In the bibliography it is possible to find different open-source prototype-based approaches enabling the use of real devices and applications such as Roofnet [16], Floornet [17], the Ad Hoc Protocol Evaluation Testbed (APE) [18], or the Reconfigurable Mobile Multi-hop Wireless Network Testbed (MINT) [19]. Although these platforms provide a higher level of accuracy, reproducing the experiments is particularly hard. Specifically, it is necessary to define the experimentation in such a way that it is controllable and observable, as far as possible, so that results can be repeatable, and thus comparable. In addition, this problem is specially critical in case of considering mobile networks, where the larger the physical distance among nodes, the more tedious and time-consuming the task of setting up a multi-hop topology. On one hand, several research projects, like APE at Uppsala University [20], have reported experiences of setting up a multi-hop wireless testbed spending a significant amount of time and work using volunteers to carry mobile devices in an orchestrated manner. However, this approach may affect the controllability and observability of the experimentation process. On the other hand, the radio range of wireless nodes operating in the 2.4 GHz band and using off-the-shelf IEEE 802.11b/g wireless network interface cards (NICs) may vary dramatically (from 50 m to 100 m). This variation depends on factors such as the way the driver provider interprets the standard specification for a given operating system and/or the presence of perturbations, such as multi-path interference or noise in the channel, among others. This may negatively impact the repeatability of experiments.

Emulation is a hybrid solution, halfway between simulation and prototype-based evaluation platforms, enabling different combinations [15]: real devices with virtual wireless NICs, real NICs deployed in emulated wireless medium, or real-world experiments in a virtual environment (e.g., virtual mobility). Being a trade-off solution, emulation is the most recommendable option to quickly evaluate developments (being mature or not) in

[1] http://www.read.cs.ucla.edu/click/nsclick

different scenarios. It considers typical elements from real deployments like the use of real devices and applications and simulates others that are difficult to recreate and repeat, like mobility. Most representative platforms are the Open-Access Research Testbed for Next-Generation Wireless Networks (ORBIT) [21], the Carnegie Mellon University Wireless Emulator (CMUWE) [22], Castadiva [23], Mobiemu [24] or Emulab [25].

2.2. Recreation of threats

Recreating as faithfully as possible the features of the environment under which ad hoc networks typically operate is a key aspect to justify the credibility of results derived from the evaluation of ad hoc routing protocols. More precisely, the dynamic features of such environments require considering in the evaluation definition aspects such as the mobility of nodes (in case of mobile devices), and the presence of internal or external threats related to the nature of the wireless medium, the limited resources of devices, and the absence of a fixed infrastructure, that may lead to the occurrence of faults in ad hoc routing protocols. This also includes, for example, misconfigurations, quality of service (QoS) aspects, and large scale disasters that may compromise the infrastructure.

It is worth noting that most of current platforms take mobility into account. Nevertheless, very few of them consider the occurrence of faults despite their importance during the execution of experiments in the evaluation process. This fact is typically avoided by main current evaluation platforms given the added difficulty to recreate the presence of faults in systems in a controllable and repeatable way. Despite that fault injection has been widely explored in the dependability domain through frameworks such as NFTAPE [26], XCEPTION [27], or FIAT [28], its application to ad hoc networks is limited to just two frameworks: MINT [19] and AVR-INJECT [29]. Nevertheless, whereas the former just exploits the impact of packet dropping, the latter only focuses on the injection of bit-flips within nodes' processor, thus obviating all potential faults affecting the interaction between nodes at the communication layer, such as those listed in Table 1. Additionally, AVR-INJECT introduces a considerable intrusiveness in the evaluation metrics, since the injection of bit-flips is emulated through the insertion of additional instructions in the assembly code. Furthermore, authors do not evaluate any routing protocol in their works.

Table 1: Accidental and malicious threats affecting the routing layer of ad hoc networks. Most of them have a transient nature given the dynamic features of ad hoc network deployments. Obviously, these are not the only threats to ad hoc networks, but they are enough in number and importance to justify the need of evaluating ad hoc routing protocols from the perspective of their resilience.

Network characteristic	**Main threats identified**
Resources limitations (hardware integrity and battery lifetime)	Physical damage, flooding attack, neighbors saturation, battery extenuation
Wireless communication medium (open shared wireless channel)	Signal attenuation, multifading, ambient noise, jamming attack, exposed node, traffic analysis, cryptanalysis
Mobility of nodes (unpredictable topology)	Sequence number replay, replay attack, Sybil attack, tampering attack, Doppler shift
Deployment issues (incorrect network configuration)	Wrong nodes distribution, wrong routing protocol configuration, peak in service demand, QoS aspects
Absence of infrastructure (lack of Public Key Infrastructure (PKI) and certification authorities)	Sink hole, black hole, selective forwarding attack, jellyfish attack

2.3. Metrics considered in the evaluation

Ad hoc routing protocols are normally characterized through metrics used by evaluation platforms. Table 2 summarizes the metrics typically considered by analyzed frameworks. It is worth noting that AVR-INJECT has not been included since it has not been used to evaluate routing protocols. Results show how the behavior of ad hoc routing protocols is massively characterized through performance metrics reporting the throughput, delay, routing overhead, or packet delivery ratio exhibited by the network. Indeed, almost 90% of the evaluation platforms considered in this study provide throughput and delay as a reference measure, which is an example of how, to date, the evaluation of ad hoc routing protocols has been generally limited to functional aspects.

Additionally, it is worth noting that non-functional aspects surprisingly remain in the background. With respect to resources consumption, few metrics are normally considered. In particular, they are only considered by those evaluation platforms that are able to recreate networks with resources limitations, such as wireless sensor networks. It is remarkable, from a resilience viewpoint, the limited presence of metrics to quantify aspects related to the ability of the system to coexist and tolerate the presence of threats in the system. Although there are platforms, like MINT, that evaluate the impact of threats in the system through the degradation of mainstream performance metrics in a process typically referred to as performability evaluation [30], it is also necessary to define and use specific resilience metrics to understand and explain the behavior of routing protocols. This lack is one of Achilles' heels in the evaluation of ad hoc networks.

Table 2: Summary of metrics considered by current evaluation platforms

Measure		*Opnet* [13]	*NS3* [31]	*Glomosim* [14]	*CMUWE* [22]	*ORBIT* [21]	*Castadiva* [23]	*Emulab* [25]	*MobiEmu* [24]	*Roofnet* [16]	*Floornet* [17]	*APE* [18]	*MINT* [19]
Performance	Throughput	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓
	Packet delivery ratio / Packet loss	✓	✓	✓	✓	✓						✓	
	Delay	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓
Resources consumption	Routing overhead	✓	✓	✓									
	Energy consumption	✓	✓	✓									
Resilience	Link flapping											✓	
	Connectivity									✓		✓	

2.4. Discussion

This section has reviewed the most widely used evaluation trends. Thus, it has been possible to identify some challenges and opportunities related to the lack of works that consider, despite its great importance, (i) the presence of faults and attacks in the network during experimentation and (ii) the use of resilience metrics addressed to evaluate the impact of those faults and attacks on routing protocols. These points, as explained in this section, could be addressed by applying the notions of resilience evaluation, an approach that can be used to determine the ability of a computing systems to perform properly in the presence of faults/attacks and to recover from any service degradation. On one hand, the deliberate introduction of faults in the system in a repeatable and controllable way (fault injection) could be useful to deal with the former issue. On the other hand, selecting a suitable set of resilience metrics may address the second issue.

Previous testbeds have improved the state of the art. In particular, emulation represents a trade-off between simulation and real-world prototyping that eases the study of real

ad hoc networks while avoiding the problem of deploying and controlling a large number of actual mobile nodes within a wide area. Most of these testbeds, such as Emulab and PlanetLab, are typically shared by multiple users. So, administrators need to define some testbed policies to limit the intrusiveness between experiments from different users. In practice, these policies are adequate for the platform governance, but unfortunately, they impose certain constraints that limit their coexistence with the fault injection strategies proposed in this paper. For example, they forbid the injection of continuous bursts of messages, thus preventing the injection of packet flooding-based faults or attacks. This is one of the reasons why resilience evaluation requires the use of exclusive resource testbeds.

In our previous conference papers we performed some preliminary steps towards fault and attack injection in ad hoc networks. In our prior work [32], we developed a methodology to experimentally evaluate and compare the behavior of routing protocols. In [33] we showed how the robustness of two security (prevention and detection) mechanisms for ad hoc routing protocols in the presence of two different attacks could be improved by properly tuning the target protocols. Further, in [34], we extended the scope of our research to evaluate additional routing protocols in presence of a wider set of 11 faults and attacks. While these works focused on obtaining experimental results, the present paper achieves a higher procedural maturity level. Indeed, the present paper focuses on the specification of suitable measures and how to obtain them in practice while addressing the definition and implementation of processes to achieve adequate levels of controllability, repeatability, and observability in the resilience evaluation of ad hoc routing protocols.

3. REFRAHN methodology

The previous section showed the impairments that limit the resilience evaluation of ad hoc routing protocols in practice. The present section proposes a methodology to cover such lacks. Thus, the interest of REFRAHN is not to merely perform resilience-related experiences. Instead, it aims at identifying the key points when addressing resilience evaluation in a very specific domain: ad hoc routing protocols. Conversely to previous frameworks or testbeds, our research focuses on the requirements to resilience evaluation. Then, Section 4 will exemplify one way, among those possible, to implement these requirements. Our goal is to share with the research community our conclusions about the need and the way to recreate more realistic conditions in the evaluation of ad hoc routing protocols. This involves proposing (i) techniques to simulate or emulate the presence of perturbations from both accidental and malicious nature; and (ii) specific resilience measures to determine the impact of those perturbations on the network behavior, thus guiding the decision of evaluators.

The REFRAHN methodology defines three basic stages: the experiments configuration, the experiments execution, and the analysis of results. Figure 1 introduces such stages.

3.1. Experiments configuration

All the parameters that characterize the target scenario must be precisely determined prior to experiments execution. As previously stated, recreating the execution environment characteristics and selecting a proper set of metrics that reflects such characteristics is essential for the quality of evaluation results. Accordingly, our methodology divides such sensitive parameters into four categories: (i) the *network profile*, which configures all the parameters related to the network deployment; (ii) the *execution profile*, that animates the network deployment with realistic workloads and faultloads; (iii) the *metrics definition* to quantify the behavior of the system in presence of such elements; and (iv) the *campaign configuration*, to delimit the statistical representativeness of the experimental execution.

The *network profile* is in charge of accurately defining the network under study, its characteristics and deployment. Among those parameters, the *topology* defines the relative

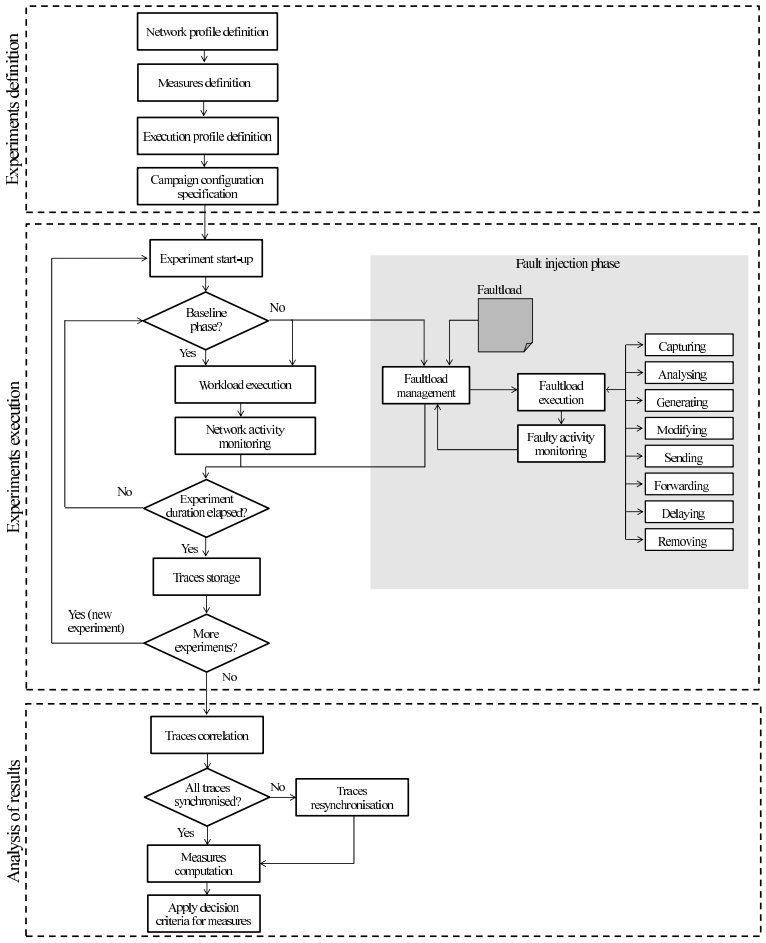

Figure 1: Summary of the experimental resilience evaluation stages.

position and initial distribution of the nodes within a delimited area. The *network size* is determined by the total number of network nodes. The *area* bounds the physical space where nodes are confined for the duration of the experiments. Finally, thanks to the *mobility pattern* (like Manhattan [35], random way point [36], or walking [37] models) and the *node speed*, it is possible to compute the trajectory followed by network nodes. The *routing protocol* to be considered is another parameter of prime importance.

The execution profile is defined in terms of the *workload* and the *faultload*. The *workload* describes the application traffic exchanged among nodes. Data flows are generated by the *applications* used by *source* and *destination* nodes. Real applications are preferred to increase the representativeness of results. Some examples of useful applications to study particular behaviors are voice over IP (VoIP) conferences, instant messaging, or peer-to-peer (P2P) file exchanging. Yet, the use of synthetic workloads could be also interesting to study data flows from a generic viewpoint. The bit rate of synthetic workloads can be constant or variable. Conversely to real workloads, synthetic ones offer a higher level

of controllability because their execution does not depend on interactions with end users. The *faultload* defines the adverse conditions the system will face during the evaluation. One of the major issues along these years of research concerns the representativeness of injected faults and attacks [38]. In other words, the faultload should include those faults and attacks that could actually be experienced during the system lifetime in order to obtain realistic evaluation results. If injected faults and attacks are not representative, then the usefulness of the evaluation process can be compromised.

The set of metrics proposed in this methodology characterizes different features of routing protocols such as performance, resources consumption, and resilience, thus providing the evaluator with different criteria to assess target protocols.

The proposed resilience evaluation approach focuses on the practical execution of experiments. Such experiments can be grouped into experimental campaigns. Several parameters define the configuration of experimental campaigns: the *warm-up time* required for the establishment of initial routes between network nodes, the *duration* of the experiments, and the *number of repetitions* determining the suitable number of experiments to be performed.

3.2. Experiments execution

During its start-up, each experiment requires an initial set-up time to reset the system to its original state, followed by a warm-up time, devoted to lead the targeted routing protocol to a controllable state. From our experience, this means that after the warm-up period, results tend to be more steady and repeatable. This strategy is compatible with the idea of evaluating the bootstrapping of a routing protocol. Indeed, it is possible to schedule the start of experiments to any instant of time t, even when t is scheduled before the protocol launching. This type of experiment configuration can be useful to determine the time devoted to the protocol to be operational.

Network nodes are configured to play their assigned role via a control network. Common nodes are programmed to generate network traffic according to the selected workload, whereas injector nodes are in charge of introducing faults in the system according to the defined faultload. Regardless whether nodes are configured as common or injector nodes, they follow the predefined topology and execute the targeted routing protocol.

3.2.1. Golden run execution

After the network initialization, the *baseline phase*, known as *golden run* in the dependability domain, starts. It consists of a number of successive experiments in which the system executes the selected workload, in absence of faults, while nodes activity is monitored. To cope with this task, common nodes will exchange application data flows according to the evaluator preferences. A number of probes will be required in each node to monitor its activity during experimentation. Basically, these probes collect information about the amount and type of packets exchanged, the timestamps identifying when packets were received and forwarded, and the energy consumed by each packet.

3.2.2. Fault injection execution

The *fault injection phase* corresponds to the execution of the workload in presence of the faultload to evaluate the impact of faults and attacks on the system behavior. Its procedure is identical to the golden run phase, but introducing a fault into the system at a particular location at a given time. The injection procedure, the injection point (or target), the trigger, and the duration of each considered fault are aspects that must be specified. The injection point determines where to inject the fault, which may require monitoring the network to identify a suitable location. Once determined, the generation, delay, removal, and modification of packets takes place. Essentially, injector nodes are prepared to carry out a set of generic basic actions that can be applied at the routing level in the right order.

Such list, introduced in the gray box in Figure 1, represents the union between the sets of actions proposed in previous works [39, 40, 41]. These actions involve (i) sniffing and storing packets from the wireless medium, (ii) processing stored information, (iii) creating packets, (iv) sending packets, (v) relaying packets, (vi) introducing a given lapse of time between one packet and the following, (vii) altering the content of legitimate packets, or (viii) deleting packets.

The combination of these basic actions can result into far more complex faults. Additionally, injector nodes can trace their malicious activity (start of fault injection, fault activation, and end of fault injection) for its subsequent analysis. The injection procedure, injection point, trigger, and duration of some of the perturbations listed in Table 1 are next presented as an example of the novelty of the methodology:

Signal attenuation

- *Injection procedure:* As our methodology emulates the location of nodes, the effect of increasing or decreasing the distance among them is recreated through packet loss. Previous works using physical attenuators such as [42], similarly obtained a degradation in the packet loss when emulating an increment in the distance between a given pair of nodes.
- *Injection point*: The packet loss is applied over routing and applicative packets received by the NIC of every node participating in the experiment campaign, thus recreating the worst possible scenario.
- *Injection trigger:* The fault is activated from the beginning until the end of the experiment.
- *Fault duration:* Although generally considered transient, this fault is bounded by the experiment duration given its activation in every network node.

Ambient noise

- *Injection procedure:* The interferences created in the wireless medium are emulated through packet corruption [43].
- *Injection point:* A packet corruption rate is applied over the routing and applicative packets received by the NIC of every node participating in the experiment campaign to recreate the worst possible case.
- *Injection trigger:* The fault is activated from the beginning until the end of the experiment.
- *Fault duration:* Although generally considered as a transient fault, its duration is bounded by that of the experiment given its activation in every network node.

Battery extenuation

- *Injection procedure:* This fault can be emulated by disabling the packet receiving and sending capabilities of injector nodes [43]. Affected nodes will be seen as fallen nodes by their neighbors.
- *Injection point:* The fault disables the functions at the victim node's wireless NIC.
- *Injection trigger:* The fault is activated from the beginning until the end of the experiment.

- *Fault duration:* The duration of the fault, typically permanent, is bounded by the experiment duration.

Traffic peak

- *Injection procedure:* This fault is injected by increasing the packet sending rate of an injector node up to the saturation point of the network (e.g., around 18 Mbps in IEEE 802.11g). This model emulates the case where tens of users share the same route to exchange data [44].
- *Injection point:* The fault, launched by an injector node, degrades the behavior of all nodes that are in the same radio range during the packet emission.
- *Injection trigger:* The fault activation is scheduled by the evaluator. By default, it is launched at the beginning of the experiment.
- *Fault duration:* The nature of a traffic peak is transient. So this fault has been defined to affect a limited percentage (generally delimited by the evaluator) of the experiment's duration.

Sink hole attack

- *Injection procedure:* The sink hole attack is recreated through the exchange of routing packets between common and injector nodes [44]. The injector node playing the role of the attacker must create routing packets their neighbors are able to understand.
- *Injection point:* The fault is launched by an injector node. First, the injector node (M) must dynamically locate a proper data flow to identify victim nodes (A, B and C). Then, the injector node tries to replace the position of one legitimate node (B) in the route. Likely, it announces itself as a suitable node so that neighbor nodes (A and C) take it into account to establish a new route, as Figure 2 illustrates.
- *Injection trigger:* This attack is triggered when the injector node is in the same radio range as the victim nodes that are forwarding a target data flow.
- *Fault duration:* This fault is generally transient given the mobility, connection, and disconnection of nodes.

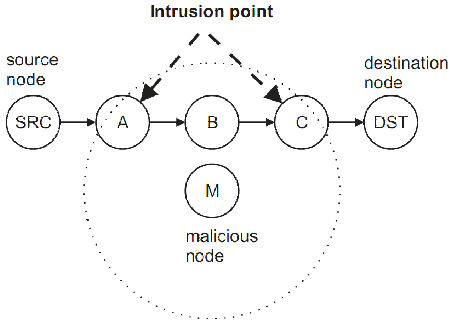

Figure 2: Intrusion point for sink hole attacks.

Tampering attack

- *Injection procedure:* The recreation of tampering attacks requires first the execution of a sink hole attack [45]. Then, the target data flow between a pair of nodes is altered.
- *Injection point:* The payload of incoming target applicative packets is modified before relaying them towards their destination.
- *Injection trigger:* This attack is triggered as soon as the sink hole attack succeeds.
- *Fault duration:* This is generally a transient fault, although its duration depends on the injector node's ability to keep in the route as far as possible.

Replay attack

- *Injection procedure:* The injector node is in charge of capturing routing packets from one zone of the network and reproducing them in a different one [44].
- *Injection point:* Captured routing protocol packets will be broadcasted by the injector node, thus affecting the topology map of all nodes within its radio range during the attack.
- *Injection trigger:* The injector node will capture routing protocol packets during the whole experimentation time. In parallel, such packets will be replayed after a fixed period of time determined by the evaluator.
- *Fault duration:* Although active for the duration of the experiment, the particular duration of this attack depends on the percentage of time it affects a given node. Given the dynamic nature of ad hoc networks, it is generally considered transient.

Selective forwarding attack

- *Injection procedure:* This attack requires the successful execution of a sink hole attack [45]. Then, the target data flow between a pair of nodes is dropped.
- *Injection point:* The injector node drops all those packets belonging to the target data flow.
- *Injection trigger:* This attack is triggered as soon as the sink hole attacks succeeds.
- *Fault duration:* The duration of this attack depends on the injector node's ability to keep in the route as far as possible.

Jellyfish attack

- *Injection procedure:* This attack requires the successful execution of a sink hole attack [45]. After that, the target data flow between a pair of nodes is delayed.
- *Injection point:* The injector node delays all those packets belonging to the target data flow a given period of time established by the evaluator.
- *Injection trigger:* This attack is triggered as soon as the sink hole attack succeeds.
- *Fault duration:* The duration of this attack depends on the injector node's ability to keep in the route as far as possible.

Flooding attack

- *Injection procedure:* The injector node will send a constant burst of packets reaching the saturation point of the network (for example, around 18 Mbps in practice for IEEE 802.11.g) [44].
- *Injection point:* The fault is launched by an injector node, thus degrading the behavior of all nodes within its radio range during the packet emission.
- *Injection trigger:* The injector node will deploy the attack from the beginning of the experiment.
- *Fault duration:* Although active for the duration of the experiment, the particular duration of this attack depends on the percentage of time it affects a given node. Given the dynamic nature of ad hoc networks, it is generally considered transient.

Neighbors saturation

- *Injection procedure:* This fault is emulated by a single injector node that sends a burst of fake routing protocol packets announcing a huge amount of different links [44]. The amount of new links announced in such burst could be statically assigned by the evaluator, or dynamically determined by the injector node itself.
- *Injection point:* As far as the packets sent by the injector node are received by all the neighbor nodes within its radio range, they will believe that each packet makes reference to a different announcing node.
- *Injection trigger:* The injector node will deploy the attack from the beginning of the experiment.
- *Fault duration:* The fault duration is limited by the number of bursts launched by the injector node. The evaluator should configure this parameter.

Sequence number replay

- *Injection procedure:* This fault is emulated by manipulating the generation of packet sequence numbers in a single node, in such a way that all packets are sent with the same packet identifier [44].
- *Injection point:* The injector node will mainly affect those nodes within its radio range.
- *Injection trigger:* The injector node will deploy the fault from the beginning of the experiment.
- *Fault duration:* The fault will be deployed for the whole duration of the experiment. However, from the network viewpoint, it can be considered transient as its practical duration is bounded by the time it affects nodes within the radio range of the injector node.

The degree of intrusiveness (either temporal or spatial) induced in network nodes by the fault injection process is negligible. Conversely to other fault injection methodologies [46], in no case the system execution is paused or forced to launch complex routines or tasks to induce a faulty behavior in common nodes. The faulty activity is monitored by the fault injection monitor.

3.3. Analysis of results

At the end of the experiments, all log files generated with information collected while monitoring the system activity are processed to enable the analysis of results. Resilience measures will depend on the logs generated by injector nodes. Once measurements are processed, resulting metrics will be shown.

4. A tool to support our methodology

REFRAHN proposes emulation as a way to reduce the physical area of experimentation and maintain the basic multi-hopping properties while considering fault injection to introduce a wide variety of faults and attacks within the ad hoc network. REFRAHN implements three different stages related to the proposed methodology: (i) the definition of all the required parameters to specify the resilience evaluation to be performed; (ii) the execution of the requested fault injection experiments while monitoring the system's execution; and (iii) the analysis of the impact of these faults into the system.

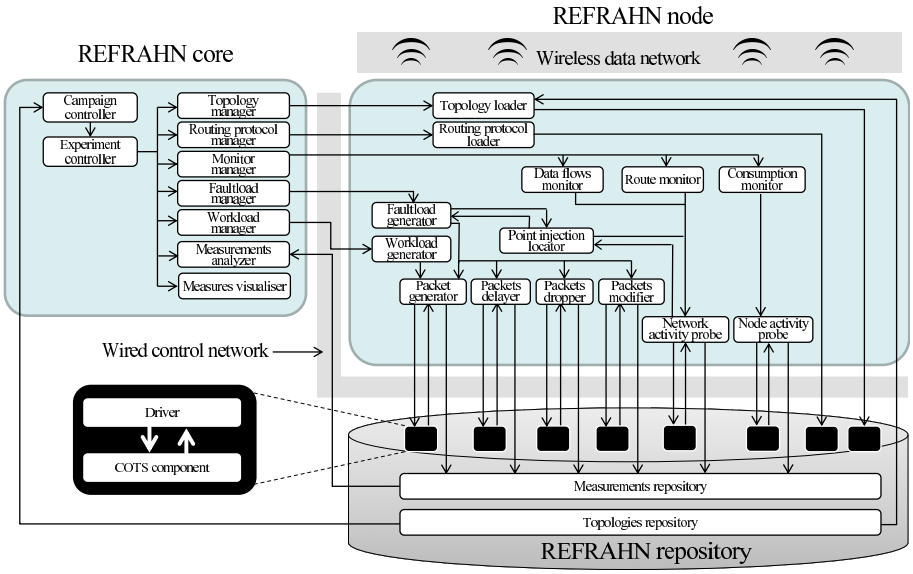

Figure 3: General architecture of REFRAHN.

As can be seen in Figure 3, the proposed architecture of REFRAHN consists of two different networks. The first one, the wireless (data) network, is the experimental ad hoc network used by nodes to exchange information and test real routing protocols. REFRAHN nodes can play the role of either common or injector nodes. The former send and forward traffic to other network nodes, whereas the latter are responsible for recreating the occurrence of faults in the network. The second network, the wired (control) network, connects all nodes with a special one, the REFRAHN core, using Ethernet technology to avoid large latencies. This node is in charge of configuring all network nodes and controlling the experimentation. The use of two different interfaces is very important to reduce the intrusiveness of the evaluation platform itself in the evaluated system. Furthermore, the architecture is completed with a shared space, named REFRAHN repository, that can be accessed by the core and network nodes to store components and data. The REFRAHN

repository has been implemented using the Network File System (NFS) technology, thus enabling the REFRAHN core and network nodes to access files over the wired network in a manner similar to how local storage is accessed. Maintaining the internal clocks of network nodes synchronized is essential so that an experiment may start at the same time in all participating nodes, avoiding significant latency effects and maximizing results accuracy. Accordingly, nodes are synchronized through the Network Time Protocol (NTP).

It is to note the flexibility offered by the proposed architecture. Since nodes mobility is emulated, the experimental platform may accommodate network nodes physically close. The emulation of mobility enables experimenters to create scenarios with dynamic topologies without physically moving the nodes. In this way, real ad hoc networks can be easily used for experimentation, since nodes behave as if they were moving around the designated area despite being static and receiving all the traffic generated by the rest of nodes. Although our methodology is generic enough to support different types of emulation configurations, the implementation of our testbed is based on emulation of mobility through packet filtering. This decision limits the scalability of our approach to tens of nodes to avoid the intrusiveness caused by an overloaded wireless medium.

Figure 4: Typical router stack configuration.

Indeed, we have performed some preliminary experiments with stacks of 20 routers without obtaining significant deviations from reality (Figure 4 shows a typical nodes stack configuration). The benefits in terms of cost, simplicity, and portability, clearly support this decision. In practice, if the physical deployment area is not a problem, surely the use of RF attenuators is more recommended from the point of view of accuracy. However, most research laboratories typically present logistic spatial limitations. In these cases, the alternative described in this section can be really useful. Figure 5 can assist the reader to better understand our approach. The considered ad hoc routing protocol can be changed to evaluate the features of different targets. Likewise, the wireless interface can also be replaced to take into account different physical wireless technologies.

REFRAHN has been defined to support IP as the base technology for communications, which makes our proposal independent from the technology used in the physical layer (such as Bluetooth, Zigbee, or WiFi). The implementation of REFRAHN eases the remote control of all devices through an intuitive application that assists the user to dynamically manage network nodes according to the desired deployment scenario.

Experiment randomness may be affected by internal causes, when referring to internal processes within the node, e.g., changes in the operating system (OS) or the central processing unit (CPU) load; and external ones, when considering changes in the unstable wireless environment. To deal with the first ones, we limit the execution of background processes to control the processing variability. Thus, the average CPU usage in network nodes is typically underneath 15%. Additionally, we reset the experimental process after each execution to discard residual randomness. To deal with external causes, we propose

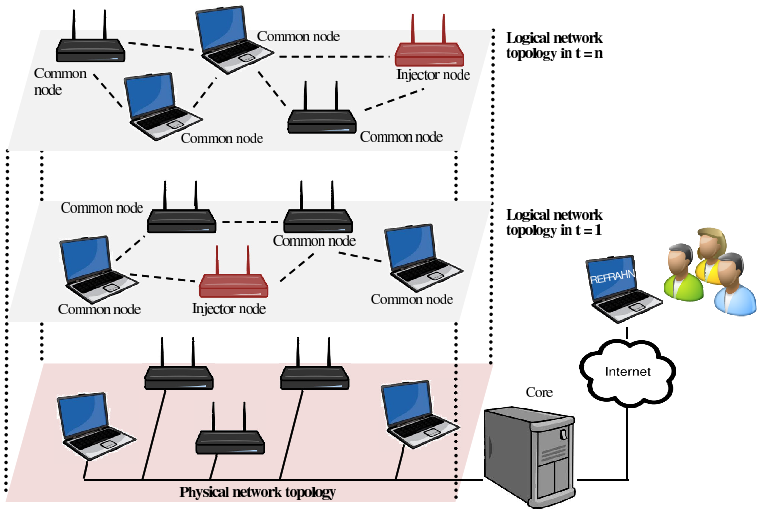

Figure 5: Basic elements of REFRAHN.

to launch experiments at night, when interferences caused by the network activity are negligible. Conversely to existing testbeds such as Emulab [25], where the platform can be concurrently accessed, the access to REFRAHN is exclusive (non-concurrent). This decision ambitions to reduce the interferences between experiments of different users, thus limiting the intrusiveness these phenomena may have on other users experiments.

4.1. Experiments definition

This process is manually launched by the user to gather all the information required to perform the desired experiments. REFRAHN's graphical user interface (GUI) will guide the evaluator along this process. This GUI enables the user to configure all the parameters introduced in Section 3.

4.2. Execution of experiments

This process controls the experiment execution flow, including the configuration stage, the execution of the workload, the observation of the system's behavior, and the injection of faults. As already stated, nodes can remain physically stationary (in a laboratory, for example) while their mobility is entirely emulated via software. *iptables* is a generic firewall for the definition of packet-filtering rules. Taking this into account, each node is provided with a *visibility script file*, which defines the set of rules that must be used to emulate the node's visibility, and thus its mobility during the experiment. REFRAHN enables the definition of different types of either real or synthetic traffic flows between pairs of nodes. On one hand, the workload has been instrumented to send real traffic by means of actual network traffic applications. By default, well-known open-source applications based on VoIP (like *Ekiga*) and File Transfer Protocol (FTP) (*FileZilla*) are supported. However, as previously stated, additional applications can be registered in REFRAHN (this is the case, for instance, of remote control applications for drones or helicopters). On the other hand, synthetic traffic is created by means of the well-known application *iperf*, an open-source packet generator for Linux to establish CBR (Constant Bit Rate) or VBR (Variable Bit Rate) UDP/TCP data flows between pairs of nodes. The only requirement for these

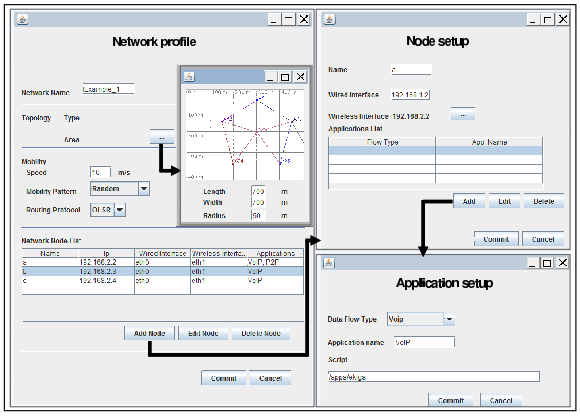

Figure 6: REFRAHN GUI: Forms to configure the network profile (a) and the execution profile (b).

tools is to support command-line interaction. Thus, their remote and automatic control, and their consequent integration within REFRAHN becomes very easy.

The fault injection procedure is executed in parallel with the workload. We rely on well-known commercial off-the-shelf (COTS) components to ease the portability and maintainability of REFRAHN. Traffic monitors are suitable tools to *capture* and *analyze* network traffic. It is to note that *tcpdump* is a good solution, among the available traffic monitors considered nowadays, given the wide variety of filtering parameters that can be tuned, like timestamp, IP and MAC addresses, ports and so on. To instantiate packet *generation* actions, *iperf* and *nemesis* [47] have been used to encapsulate our malicious payloads. Packet *modifying* and *forwarding* actions can be implemented through tools such as *tcprewrite* and *tcpreplay* respectively, both included within the *tcpreplay* suite. Such tools edit packets already captured and replay them at arbitrary sampling speeds onto the network. Finally, network emulation libraries such as *netem* [48] are very useful to recreate packet *removing* and *delaying*. *netem* emulates variable delay, loss, duplication and re-ordering of packets in the node's NIC, and is generally enabled by default in Linux-based kernels. These tools can be easily controlled via command-line interfaces, which

eases their automatic configuration without human intervention.

Table 3: Description of the faultload considered for the evaluation.

Ref.	Fault model
F1	***Signal attenuation***: The percentage of packet loss induced by this fault is emulated by the *netem* tool following a normal distribution. By default, this percentage is set to 5% for the duration of the experiment.
F2	***Ambient noise***: The percentage of packet corruption induced by *ambient noise* is introduced by the *netem* tool following a normal distribution. By default, this percentage is set to 5% for the duration of the experiment.
F3	***Battery extenuation***: The effects of *battery extenuation* are emulated by means of the *netem* tool by introducing a packet loss in both receiving and sending packets. By default, packet loss is set to 100% for the duration of the experiment.
F4	***Traffic peak***: The massive generation of packets is emulated using the *iperf* tool. The effect of peaks has been recreated by default by creating 2-second bursts of 18 Mbps every 10 seconds.
F5	***Sink hole attack***: The selection of the injection point to launch the intrusion depends on the sensitive information provided by *tcpdump*. Once victim nodes have been identified, the route intrusion process requires knowing the syntax and semantics of the packets exchanged by the target routing protocol. The fault injector node is in charge of providing *nemesis* correct payloads, in the right order, to successfully join the target route.
F6	***Tampering attack***: This attack requires the successful execution of a *sink hole attack*. Once the injector node intrudes the communication route, it uses the *tcprewrite* tool to edit the payload of the applicative packets it should forward. By default, the original payload is changed by a string set to 0s.
F7	***Replay attack***: The injector node first uses *tcpdump* to capture routing packets exchanged in the network. Filtering those packets by port will be useful to identify target routing protocol packets. Then, by default, captured routing packets will be replayed 30 seconds later using the *tcpreplay* tool for the duration of the experiment. Replaying routing packets aims at inducing topology incoherences in the network.
F8	***Selective forwarding attack***: This attack can only take place after a successful *sink hole attack*. After that, the injector node configures the *netem* tool to drop all applicative packets belonging to the target data flow.
F9	***Jellyfish attack***: This attack requires a successful *sink hole attack*. After that, the injector node delays all data packets using the *netem* tool. By default, this delay has been configured to 2 seconds since it is time enough for its effects to show.
F10	***Flooding attack***: In this attack, the injector node must generate a heavy network traffic. Conversely to the effects of *traffic peak*, this attack generates a continuous broadcast data flow using *iperf*. This data flow is configured by default to send a rate of 18 Mbps for the duration of the experiment.
F11	***Neighbors saturation***: Injector nodes emulate this fault using *nemesis* to randomly forge fake routing packets. By default, such routing packets are sent in a burst announcing 400 new fake links between nodes.
F12	***Sequence number replay***: Injector nodes use the *nemesis* tool to forge routing packets with exactly the same sequence number. By default, the duration of the fault was set to that of the experiment.

The fault injector configures previously presented tools to orchestrate the faulty activity deployed by injector nodes. The proposed faultload takes into account a non-exhaustive but sufficient set of well-known threats that affect the behavior of routing protocols. Some of them such as *signal attenuation* [43], *ambient noise* [43], *battery extenuation* [43], and *traffic peak* [44] are protocol-independent faults, while some others depend on the particular nature of each routing protocol, like *sink hole attack* [44], *replay attack* [44], *tampering attack* [45], *selective forwarding attack* [45], *jellyfish attack* [49], *flooding attack* [44], *neighbors saturation* [44], and *sequence number replay* [44]. The particular parameterization used in REFRAHN for each threat is illustrated in Table 3.

The faultload implemented in this paper is, to the best of our knowledge, one of the widest sets in the literature of evaluation frameworks for ad hoc routing protocols. However, although we are aware that this set faults and attacks is far from being complete, we claim that it can become an adequate resilience benchmark for ad hoc routing protocols given the heterogeneous origin of considered threats. Indeed, this faultload can be adapted to the evaluator needs by configuring their injection procedure, injection point, injection trigger, and injection duration. Furthermore, our faultload presents threats from an accidental and malicious nature, that can affect different characteristics of routing protocols.

4.3. Analysis of results

The analysis of results is automatically launched after the execution of the experiments. Monitoring probes are executed in parallel with the workload and faultload. Consequently, different logs are generated by nodes while monitoring the behavior of the ad hoc network during the experiments. Typically, these logs are obtained from both the output of workload and faultload generators and additional monitoring tools (e.g., the ping utility). In

Table 4: Selected metrics.

Performance	**Description**	**Formula**	**Interpretation**
Packet delivery ratio (%)	% of packets that arrived from source to destination in a communication route.	$\frac{\#\text{received packets}}{\#\text{generated packets}} \cdot 100$	The higher the better
Packet loss (%)	% of packets that were lost in a communication route.	$\frac{\#\text{lost packets}}{\#\text{generated packets}} \cdot 100$	The lower the better
Delay (ms)	Mean time required by a packet to get from source to destination.	$\frac{\text{delay of received packets}}{\#\text{received packets}}$	The lower the better
Resources consumption	**Description**	**Formula**	**Interpretation**
Energy consumption (J)	Energy consumed by the NIC of a node that takes part in a communication route.	$E = E_{Sending} + E_{Overhearing} + E_{Receiving}$	The lower the better
Resilience	**Description**	**Formula**	**Interpretation**
Route availability (%)	% of time the target route was ready to be used.	$\frac{\text{time the route worked}}{\text{experiment duration}} \cdot 100$	The higher the better
Packet integrity (%)	% of packets received at destination whose content (or payload) was not illegitimately altered.	$\frac{\#\text{non-altered received packets}}{\#\text{received packets}} \cdot 100$	The higher the better
Threat exposure (%)	% of time the communication route was exposed to any fault.	$\frac{\text{time the fault was activated}}{\text{experiment duration}} \cdot 100$	The lower the better
Fault effectiveness (%)	% of the threat exposure time the fault actually impacted the network behavior.	$\frac{\text{time the fault was activated}}{\text{time the fault was launched}} \cdot 100$	The lower the better

order to ease the integration of new COTS in REFRAHN, an interface must be implemented for each one of these tools so that results can be parsed to an equivalent format. Figure 3 illustrates this principle. After being automatically formatted, the whole set of data stored in these logs must be processed, correlated, and analyzed to extract those values needed to estimate desired metrics. Table 4 summarizes the purpose, formula, and interpretation of the different metrics considered in our case.

Generic performance metrics, typically taken into account in current literature such as the *packet delivery ratio*, *packet loss*, and *delay*, have been selected. The resources consumption is computed throughout the *energy consumption*. Finally, this subset of metrics is completed with some resilience metrics: *route availability*, *packet integrity*, *threat exposure time*, and *fault effectiveness*.

The *common node log* is useful to compute performance- and resources-consumption-related metrics. The other two logs, the *ping log* and *injector node logs*, are required to estimate resilience-related metrics. The rest of this section details how defined metrics for performance, resources consumption, and resilience can be deduced from all these experimental measurements.

4.3.1. Performance metrics computation

Each node must collect all the information related to the workload activity (comprising the application traffic sent or received) using *common node logs*. Such logs will be used to compute the *packet delivery ratio*, *packet loss*, and *delay*. Tools, like *tcpdump*, are good candidates to generate these logs given their flexibility (e.g., -vv option in *tcpdump* prints a wide variety of useful information, like the Time To Live (TTL), ID, or the total length of packets). Regarding the intrusiveness, *tcpdump* is a light process running through command line, which monitors all the network activity with a low CPU and memory usage.

As previously indicated in Table 4, the *packet delivery ratio* is computed as the relationship between the amount of packets delivered to the destination node and the total amount of packets sent by the source node. The amount of packets sent can be computed from the source node's log whereas the set of packets received is estimated from the destination node's log with respect to the same data flow.

Packet loss is computed as the relationship between the amount of packets not delivered to the destination node and the total amount of packets sent by the source node (see Table 4). The amount of packets not delivered can be computed as the difference between the set of packets sent (from the source node's log) and the set of packets received (from the destination node's log) with respect to the same data flow. It can be alternatively estimated as 100-*packet delivery ratio*.

The traffic log of source and destination nodes is required to compute the average *delay* of data flows (see Table 4) as the difference between the timestamps of packets sent and received.

4.3.2. Resources consumption metrics computation

The *energy consumption* is the measure considered to estimate the resources consumption. It is computed as the energy required by common nodes' NIC to send, receive, and overhear packets (see Table 4). To estimate this measure, it is necessary to compute the amount of packets sent, received, and overheard by a node from its neighborhood (including both applicative and routing packets). The notion of overheard traffic makes reference to those packets listened by a node even when it is not their addressee. Accordingly, it is necessary to filter from the traffic log of each node (i) those packets sent by the node, (ii) those packets addressed to the node itself, and (iii) those which are not. Then, those values are multiplied by the energy required to send, receive, and overhear a packet, respectively. To improve accuracy, we obtained this value directly from the wireless NIC using hardware probes.

A generic expression to estimate the energy consumed (in Joules) to send a packet p from a wireless NIC is $E_s(p) = P_s(p) * t_s(p)$, where $P_s(p)$ is the power consumed to send a packet and $t_s(p)$ is the time required to transmit it. $P_s(p)$ can be computed as $P_s(p) = V_{in} \frac{v_s(p)}{R}$ [50], where V_{in} is the input voltage (e.g., about 3.3 V for current laptops), $v_s(p)$ is the voltage required to send a packet, and R is a test resistance (1 Ω is generally enough). Accordingly, we get $E_s(p) = 3.3 \frac{v_s(p)}{1} t_s(p)$ Joules for our example, where $t_s(p)$ depends on the packet size (the larger the packet, the longer the time to send it). Finally, the total energy consumed by a wireless card to send packets during experimentation can be approximated as $E_s = E_s(p) * N_s$, where N_s is the total amount of packets sent. Similarly, we can also compute the energy consumed by receiving (E_r) and overhearing (E_o) packets, to estimate the total energy consumed by a wireless card as $E = E_s + E_r + E_o$.

This model applies to single stream NICs. Our future work will extend this approach to include multi-streams (MIMO) Wi-Fi nodes.

4.3.3. Resilience metrics computation

The *route availability* metric represents the average probability of a packet to be delivered from source to destination nodes (see Table 4). In order to estimate this probability it is necessary to deploy some mechanism in the ad hoc network to determine if the communication between source and destination nodes is possible at any given time. Since the network workload may not ensure that applicative packets are continuously exchanged between nodes, ICMP ECHO REQUEST (ping) messages are continuously sent from source to destination. This activity is reported by *ping logs*.

The rest of resilience metrics (*packet integrity*, *threat exposure*, and *fault effectiveness*), can be derived from the information provided by the injector node. As shown in Table 5, the *injector node log* lists all the events the injector node induces on the network. The first element of each log's line is a timestamp stating when an event occurred. This is followed by the event identifier and description. In the particular case of the injector log reported in Table 5, the reader can see the log of a *selective forwarding attack*. Event E1 notifies the detection of a target data flow, whereas Events E2 and E3 notify the execution of the *sink hole attack* and its successful intrusion, respectively. Finally, Event E4 confirms the packet dropping of the target data flow. In this case, the injector node only disrupts packets (i.e. data flows) sent from node 192.168.2.56 to node 192.168.2.55 in a port range between 5000 and 5099.

Using this log, the *packet integrity* of a data flow (see Table 4) is computed as the ratio between (i) the time the data flow is not affected by attacks that alter the content of packets and (ii) the duration of the experiment.

Table 5: Sample injector node log for a selective packet dropping attack.

timestamp	event notification
1233226824.839366	E1 DETECTION data flow between 192.168.2.56 and 192.168.2.55
1233226836.093937	E3 OFF INTRUSION data flow between 192.168.2.56 and 192.168.2.55
1233226836.116471	E2 GENERATING malicious routing packets between 192.168.2.56 and 192.168.2.55
...	...
1233226858.660319	E3 ON INTRUSION data flow between 192.168.2.56 and 192.168.2.55
1233226858.684511	E2 GENERATING malicious routing packets between 192.168.2.56 and 192.168.2.55
1233226859.797914	E4 ON DATA FLOW DISRUPTION between ports 5000 and 5099

The *threat exposure* can be computed by analyzing the intervals of time when common nodes are in the radio range of injector nodes, i.e., when they are susceptible to suffer the fault effects (see Table 4). This information can be easily extracted from injector node's logs. It is calculated as the ratio between the time the fault is activated and the duration of the experiment. In the example of Table 5, the time the fault is activated refers to the difference between Event E1 (the injector node detects the data flow) and Event E3 ON (the intrusion has been successfully completed).

Finally, *fault effectiveness* is computed as the ratio between the time the fault is activated and the time spent by the injector node to successfully activate the fault (see Table 4). In the example of Table 5, the time required by the injector node to activate the fault is computed as the difference between Event E2 (the injector node starts generating fake packets addressed to the routing protocol) and Event E3 ON (the intrusion has been successfully completed).

5. Case study

This section illustrates the feasibility of REFRAHN through a simple case study. All fault models implemented by REFRAHN have been injected in different experiment campaigns to determine whether our approach can be effectively used for the resilience evaluation of ad hoc routing protocols. For the sake of representativeness, the experimental target routing protocol considered in this case study is well-known by the ad hoc networks community and extensively used for experimentation: the Optimized Link-State Routing (OLSR) protocol. OLSR [51] is a proactive protocol which maintains routing information within network nodes to ease the quick establishment of routes. Network nodes continuously disseminate HELLO messages to announce their presence to their neighborhood, and Topology Control (TC) packets to disseminate such information to the rest of the network. So as to ensure that every single node shares the same vision of the network topology, all this information is distributed by following an optimized flooding procedure. This study will consider different versions of an OLSR implementation called *olsrd* (from *www.olsr.org*), which instruments a Link Quality extension to compute the minimum path between two nodes. Versions *v.0.4.10* (released in 2006), *v.0.5.6* (released in 2008), and *v.0.6.0* (released in 2010) have been considered. Moreover, an additional version implementing a MD5 encryption-based mechanism has been included in the case study to evaluate its effectiveness versus default versions. The proposed mechanism consists of a plugin added to *olsrd v.0.6.0* (from now on *v.0.6.0+md5*) in charge of establishing a 3-way handshake between every pair of neighbor nodes. To verify the exchange of routing packets, a signature (or hash) is included at the end of any routing packet. The signature is computed by cyphering the content of the outgoing routing packet with a timestamp and a 128-bits symmetric key. This symmetric key should be known by all legitimate network nodes. As far as the signature of each incoming routing packet is checked, the routing

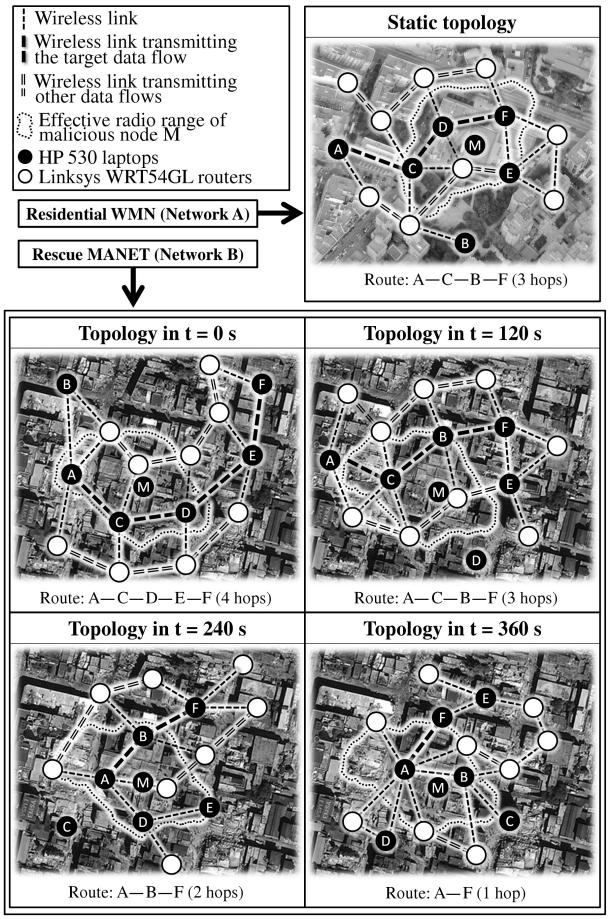

Figure 7: Topology evolution in considered scenarios.

integrity is protected against malicious outsiders. It is worth noting that although legitimate users would still be able to deploy attacks in the system, the interest of selecting this mechanism is to show the severe impact that accidental faults or attacks launched by intruders may have on the network behavior. Since none of the attacks provided by REFRAHN focuses on secret key exploitation, we assume a default secret key is provided by the network administrator. In no case the study of securely secret key sharing is in the scope of this paper.

5.1. Experimental testbed

Figure 7 depicts the nodes deployment for two different scenarios. The considered experimental testbed consisted of 7 HP 530 laptops (regular devices) and 10 Linksys WRT54GL routers (tiny devices). These nodes were equipped with both a wired Ethernet and a wireless IEEE 802.11b/g interface. The controller used to orchestrate the interactions between network nodes was a mainstream desktop PC.

The goal is showing the flexibility of REFRAHN to recreate both static (*Network A*) and mobile (*Network B*) scenarios. The topology and mobility of the considered scenarios have been configured accordingly and represent, as close as possible, the behavior of real ad hoc networks. For instance, the topology specified for *Network A* has been defined so that nodes A and F are 3 hops distant. As *Network B* represents a scenario addressed to people, speeds ranging from 0 to 3 m/s have been considered adequate for nodes mobility. The topology evolves dynamically (1 to 4 hops) as depicted in Figure 7. For the sake of simplicity, a snapshot of this evolution is shown every 120 seconds. In *Network B* the number of hops along the route formed by nodes A and F ranges from 1 to 4 hops. This decision has been taken considering that, in practice, real routes rarely expand beyond 4 hops [52]. Furthermore, it is to note that precomputed topology changes cause nodes involved in traffic forwarding to move away from the route, whereas new nodes appear as suitable candidates for routing. As far as these nodes are not identified yet by their new neighbors, the routing protocol requires some time to form a route again. This is the worst case a communication route may face and, by following this approach, routing protocols may be assessed under extreme conditions.

5.2. Execution profile configuration

Network A recreates the case of a Wireless Mesh Network (WMN) that provides low-cost Internet access to a residential district. The application traffic was defined in terms of synthetic UDP constant bit rate data flows of 200 Kbps, which is similar to the rates observed in real daily scenarios (*http://dashboard.open-mesh.com/overview2.php?id=Hillsdale*). *Network B* recreates a rapid Mobile Ad Hoc Network (MANET) deployment to assist the victims of a natural disaster. The workload defined for *Network B* consisted of synthetic UDP constant bit rate data flows of 2 Mbps, which was specially conceived to exchange a huge amount of information (e.g., real-time video streaming). Three different data flows are established for each experiment. Nevertheless, measurements are collected only from the data flow established between laptop nodes labeled A to F (see Figure 7), where the intrusiveness of monitoring probes is lower than in routers. The rest of data flows are exchanged between router nodes which are 1-hop neighbors of nodes A to F. This intends to emulate the real conditions of a wireless network, where the transmission, reception, and overhearing of packets are influenced by the traffic conditions imposed by nodes in the same radio range. All the faults implemented by REFRAHN have been used as faultload for this case study. In total, a set of 11 representative faults in the domain of ad hoc routing protocols has been introduced in the system: *signal attenuation*, *ambient noise*, *battery extenuation*, *traffic peak*, *olsrd tampering attack*, *olsrd replay attack*, *olsrd selective forwarding attack*, *olsrd jellyfish attack*, *flooding attack*, *olsrd neighbors saturation* and *olsrd sequence number replay*. It is worth noting that the *sink hole attack* was not injected alone but as a prerequisite to launch intrusion-based attacks. In our case, node M from Figure 7 will play the role of the injector node for all faults but *signal attenuation* and *ambient noise*. These particular faults are executed by every single node because they typically affect wider zones of the network. It is worth mentioning that routing protocol-dependent faults have been instantiated to *olsrd*, our target routing protocol in this case study.

5.3. Number and duration of experiments

Results were obtained from 1440 experiments, considering 2 network types × (1 Golden run + 11 fault models) × 4 target routing protocols (*v.0.4.10*, *v.0.5.6*, *v.0.6.0*, and *v.0.6.0+md5*), resulting in a total of 96 different experiments configurations which were executed 15 times to increase the statistical representativeness of results. Experiments lasted 9 minutes each, with 1 minute (empirically established) devoted to the warm up of the protocols, and 8 minutes for running the workload and collect the measurements. Table 6 summarizes the main parameters of the experimentation.

Table 6: Parameters considered in the experimentation.

Parameter	Value
Routing protocol	olsrd v.0.4.10 / v.0.5.6 / v.0.6.0 / v.0.6.0+md5
Network mobility	WMN (no mobility) / MANET (0-3 m/s)
# of nodes	17 (10 routers + 7 laptops)
Workload	WMN (200 Kbps UDP flows) / MANET (2 Mbps UDP flows)
Faultload	*signal attenuation / ambient noise / battery extenuation / traffic peak / olsrd tampering attack / olsrd replay attack / olsrd selective forwarding attack / olsrd jellyfish attack / flooding attack / olsrd neighbors saturation / olsrd sequence number replay.*
# of experiments	1440

5.4. Analysis of results

Figure 8 depicts the results of experiments. To enrich results, the percentage of packet loss has been categorized. We distinguish whether packets were lost due to the mobility of nodes or other causes, being then classified as short service interruptions ($< 15s$), and long service interruptions ($> 15s$). The same applies to delay. Although it is measured in ms, it is characterized according to its duration, i.e., percentage of normal delays ($< 400ms$), long delays ($400 - 1200ms$) and very long delays ($> 1200ms$).

5.4.1. Mobility analysis

When considering the effect of mobility in both scenarios, an immediate conclusion from the viewpoint of performance is that, regardless the *olsrd* version used, results obtained in (static) *Network A* are generally better than those obtained in (mobile) *Network B*. Nodes in *Network B* must rebuild network routes several times, which greatly affects those *olsrd* versions requiring more time to establish routes. According to the golden run phase (fault-free execution of the experiment), *v.0.4.10*, *v.0.5.6*, and *v.0.6.0* required an average time of $22.71s$, $37.40s$, and $40.71s$, respectively, to establish a 3-hops route. However, these two last versions may be considered as equivalent, as their standard deviation overlaps. Finally, *v.0.6.0+md5* needed $59.40s$ to establish a route, mainly due to the underlying handshake mechanism. The time devoted to establish a route has a significant impact on the service delivered. The longer it is, the lower the performance exhibited by the ad hoc network.

From the viewpoint of resilience, this study reflects that the mobility in *Network B* may result either beneficial or harmful depending on the considered routing protocol version. In general, results are very similar for those threats that affect the whole network, e.g. ambient faults like *signal attenuation* and *ambient noise*. In most cases, mobility assists routing protocols in leaving the area of influence of nodes originating the fault. For example, the resilience of non-secure protocol versions in presence of intrusion-based attacks improves as the attacker (node M) leaves the radio range of victim nodes. Consequently, despite the waste of time devoted to establishing new routes, the threat exposure rate decreased from 20 to 50 percentage points (which is proportional to the time node M was in the vicinity of the route). Then, the threat exposure gap between non-secure and secure protocol versions is reduced in presence of mobility for intrusion-based attacks (i.e., *selective forwarding attack*, *tampering attack*, and *tampering attack*). Regarding the fault effectiveness, it can be seen that threat exposure rates are not directly related to the impact caused on the network behavior. For example, in the case of *neighbors saturation*, the threat exposure time was only of $10s$, but the effects of the fault persisted in the route even when the threat was not longer active (3857.1% of the threat exposure rate for *Network A* and 1904.1% for *Network B*). Furthermore, *v.0.6.0+md5* presents the lowest fault effectiveness for all the considered threats, but for *battery extenuation* and *traffic peak*, due to its lower adaptiveness to establish new routes. If we focus on the effectiveness

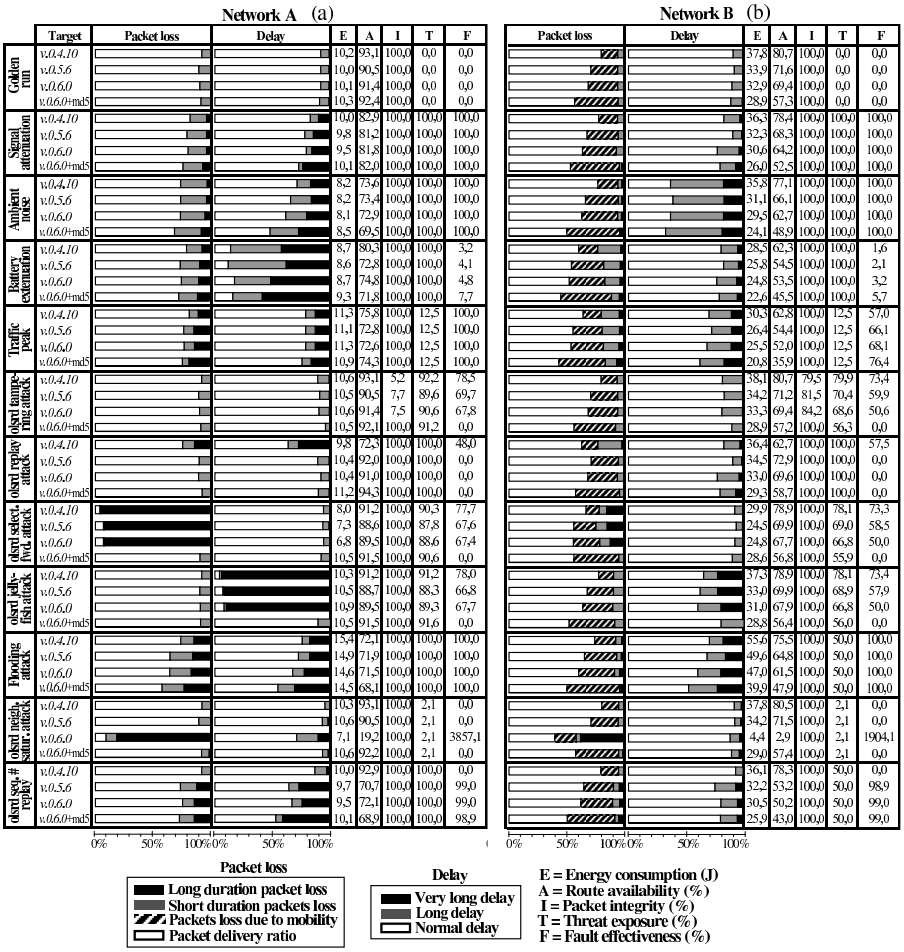

Figure 8: Metrics obtained during experimentation.

of the secure mechanism, it can be seen that intrusion-based attacks like *jellyfish*, *selective forwarding*, and *tampering attacks* were useless against *v.0.6.0+md5*. This is specially perceptible in the delay, packet loss, and integrity of *Network A*, which are a 90% worse than that of the golden run phase for non-secure protocol versions. Nevertheless, we can highlight that, although faults provoking long service interruptions in terms of packet loss and delay (*flooding attack*, *traffic peak*, *battery extenuation*, *ambient noise*, and *signal attenuation*) impact all the considered *olsrd* versions, *v.0.6.0+md5* is, curiously, the most affected one. This result shows that despite protecting routing with cryptography, it is not exempt of performance problems, which are in some cases, paradoxically more severe than for non-secure versions.

5.4.2. Limited resources analysis

The energy consumption is clearly related to the number of packets sent, received, and overheard by nodes, and the amount of information they contain. As including a

Table 7: Average energy consumption for NICs when nodes in Network B execute *olsrd v.0.4.10* in the golden run

Category	Energy (J)	Percentage
E_s (Sending energy)	13,13	35%
E_r (Receiving energy)	9,80	26%
E_o (Overhearing energy)	11,60	31%
Protocol overhead	3,28	9%
Total	37,79	100%

128-bits signature, routing protocol packets for *v.0.6.0+md5* are larger that for the rest of versions. Furthermore, the execution of the 3-way handshake increases the amount of packets exchanged between nodes. As a consequence, the energy consumed by the routing protocol traffic increases a 20%.

Table 7 shows an example of how the total energy consumption is computed in a golden run experiment. When estimating the average energy consumption of each routing protocol in both networks, the routing protocol traffic (also referred to as protocol overhead) represents less than 10% of the total energy consumed. However, this amount is important enough to pose a serious problem for nodes lifetime in Wireless Sensor Networks, thus compromising the availability of the network.

Regarding the application traffic, it must be noted that the energy consumed by nodes along the target route increased just by overhearing traffic from other data flows. According to our topology configuration in Figure 7, nodes in our target route are subjected to the overhearing effect of 2 additional data flows. In particular, 31% of the energy consumed is devoted to overhearing in the example of Table 7. However, this value increases up to 42% when computing the average for all the protocols.

It is also worth noting the higher energy consumed by *Network B*, obviously motivated by its heavier workload. As the best possible scenario is that where nodes consume the least, readers may misunderstand the results when analyzing them just from a strict energy viewpoint. For instance, some faults (like *ambient noise* or *battery extenuation*), decrease the energy consumed by nodes with respect to that consumed by the golden run phase. However, this energy saving is related to packet loss as can be deduced from correlating this information with availability (if packets are not flowing along the route, nodes do not consume any energy). Hence, although faults may benefit nodes by reducing their energy consumption, this cannot be really considered a benefit for the network, as faults affect the final service provided to the user. On the other hand, *flooding attack* and *traffic peak* are the only faults increasing the energy consumed by nodes. This increase is specially important for *flooding attack* (around 50%), thus highlighting again the importance of the overheard traffic.

5.4.3. Software bugs analysis

Results from Section 5.4 showed a significant packet loss in the network when subjecting *olsrd v.0.4.10* to the presence of a *replay attack* (around 20% higher with respect to the golden run phase). Nevertheless, the surprising result is that the rest of protocol versions were not affected at all. After studying the source code of the different versions in more detail, we found that *v.0.4.10* was the only one that does not implement a mechanism to reject packets whose sequence number has been already received and, thus, takes them into account to establish a (probably fake) route. *v.0.5.6* and *v.0.6.0* (as expected) reject packets with duplicated identifiers emitted by the injector node M, thus relying on other nodes to establish the target route and increasing the delay. Paradoxically, *olsrd v.0.4.10*

benefits from this vulnerability when the network is subjected to the presence of a *sequence number replay* fault. As the rest of the content of replayed packets is legitimate, *olsrd v.0.4.10* processes a wider amount of information with respect to the other protocol versions (that directly reject those packets), which involves an important improvement in the overall network behavior (see this effect on Figure 8).

The analysis of results obtained from Section 5.4 showed another interesting result. Surprisingly, the oldest version of *olsrd* (*v.0.4.10*) required the shortest time to establish a 3-hops route (22.71s), which represents from 15 to 18 seconds less than recent versions (*v.0.5.6* and *v.0.6.0*, respectively). Although this result improves the performance of *olsrd v.0.4.10*, the protocol is less robust against the occurrence of intrusion attacks. To study more in detail this phenomenon, Figure 9 presents the instantiation of the *sink hole attack* for *olsrd*. The injector node (node M) induces a propitious network topology to intrude the target route. To reach that goal, the injector node induces a possible routing link between target victim nodes (nodes A and C) by faking HELLO and Topology Control (TC) olsrd messages. Such messages enable nodes to determine optimal routes for their communications. Thus, the injector node forces its intrusion in the route by (maliciously) misusing the aforementioned messages. It starts injecting in the network HELLO messages declaring victim nodes as its neighbors. To obtain a symmetric link, the injector node needs victim nodes to generate TC messages acknowledging reciprocal links. To fake the generation of these messages at the victims side, the injector node forges fake TC messages announcing such links.

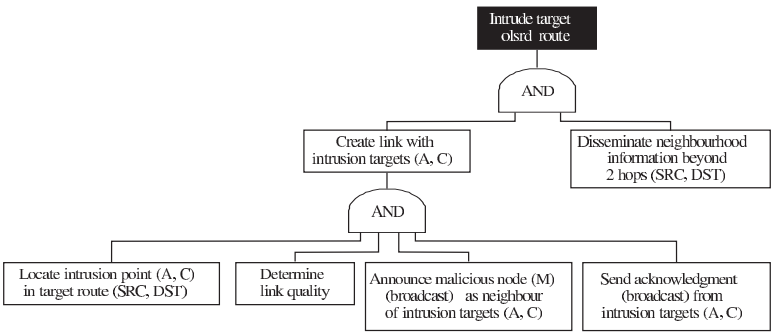

Figure 9: OLSR route intrusion procedure.

Experiments showed that, in all considered cases, omitting one of these basic actions prevented the malicious node from intruding the route but in one particular case: when acknowledgment from intrusion target was not broadcasted. Table 8 shows which was the content of routing tables of the nodes selected as the intrusion point (A and C) for *olsrd v.0.5.6* and *olsrd v.0.6.0* when omitting the acknowledgment from intrusion targets. It must be noted that the best possible value for the field *link quality* is 1.00, whereas INF (infinity) states that there is not a link between these nodes. In this case, although each common node creates a link with the malicious one, they are not aware of the link created by the other node. This is the behavior defined in the protocol specification.

Nevertheless, routing tables of target nodes for *olsrd v.0.4.10* (see Table 9) show very different results: the route A $\leftrightarrow$ B $\leftrightarrow$ C has been dismissed in favor of a new route A $\leftrightarrow$ M $\leftrightarrow$ C, since these new links present the best possible link quality. This procedure reveals a possible vulnerability, according to the protocol specification, in the implementation of this version.

In other words, *olsrd* version *v.0.4.10* is not fully OLSR compliant, while *olsrd v.0.5.6*

Table 8: Routing tables for *olsrd v.0.5.6* and *olsrd v.0.6.0*.

Node A			***Node C***		
DST	**Next**	**LQ**	**DST**	**Next**	**LQ**
A	M	1.00	C	M	1.00
C	M	INF	A	M	INF

DST = destination node, *LQ* = link quality

Table 9: Routing tables for *olsrd v.0.4.10*.

Nodes A and C					
Before intrusion			***After intrusion***		
DST	**Next**	**LQ**	**DST**	**Next**	**LQ**
A	B	2.34	A	M	1.00
B	A	1.68	M	A	1.00
B	C	1.57	M	C	1.00
C	B	1.52	C	M	1.00

DST = destination node, *LQ* = link quality

and *olsrd v.0.6.0* are. This can be considered as a vulnerability in protocol version *v.0.4.10* that can be exploited by attackers to intrude a route more easily (with a reduced set of actions). Yet, the side effect seems to be a reduction in the time required to create new communication links, which explains the better packet delivery ratio of this version with respect to the rest of considered versions (see Figure 8).

5.5. Discussion

The type of results obtained in this section has shown that, beyond the scale of the network deployment considered, there are certain conclusions that can be generalized. Thanks to REFRAHN, we have seen the impact that mobility may have on network deployments, the importance of resources consumed by network nodes, or the effect of software bugs in the expected behavior of routing protocols. In summary, we have shown the feasibility of REFRAHN to enable researchers and practitioners to gain knowledge about the characteristics of ad hoc routing protocol through realistic resilience evaluation.

6. Conclusions

This paper has explored existing gaps in the practical evaluation of ad hoc routing protocols, which are the most sensitive elements of ad hoc networks. In such a way, if the routing protocol fails, communications will be rarely possible beyond one hop. The research developed around these problems has led us to propose a novel experimental framework. Such framework, named REFRAHN (Resilience Evaluation FRamework for Ad Hoc Networks), defines a methodology and implements a tool to carry out the resilience evaluation of ad hoc routing protocols, while addressing essential issues such as (i) the definition of experiments that emphasize the need for recreating the dynamic characteristics of real ad hoc networks deployments, specially the mobility of nodes and the occurrence of faults; (ii) the execution of experiments, considering the use of real devices executing real (non-simulated) routing protocols; and (iii) the ulterior analysis of measurements to deduce a complete set of performance, resilience, and resources consumption metrics.

REFRAHN pays special attention to the practical aspects of the evaluation. With respect to controllability, REFRAHN is able to define and manage experiments campaigns

manipulating all type of parameters concerning nodes configuration, such as their spatial location, mobility pattern, the considered target routing protocol, the workload, and faultload, among other parameters. Furthermore, the execution of experiments is totally automated and can be remotely controlled. The degree of control achieved through the mobility emulation and the accurate fault injection, enables REFRAHN to execute repeatable experiments. This feature makes REFRAHN an interesting testbed to conduct reproducible fault injection experiments for ad hoc routing protocols in the domain of ad hoc networks. REFRAHN also provides a good level of observability. Our tool uses methods based on the analysis of experiments to collect packet traces from different types of nodes (both common and injector nodes), at different levels (software and hardware). Taking scalability issues into account, REFRAHN is able to deploy network topologies formed by tens of nodes in a reduced space. Furthermore, REFRAHN is (i) flexible enough to enable the injection of the most representative faults in the domain of ad hoc routing protocols, such as *ambient noise*, *flooding attacks*, or *intrusion-based attacks*, among others, and (ii) scalable enough to consider and include new types of faults in the future. This is a good balance between the need for scaling network deployments and the space limitations of most research laboratories. Finally, the low intrusiveness of REFRAHN is guaranteed not only because management operations between the core and network nodes use a different control network, but also because the tools installed in network nodes deploy light processes, which in no case saturate the nodes capacity.

On the one hand, REFRAHN enables the characterization of ad hoc routing protocols according to a number of performance and resilience attributes. This information could be used by developers to improve their protocol implementation and correct any vulnerabilities attackers may benefit from (as already seen in this paper), but also by network administrators to fine tune routing protocol parameters to meet their network requirements, or security solutions testers to evaluate the goodness of their proposals in presence of real faults to improve false positives and false negatives ratios. On the other hand, this initial evaluation process will enable the comparison, with selection purposes, of these routing protocols. This performance and resilience benchmarking procedure should define precise guidelines for interpreting obtained metrics in order to determine which is the most suitable routing protocol, implementation version, and configuration for a particular ad hoc network. In addition, obtaining concrete information about attacks is of great interest for designing new, and improving existing, intrusion detection and tolerance mechanisms for ad hoc networks. The validation of these mechanisms could also be performed by following the proposed approach. Finally, results obtained by this approach could also be used to animate ad hoc network models, and to enrich existing simulators, with actual data collected from experimentation.

Approaches aimed at experimentally balancing resilience, performance, and cost will gain in importance as ambient intelligence-based solutions, and thus ad hoc networks, become more common in our everyday lives.

Acknowledgements

This work was supported in part by the Spanish Project ARENES under Grant TIN2012-38308-C02-01 and in part by the Programa de Ayudas de Investigacin y Desarrollo through the Universitat Politcnica de Valncia, Valencia, Spain.

References

[1] Global Technological and Societal Trends, online: http://www.internet-of-things-research.eu/documents.htm, 2014.

[2] K. R. Chowdhury, I. F. Akyildiz, Crp: A routing protocol for cognitive radio ad hoc networks, IEEE Journal on Selected Areas in Communications 29 (2011) 794–804.

[3] A. Boukerche, B. Turgut, N. Aydin, M. Z. Ahmad, L. Bölöni, D. Turgut, Routing protocols in ad hoc networks: A survey, Computer Networks 55 (2011) 3032–3080.

[4] A. H. Mohsin, K. A. Bakar, A. Adekiigbe, K. Z. Ghafoor, A survey of energy-aware routing protocols in mobile ad-hoc networks: trends and challenges, Network Protocols and Algorithms 4 (2012) 82–107.

[5] J.-C. Laprie, Resilience for the scalability of dependability, in: Proceedings of the Fourth IEEE International Symposium on Network Computing and Applications, NCA '05, 2005, pp. 5–6.

[6] Y.-C. Hu, D. B. Johnson, A. Perrig, Sead: Secure efficient distance vector routing for mobile wireless ad hoc networks, Ad Hoc Networks 1 (2003) 175–192.

[7] C. Adjih, T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, D. Raffo, Securing the olsr protocol, in: Proceedings of Med-Hoc-Net, 2003, pp. 25–27.

[8] M. G. Zapata, Secure ad hoc on-demand distance vector routing, ACM SIGMOBILE Mobile Computing and Communications Review 6 (2002) 106–107.

[9] S. Buchegger, J.-Y. Le Boudec, Performance analysis of the confidant protocol, in: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing, ACM, 2002, pp. 226–236.

[10] S. Kurkowski, T. Camp, M. Colagrosso, Manet simulation studies: the incredibles, SIGMOBILE Mob. Comput. Commun. Rev. 9 (2005) 50–61.

[11] A. Jabbar, H. Narra, J. P. Sterbenz, An approach to quantifying resilience in mobile ad hoc networks, in: Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the, IEEE, 2011, pp. 140–147.

[12] Network Simulator, [Online]. Available: http://www.nsnam.org/, 2014.

[13] OPNET Simulator, [Online]. Available: http://www.opnet.com, 2014.

[14] GloMoSim, [Online]. Available: http://pcl.cs.ucla.edu/projects/glo-mosim/, 2014.

[15] S. Bouckaert, W. Vandenberghe, B. Jooris, I. Moerman, P. Demeester, The w-ilab.t testbed, in: Testbeds and Research Infrastructures. Development of Networks and Communities, Springer, 2011, pp. 145–154.

[16] Roofnet testbed, [Online]. Available: http://pdos.csail.mit.edu/roofnet, 2014.

[17] Floornet testbed, Floornet: A Wireless Multihop Testbed, [Online]. Available: http://http://floornet.org/, 2014.

[18] E. Nordstrom, P. Gunningberg, H. Lundgren, A testbed and methodology for experimental evaluation of wireless mobile ad hoc networks, in: Testbeds and Research Infrastructures for the Development of Networks and Communities, Italy, 2005, pp. 100–109.

[19] C. Mitchell, V. Munishwar, S. Singh, X. Wang, K. Gopalan, N. Abu-Ghazaleh, Testbed design and localization in mint-2: A miniaturized robotic platform for wireless protocol development and emulation, in: First International Communication Systems and Networks and Workshops (COMSNETS), 2009, pp. 1 –10.

[20] H. Lundgren, D. Lundberg, J. Nielsen, E. Nordström, C. Tschudin, A large-scale testbed for reproducible ad hoc protocol evaluations, in: WCNC'02: Proceedings of the IEEE Wireless Communications and Networking Conference, 2002, pp. 412–418.

[21] ORBIT project, The ORBIT radio grid emulator (ORBIT), [Online]. Available: http://www.orbit-lab.org/, 2014.

[22] The Carnegie Mellon University Wireless Emulator, [Online]. Available: http://www.cs.cmu.edu/ emulator/, 2014.

[23] J. Hortelano, M. Ncher, J.-C. Cano, C. Calafate, P. Manzoni, Castadiva: A Test-Bed Architecture for Mobile AD HOC Networks, in: IEEE Symposium on Personal, Indoor and Mobile Radio Communications, 2007, pp. 1–5.

[24] Mobility Emulator (MobiEmu), [Online]. Available: http://mobiemu.sourceforge.net/, 2014.

[25] Emulab - Network Emulation Testbed, [Online]. Available: http://www.emulab.net/, 2014.

[26] D. T. Stott, B. Floering, D. Burke, Z. Kalbarczpk, R. K. Iyer, Nftape: a framework for assessing dependability in distributed systems with lightweight fault injectors, in: Computer Performance and Dependability Symposium, 2000. IPDS 2000. Proceedings. IEEE International, IEEE, 2000, pp. 91–100.

[27] J. Carreira, H. Madeira, J. G. Silva, Xception: Software fault injection and monitoring in processor functional units, Dependable Computing and Fault Tolerant Systems 10 (1998) 245–266.

[28] J. H. Barton, E. W. Czeck, Z. Z. Segall, D. P. Siewiorek, Fault injection experiments using fiat, Computers, IEEE Transactions on 39 (1990) 575–582.

[29] M. Cinque, D. Cotroneo, C. Di Martino, S. Russo, A. Testa, Avr-inject: A tool for injecting faults in wireless sensor nodes, in: IEEE International Symposium on Parallel Distributed Processing (IPDPS) 2009, 2009, pp. 1–8.

[30] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Transactions on Dependable and Secure Computing 1 (2004) 11–33.

[31] Network Simulator 3, [Online]. Available: http://www.nsnam.org/, 2014.

[32] J. Friginal, D. de Andrés, J.-C. Ruiz, P. Gil, Towards benchmarking routing protocols in wireless mesh networks, Ad hoc networks 9 (2011) 1374?1388.

[33] J. Friginal, D. de Andrés, J.-C. Ruiz, P. Gil, Using performance, energy consumption, and resilience experimental measures to evaluate routing protocols for ad hoc networks, in: Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on, IEEE, 2011, pp. 139–146.

[34] J. Friginal, D. de Andrés, J.-C. Ruiz, P. Gil, Resilience-driven parameterisation of ad hoc routing protocols: Olsrd as a case study, in: Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on, IEEE, 2011, pp. 85–90.

[35] J.-K. Chen, C. Chen, R.-H. Jan, H.-H. Li, Expected link life time analysis in manet under manhattan grid mobility model, in: Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems, MSWiM '08, 2008, pp. 162–168.

[36] C. Bettstetter, H. Hartenstein, X. Pérez-Costa, Stochastic properties of the random waypoint mobility model, Wirel. Netw. 10 (2004) 555–567.

[37] M. McNett, G. M. Voelker, Access and mobility of wireless pda users, SIGMOBILE Mob. Comput. Commun. Rev. 9 (2005) 40–55.

[38] R. Natella, D. Cotroneo, J. A. Duraes, H. S. Madeira, On fault representativeness of software fault injection, IEEE Transactions on Software Engineering 99 (2011).

[39] P. Ning, K. Sun, How to Misuse AODV: A Case Study of Insider Attacks Against Mobile Ad-Hoc Routing Protocols, in: IEEE Information Assurance Workshop, 2003, pp. 60–67.

[40] A. Morais, A. Cavalli, E. Martins, A model-based attack injection approach for security validation, in: Proceedings of the 4th international conference on Security of information and networks, SIN '11, 2011, pp. 103–110.

[41] Y. an Huang, W. Lee, Attack analysis and detection for ad hoc routing protocols, in: In Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID, 2004, pp. 125–145.

[42] J.-S. Yeom, N. Wisitponghan, S. Panichpapiboon, O. K. Tonguz, A testbed emulator for cross-layer studies in mobile ad hoc wireless networks, in: International Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities (TridentCom), 2007, p. 10.

[43] D. Macedo, L. Correia, A. dos Santos, A. Loureiro, J. Nogueira, Evaluating fault tolerance aspects in routing protocols for wireless sensor networks, in: Fourth Annual Mediterranean Ad Hoc Networking Workshop, 2005, pp. 285–294.

[44] B. Wu, jianmin Chen, J. Wu, M. Cardei, A survey on attacks and countermeasures in mobile ad hoc networks, in: Wireless/Mobile networks security, Springer-Verlag, 2006, pp. 103–135.

[45] A. Wood, J. Stankovic, A taxonomy for denial-of-service attacks in wireless sensor networks, Chapter 32. CRC Press LLC, 2005.

[46] M.-C. Hsueh, T. K. Tsai, R. K. Iyer, Fault injection techniques and tools, IEEE Computer 30 (1997) 75–82.

[47] Nemesis tool, [Online]. Available: http://sourceforge.net/projects/nemesis/, 2014.

[48] Network Emulator (Netem), [Online]. Available: http://www.linuxfoundation.org, 2014.

[49] I. Aad, Impact of denial of service attacks on ad hoc networks, IEEE/ACM Trans. Netw. 16 (2008) 791–802.

[50] L. M. Feeney, An energy consumption model for performance analysis of routing protocols for mobile ad hoc networks, Mob. Netw. Appl. 6 (2001) 239–249.

[51] T. Clausen and P. Jacquet, Optimized Link State Routing Protocol(OLSR), RFC 3626 (2003).

[52] Y. Sun, I. Sheriff, E. M. Belding-Royer, K. C. Almeroth, An experimental study of multimedia traffic performance in mesh networks, in: Workshop on Wireless traffic measurements and modeling (WiTMeMo), 2005, pp. 25–30.