Improving The Fault Tolerance of Ad Hoc Routing Protocols using Aspect-oriented Programming

Antonio Bustos Rodríguez

Advisors:

Dr. Juan Carlos Ruiz García

Dr. David de Andrés Martínez

Mr. Jesús Friginal López

Master's thesis

Departamento de Informática de Sistemas y Computadores

Universitat Politècnica de València

September, 2012

Acknowledgements

First and foremost, I would like to thank to my advisors, Dr. Juan Carlos Ruiz García and Dr. David de Andrés Martínez from the Institute for the Applications of Advanced Information and Communication Technologies (Instituto de Aplicaciones de las Tecnologías de la Información y las Comunicaciones Avanzadas - ITACA) of the Technical University of Valencia (Universitat Politècnica de València - UPV), for their advice, encouragement, endless patience and constant support. Furthermore, I would like to express my deep-felt gratitude to my advisor Mr. Jesús Friginal López from the Fault-Tolerant Systems Group (GSTF) of the Technical University of Valencia (Universitat Politècnica de València - UPV) for his advice and offering invaluable assistance, support and guidance. Their suggestions, comments and remarks resulted extremely helpful in providing the additional guidance and expertise to the completion of this work. Finally, an honorable mention goes to my family and friends for their understandings and supports in completing this master thesis.

Abstract

Ad hoc networks are distributed networks consisting of wireless mobile nodes that can freely and dynamically self-organize into arbitrary and temporary topologies, through the operation of routing protocols. These networks allow people and devices to seamlessly interconnect rapidly in areas with no pre-existing communication infrastructure and with a low cost. Many studies show that ad hoc routing protocols are threatened by a variety of accidental and malicious faults, like neighbour saturation, which may affect any kind of ad hoc network, and ambient noise, which may impact all wireless networks in general. Therefore, developing and deploying fault tolerance strategies to mitigate the effect of such faults is essential for the practical use of this kind of networks. However, those fault tolerance mechanisms are usually embedded into the source code of routing protocols which causes that i) they must be rewritten and redeployed whenever a new version of a protocol is released, and ii) they must be completely redeveloped and adapted to new routing protocols. This master thesis explores the feasibility of using Aspect-Oriented Programming (AOP) to develop and deploy fault tolerance mechanisms suitable for a whole family of routing protocols, i.e. existing and future versions of a given protocol (OLSR in this case). Furthermore, a new methodology is proposed to extend these mechanisms to different families of proactive protocols (OLSR, B.A.T.M.A.N and Babel) using a new concept in AOP, the *meta-aspect*. The feasibility and effectiveness of the proposal is experimentally assessed, thus establishing a new method to improve the deployment, portability, and maintainability of fault tolerance mechanisms for ad hoc routing protocols and, therefore, the dependability of ad hoc networks.

Table of Content

List of Tables

List of Figures

Listings

Chapter 1

Introduction

Ad hoc networks are a recent networking paradigm. They allow for the rapid creation of spontaneous, low-cost, wireless data networks without using any pre-existing communications infrastructure. Consequently, ad hoc networks enable the deployment and use of communication networks in places where it was not possible (for geographical issues) or cost-effective (price or investment of time) to deploy a fixed infrastructure. Future wireless networks are expected to coexist with existing wired ones creating intelligent environments in which citizens can consume IT services naturally, for example, without having to worry about their current location. The concept of ubiquitous computing has evolved in recent years to a point where now it is feasible to consider the development of increasingly intelligent spaces. Generally, this means talking about smart homes, buildings or cities. ”Little things” around us, which is the seed for the development of a new industry that turns around the concept of *Internet of Things* (IoT), that enables IoT systems to establish multi hop communications between those devices willing to communicate in a transparent way from the user viewpoint. Illustration 1.1 represents a set of mobile devices able to integrate an ad hoc network.

This new network technology emerged from military and rescue mission application domains [1], and today focuses the attention of researches in a variety of domains, like Mobile Ad Hoc Networks (MANET), Wireless Mesh Network (WMN) and Wireless Sensor Network (WSN).

Other ad hoc networks are still under investigation, such as vehicular networks, commonly known

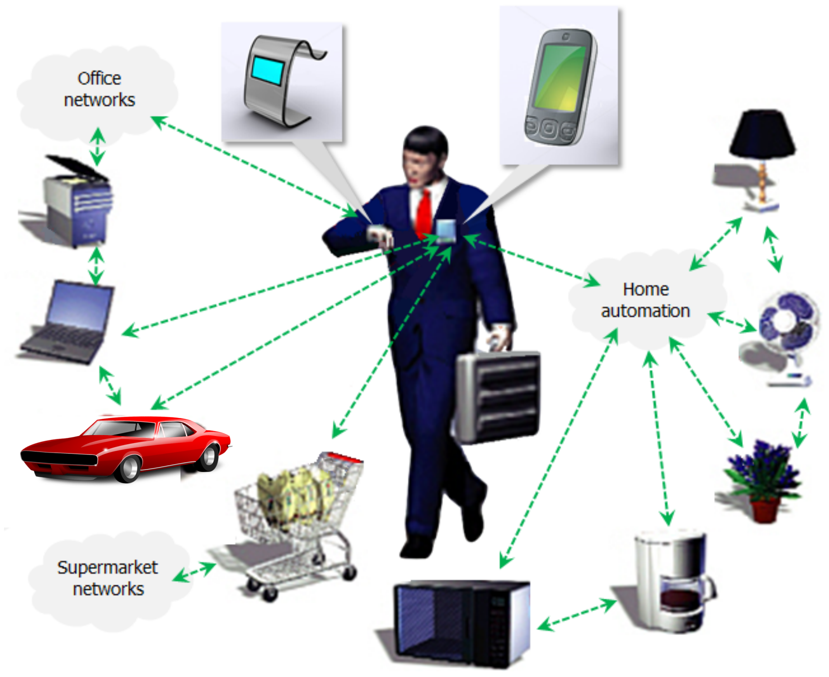

Figure 1.1: Devices integrated in an ad hoc network

as Vehicular Ad hoc Networks (VANET). The challenge is therefore to ensure that all these types of ad hoc networks offer not only a performance acceptable to its users, but also a sufficient degree of dependability so that they can use them reliably and securely.

Ad hoc networks are formed by nodes and wireless communication links. In more detail, a node is a device capable to transmit data coming from applications, but also related to the current state of the network. Communication links serve as the union of two nodes who can communicate directly between them. The set of interconnected nodes, form the ad hoc network. Each node is capable to determine which are the best links to transmit a data flow until its destination. Routing protocols are responsible to work in this layer to orchestrate how network nodes must collaborate to establish communication.

Specially, if taking into account the limited wireless transmission power, computation and available memory of current devices, and despite the promising future of ad hoc networks, there are still important challenges to address before their mass exploitation in IoT systems. Both, accidental faults and attacks are serious threats to ad hoc networks. Today, and despite the efforts done in the community of ad hoc networks to develop the first implementations of routing protocols, many issues concerning their dependability and security remain unexplored, thus limiting the practical interest and exploitation of this type of networks in real life products.

In this work, our attention goes to the exploration of two common situations in ad hoc networks that can eventually provoke failures in nodes: the network saturation and the ambient noise problems. The neighbourgh saturation problems appears when multiple nodes concentrates in a small geographical area. As far as nodes can freely move within a given scenario, it is possible to find zones in the network with a high density of neighbours in any moment. This situation can be seen as a great opportunity for ad hoc networks to create multiple communication links among nodes, thus increasing the available effective network bandwidth and decreasing existing communication. The massive exchange of routing information between the nodes can degrade performance in nodes.

To a point where they can even fail by hanging or aborting their regular computation. In this document we will show that surprisingly this problem exists in most recent implementations of ad hoc routing protocols and we will propose a solution to face the type of errors derived from this particular situation.

As far as ambient noise is concerned, this document evaluates how this phenomenon degrades the QoS of an ad hoc network thus provoking long convergence links, and inefficient routing. As it will be argued, the effect is similar to the one provoked by a flooding attack. In fact, ambient noise is the result of increasingly saturated radio frequency spectrum with other types of wireless communications where ad hoc networks operates. This obviously results in communication interferences leading to excessive loss rates or packet delays.

As in the case of neighbourgh saturation, ambient noise and flooding attacks will be studied from a dependability perspective, thus leading to detect which are, and where are located, dependability bottlenecks and how can they be mitigated with new performance and battery-effective fault tolerance mechanisms.

It is worth noting that there are several fault tolerance techniques to help ad hoc networks to improve their mechanisms to mitigate the effects of accidental faults or malicious attacks. However, to date, most of them have just been tested under simulation, thus obviating the practical impairments of such mechanisms in real implementations (e.g. memory or CPU consumption).

The reader must understand that fault tolerance mechanisms can be useful to mitigate, to some extent the effects of fault and attacks on the network. In the practical, each fault tolerance mechanism is designed and implemented for a particular routing protocol. This fact leads to one of the problems when implementing fault tolerance techniques is the wide variety of existing routing protocols, which limits the portability of the implementations from one routing protocol to another. *OLSR*, *B.A.T.M.A.N.*, *Babel* are just an example of some of the current routing protocols that can be found in the bibliography.

Aspect-Oriented Programming (AOP) [15] is a technology that facilitates the separation of concerns. The paradigm of aspect-oriented programming has been successfully used in software engineering to improve deployment of functionalities in several prototypes. However, this paradigm has been rarely used in the design of fault tolerance mechanisms, and this is the first time it applies to for ad hoc routing protocols. In this context the goal is to enable the portability and maintainability of the proposed fault tolerance solutions. This is a way of easing the role of system administrator, when integrating new components regardless of ad hoc routing protocol without worry about notions of reliability and safety.

In summary, the ambitions of this research are twofold. On the one hand, it pursuits the design and provision of practical mechanisms for tolerating situations that ad hoc networks may face in their everyday use. Since performing a complete analysis of all the known threats affecting to this family of networks is unfeasible, our attention will go to two basic and common potential problems: network Saturation and ambient noise. On the other hand, the portability and reuse of the fault tolerance mechanisms finally proposed will be improved through the use of AOP, a design and implementation approach used in the domain of highly modular software.

The outline for the remainder of the this master's thesis is as follows. Chapter 2 describes the current state of research related to the ad hoc networks, their problems and already existing fault tolerance mechanisms. Chapter 3 introduces the concept of aspect-oriented programming paradigm, as a way to implement fault tolerance mechanisms. Chapter 4 and 5 present the study and treatment of the two problems previously mentioned. Finally, Chapter 6 discuses the main conclusions from this work.

Chapter 2

Ad hoc networks: background and dependability challenges

This chapter provides an overall view of what is an ad hoc network and it enumerates, among most common and well-known threats (i.e. accidental and malicious faults) affecting their nominal behaviour, the ones considered for this research.

Ad hoc networks are infrastructureless self-configuring networks that offer a quick, easy, and low-cost solution to provide multi-hop communication networks. The origin of these networks is based in the growing reliance of our society on mobile computing, which motivates the increasing interest of incorporating new networking capabilities into modern devices. Ad hoc networks enable the seamless creation of spontaneous data networks without the need of any pre-existing communication infrastructure. As an example, data flows between nodes intercommunicating in an ad hoc network appears in figure 2.1. Although current ad hoc networks typically offer lower performance and dependability than their wired counterparts, consumers living off-the-grid are increasingly demanding access to the same types of services and quality access than those living in suburban areas. This is the main reason leading why ad hoc networks, and their capabilities to easily and rapidly deploy low cost networks, have to arouse the interest of industry. Currently, companies like Meraki (http://meraki.com) and

TerraNet (http://terranet.se), have focused their efforts in providing low-cost internet access, without dependency on base stations, antenna towers or other legacy hardware.

Omnipresent wireless technologies like (IEEE 802.11) Wi-Fi, have facilitated that ad hoc have become practically feasible. Today most wireless devices, such as laptops, smartphones, tablets, and even wireless routers, access points and other more specific gadgets, support the IEEE 802.11 communication standard. This situation easies the obtention of low-cost material for the deployment of ad hoc and spreads possibility of ad hoc creation anywhere.

Routing protocols are cornerstones of ad hoc networks. They are responsible for the efficient creation of multi-hop wireless communication routes among distant nodes. Previously to the study of the different threats affecting the nominal behaviour of routing protocols, next subsection establishes the behavioural and structural model of an ad hoc routing protocol that will be considered allthrough the rest of this document.

2.1 Routing Protocols

Routing protocols are basic bricks for ad hoc networks. Routing protocols orchestrates how network nodes must collaborate to create communication. They provide mechanisms for discovering, establishing and maintaining communication links among neighbour nodes in an ad hoc network.

The network is created through the exchange of control routing packets that are broadcasted by each network node to know topology state. Control routing packets are used by routing protocols to i) control how information is routed among network nodes, and ii) guarantee communication despite mobility or failure of nodes. In order to promote the dissemination of routing information, nodes must cooperate and rely on each other to provide routing services.

To allow the creation of communication links between any two network nodes, nodes can function both as end-hosts within each nodes radio range, and as intermediate routers for other network nodes far apart. Due to their key role, the performance and robustness of an ad hoc network will greatly depend on that exhibited by the selected routing protocol.

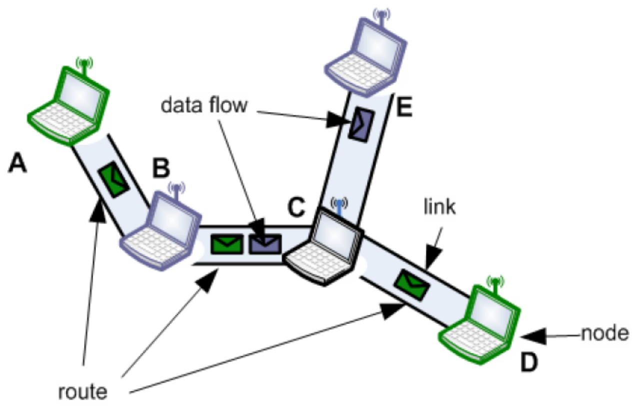

Figure 2.1: Example basic ad hoc network

Depending on the strategy considered to compute such routes, routing protocols can be link-state or distance vector. Link-state routing requires each node to maintain at least a partial topology map of the whole network. Conversely, in distance vector protocols, a given node only knows their 1-hop neighbours, but not the rest of the route.

According to their proactiveness, routing protocols can establish network routes following a reactive or a proactive approach. Reactive routing protocols find a route on demand by flooding the network with Route Request packets. This second type of protocols are typically used in application domains where battery of nodes must be preserved as much as possible, like sensor networks. Conversely, proactive routing protocols maintain fresh lists of destinations and their routes by periodically distributing routing tables using the so-called hello and topology control packets. These protocols are those more widely used in the industry due to the higher communication bandwidths they provide and their intrinsic ability to cope with topology changes derived from mobility of nodes. As a result, this document will place the spotlight on proactive routing protocols. Despite the big number of existing proactive proposals for routing information in ad hoc networks, few of them has been really implemented, and are ready to run, in a real device. Among them, OLSR, B.A.T.M.A.N. and Babel are the most well-known ones. For the sake of making this document more readable, all the necessary details relating to the different protocols under study will be introduced in the following chapters, when the information is really necessary for the understanding of each formulated proposal.

Let us however introduce here a general view of the internal architecture of a proactive ad hoc protocol (see Figure 2.2). As can be seen, the first important thing is to establish a clear difference between the packets generated at the applicative layer (*applicative packets*) from those that are generated by the routing protocol (*routing packets*) in order to keep the network nodes interconnected. Proactive routing protocols rely on a *packet generator* which is in charge of periodically creating and sending routing packets.

A core element of any proactive routing protocol is its *routing manager*. It is the responsible for handling (search, add, update and remove) existing and new routes attending to the different situations of the network. When computing a route from a source to a given destination node, it is possible to distinguish three different situations. Obviously the worst case consists in not finding any route, in such case, the communication will not be possible. Other situation may be finding one single route, but in case one node in the route fails, the communication will be compromised. Finally, the best case consists in discovering different alternative routes. In this situation, the *routing decision maker* will rank the different alternative routes aiming at selecting which is the best option, and which ones remain as backup routes in case the best one fails.

The *task scheduler* manages all the internal protocol timers, which are used to periodically trigger the broadcasting of control routing messages to the network and the expiration of a link. Valid incoming routing packets containing routing information are processed and stored by the *packet processor* in the protocol internal routing information repository. The content of incoming packets and the information stored in the routing repository is different in distance vector and link state protocols. However, in both cases, when a change in the state of a link is discovered (i.e, a new link has been created, updated or removed), the *route proxy* reflects such change in the *routing tables* of the node, which are located in the network stack which is not part of the routing protocol.

Once introduced the concept of proactive routing protocol in a general way, let the document explore the threats to the nominal behaviour of this type of protocols that are considered, among all the existing ones, in the present research.

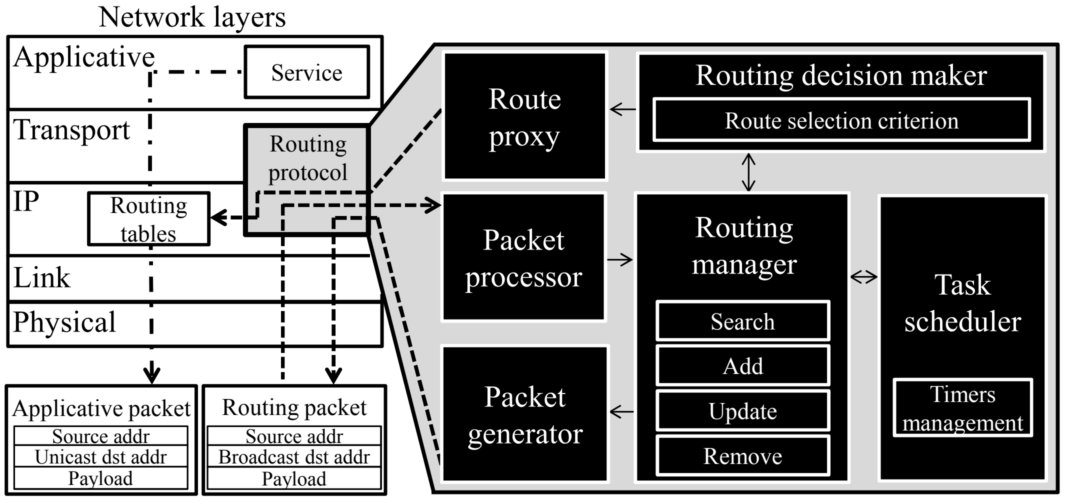

Figure 2.2: Example of events carried out by the routing protocol

2.2 Threats in ad hoc networks

Although the wireless nature of ad hoc communications promotes the mobility of nodes, and despite ad hoc routing protocols are specifically designed to handle mobility, a number of well-known problems still remains without a solution. Most of the research efforts so far, have been focused on improving the security of protocols using cryptographic primitives addressed to preserve the integrity of communications. Improvements achieved in this direction, have obviated accidental problems that can also negatively impact their resilience.

The main strengths of ad hoc networks may also become their main weaknesses when facing perturbations, like ambient noise, signal attenuation, neighbour saturation or flooding attacks, among others. Such perturbations may distort routing information, partition the network, induce a certain traffic overload, or even cause retransmissions or inefficient routing. As a result, the confident use of these networks strongly relies in our ability to produce and evaluate dependable and secure ad hoc routing protocols.

Among all the aforementioned threats in table 2.1, ambient noise is a common problem to all wireless networks, while neighbour saturation is particular to ad hoc networks. At the same time coping with these faults is basic for any ad hoc routing protocol since they are issue from accidental situations

Table 2.1: Attacks and fault-load in ad hoc networks

Signal attenuation	The signal quality depends on different aspects like the distance between nodes and the signal power. As our testbed emulates the location of the nodes, the effect of an increasing distance among nodes of Network B is emulated by inducing a random packet loss of 3% in every network node for the duration of experiments, thanks to the netem tool
Ambient noise	Since the wireless medium is simultaneously shared by multiple devices, communications are susceptible to suffer interferences from ambient noise. In this case, a packet corruption rate of 5% for every network node emulates the ambient noise
Battery extenuation	Ad hoc networks nodes, and especially those used on Wireless Sensor Networks (WSN), are characterised by their limited resources. They integrate short-duration batteries which limit to some extent the nodes lifetime. This problem can be emulated by disabling the nodes wireless interface card (NIC). In this case study, malicious node is switched off for the whole duration of the experiment
Traffic peak	Non-predicted peaks in the service demand may exhaust the local resources of nodes and result in network congestions. This fault is injected by increasing the packet sending rate of a source node up to 18 Mbps (the saturation point of the network) for the first 60 seconds of experimentation. This emulates the case where 90 users (in Network A) and 9 users (in Network B) share the same route to exchange data
Tampering attack	This attack violates the principle of data integrity by modifying the content of certain packets (workload packets in our case). As most intrusion-based attacks, it first requires a successful sink hole attack (locate a target data flow and exchange faked routing messages with victim nodes to intrude the route). After that, the payload of incoming target packets is changed before relaying them to their destination.
Replay attack	The attacker overhears the network traffic to (i) capture and (ii) reproduce these packets when they have already expired. In our case, routing protocols packets will be captured during the whole experimentation time, and will be replayed 30 seconds later (partially supported by tcpreplay, http://tcpreplay.synfin.net). This behaviour makes difficult the maintenance of legitimate routes.
Selective forwarding attack	This attack can only take place after a successful sink hole attack. After that, the malicious node drops all those packets belonging to a target data flow.
Jellyfish attack	This attack requires a successful sink hole attack. After that, the malicious node delays all data packets for 2 seconds.
Flooding attack	Nodes must overhear all messages in their radio range to determine whether they are the addressee. As a result, an attacker can saturate the medium with broadcasting storm just to keep nodes busy, consuming their resources and preventing communication. In our case, the attacker will send 18 Mbps of broadcast traffic (the saturation point of the network) during all the experimentation.
Neighbours saturation	Routing protocols store the topology information in internal routing tables, which grow proportionally to the amount of links stored. In this way, large routing tables may result difficult to manage. This problem is emulated by sending a burst of fake routing packets announcing 400 new links between nodes, at the beginning of the experimentation, in the radio range of malicious node.
Sequence number replay	Nodes may start sending routing protocols packets without updating its identification sequence number. This fault is emulated, in our case, by manipulating the generation of packet sequence numbers in malicious node, for the duration of the experiments.

that any ad hoc network may face during its lifetime. In other words, the absence of consideration of these basic threat may limit the usefulness of any ad hoc routing protocol in practice. This why our purpose in the present research is limited to the consideration of neighbour saturation and ambient noise.

Let's explain in more detail which are these threats.

2.2.1 Neighbour saturation

Considering redundant candidate nodes to automatically and transparently replace any intermediate node in case of a failure improves the dependability of ad hoc networks, since network availability increases as the number of communication links does. Consequently, the high density of nodes in the network is typically a very welcome situation for the network. Additionally, as many studies conclude, increasing the number of nodes reduces the number of hops between source and destination nodes [12], which can potentially improve the performance of the network.

However, as far as the network connectivity depends on the continuous exchange of routing packets, an extremely high density of nodes could overflow the nodes' capacity to manage a massive reception of information and may be the source of important interferences affecting the available network bandwidth, thus defying the capacity of network nodes for managing such avalanche of information [13].

Neighbour saturation is a well-known problem rarely addressed in practice, probably due to the difficulties to recreate the conditions of this problem in current test-beds. As far as nodes can freely move within a given scenario, it is possible to find zones in the network with a high density of neighbours in any moment. Some previous works have addressed this problem, particularly in the field of sensor networks, which are specially sensitive to the extenuation of resources [20]. Regardless whether originated accidentally (due to a wrong distribution of nodes) or maliciously (if an attacker exploits a propitious situation to deploy packet flooding attacks), it could (i) increase the resources consumption of nodes (CPU, memory, battery, etc), and (ii) the packet loss and delay of communications, thus resulting counter-productive to the network behaviour.

Some previous works have addressed this problem, particularly in the field of sensor networks, which are specially sensitive to the extenuation of resources [20]. Nevertheless, apart from the fact that most studies are based on simulation, they limit their scope to the analysis of reactive routing protocols which establish routes on demand, like AODV [4]. To date, to the best of the authors' knowledge, no study has been oriented towards studying the effect of neighbour saturation on real routing protocols executed by real devices.

The real question is to what extent this situation can degrade the dependability of a proactive ad hoc routing protocol. In case of a significant degradation level, a second question must be addressed. It refers to how to cope with this issue, i.e. which type of fault tolerance mechanisms will be more suitable and how should it be deployed on the actual implementation of the protocol. These are the concrete issues that will be addressed in the third chapter of this document.

2.2.2 Ambient noise

On the last decades, wireless communications have gained importance in our daily life. As the number of wireless networks increases, the environment that these networks share becomes increasingly saturated with signals from a wide variety of sources. As a result, an increasing number of transmissions interact and interfere with one another and cause unforeseen problems for communication protocols. This phenomena is typically known in the bibliography as background traffic or *ambient noise* [9].

The existence of *ambient noise*, and its negative effect in wireless mesh networks, is something indisputable. Since the industrial, scientific and medical (ISM) radio bands are unlicensed, many different types of equipment may use (cordless phones, cell phones or radio-frequency-based remote controllers typically supporting e.g. the IEEE 802.11x and the IEEE 802.15.x standards) or generate noise (e.g., microwave ovens and other electrical devices) in these frequencies, either accidentally or deliberately, that may end up increasing the amount of lost or dropped packets. Such packet dropping originated by ambient noise is one of the problems that might lead wireless networks in general, and ad hoc networks in particular, to suffer from link removals. This event is the most critical from the viewpoint of network connectivity. Link removal becomes potentially probable if the links maintained by the routing protocol stop being updated. This typically happens when there is any persistent problem

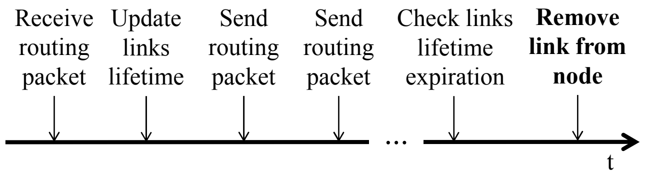

Figure 2.3: Example of events carried out by the routing protocol to remove link from node

affecting the reception of routing packets, for instance ambient noise, thus causing the link lifetime expiration (see Figure 2.3). Unfortunately, this effect may increase the risk of network partitioning specially if there is just one single route available between source and destination, consequently isolating certain regions of the network.

To face the *ambient noise* threat, keeping links active when link quality starts to degrade is a major goal to avoid topology control disable links affectted. Since link quality is basically computed by each node according to the amount of routing information received from its neighbourhood: the higher this reception rate the better.

The reaction speed exhibited by the routing protocols against noise is also an aspect that must be attentively studied. Overly agile protocol reactions may lead to route flapping [18] and must be avoided. In fact, they may increase the network overhead by flooding the route repair control messages without gaining much throughput. One possible way to counter this effect is to use flap damping. The goal is to limit the global impact of unstable routes by temporarily suppressing routes with rapid changes over short time periods. However this technique may cause persistent oscillation in the network due to the adverse interactions between flap damping and route convergence [8]. On the other hand, if the reaction of the protocol against noise is too slow it may entail the loss of existing communication links in the network. The main consequence is the activation of the self-configuration capabilities provided by the routing protocols to establish new communication routes among affected nodes. When many links result affected, the convergence time increases, which reduces the network stability and availability [7]. Shorter convergence time also means smaller resource usage. It is thus suitable to keep communication links alive as long as possible in case of ambient noise.

As far as ambient noise is concerned, the challenging questions addressed in this research are: how to cope with ambient noise in order to mitigate its impact on routing protocols, and how to design and implement the necessary mechanisms to enable their reuse in multiple and diverse protocols implementations.

2.2.3 Dependability challenges

Beyond the particular threats covered by our research, it must be noted that one of the biggest challenges in the design and implementation of fault tolerance mechanisms, is the promotion of their automatic reuse in different target components reuse. Typically, in ad hoc routing protocol context, mechanisms are usually implemented to be applied in a specific protocol which creates a dependency of the implementation to the protocol. The abstraction of the parts of a mechanism allows achieving the separation between functional protocol mechanisms (those specifying how the protocol must behave) from non-functional ones (introducing fault tolerance in the protocol). Separation of concerns allows the reuse and portability of a mechanism in different ad hoc protocols and facilitates the maintainability and deployment of the implementation.

Although the benefits of portability and reusability of fault tolerance seems quite intuitive and understandable, the need of automation in the deployment of related mechanisms is not so obvious. Basically, the idea is to avoid the intervention of developers with limited dependability skills in the design and implementation of fault tolerance mechanisms. This asks for a clear separation of concerns and requires the provision of means to automate the deployment, and adapt the implementation, of fault tolerance mechanisms to the particular features of each particular target, in our case protocol.

In next chapter, aspect oriented programming is defined as a paradigm for supporting the deployment, maintainability and implementation of fault tolerance mechanisms. This fundamental technology for our work has no relation with ad hoc networking and this is why it is introduced in a separated chapter, despite the fact that it is part of the background required for understanding the contributions included in the reported investigation.

Chapter 3

Aspect Oriented Programming

Aspect-Oriented Programming (AOP) is a paradigm that promotes a separation in the design and implementation of non-functional and functional mechanisms, thus enabling the reuse of non-functional mechanisms (aspects) on different solutions through the separation of concerns [15]. The paradigm of aspect-oriented programming has been widely used in software engineering to improve deployment of functionality in several prototypes [11].

Aspect oriented programming enables modularization of crosscutting concern, concern separation in imperative programs, creation of highly customizable software, enhance development of software product lines. Main functionality is maintainability of external mechanisms, from the time external mechanisms remain separate of main code.

AOP has been implemented to several programming languages, specially those applying the notion of ”class” like Java, with AspectJ or C++, with AspectC++. Their capability to create templates and inherit functionalities eases the application of AOP. However, despite not considering the notion of ”class”, languages (like C), can also benefit from the application of this paradigm. AspectJ is a simple and practical extension of aspect oriented programming in Java language to resolve the cross-cutting between classes and aspects. The C programmming language has several available implementations:

AspectC++ [1], XWeaver project [2], FeatureC++ [3], AspectC [4], AspeCt-oriented C [5], Apiscere [6],...

3.1 Basic elements of AOP

In AOP appears four important definitions: *advices*, *pointcut*,*joinpoint* and *weaver*. Instead of writing the code for a mechanism as part of the target program functions, it can be written as independent code segments called *advices*. Along with an advice, a *pointcut* is declared stating when in the target program the advice code should be executed. An accessible point during the target program execution where advices can be inserted is known as a *joinpoint*. The process of combining advices with the target program sources is called weaving and is done by a compiler known as the AOP language *weaver*. When an advice is inserted, the weaver handles the connection between the advice and the target program. Obviously, the target program source conditions the AOP language compiler, that must be written in the same language. It must be noted that despite the generic presentation of all the above concepts, the specification of advices, the level of expressiveness of joinpoints, and the way they are weaved to programs, vary from one programming language to another and from one AOP compiler to another. An example of previous four elements are described in 3.1 illustration.

Listing 3.2 contains a simple example of the application of AOP in code written in C language. To show the functionality of aspects, there are two simple advices written in listing 3.1 inside the aspect code. Advice 1 contains a piece of code executed before `int main()` function. Advice 2 contains a piece of code similar to the previous, but this piece is executed after `int main()`. The aspect with the two advices, is weaved with the code in the compile time. Output of the execution of main code weaved with the aspect appears in listing 3.3.

A basic example of aspect oriented programming is tracing specific calls of a function in execution

[1]http://www.aspectc.org/
[2]http://www.pnp-software.com/XWeaver/
[3]http://wwwiti.cs.uni-magdeburg.de/iti_db/fcc/
[4]http://www.cs.ubc.ca/labs/spl/projects/aspectc.html
[5]https://sites.google.com/a/gapp.msrg.utoronto.ca/aspectc/
[6]http://mcis.polymtl.ca/ bram/aspicere/index.html

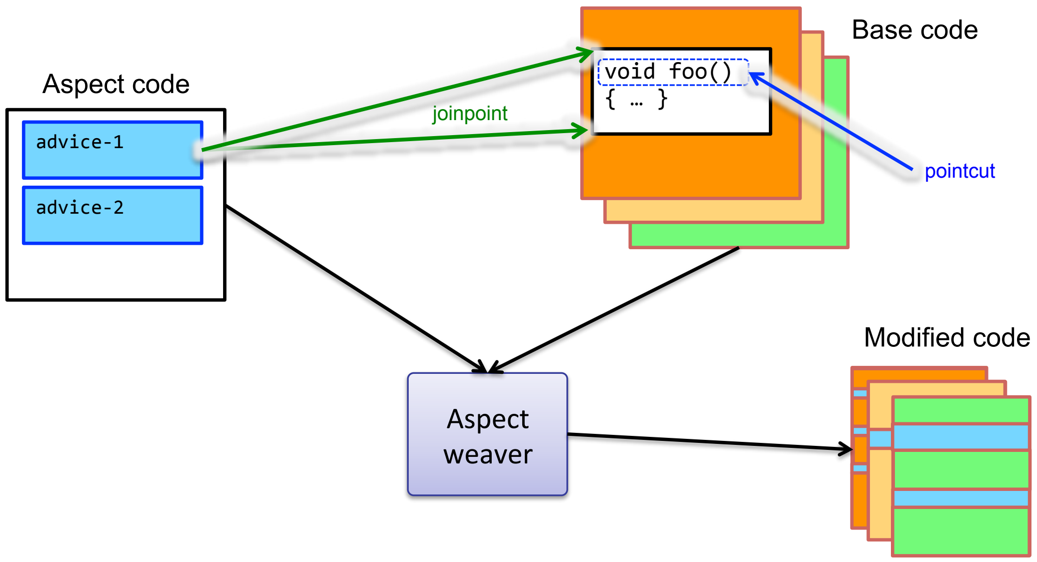

Figure 3.1: Conceptual scheme of the aspect-oriented programming paradigm

```
/* Advice 1 */
before(): execution(int main()) {
    printf("Hello ");
}

/* Advice 2 */
after(): execution(int main()) {
    printf(" from AOP example ! \n ");
}
```

Listing 3.1: Aspect code example with two advices defined

```
int main() {
    printf("world");
}
```

Listing 3.2: Main code example

```
Hello world from AOP example !
```

Listing 3.3: Console output of aspect execution

```
before(): call($ $(...)) {
    printf("Calling a function \n ");
}
```

Listing 3.4: Advice trace calls of any function

time. Declaring an advice of the aspect consists printing message with desired information and pointcut is set up before target function is called. A collection of messages will be printed when application executes. After execution, processing information collected improves knowledge about behaviour application, as a different way to debug an application when a problem appear. Advice to trace any call is exhibited in listing 3.4. With the character ”$” is possible to match any type identifier or any continuous length string and basic example of listing 3.4 can be improved to display useful information, for instance, caller function name of the function of the matched joinpoint, string representation of the function return type, source file name of the matched joinpoint,...

Sometimes it is necessary to log information of critical operations, for example, in a banking application, transfer money. Instead to modify validated code (security considerations) of transfer money operation, a simple aspect can log information after transfer money operation executes successfully. This aspect consists in print function to write simple message in a log file, without override original code of transfer money operation.

3.2 AOP in ad hoc networks

Despite being rarely considered, the use of AOP in ad hoc networks opens a door to ease the design and improve the maintainability of the fault tolerance mechanisms of ad hoc routing protocols. However, there is still much work to be done in this subject. Cuppens investigates in paper [6] about using aspect-oriented programming to study availability issues in proactive routing protocols of mobile adhoc networks (MANET). However, studying the practical impairments behind this theorist approach could be interesting to complement this interesting research.

The present master’s thesis makes a step forward to address not only the design of fault tolerance mechanisms based on AOP, but also its implementation and the issues in terms of intrusiveness that limit, to date, their use in real implementations of ad hoc routing protocols.

In next chapter, we introduce our first mechanism to mitigate concrete problem in ad hoc networks, neighbour saturation. OLSR is sensitive to be affected by this pathology, so we will explain in more detail how the problem affects to this routing protocol. An AOP approach will be used to design and implement this mechanism.

Chapter 5 studies the effects of ambient noise in proactive ad hoc routing protocols, and how to mitigate them generically. In a more sophisticated level, generic fault tolerance mechanism is designed and implemented with aspect-oriented programming to be able to deploy in several ad hoc routing protocols.

Chapter 4

Studying the neighbourgh saturation threat

The previous chapters have established that our interest goes to proactive routing protocols and the use of aspect oriented programming, from now on AOP, in order to develop mechanisms able to tolerate the various fault that may threat the nominal behaviour of such protocols. As already mentioned, our goal is two-fold. On one hand, we will show how to develop a fault tolerance mechanisms suitable for use in a family of routing protocols. On the other hand, we will introduce a methodology to enlarge the spectrum of application of the approach to different types of proactive protocols.

The variety of the problems that can support the illustration of the use of AOP for fault tolerance in ad hoc networks is very large, as already describe in Chapter 2. But, for the sake of conciseness, our purpose in the present research will be limited to the problems of neighbour saturation and ambient noise.

More precisely, this chapter defines an aspect suitable to mitigate the negative effects that situations of neighbour saturation can have over a family of proactive routing protocols. The selected target is the one with the largest open source community in the world, named OLSR.

The chapter structures as follows. Section 4.1 introduces the main characteristics of OLSR and the experimental setup deployed for the study of its behaviour in situations of big neighbour densities. This study is carried out in subsection 4.1.2 where the pathology exhibited by the various versions of the protocol under study is taken into consideration. Then, section 4.2 designs and implements a fault tolerance aspect suitable to cope with the identified pathology. Finally, section 4.3 assesses the level of effectiveness of the aspect once deployed over the various implementations of the various OLSR versions under study and section 4.4 presents conclusions.

4.1 Case study

Optimized Link State Routing (OLSR) [23] is the most well-known link state protocol and one of the most widely-used proactive routing protocols nowadays. OLSR employs an optimised flooding mechanism, where only special nodes called Multi-Point Relay (MPR) are responsible for broadcasting the routing information along the network. Although the initial specification of OLSR (RFC 3626) established route computation using hop-count as metric, current specification OLSRv2 promote the use of link quality extensions, like ETX.

Let us imagine an actual scenario similar to those that can be found in smart museums [2], where the problem of neighbour saturation may happen. Visitors' audio guides or mobile devices integrate an ad hoc network to receive information related to the piece of the exhibition they are looking at. In such cases, the high concentration of nodes in a particular area, far from being a benefit, may result counter-productive for the network behaviour.

The popular implementation of OLSR, *olsrd* (available at *http://www.olsr.org*), was considered as the experimental routing protocol target, concretely last steady version v.0.6.0 from 2010. For four years, this protocol has been developed by the currently most active and widest world-wide open-source community in the domain of ad hoc networking. As far as it is written in C language, it supports today a large number of devices (smartphones, laptops and tiny devices, such as low-cost WIFI routers).

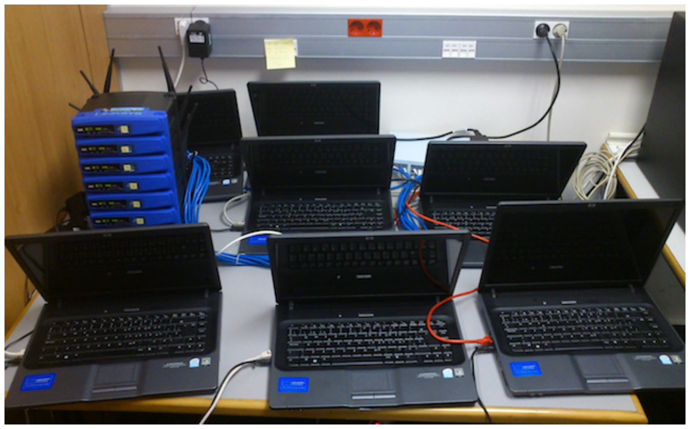

Figure 4.1: Devices used in the experimental set-up

To execute *olsrd* in our experimental set-up, we considered regular nodes implemented by HP 530 laptops with a processor of 1.6GHz and 512MB of RAM running Ubuntu 7.10 OS. Ilustration 4.1 shows devices used in our experimental set-up.

To illustrate this problem, we created a simple ad hoc network increasing the amount of real nodes in the same radio range (up to 10 nodes). Furthermore the massive appearance of new nodes in the network was emulated by externally announcing a large number of new links. As previously commented, ad hoc routing protocols are designed to support networks with a large number of nodes and links. According to our experiments, network nodes collapse when reaching a number of 100 neighbours. Why is this happening? This issue will be explained in the rest of this section.

4.1.1 Pathology identification

At first sight, it seems obvious that the problem could be exclusively related to the massive announcement of new links but, some more experimentation, showed that there were more things to analyse. In order to determine the precise behaviour of the protocol in presence of neighbour saturation, a simple experiment with just one node was performed. The announcement of new links in the network was carried out by an auxiliary node executing a modified version of *olsrd* where the default function of sending packets was altered to send a massive amount of routing packets. As finally all the routing packets are exchanged through the wireless medium, in practice, emulating the presence of a huge

Table 4.1: Characterisation of failure modes

Failure mode	Routing process under execution (*olsrd* process running)	Routing capabilities enabled (watchdog timer running)
Normal behaviour	Yes	Yes
Hang	Yes	No
Crash	No	No

amount of neighbours via one single node is the same that having a huge amount of real neighbours announcing themselves. This assumption is possible considering that our abstraction level is located at the routing layer, above the MAC one. Surprisingly, the node was not affected at all by the packet sent and continued its normal operation without any problem. After analysing the traffic generated among nodes using the *tcpdump* utility, we could determine that it was not the announcement of new nodes what directly caused the instability of the routing protocol, but the ulterior massive exchange of these links among neighbour nodes.

The effect induced by a high density of neighbour in the vicinity of a given node can be characterised by its failure mode. It is possible to distinguish three different failure modes: normal behaviour, hang or crash. Typically, hang mode can be identified because despite the protocol remains operating, its communication capabilities to send and receive routing packets has been disabled persistently. Conversely, crash mode directly involves stopping the protocol execution. Table 4.1 summarises considered failure modes.

In order to identify the occurrence of previous failure modes, different probes have been introduced in the system. Concretely, to monitor whether the process associated to the routing protocol is alive, the *top* tool has been used. Furthermore, an *olsrd*'s plugin called *watchdog* has been instantiated to determine when the protocol hangs. In essence, this plugin periodically saves the system timestamp while the protocol works properly. So, by comparing the last watchdog timestamp with the current system timestamp, it is possible to determine whether *olsrd* hanged during the experimentation, and when.

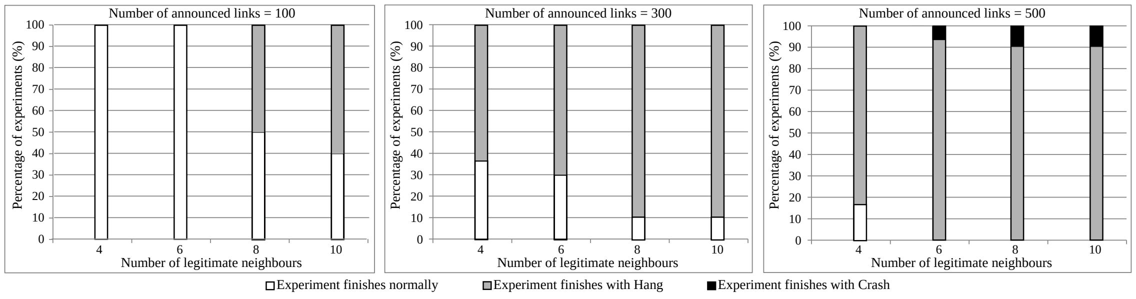

Figure 4.2: Impact of *Neighbour Saturation* according to the number of nodes and of new links announced

4.1.2 Pathology analysis

Once the failure modes defined and how to identify them in practice, different experimental campaigns were configured to study the impact of neighbour saturation in more detail. More than 200 experiments were executed considering 3 rates of massive exchange of routing links (100, 300, 500 new nodes), representing the advertisement of new nodes in the neighbourhood.

The results obtained are shown in Figure 4.2. Increasing the number of announced new links from 100 to 300 caused *olsrd* to crash in 60% of the experiments with just 4 nodes in the vicinity, whereas announcing 500 new links just collapsed the network and increased the percentage of hangs. So, it can be observed that networks with an increasing number of nodes are very likely to cause stability problems in the protocol, which may result in either hang or crash, depending on the density of nodes in the same neighbourhood. So, not only the size of the network is important, but the density of nodes in the same neighbourhood is critical. Experiments performed in a worst-case scenario consisting in a neighbourhood of 10 legitimate nodes with an increasing number of announced links, showed that protocol hangs begin to appear just by announcing 50 new links. Experimentation carried out with a more recent version of *olsrd* (v.0.6.1) was additionally considered obtaining similar results.

Although the origin and the effect of the problem is determined, the error propagation mechanism that leads to those particular failure modes remains unexplored. To increase our understanding of this error mechanism, experiments were repeated using the *ddd* debugging tool. After this process, the critical section where the fault (the massive announcement of neighbours) becomes a failure (either crash or hang) could be determined within the source code. Given the link-state nature of OLSR, *olsrd*, builds a full map of the connectivity to the network, showing how nodes are connected. Indeed, after analysing the obtained traces, the problem was tracked down to the management of the neighbours table when it is big enough (*olsr_{lookup, delete, insert}_neighbour_table* functions). The official *olsrd* bug tracker does not list this problem, and it is today an issue to be corrected.

4.2 Mitigating the effect of the threat through fault tolerance

This section presents the approach used to inhibit neighbour saturation. In the first, the approach proposed is explained in a high level and afterwards its implementation is introduced with the details of integration of the mechanism as an aspect, to be deployed in both versions of *olsrd*.

Taking everything into account, the compilation process of an aspect is explained in the last point of the section. The aspect compiler tool is explained to understand the weaving process of main source code of the routing protocol and the designed aspect.

4.2.1 Design

The design of a fault-tolerance aspect to temporally limit the number of new neighbours requires a mechanism able to act when incoming routing packets are received in a node. If a node has reached its limit of neighbours stored, it will avoid processing any announcement of new neighbours in the network, discarding these routing protocol packets.

Instead of modifying the source code of routing protocols in versions affected by the problem, aspect-oriented programming has been elected as a suitable paradigm programming to face this problem. The aspect designed will have the ability to inhibit the problem in any version of the protocol. Without using aspect-oriented programming, the mechanism should have been written in each function of the code affected of each version of *olsrd*, and future changes in the mechanism should be modified in all files affected. AOP solves this extra-costs in the implementation, improving development and deployment costs of the mechanism.

```
/* Advice to count the number of links in the links table */
after: call(void set_loss_link_multiplier(...)
if(strcmp("add_link_entry", this->funcName) == 0){
    links_counter++;
}
if(strcmp("remove_link_entry", this->funcName) == 0){
    links_counter--;
}

/* Advice to temporally discard the reception of new neighbours */
ssize_t around(): call(ssize_t olsr_recvfrom(...))
if(strcmp("olsr_input", this->funcName) == 0){
    if(links_counter < LIMIT){
        return proceed();
    }else{
        return 0;
    }
}
```

Listing 4.1: Aspect to inhibit neighbour saturation

The main responsibility of the aspect is control incoming routing protocol packets and the amount of links managed in the routing table. The aspect will contain two advices with different objectives. In the first advice, it is necessary to define a counter which dynamically increases and decreases as long as the number of links in the links table is modified. Then it will be necessary a second advice to limit the number of links in the links table, which is the actual flaw in the code we want to correct. The aspect will try to prevent from nodes of the ad hoc network entering into saturation state. Aspect will be weaved in the main source code with the aspect-oriented tool introduced later in the subsection 4.2.3.

4.2.2 Implementation

With respect to the first advice, the joinpoint where the links list is modified for the first time involves the invocation of the "set_loss_link_multiplier" function. If this call is made by the "add_link_entry" function, the links counter is increased in one unit. Otherwise, in case it is called by the "remove_link_entry" function, the counter will be decremented in one unit. Listing 4.1 illustrates the code of this advice to count the current number of links of *olsrd*.

With respect to the second advice, we set the reception of OLSR packets as the join-point (using the function named “olsr_recvfrom”). This is the point where *olsrd* processes incoming data from the network interface card. One link always refers to one particular neighbour, so in practice, we consider the notions of neighbours and links as equivalent. In this way, if limiting the reception of routing protocol packets advertising new neighbours through the definition of a threshold value (LIMIT), the routing protocol packets responsible for overloading the system will be discarded and consequently not parsed. Thus, the problem caused by the saturation in the links reception could be mitigated. The code of this advice is detailed in listing 4.1.

As far as LIMIT may vary with respect to the node’s capability to manage new nieghbours, it is important not to fix its value. Conversely, it should be a parameter depending on the instantiation of the routing protocol. In addition, it is worth noting that this mechanism does not prevent the routing protocol from updating those links previously established.

4.2.3 The AspeCt-oriented C compilation process

AspeCt-oriented C [14] is a aspect-oriented software development focused in C programming language. This project is conducted by the Middleware Systems Research Group at the University of Toronto. AspeCt-oriented C (ACC) ships with a set of compiler tools, particularly an aspect-oriented programming weaver which produces ANSI-C code with aspect files and ANSI-C source code.

ACC is an perfect tool to develop aspects in C programming language, due to its simple but powerful aspect syntax and compatibility of generated code with GCC. ACC has been proved to support aspect-oriented language designs for other languages, to evidence its robustness to support aspect-oriented programming. Because most of routing protocols in ad hoc networks are written in C programming language, ACC is a good choice to develop aspects to enhance capabilities of routing protocols.

Aspect weaver is a fundamental key in aspect oriented programming to do intersection advice code in main code. Each programming language dispose several aspect weaver to aspect compilation. Semantic of advices, joinpoints and pointcuts are restricted to each aspect weaver, despite aspect oriented programming semantic is very similar in most weavers.

Weaving process can be resumed in parsing advice code, semantic checking of advice and join point matching in main target code. Aspect code and main code processing phase ends when weaving process generates source code ready to compile. For example, source code generated is capable to be compiled with gcc. The process to weave the aspect implemented and source code of *olsrd* routing protocol is described in illustration 4.3. First, source code of *olsrd* and aspect are preprocessed with GCC compiler. After this stage, source code is ready to be weaved with aspect-oriented tool ACC. In the second stage, source code files of main code (*olsrd* routing protocol) are selected if they contain the pointcuts declared in advices. Selected source code files are weaved with the aspect file, to include the aspect in the correct place of the code. Finally, weaved source code files and remaining source code of routing protocol are compiled with GCC compiler to generate resilient version of *olsrd* binary.

4.3 Approach verification

In order to assess the effectiveness of the developed aspect for tolerating saturation problems, the aspect is automatically deployed by the Aspect-C weaver on the implementation of different versions of the protocols under study, OLSR. As in the pathology analysis scenario, some fault injections experiments are carried out in order to check the level of improvement issued from the use of the produced aspect. This section reports on the results obtained.

This subsection is devoted to analyse which is the proper value of LIMIT for the particular test-bed considered in this chapter. Consequently, the conclusions obtained are limited to the nodes considered in such deployment.

In order to assess the behaviour of the proposed fault-tolerance aspect, more than 580 experiments were executed considering the following factors: two versions of *olsrd* (v.0.6.0 and v.0.6.1), four values for LIMIT (50, 100, 200 and 300 neighbours) and six network deployments (1, 2, 4, 6, 8 and 10 nodes). Each combination of factors was repeated 8 times to enhance the statistical significance. As the announcement of 500 new links resulted the worst experimental case, this same value was selected to ease the comparison of results.

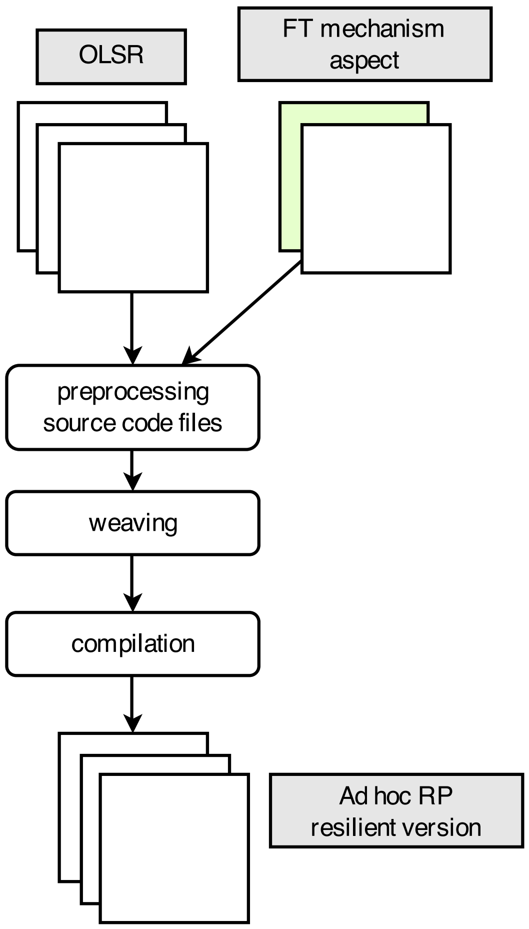

Figure 4.3: Aspect-oriented compilation stages

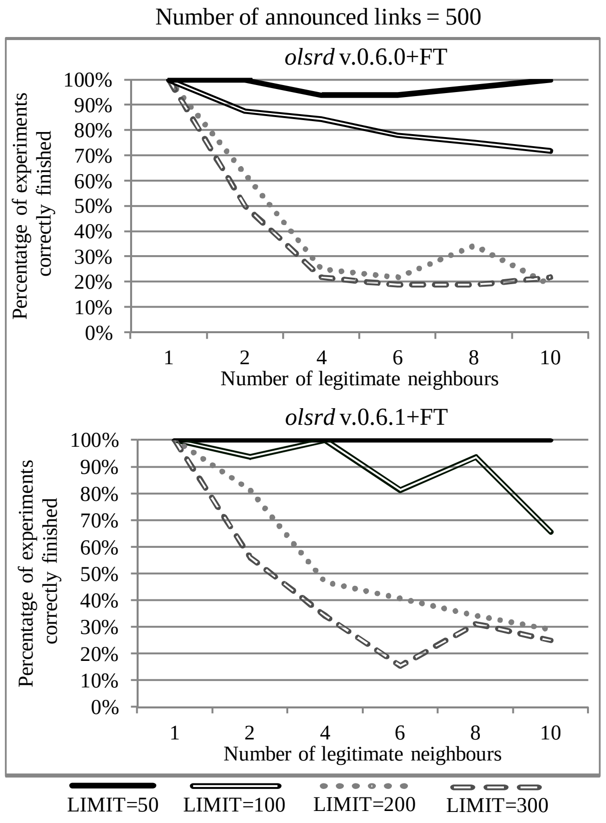

Figure 4.4: Percentage of survivor nodes to experimental campaigns with fault tolerance mechanism enabled

Table 4.2: Failure modes observed (C - Crash, H - Hang) considering the fault-tolerance mechanism in presence of 500 new neighbours for 10 real nodes

olsrd version	LIMIT							
	300		200		100		50	
	H	**C**	**H**	**C**	**H**	**C**	**H**	**C**
v.0.6.0+FT	21%	59%	29%	46%	8%	16%	2%	1%
v.0.6.1+FT	18%	54%	7%	47%	5%	9%	0%	0%

Figure 4.4 illustrates the evolution of the percentage of successful experiments when considering previous thresholds. Despite not showing identical values, what is worth mentioning is that both *olsrd* versions (v.0.6.0 and v.0.6.1) present similar trends, which is interesting to validate the behaviour of our fault-tolerance mechanism. As seen, the minimum value to totally mitigate the impact of neighbour saturation is limited to 50 nodes.

Table 4.2 shows the complementary percentage of experiments which finished in hang or crash. As can be observed the percentage of hangs and crashes for LIMIT=50 is negligible. However, beyond this value, the percentage of correct experiments decreases as fast as the amount of hangs and crashes increases. Indeed, when limiting the nodes filter to 100 new links (LIMIT=100), the percentage of correct experiments correctly finished decreases around 70% in the worst case of considering a neighbourhood of 10 legitimate nodes. Finally, considering limits higher than 200 links becomes practically infeasible as far as they obtain values which are comparable to those obtained with the default version of *olsrd* (not considering the presence of the fault-tolerance mechanism in both version v.0.6.0 and v.0.6.1).

As results from Table 4.3 show, the binary size resulting from applying the aspect version of the fault-tolerance mechanism, increments less than 6% with respect to the default versions of *olsrd*, which is practically negligible. During runtime and considering normal conditions of density of nodes, the increment of CPU consumption is null in both the default versions of *olsrd* and those where the fault-tolerance mechanism has been introduced. It means that our mechanism has no cost in terms of CPU when its activation is not required. When the density of nodes increases through the announcement of a massive amount of links, the default versions of *olsrd* strongly increment the CPU consumed when facing a high density of nodes. Conversely, fault-tolerant versions of *olsrd* automatically trigger the

Table 4.3: Intrusiveness introduced by the fault-tolerance mechanism (FT) to temporally discard the reception of new neighbours

***olsrd* version**	**Binary size**	**CPU when there is a normal density of nodes**	**CPU when there is a very high density of nodes**
v.0.6.0	968 KB	1%	35%
v.0.6.0+FT	1028 KB	1%	1%
v.0.6.1	1008 KB	1%	30%
v.0.6.1+FT	1028 KB	1%	1%

links filtering when detecting such an increment.

4.4 Discussion

Despite the simplicity of the aspect-oriented approach adopted, the preliminary results shown are promising and consolidate our idea of using aspect oriented programming to enhance dependability of ad hoc routing protocols. However, the applicability of AOP in the domain of ad hoc networks goes beyond the mechanism deployed in this approach and opens a wide spectrum of possibilities, for example, to dynamically track the behaviour of routing protocols and propose countermeasures in case of detecting an anomalous behaviour.

Fault tolerance approach to temporally limit the number of new neighbours is implemented and deployed specifically to two versions of *olsrd*. If current pathology affects more routing protocols, mechanism will be implemented and deployed in a specific way for each routing protocol. Implementing specific mechanisms can lead to complications and useless waste of time. If a pathology affects to considerable amount of routings protocols, specific aspect implementation of fault tolerance mechanism is not a efficient solution.

Given these implementation dependencies that limit the application of one fault tolerance mechanism from one routing protocol to another, we ambition at defining a generic aspect, referred to as meta-aspect from now on, to ease the deployment of fault tolerance mechanisms regardless the instantiation particularities of the routing protocol targeted. The idea of *meta-aspects*, that goes beyond the classical application of the AOP paradigm will be illustrated in next chapter.

Chapter 5

Addressing the ambient noise threat

Ambient noise is one of the most common problems impacting the behaviour of ad hoc routing protocols. To face this problem the goal of this chapter is twofold. On one hand, the present chapter designs a best-effort adaptive fault tolerance mechanism to mitigate the effects of this ambient noise. On the other, this fault tolerance mechanism is implemented using a *meta-aspect* approach, that goes beyond the traditional use of AOP to enable the deployment of the mechanism in any proactive routing protocol regardless its implementation.

Thanks to the use of such *meta-aspects* we will be able to cope with the second goal of this research: showing the feasibility of using AOP to develop FT aspects applicable to different types of proactive routing protocols.

5.0.1 Proactive Routing Protocols Under Study

Proactive routing protocols in ad hoc networks are responsible for determining the optimal route among network nodes. To determine optimal routes, proactive routing protocols exchanges routing packets periodically among nodes to create and maintain the topology. These routing packets are divided in packets to discover nodes in 1-hop (hello packets) and packets to propagate routing information beyond 1-hop neighbours (control packets)

Three state-of-the-art proactive routing protocols have been selected as targets to study ambient noise problem. Their names are Optimized Link State Routing (OLSR), the Better Approach To Mobile Ad hoc Networking (B.A.T.M.A.N) and Babel.

OLSR was described in section 4.1 of the previous chapter. Nevertheless, it is interesting to describe the current specification OLSRv2 promote the use of link quality extensions, like ETX. The initial specification of OLSR (RFC 3626) established route computation using hop-count as metric.

B.A.T.M.A.N [17] is a novel proactive distance-vector routing protocol. For each node, B.A.T.M.A.N periodically sends out broadcast messages to inform neighbours of its existence. This process is repeated until the routing information reaches all network nodes. For each link the routing packets arrival rate is advertised so that neighbour nodes can determine the link quality. B.A.T.M.A.N uses the TQ metric to estimate the quality of network links.

Babel (RFC 6126) [5] is the most recent protocol under consideration. It is a distance-vector routing protocol that has two main characteristics to optimise its relay mechanism. On one hand, it uses history-sensitive route selection to minimise the impact of route flaps. In such a way, the route selection favours the previously established path rather than alternating between two routes. On the other hand, it forces a request for routing information each time it detects a link failure from one of its preferred neighbours. This is a best-effort mechanism to reduce the convergence time of the network. Like OLSR, Babel uses ETX as a metric of quality for network links.

5.0.2 Link Quality Metrics in Routing Protocols

The traditional Ethernet philosophy of selecting a given communication link towards a destination, among those available, with the criterion of minimising the number of remaining hops (*hop-count*) is a poor choice. Due to noise, the quality of all communication links between mesh nodes is not the same, which advices against the use of such a simple metric.

Since mid-00s, the notion of link quality is basically computed by each node according to the amount of routing information received from its neighbourhood: the higher this reception rate the better.

The *Expected Transmission Count* (ETX) is without any doubt, the most well-known metric for characterising the quality of a link [16]. It reflects the number of expected transmissions of a packet to be received without error at its destination. This number varies from one to infinity. An ETX of one indicates a perfect transmission medium, whereas an ETX of infinity represents a non-functional link. In practice, ETX can be defined as $ETX(i) = (1/(RPAR(i) * NRPAR(i))$. Given a sampling window in link i, $RPAR(i)$ is the Routing Packets Arrival Rate seen by a node, and $NRPAR(i)$ is the $RPAR(i)$ seen by the neighbour node. An alternative quality metric is proposed in [17] and it is called *Transmission Quality* (TQ). This second metric shares the same principles established by ETX but it is computed in another way. In essence, receivers calculate the number of routing packets received against those expected following the expression: $TQ(i) = (RPAR(i) * NRPAR(i) * p(i))/(MAX\_LQ^2)$. In the expression $RPAR(i)$ and $NRPAR(i)$ have the same meaning than in ETX, and MAX_LQ is a constant which bounds the ideal maximum quality. The term $p(i)$ refers to the penalty that is applied to unidirectional links, to promote those that are bidirectional. The higher the TQ, the better. In table 5.1 appears a brief description of previous measures.

For the sake of simplicity, the rest of this chapter will denote the quality of a link i as lq_i and it will be always interpreted as the higher the lq_i the better. Under such assumption lq_i should be viewed as $1/ETX$.

As a result, whenever two communication links towards a destination i and j are available, the protocol selects the one providing the better link quality. From this viewpoint, and since such link quality metrics are periodically recomputed, one can say that presented approaches adapt to variations of quality derived from ambient noise. However such capacity is not enough to keep communication links alive whenever such noise persist along the time or it is very high.

5.1 Experimental setup

The previous protocols introduced are integrated within a real experimental test-bed. After that, these routing protocols are assessed to show consequences of ambient noise and particular set of parameters involved. Such parameters will be finally analyzed to determine the fault tolerance algorithm.

Before subjecting the commented routing protocols to the presence of ambient noise, it is necessary to characterize it in practice. Thanks to this experimentation, we could distinguish three different levels of ambient noise: a high ambient noise coinciding with the workday involving a packet loss ranging from 35% to 50%, a moderate one matching with the lunch breaks from 5% to 35% and a low one from 0% to 5% at night. Despite not being so frequent, the range from 50% to 100% typically representing additional external perturbations like malicious attacks from signal inhibitors or accidental faults like interferences from microwave ovens, completes this model.

As ambient noise is eventually manifested in terms of packet loss [19], the emulation of ambient noise through packet loss is a possible solution to recreate the exact conditions of noise in an experimental environment. Thus, it is possible to enhance the repeatability and reproducibility of the experimentation, and the accuracy of the results provided. So, the use of tools such as netem [1] for the injection of a given rate of packet loss in the network has been required.

The implementation targets considered for our experimentation are last steady open-source versions of the routing protocols previously described in Section II-C. Accordingly, well-known implementations of OLSR (olsrd v.0.6.0 [2]), Babel (babeld v.1.1.1 [3]) and B.A.T.M.A.N (batmand v.0.3.2 [4]) have been taken into account.

The testbed used to carry out our experimentation has been explained before in section 4.2, and in this case study experiments are executed with the same setup. To carry out our experiments and assess the robustness of routing protocols against ambient noise threat, same network deployment as in the chapter 4 was considered.

[1] http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

[2] www.olsrd.org

[3] www.pps.jussieu.fr/jch/software/babel/

[4] www.open-mesh.org

This experimental setup takes into consideration the two basic types of routes, route A-B with alternative and route A-C without alternative paths. In the first case, the criterion to switch from one route to another is always changing the old best route by the new best one from the set of available alternative routes. The study of this case has been simplified by considering just two alternative routes, which is the minimum number of routes to perform a route switch. Thus, route A-B can be established through nodes x_i and y_i respectively. However, it is worth noting that the ambient noise may not be neither constant in time nor homogeneous in space. Given the unavoidable presence of physical obstacles like walls or other objects, the studied routing protocols usually find the best route from A to B (and vice-versa) through x_i nodes most of the times, rather than through y_i nodes.

An UDP constant bit-rate data flow of 200 Kbps was established from source A to destination B to compute the Effective Packet Delivery Ratio (EPDR) of the route. As previous works indicate [21], the Effective Packet Delivery Ration (EPDR) is one of the measures that better represents the impact of ambient noise in the network behaviour. The EPDR for a given route r is expressed as the percentage of applicative packets received from the total sent.

In order to limit the influence of real ambient noise in our results, our experimentation was carried out at night, assuming an acceptable intrusiveness of 0% to 5% of real (non-emulated) packet loss for wireless networks. In summary, 600 experiments of 300s each divided in two experimental campaigns (one per type of experiment) were executed.

5.1.1 Pathology identification and analysis

In the experiments, the EPDR decreases proportionally with the amount of ambient noise introduced. The case representing the ideal EPDR loss should involve an identical decrement with respect to the ambient noise introduced. Any case where the EPDR decreases faster than ambient noise introduced, necessarily involves the presence of any additional effect impacting on the EPDR.

The results of previous experiments are not casual. After analysing the code in detail, we observe that behind the name of different variables and constants (always consistent with their respective specification), there are three common parameters characterising the behaviour of proactive routing pro-

Effective Packet Delivery Ratio (EPDR)	Percentage of applicative packets received from the total sent
T	Default period to send a routing packet advertising a given link
Twindow	Validity time determining the temporal window after which the protocol decides whether discarding a link or not
Minimum Quality Threshold (MQT)	Minimum acceptable quality before flushing a link

Table 5.1: Measures to characterize behaviour of the routing protocols in presence of ambient noise

tocols against ambient noise. Table 5.2 identifies them and state their default values. T, is the default period to send a routing packet advertising a given link, and Twindow is the validity time determining the temporal window after which the protocol decides whether discarding a link or not. Basically, the use of these two time-related parameters is located within the task scheduler module already presented in figure 2.2. The Minimum Quality Threshold (MQT) defines the minimum acceptable quality before flushing a link. This quality-based parameter is used within the routing manager to decide about the feasibility of a given link. Table 5.1 shows previous measures used to characterize the behaviour of routing protocols in presence of ambient noise.

After analysing the targeted routing protocols, the conditions that must be satisfied to purge a link have been ordered from the most reactive to the most conservative as follows: *olsrd* and *babeld* require (i) the expiration of the Twindow or (ii) achieving the MQT. Conversely, *batmand* only requires the expiration of Twindow before removing the link. Surprisingly, the notion of MQT is never taken into consideration in *batmand*. Unlike *babeld* and *olsrd*, the link quality is only updated in *batmand* when getting new information through incoming routing packets, and as it cannot be 0, *batmand* must consider the MQT is not necessary. Let us now focus on T. If we additionally normalise T in the different configurations shown in Table 5.2 with respect to 1 sec. to compare their relative Twindow, we observe that *olsrd* only can send 6 packets to update the validity time of the link, whereas *babeld* admits up to 15 packets, and *batmand* has 200 new opportunities.

Table 5.2: Critical parameters for the route availability in ad hoc networks

Protocol implementation	Twindow (s)	T (s)	MQT (%)
olsrd v.0.6.0	30	5	10
batmand v.0.3.2	200	1	none
babeld v.1.1.1	60	4	0

5.2 Fault tolerance approach

This section faces the problem of ambient noise in proactive routing protocols proposing a generic adaptive strategy which enables the routing protocol to replicate the routing packets only when ambient noise is detected. Replication is a well-known technique in the domain of fault tolerance. The use of packet replication in this chapter is devoted to ensure the reception of the routing information even when the presence of a high level of ambient noise disturbs the communication between nodes. So, this approach can be useful in environments affected by ambient noise when links are in risk of disappearing or it is necessary to speed up the convergence time.

5.2.1 Design

This technique is based on the principles of T, Twindow and MQT previously identified in the last section. The algorithm proposed in this section is applied from the default configuration of the routing protocols. Regarding their regular behaviour, routing protocols send packets every T seconds and passively wait the time defined by Twindow to remove a link in case it is necessary. Our technique consists not in waiting passively the expiration of Twindow, but in analysing the trend for such link quality, and predicting its value in Twindow in order to react (in time) against a possible link removal.

In essence, the point of our proposal can be easily understood through the graphic in Figure 5.1. If applying basic algebraic notions, given two points A (x_2, y_2) and B (x_1, y_1) in cartesian axis, it is possible to determine the equation of the linear function as Formula 5.1 shows.

$$y - y_1 = m(x - x_1) \quad | \quad m = \frac{y_2 - y_1}{x_2 - x_1} \tag{5.1}$$

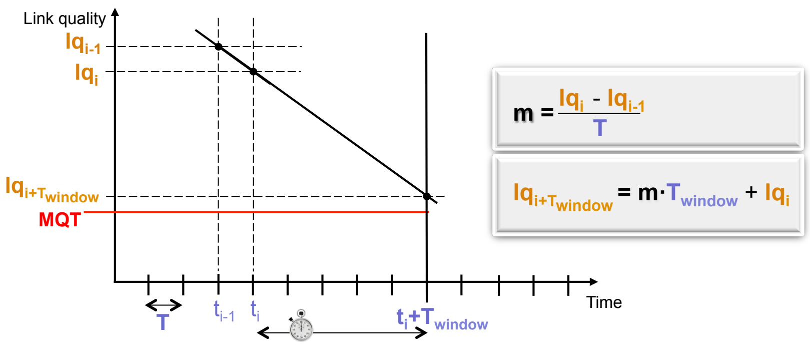

Figure 5.1: Link-quality-based adaptive packet replication technique

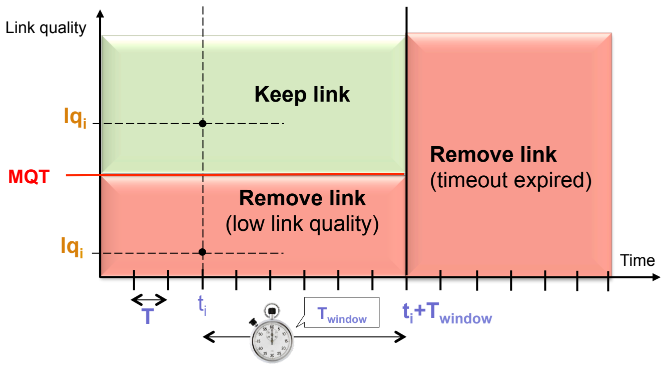

Figure 5.2: LARK parameters

If replacing points A and B by (t_i, lq_i) and (t_{i-1}, lq_{i-1}) where t_i represents the current time (T) and lq_i is its respective link quality and t_{i-1}, lq_{i-1} represent the last information but in last T, we would obtain Formula 5.2.

$$lq_{(t_{i+Twindow})} = m \cdot Twindow + lq_i \quad | \quad m = \frac{lq_i - lq_{i-1}}{t_i - t_{i-1}} \tag{5.2}$$

According to Formula 5.2, it is possible to predict the link quality $lq_{i+Twindow}$ for a given time $t_{i+Twindow}$ according to the trend pointed by lq_i and lq_{i-1}. Thus, in case this estimation is underneath the MQT, the activation of the link-quality based packet replication is triggered.

To avoid the loss of links and its effects on the network, a Link-quality-based Adaptive Replication of pacKets (LARK) algorithm is proposed. The main key features of the algorithm is the replication and the adaptation. Routing message replication may mitigate the impact of noise on link quality. Level of noise varies along the time, for this reason level of replication is adequate to the level of ambient noise. Tuning the level of replication has to be done dynamically when the routing protocol is in a running state, depending of the future prevision calculated with LARK algorithm.

To continue with previous work with aspect-oriented programming in the preceding chapter, LARK algorithm will be implemented using AOP. This paradigm will help to deploy a unique fault tolerance mechanism in any routing protocol, maintaining the same implementation.

Our technique is included within the routing manager module of the routing protocol (see Figure 2.2). Listing 5.1 shows the pseudo-code that has been implemented in C language the fault tolerance mechanism. The algorithm is included is generic for any routing protocol. However, there are parts of the algorithm, for example, send replicas call which depends of each routing protocol. These generic parts are encapsulated to improve maintainability of the external parts of the algorithm and this concept will be introduced in next subsection.

The real conditions of the network in practice impose limiting the amount of replicas to N_{max}. If considering a very severe ambient noise, the fact of sending more and more replicas will only contribute to increase, even more the effect of ambient noise. The value of N_{max} has been empirically computed for our deployment in order not to exceed the overhead obtained when applying the *batmand*-like configuration beyond 150$\%$.

As previously stated, *batmand* presented certain limitation like the absence of a MQT. However, given the genericity of our approach, nothing impairs assigning a $MQT = 0$ to *batmand* in our algorithm, or to any other proactive routing protocol which does not consider its use. Our technique is applied before the moment of sending a routing packet every time T. Then, the algorithm proposed must obtain the value of lq_i and lq_{i-1}. The value of lq_i can be easily obtained from the current state of the routing manager module. However, not all the protocols consider storing the previous state. Accordingly, the algorithm must store lq_i to provide lq_{i-1} in the next iteration of T. Also the algorithm must store amount of replicas calculated previously for this link, $l.nReplicas_{i-1}$ and the

```
Nmax=...
/* every time a new routing packet is forged */
for each 1-hop Link l announced in the packet you send
    /* obtain the value of the variables needed */
    load l.lq[i]
    load l.lq[i-1]
    load l.nReplicas[i-1]
    load l.t[i]
    load l.t[i-1]
    /*deduce future link quality*/
    m=(lq[i]-lq[i-1])/(t[i]-t[i-1])
    lq[i]+Twindow=m*Twindow+lq[i]
    /* determine the number of replicas to send */
    if (lq[i]+Twindow <= MQT) then
       if (l.nReplicas[i] < Nmax) then l.nReplicas[i]++
    else
        if (l.nReplicas[i] > 0) then l.nReplicas[i]--
    /* save the variables for the next iteration */
    store l.lq[i]
    store l.nReplicas[i]
    /* update the worst case of replicas to send */
    if(l.nReplicas[i] > replicasToSend)
       replicasToSend = l.nReplicas[i]
/* send the replicas required [0,limit] */
send(replicasToSend,routingPacket)
```

Listing 5.1: Pseudo-code LARK Algorithm

timestamp of previous amount replicas calculated, load $l.t_{i-1}$. Actual timestamp is saved in load $l.t_i$ to use in deduction of future link quality. Cost of storing previous values is negligible in terms of memory footprint even for the tiny devices considered in our experimentation.

The next step involves computing $lq_{i+Twindow}$ is calculated through the expression in Formula 5.2. In case this value is underneath MQT, the value of $nReplicas$ indicating the number of replicated packets that will be sent in T, is progressively increased only if its current value is lower than N_{max}. Otherwise, in case the link has overcome from the risk of disappearing, the number of replicas is progressively reduced up to 0, thus restoring the default behaviour of the protocol. The value of $nReplicas$ is also stored to increase or decrease it in the following iteration, depending on the state of the link.

Number of replicated packets corresponds to the highest number of $nReplicas$ for each 1-hop link announced in the packet which will be sent. Only in the case of $nReplicas_i$ is higher than $replicasToSend$ previous value, value stored in $replicasToSend$ will be updated. After loop go over into all 1-hop links announced in the packet, the packet will be replicated the number of times indicated in $replicasToSend$.

In any case, all the replicated packets send the same information, so, any packet already received will be discarded. Our goal thus is not sending new packets with further information, but increasing the probability of broadcasting the same information at least once. As all the protocols natively implement the mechanism to discard replicas, no additional strategy has been required to be introduced in our algorithm in the reception of packets.

Given the simplicity of the operations considered, and the time elapsed between the iterations, the overhead introduced in the protocol in terms of CPU is also negligible (less than 1%).

5.2.2 LARK implementation using Aspect-oriented programming

Algorithm has to be executed when routing protocol packet is sent from the routing protocol manager code (Figure 2.2). These points in the source code are the joinpoint of the aspect which will contain

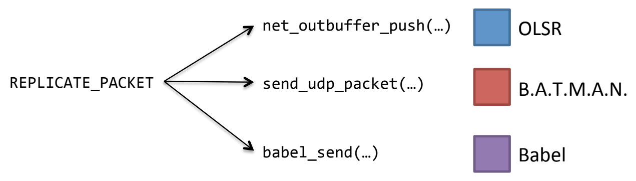

Figure 5.3: Resolution of a *meta-tag* in preprocessing phase

LARK algorithm implementation. Each routing protocol needs specific joinpoint, however the aspect cannot contain dependent parts for each routing protocol. To solve this problem, dependent parts of the aspect, for example the joinpoint, are encapsulated in *meta-tags* to create out *meta-aspect*.

A *meta-tag* is a declaration of code which each routing protocol has to define. The *meta-tags* are resolved in preprocessing phase with the own header files for the fault tolerance mechanism. Without using the concept of *meta-tag*, a simple aspect has to be modified in the specific lines which depend of the implementation of the target routing protocol. Thanks to the concept of *meta-tag*, dependent parts of the aspect are encapsulated, which improves the cost of implementation, facilitates deployment in any routing protocol and it will reduce efforts in the maintainability. A example of *meta-tag* is parsing packet to get list 1-hop neighbours, which code of this *meta-tag* is different for each routing protocol. In figure 5.3 appears a example of *meta-tag* of the replication action which is resolved for each of the three proactive routing protocols (OLSR, B.A.T.M.A.N and Babel).

A *meta-aspect* is an aspect which contains *meta-tags* in its code, in order to be generic and be deployed in any system. In our case study, the *meta-aspect* allows deploy the fault tolerance mechanism in any proactive routing protocol with a single implementation of the code. LARK algorithm is encapsulated as a fault tolerance mechanism inside the *meta-aspect*.

Each routing protocol is responsible of the definitions of the *meta-tags*. Separation of the *meta-tags* of routing protocol layer from LARK algorithm, creates a different concept in the implementation of fault tolerance mechanisms. Experts in fault tolerance mechanisms can design new techniques and algorithms regardless the integration in a particular routing protocol. Deployment and development of new fault tolerance mechanisms in routing protocols is easy goal with the generic implementations. Using *meta-aspects*, experts in routing protocols have to integrate fault tolerance mechanisms in a particular routing protocol filling the *meta-tags* with the specific code of target routing protocol.

LARK algorithm integration inside the aspect

As introduced in last subsection, the *meta-aspect* is activated when a routing protocol packet after it is sent. If the packet contains a list of sender 1-hop neighbours list, the packet will be processed.

LARK algorithm is integrated inside the advice of the *meta-aspect*. At the beginning of the *meta-aspect*, the joinpoint commences using *meta-tags* to encapsulate specific parts of the source code of each routing protocol. After the joinpoint, the advice starts with the initialization of the variables. Variables dependent of each routing protocol are encapsulated into `INITIALIZE_SPECIFIFC_VARIABLES` *meta-tag*.

Advice code needs to parse the data of the packet to access to their fields easily, depending of each routing protocol. Verification of the parsed is necessary to ensure packet data is valid and it has not been affected by any error.

The main information of the packet is in its list of neighours achievable in 1 hop. In the `GO_OVER_NEIGH_1_HOP_PACKET_LIST` *meta-tag*, advice go over this list and in each iteration, information about link quality and address of the neighbour is pulled.

In addition with the list of 1 hop neighbours of each packet, there is one more list. The link quality list is used in the advice, which contains information of the 1 hop neighbours of the node. Link quality list is updated with neighours information of the packet, for example, adding new neigbour in the list. Timestamp, address, number of replicas and link quality of each neighbour announced in the packet, is stored in the link quality list. Timestamp remember the last time link quality of the neighbour was updated. Neighbour address identifies each node when the advice go over the list. Finally, number of replicas keeps the track of number of replicas necessary in the LARK algorithm.

With the reference to the information of a neighbour from the packet, it is searched in our link quality list. If the address of neighbour packet matches with the address of the neighbour stored in the link quality, it is the moment to predict the future link quality with LARK algorithm. If the future link quality is lower than *Minimum Quality Threshold* (MQT), number of replicas is incremented. In the contrary case, future link quality predicted is higher than MQT, number of replicas is decremented. Actual information of the link quality is saved in the list, beside the timestamp information. At last, the `replicas_to_send` value indicates the worst number of replicas to each neighbour of the packet list. If the neighbour has not been found in the link quality list, it will be added at the last of the list.

When future link quality has been predicted for all neighbours of the packet, the packet is replicated `replicas_to_send` times. When ad hoc network is stable, the most likely scenario is no replication of the routing protocol packets because future link quality of the neighbours exceed the *Minimum Quality Threshold*. In the listing 5.2 appears the code of the *meta-aspect* with LARK algorithmm integrated within it.

5.3 Assessing Fault Tolerance Aspect Approach

New version of the routing protocol with LARK algorithm integrated with the aspect, have been assessed with the same experiments as mentioned in the section 5.1. Default version of the routing protocol are evaluated with the versions with LARK aspect integrated to distinguish the improvements in terms of routing overhead, route availability and Effective Packet Delivery Ratio.

Additional experimental campaigns were required to show the effectiveness of the algorithm proposed. Concretely, let us first analyse the Figure 5.4, which represents the routing overhead introduced by the routing protocols implementing the LARK algorithm. Basically, when applying our technique, all the routing protocols balance their overhead to adapt their behaviour in a context-aware way.

In the first case (see Figure 5.4b), no additional routing packet is sent, so the intrusiveness introduced in the network is null. Conversely, when the ambient noise increases and the routing protocol requires a major effort to maintain their routes, it is allowed to increment the amount of routing information sent. Unlike the regular behaviour of the routing protocols considered, what is constant using our technique is not the rate of routing packets sent, but the rate of received ones. The goal thus, is maintaining the routing capability as longer as possible, even with a severe amount of ambient noise.

```
after(ARGUMENTS_POINTCUT): call(RETURN_TYPE_POINTCUT FUNCTION_POINTCUT(
    ARGUMENTS_TYPE_POINTCUT)) && args(ARGUMENTS_NAME_POINTCUT) {
    int timestamp = 0; float lq = 0.0;
    char neigh_addr[ADDR_STR_LEN];
    int replicas_to_send = 0; int i = 0; int found = 0;
    double m = 0; double future = 0;
    INITIALIZE_SPECIFIFC_VARIABLES
    PARSE_DATA_ROUTING_PROTOCOL_PACKET
    if(VERIFICATION_DATA_ROUTING_PROTOCOL_PACKET == 0) {
        GO_OVER_NEIGH_1_HOP_PACKET_LIST {
            strcpy(neigh_addr, GET_ADDRESS_NEIGH_PACKET_LIST);
            lq = GET_LQ_NEIGH_PACKET_LIST;
            GO_OVER_NEIGHBOUR_LQ_LIST {
                if(strcmp(LQ_NEIGH_ADDR, neigh_addr) == 0) {
                    found = 1;
                    timestamp = get_timestamp();
                    if(LQ_NEIGH_TIMESTAMP != timestamp) {
                        m = (lq - LQ_NEIGH_LQ_PRE)/(timestamp - LQ_NEIGH_TIMESTAMP);
                        future = (m * T_WINDOW) + lq;
                        if(future <= MQT) {
                            if(LQ_NEIGH_NREPLICAS < N_MAX_REPLICAS) {
                                LQ_NEIGH_NREPLICAS_INCREMENT }
                        }else{
                            if(LQ_NEIGH_NREPLICAS > 0){
                                LQ_NEIGH_NREPLICAS_DECREMENT }
                        }
                        LQ_NEIGH_LQ_PRE = lq;
                        LQ_NEIGH_TIMESTAMP = get_timestamp();
                        if(LQ_NEIGH_NREPLICAS > replicas_to_send){
                            replicas_to_send = LQ_NEIGH_NREPLICAS; }
                    }
                    break;
                }
            }
            if(found == 0) {
                ADD_NEIGH_PACKET_LIST_TO_LQ_TABLE    }
            for(i=0; i < replicas_to_send; i++) {
                REPLICATE_PACKET }
        }
    }
}
```

Listing 5.2: Code of Aspect Fault Tolerance Mechanism

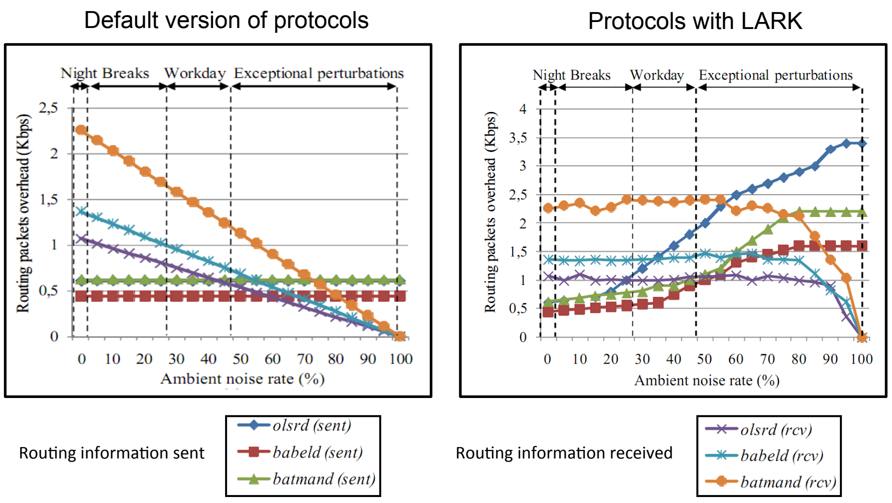

Figure 5.4: Routing overhead in proactive routing protocols

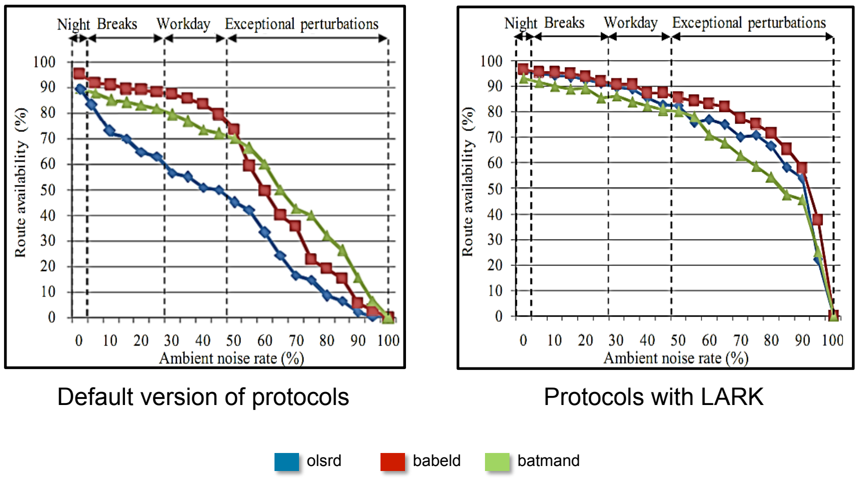

Figure 5.5: Route availability in proactive routing protocols

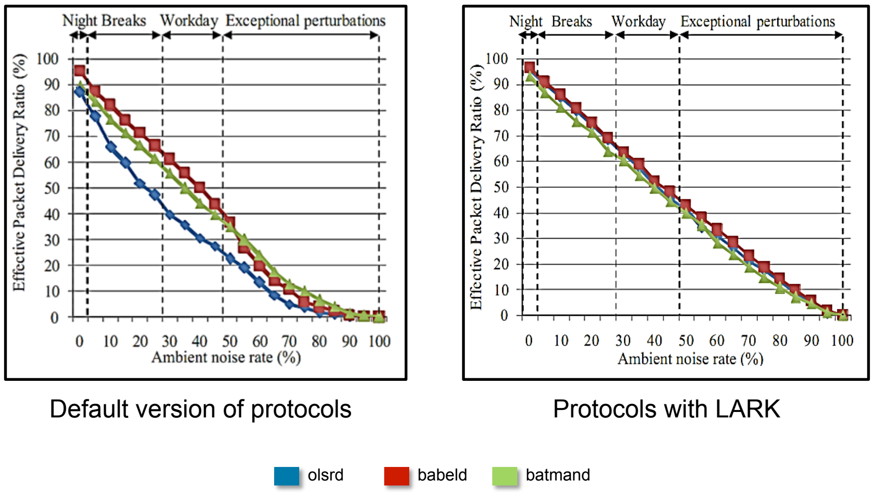

Figure 5.6: Effective Packet Delivery Ratio in proactive routing protocols

If comparing the new results with the previous overhead involving the packets sent, all the protocols reduce the amount of packets sent up to around 70% packet loss caused by ambient noise. In this condition of extreme necessity for the links survival, the routing protocols using our technique are forced even to increase the overhead introduced in the default configuration. Indeed, *olsrd*, *babeld* and *batmand* increment 13%, 15% and 150% in this point. However, the major difference is that now, the RA increases in these conditions. Beyond this ambient noise rate, the routing packets received decrease given the practical bound imposed by N_{max} to limit the packet replication indefinitely.

If taking these results in mind when comparing the route availability obtained when applying our technique, with those provided previously in Figure 5.5 (without LARK algorithm), the benefits of our technique can be observed.

Additionally, it is worth noting that thanks to this technique *batmand* speeds up its converge time, and consequently increments its route availability with respect to its default configuration from 5% to 10%. Results are very similar for *olsrd* and *babeld* (less than 3% of difference), but taking into account that the overhead introduced has been widely reduced for the ranges of breaks and workday (more than 150% in all the cases), where the protocols will operate most of the time.

All these improvements are observed in terms of the EPDR in Figure 5.6, thus enhancing the general behaviour of the ad hoc networks with respect to the regular behaviour of the protocol.

5.4 Conclusion

Contribution provided in this chapter goes in the direction of improving the network converge time resulting from a high level of ambient noise, while reducing the cost of routing overhead. As discussed before, route availability increment with rates of ambient noise lower than 70%.

Although results have been obtained from OLSR, Babel and B.A.T.M.A.N, a number of conclusions can be generalised and applied to any type of proactive routing protocol deploying *meta-aspect* with Link-quality-based Adaptative Replication of pacKets algorithm (LARK).

Firstly, LARK algorithm is a novel strategy to fight against ambient noise, proposed as a complement to the existing solutions. Concretely, its novelty is on promoting the dynamic adaptiveness of the routing protocol to the network environment to determine the optimum amount of routing information that must be exchanged among nodes in a given moment.

Secondly, major contribution of this chapter is reengineering the proposed link-quality-based replication algorithm as an *meta-aspect*. The *meta-aspect* has the potential of improving its portability to other different proactive routing protocols. Aspect-orientation has already showed its value in other contexts of use, including embedded systems. However, the complexity of redeploying our algorithm using AOP is something that has been achieved successfully. Aspect-oriented programming has demonstrated its enormous capacity in design and implementation of new fault tolerance mechanisms.

In the figure 5.7 represents our proposal in the design of generic aspects, the *meta-aspects*. The design phase of fault tolerance *meta-mechanisms* is done manually adapting the code of the fault tolerance mechanisms. A *meta-mechanism* is included into the *meta-aspect* is able to be instantiated for any routing protocol automatically. After preprocess the *meta-mechanism* with routing protocol and weaving the files, it is possible to continue instantiating more *meta-mechanisms* in the routing protocol, to resolve problems of the routing protocol. Finally, source files generated in previous phases are compiled to obtain the routing protocol resilient version. Designers of fault tolerance do not have

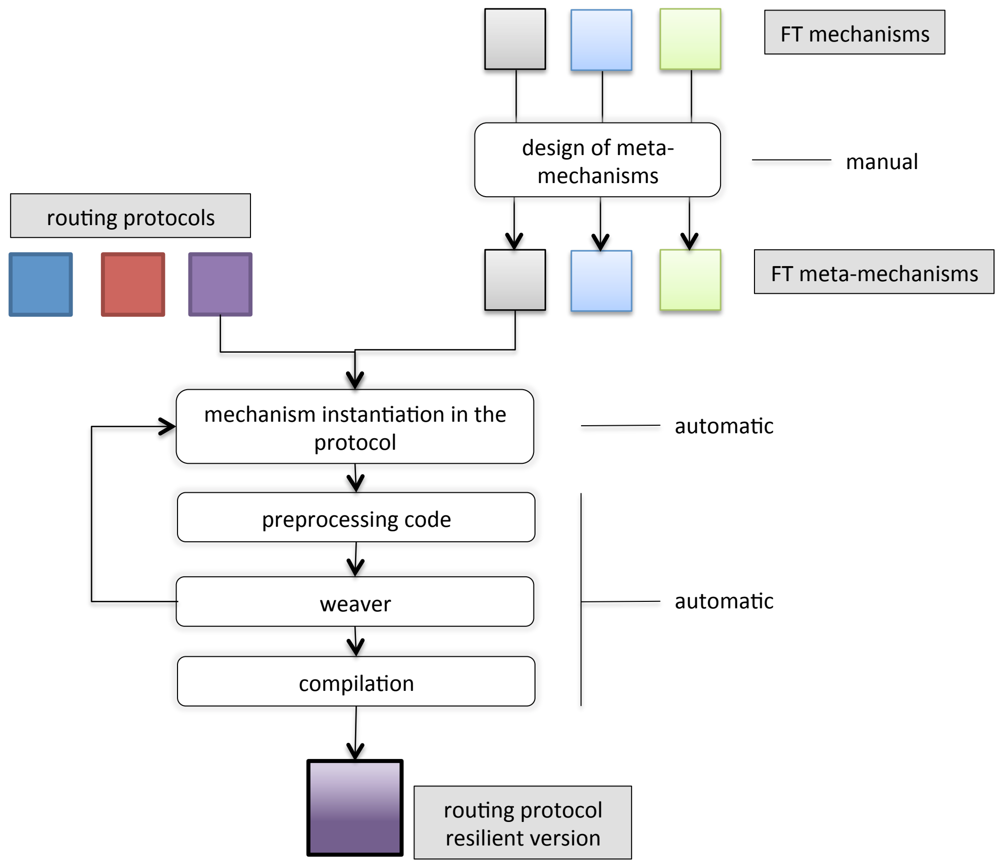

Figure 5.7: Transformation phases of the *fault tolerance meta-mechanisms* and routing protocols

to concern in the routing protocols where the mechanisms will be applied. When the fault tolerance mechanism is designed, it is included into a *meta-aspect* to represent the *meta-mechanism* ready to be instantiated. Our approximation goes beyond of the application of a mechanism to mitigate the specific problem of ambient noise, our approximation has been designed transform the manner to design, implement and deploy fault tolerance mechanisms.

Chapter 6

Conclusions

At first of the work, we have encountered with the limitations in the deployment of the fault tolerance mechanisms in the routing protocols. Firstly, each fault tolerance mechanism has to be implemented specifically for a routing protocol. The cost of reusing a mechanism in multiple routing protocols is considerable, even more maintaing code of a mechanism in multiple version of it in the protocols. This leads to an enormous efforts and a consumption of time if changes are necessary in the implementations. Furthermore, the problem of specific implementations to enhance a system and problems reusing the implementation, appears in other areas of engineering.

In previous chapters, 4 and applicability of aspect-oriented programming has been introduced to improve dependability of ad hoc routing protocols. Since a modest problem where the potential of aspect-oriented programming has been exhibited, a serious pathology has been mitigated in a elegant manner following the same programming paradigm, without any limitation of proactive routing protocols. Aspect-oriented programming remains a paradigm rarely used in fault tolerance in ad hoc networks, but this master thesis shows an interesting way to continue making progress in the area. AOP benefits portability and maintainability of the fault tolerance mechanisms code integrated into aspects. However, in the work of the adaptation of the two fault tolerance mechanisms into aspects, we have been located the real potential of aspect-oriented programming to enhance resilience in ad hoc networks.

In this work, we demonstrated the limitation of the fault tolerance mechanisms for a specific routing protocol vanishes with the use of *meta-mechanisms* integrated into the *meta-aspects*. The newfangled *meta-aspects* has been proposed as efficient manner to implement and deploy a fault tolerance mechanism in multiple real proactive routing protocols. Separation of the concerns between fault tolerance experts and routing protocol designers has been achieved. *Meta-tags* are the key in the separation of concerns, and they are filled by routing protocol experts depending peculiarities of the proactive routing protocols. Advantages as for example, ease of deployment, design, development and maintainability, are attributes that have been enhanced throughout this work.

Future work is related in the development of fault tolerance mechanisms under the notion of *meta-aspects*, achieving the generic *meta-mechanisms*. This new concept can be applied in the design of new mechanisms as the encryption in the communications of the ad hoc routing protocols. As well still remaining mechanisms which can be reused protocols of different nature (proactive vs. reactive). Evaluation of performance of resilient versions of routing protocols in new kinds of devices also still pending.

Actually, the use of genericity in the mechanisms to be integrated in different systems, is goal that cover many fields in the engineering which can be resolved with the concepts of the aspect-oriented programming.

Chapter 7

My publications

The following publications have been issued from the work developed in this master thesis.

- Jesús Friginal, Juan-Carlos Ruiz, David de Andrés and Antonio Bustos: Mitigating the Impact of Ambient Noise on Wireless Mesh Networks Using Adaptive Link-Quality-based Packet Replication, 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012), 2012, Boston (Massachusetts USA), Pages 8. ISBN 1-4673-1625-5. Conference CORE A* [10]
- Antonio Bustos, Jesús Friginal, David de Andrés, Juan-Carlos Ruiz: An Aspect-Oriented Approach to Face Neighbour Saturation Issues in Proactive Ad hoc Routing Protocols: olsrd as a case study, 1st International Workshop on AppRoaches to MObiquitous Resilience (ARMOR), 2012, Sibiu (Romania), Pages 6, ISBN 978-1-4503-1150-2/12/05 [3]

Bibliography

[1] *The Federal Response to Hurricane Katrina: Lessons Learned*. U.S. Executive Office of the President, 2006.

[2] Erich Bruns and Oliver Bimber. Phone-to-phone communication for adaptive image classification. In *Int. Conf. on Advances in Mobile Computing and Multimedia*, pages 276–281, 2008.

[3] Antonio Bustos, Jesús Friginal, David de Andrés, and Juan-Carlos Ruiz. An aspect-oriented approach to face neighbour saturation issues in proactive ad hoc routing protocols: olsrd as a case study. In *Proceedings of the 1st European Workshop on AppRoaches to MObiquiTous Resilience*, ARMOR '12, pages 3:1–3:6, New York, NY, USA, 2012. ACM.

[4] C. Perkings. Ad hoc On-Demand Distance Vector(AODV) Routing. *RFC 3561*, 2003.

[5] Juliusz Chroboczek. BABEL. [Online]. Available: http://www.pps.jussieu.fr/ jch/software/babel/, 2011.

[6] Frédéric Cuppens, Nora Cuppens-boulahia, Tony Ramard, and Julien A. Thomas. Misbehaviors Detection to Ensure Availability in OLSR. In *Mobile Sensor Networks*, pages 799–813, 2007.

[7] Gang Feng, Fei Long, and Yide Zhang. Hop-by-hop congestion control for wireless mesh networks with multi-channel mac. In *Proceedings of the 28th IEEE conference on Global telecommunications (GLOBECOM)*, pages 242–246, 2009.

[8] Joanna Geibig and Dirk Bradler. Self-organized aggregation in irregular wireless networks. In *Wireless Days*, pages 1–7, 2010.

[9] Yong He, Ruixi Yuan, Xiaojun Ma, and Jun Li. Analysis of the impact of background traffic on the performance of 802.11 power saving mechanism. *IEEE Communications Letters*, 13(3):164 –166, 2009.

[10] David de Andrés Antonio Bustos Jesús Friginal, Juan-Carlos Ruiz. Mitigating the impact of ambient noise on wireless mesh networks using adaptive link-quality-based packet replication. *42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, page 8, 2012.

[11] G. Kiczales. Aspect-oriented programming. *ACM Comput. Surv.*, 28(4es), December 1996.

[12] D. Lundquist and A. Ouksel. Distributed delay: Improving network throughput by reducing temporal saturation. In *3rd International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1 –5, 2009.

[13] Gustavo Marfia, Giovanni Pau, Enzo De Sena, Eugenio Giordano, and Mario Gerla. Evaluating vehicle network strategies for downtown portland: opportunistic infrastructure and the importance of realistic mobility models. In *Proceedings of the 1st workshop on Mobile opportunistic networking*, pages 47–51, 2007.

[14] Charles Zhang Michael Gong and Hans-Arno Jacobsen. Aspect-oriented c for systems programming with c. *AOSD 2007 Software Demonstration*, 2007.

[15] G.C. Murphy, R.J. Walker, and E.L.A. Banlassad. Evaluating emerging software development technologies: lessons learned from assessing aspect-oriented programming. *IEEE Transactions on Software Engineering*, 25(4):438 –455, 1999.

[16] Xian Ni, Kun-chan Lan, and Robert Malaney. On the performance of expected transmission count (etx) for wireless mesh networks. In *Proceedings of the 3rd International Conference on Performance Evaluation Methodologies and Tools (ValueTools)*, pages 77:1–77:10, 2008.

[17] Open Mesh, Better Approach To Mobile Ad hoc Networking (B.A.T.M.A.N.). [Online]. Available: http://www.open-mesh.net/, 2011.

[18] Krishna Ramachandran, Irfan Sheriff, Elizabeth Belding, and Kevin Almeroth. Routing stability in static wireless mesh networks. In *Proceedings of the 8th international conference on Passive and active network measurement (PAM)*, pages 73–83, 2007.

[19] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. In *IEEE Wireless Communications and Networking*, volume 3, pages 1664 – 1669, march 2005.

[20] E.M. Royer, P.M. Melliar-Smith, and L.E. Moser. An analysis of the optimum node density for ad hoc mobile networks. In *IEEE International Conference on Communications (ICC)*, volume 3, pages 857 –861, 2001.

[21] Xu Su and Rajendra V. Boppana. On the impact of noise on mobile ad hoc networks. In *Proceedings of the 2007 international conference on Wireless communications and mobile computing (IWCMC)*, pages 208–213, 2007.

[22] Yuan Sun, Irfan Sheriff, Elizabeth M. Belding-Royer, and Kevin C. Almeroth. An experimental study of multimedia traffic performance in mesh networks. In *Workshop on Wireless traffic measurements and modeling (WiTMeMo)*, pages 25–30, 2005.

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