João Carlos Andrade de Almeida

Licenciado em Engenharia Informática

Intrusion Tolerant Routing with Data Consensus in Wireless Sensor Networks

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador :	Prof. Doutor Henrique João Lopes Domingos, Professor Auxiliar, DI-FCT-UNL
Co-orientador :	Prof. Doutor Vitor Manuel Alves Duarte, Professor Auxiliar, DI-FCT-UNL

	Júri:
Presidente:	Prof. Doutor Pedro Manuel Barahona
Arguente:	Prof. Doutor António Casimiro da Costa
Vogal:	Prof. Doutor Henrique João Lopes Domingos

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Outubro, 2013

Intrusion Tolerant Routing with Data Consensus in Wireless Sensor Networks

Acknowledgements

Ao longo da realização desta dissertação, várias foram as pessoas e instituições que me apoiaram. Dirijo às mesmas o meu mais sincero apreço e agradecimento:

Ao meu orientador, Professor Doutor Henrique João Lopes Domingos por toda a disponibilidade, por todo o aconselhamento, pela revisão e pelos valiosos contributos que contribuiram para que este trabalho tivesse o devido valor e contribuísse para o projecto no qual está integrado;

Ao meu co-orientador, Professor Doutor Vitor Manuel Alves Duarte por toda a disponibilidade e empenho em contribuir para este trabalho, dando valiosos conselhos e sugestões para o mesmo;

À Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, em particular ao Departamento de Informática, por ter sido a minha segunda casa ao longos destes últimos anos e que sempre proporcionou ferramentas excepcionais para o sucesso dos seus alunos;

A todos os meus colegas de trabalho com os quais ao longo destes últimos anos me fui cruzando, pela sua amizade, pelas discussões técnicas e apoio mútuo nestes anos importantes das nossas vidas;

Aos meus pais, por me terem dado a possibilidade de chegar até aqui, por me terem permitido escolher o meu próprio caminho e por terem acreditado em mim;

Aos meus amigos pela companhia nos bons momentos e pelo apoio nos maus, em especial à Sara Ralha pelo interesse demonstrado, pelo apoio, companhia, compreensão, confiança e sábios conselhos.

If you have an apple and I have an apple and we exchange these apples then you and I will still each have one apple. But if you have an idea and I have an idea and we exchange these ideas, then each of us will have two ideas.
– George Bernard Shaw

Abstract

Wireless sensor networks (WSNs) are rapidly emerging and growing as an important new area in computing and wireless networking research. Applications of WSNs are numerous, growing, and ranging from small-scale indoor deployment scenarios in homes and buildings to large scale outdoor deployment settings in natural, industrial, military and embedded environments. In a WSN, the sensor nodes collect data to monitor physical conditions or to measure and pre-process physical phenomena, and forward that data to special computing nodes called Syncnodes or Base Stations (BSs). These nodes are eventually interconnected, as gateways, to other processing systems running applications.

In large-scale settings, WSNs operate with a large number of sensors – from hundreds to thousands of sensor nodes – organised as ad-hoc multi-hop or mesh networks, working without human supervision. Sensor nodes are very limited in computation, storage, communication and energy resources. These limitations impose particular challenges in designing large scale reliable and secure WSN services and applications. However, as sensors are very limited in their resources they tend to be very cheap. Resilient solutions based on a large number of nodes with replicated capabilities, are possible approaches to address dependability concerns, namely reliability and security requirements and fault or intrusion tolerant network services.

This thesis proposes, implements and tests an intrusion tolerant routing service for large-scale dependable WSNs. The service is based on a tree-structured multi-path routing algorithm, establishing multi-hop and multiple disjoint routes between sensors and a group of BSs. The BS nodes work as an overlay, processing intrusion tolerant data consensus over the routed data. In the proposed solution the multiple routes are discovered, selected and established by a self-organisation process. The solution allows the WSN nodes to collect and route data through multiple disjoint routes to the different BSs, with a preventive intrusion tolerance approach, while handling possible Byzantine attacks and failures in sensors and BS with a pro-active recovery strategy supported by intrusion and fault tolerant data-consensus algorithms, performed by the group of Base Stations.

Keywords: Wireless Sensor Networks (WSN), Routing Services and Protocols for WSN, Reliability, Consensus Protocols, Intrusion Tolerance, Simulation Environments for WSN.

Resumo Alargado

As Redes de Sensores Sem Fios (RSSF) são compostas por pequenos dispositivos computacionais (sensores), que podem ser distribuídos por uma grande área geográfica, de modo a medirem e transmitirem dados relativos às condições actuais ou acontecimentos físicos no meio ambiente envolvente. Os dados recolhidos pelos sensores são então encaminhados para nós especiais chamados Estações de Base (que por sua vez poderão estar conectados a outros nós onde executam as aplicações).

As RSSF de grande escala podem envolver uma enorme quantidade de nós, possivelmente operando em vasta áreas geográficas e em condições limitadas de acessibilidade. Estas redes funcionam como redes auto-organizadas e sem supervisão humana, tendo por base arquitecturas de rede com encaminhamento ponto-a-ponto (ou sensor-a-sensor). Os sensores são em geral muito limitados nos seus recursos de computação, memória, comunicações e energia. Estas limitações tornam a concepção e o desenvolvimento de serviços e aplicações para estas redes um grande desafio. A abordagem de serviços de segurança e fiabilidade, num contexto de RSSF confiáveis, torna este exercício ainda mais exigente. Sendo os sensores limitados nos recursos acima referidos e podendo ser materializados em dispositivos computacionais miniaturizados, tal permitirá que se criem condições que os tornem muito baratos no futuro. Desta forma, uma hipótese que permite endereçar soluções resilientes com tolerância a falhas ou intrusões pode ser baseada na adopção de redes com um grande número de nós, com replicação de dados e processamento. Uma RSSF segura terá de oferecer suporte a confiança na rede, bem como requisitos de segurança suportados por serviços de encaminhamento seguros tolerantes a falhas ou intrusões. A abordagem de soluções deste tipo para RSSF de grande escala tem sido recentemente uma área com bastante interesse por parte da comunidade científica.

O objectivo principal desta dissertação é o de propor, implementar e testar um serviço de encaminhamento seguro e tolerante a intrusões para RSSF *multi-hop* de larga escala. Este serviço é baseado num algoritmo de encaminhamento com múltiplas rotas e estruturado em árvore, estabelecendo múltiplas rotas disjuntas *multi-hop* entre os nós e um conjunto de Estações de Base ou nós de recepção e agregação de dados. As rotas são descobertas, seleccionadas e estabelecidas durante o processo de auto-organização da rede, podendo ser estabelecidas e mantidas com base em optimização e balanceamento de métricas específicas, tais como: condições de cobertura da rede, latência, consumo de energia, distribuição de carga, fiabilidade ou níveis de tolerância a intrusões. As Estações de Base podem ser vistas como uma rede sobreposta com um grupo de nós confiáveis que processam serviços de consenso sobre os dados recebidos para tolerância a intrusões.

A solução apresentada permite aos nós da RSSF obter e transmitir dados através de múltiplas rotas disjuntas para as diferentes Estações de Base, com uma tolerância a intrusões preventiva. Ao mesmo tempo, lida-se com possíveis ataques ou falhas bizantinas nos sensores ou nas Estações de Base através de uma estratégia de recuperação pró-activa, suportada por um serviço de consenso de dados distribuído e tolerante a falhas e intrusões.

Palavras-chave: Redes de Sensores Sem Fios (RSSF), Serviços de Encaminhamento para RSSF, Confiança, Protocolos de Consenso, Tolerância a Intrusões, Ambientes de Simulação para RSSF

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context and Motivation

Wireless Sensor Networks (WSN) are formed by a set of small devices (sensor or motes[1]) distributed across the space, where each sensor node communicates with the others through a wireless communication environment. These networks can operate without human intervention, being deployed as autonomous systems and forming self-organized or ad-hoc networks.

Sensor nodes have some important limitations: they can communicate just with their (short-range) neighbours, have small computational processing capacities, low memory resources and limited energy (sometimes a non-rechargeable finite resource).

WSN became an interesting field in the research community. Because of its characteristics [1], this kind of networks are adequate to a enormous set of applications, like: military monitoring in battlefields or intrusion detection in military areas [2]; critical industries vigilance [3]; biomedical indicators monitoring [4]; nature habitats monitoring [5] and environmental conditions [6]; control of vulcanological and seismological phenomenons [7]; infrastructures monitoring in civil engineering [8]; control and vigilance in people and goods location [9].

However, there are important security issues we must be aware as the applications supported in the network could require. In this way, WSN should support flaws or attacks that can occur at communication level or caused by intrusions in the nodes.

Indeed, in different scenarios of applications managing critical data, goods, lives and livelihoods may depend on the timeliness and correctness of the sensor data obtained

[1]A node in a wireless sensor network that is capable of performing some processing, gathering sensory information and communicating with other connected nodes in the network.

from dispersed sensor nodes working without human supervision. As a result, such WSNs must be secured to prevent an intruder from obstructing the delivery of correct sensor data or correct node level processing, and from forging sensor data, that will cause major damage to the supported applications. To address the problem, end-to-end data authentication, confidentiality or integrity protection, as well as, reliable and intrusion-tolerant post-processing of sensor data must be guaranteed, to identify and to correct forged sensor data.

Within the scope of this thesis, we are particularly interested in the security and reliability concerns of large-scale networks (in the magnitude from hundreds to thousands of sensor nodes), as two complementary dimensions of dependability solutions for WSNs. In this context, the main objective is to design, to implement, and to assess, with an experimental environment, a secure intrusion tolerant routing service for large scale WSNs.

The proposed solution combines multiple disjoint routes, selected and established in an ad-hoc way over multiple Base Stations and data consensus mechanisms performed by those Base Stations as a mechanism to support intrusion tolerance properties. Thus, the design of the routing protocol follows a resilient approach using disjoint multi-path routes established from each sensor to each different Base Station (BS), as a preventive intrusion tolerant approach, constructing forwarding tables at each node to facilitate communication between sensor nodes and the multiple base stations. These routes are formed when the WSN is in its self-organization process, following an ad-hoc model. Data received by the multiple base stations are subjected to a data-consensus verification mechanism implemented by a pro-active intrusion-tolerant consensus protocol. After this verification, data can be used safely by the final applications.

The main idea behind the solution is to minimize computation, communication, storage, energy consumption and bandwidth requirements at the level of sensor nodes, at the possible expense of increased computation, communication, storage, energy availability and bandwidth requirements at the Base Stations running the data-consensus algorithms.

The present thesis is integrated in a task of the SITAN (Services for Intrusion Tolerant AdHoc Networks) research project [10], which is funded by the *Fundação da Ciência e Tecnologia, MCES* (ref. PTFC/EIA/113729/2010), inserted on the goal of the development of data dissemination services resilient to intrusions in ad-hoc networks.

SITAN project is developed at CITI *(Centro de Informática e Tecnologias da Informação)*, FCT-UNL *(Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa)* in collaboration with LASIGE *(Laboratório de Sistemas de Informação de Grande Escala)*, FC-UL *(Faculdade de Ciências da Universidade de Lisboa)*.

1.2 The thesis problem

Security is critical for a variety of WSN applications and deployment environments, such as home security monitoring, critical infrastructures monitoring or military deployments. In these applications, each sensor node is highly vulnerable to many kinds of attacks, both physical and logical, due to each node's cost, energy limitations, wireless communications and exposed location on the field. In large-scale scenarios, security requirements are particularly emphasized, considering certain operation criteria of WSNs as ad-hoc or autonomic distributed systems, working with no human supervision. As a result, mechanisms to achieve security concerns (including both fault and intrusion tolerance) are necessary for sensor networks, regarded as a dependable distributed system.

Although intrusion and fault tolerance has been extensively studied in the context of wired networks, ad-hoc wireless sensor networks introduce a combination of threats that are not usually faced by wired networks, or considered by their adversary and failure models. The intrinsic radio broadcast nature of the wireless communication medium significantly enhances the capabilities of an adversary to eavesdrop, tamper transmitted packets, inject malicious packets or to initiate denial-of-service (DoS) attacks. These susceptibilities also apply to wireless LANs such as IEEE 802.11 and mobile ad-hoc networks. However, sensor nodes are highly resource constrained, with limited energy lifetime, low-powered micro-sensors and actuators, using slow embedded processors, limited memory, and low-bandwidth with short-range radio communications. This limits the ability for sensor nodes to perform heavyweight public key cryptography, though elliptic curve cryptography offers a promising course of possible application in the future. The relatively weak defences of sensor nodes are susceptible to external attacks by adversaries equipped with more powerful computing and communication equipment, used in the WSN deployment area. But more important, sensor nodes are distributed in the field without physical security monitoring conditions, as available to most wired, wireless LANs and other forms of wireless networks. As a result, WSNs are highly susceptible to the physical compromise and possible capture of one or more sensor nodes. Once compromised, the sensor node(s) are used as "internal" malicious nodes, exploited by intruders to damage the WSN through DoS, jamming, spoofing, malicious routing processing and several other attacks as better explained on Chapter 2.

1.3 Objectives and focus

Considering a conventional WSN software stack, the security problems cover different levels of approach:

- Physical layer protection;
- Data-link layer security, to avoid medium-access control vulnerabilities and to protect wireless communication from external attacks;
- Network-level security services including secure network discovering and ad-hoc organization services, as well as dependable routing protocols and services supporting reliable and intrusion-tolerant data dissemination;

- Application-level security issues, including secure and reliable data aggregation and other in-network processing capabilities.

The main focus of the thesis is on the security solutions for dependable WSN at a network-level approach. The objective is particularly focused in the research of novel solutions for intrusion tolerant routing services for dependable WSNs operating in large scale environments as multi-hop systems that collect and forward sensor data to information sinks, usually implemented by Base Stations (BSs) or syncnodes with more resources and data-processing capabilities than the sensor nodes. Syncnodes and BSs also act as gateways to WLANs or wired Internet environments, where final applications are running to finally process the data received from the WSN. In this vision, WSNs may be regarded as monitoring islands with primary sensing, routing and processing capabilities, interconnected with more complex and scalable monitoring infra-structures composed by different WSNs, used in the context of distributed applications.

As a possible application scenario, we can anticipate applications inspired in monitoring events with measurement of values in the WSNs and then transmitting the values through the WSN to the syncnodes or BSs. We consider applications with "soft real-time" constraints, adapted to the communication latencies imposed by the multi-hop routing structure and communication characteristics of WSN standards (IEEE 802.15.4 [11] and/or Zigbee [12], working in ad-hoc settings with Medium Access Control based on CSMA/CA[2]).

In the above scenario, the communications pattern in the WSN is relatively simple when compared to a traditional wired or conventional ad-hoc wireless network. Data transmission in the WSN itself is dominated by local communication between sensor nodes in a limited transmission range, and multi-hop forwarding between sensor nodes and the multiple syncnodes or BSs. Primarily, data is transmitted from sensor nodes to one or more Base Stations or syncnodes. In general, the number of Base Stations in a WSN will be significantly less than the amount of sensor nodes. For this thesis purposes, the base stations are relatively resource-rich when compared with the sensor nodes, in terms of processing, storage, energy, and communication capabilities. The large number of resource-constrained sensor nodes and the small number of resource-rich base stations collectively form an asymmetric network. In this network, the group of syncnodes and Base Stations may be also regarded as an overlay.

While other sensor network architectures and routing protocols for those architectures have been proposed, our focus in the thesis is on the common asymmetric tree-structured multi-path routing architecture associated to this overlay vision.

[2]Carrier Sense Medium Access with Collision Avoidance

1.4 Approach to the system and adversary models

According to our objectives, the dependable routing service constructs a secure and efficient tree-structured intrusion tolerant routing network for WSNs, adapted to the asymmetric architecture and resource constraints, between sensor nodes and Base Stations or syncnodes, as introduced above. A key objective is to circumvent (with a preventive intrusion tolerance approach) the possible damage caused by an intruder who has compromised one or more deployed sensor nodes. Such an intruder could inject, modify, or block data packets. The routing service therefore is designed to prevent from these intrusions, limiting the ability of an intruder to cause mischief through a combination of distributed lightweight security mechanisms, in which the possible use of multiple disjoint routes is a key strategy.

Complementarily, we consider that BS nodes can be also attacked by intrusion, causing incorrect processing at the Base Station level, forging or faking data flowing from the WSN to the final applications.

The scope of the mechanisms studied in this thesis is bounded in the following research directions. We explore a solution for securing upstream data traffic – from leaf sensor nodes, through the multi-path tree-structured routing topology, until one or more Base Station sinks. Peer-to-peer secure communications among sensor nodes is out of the scope of our goals, and is not viewed as usual for flat peer-based networks. Downstream traffic beyond what is needed to securely set up upstream routing trees (during the network's configuration) is not a focus of our work itself. We consider the adoption of efficient and well-adapted light-weight cryptographic mechanisms and algorithms, as well as key-distribution protocols to establish pair-wise keys as solutions that can be devised, subjacent to the routing service itself.

Another assumption in our network model is that sensor nodes can have only limited mobility after their initial deployment, which we believe to be the commonest case in many situations of large scale WSNs. The topology discovery and network organization process and set up is designed to be rerun periodically, in order to update changes in the topology due to faults, to refresh the topology and network organization; the same process can be applied to accommodate "limited mobility". Continuous or strong mobility concerns during and after set up is out of the scope of this thesis.

In our system and adversary model we consider that intrusions (or failures) following a Byzantine setting are possible in sensor nodes and Base Stations. However, we consider that these possible intrusions are independent; thus we don't support an adversary that simultaneously attacks sensor nodes and Base Stations (or in a selective way controls the entire topology, by controlling a number of sensor nodes and by also controlling a number of Base Stations). In this way, we consider that an adversary can only randomly compromise a limited number of sensor nodes somewhere in the topology or a limited number of Base Stations.

1.5 Thesis Goals

The objective of this thesis will be the experimental assessment of the intrusion tolerant routing service, namely based on the multi-path tree-based routing strategy and data consensus mechanisms performed by the Base Stations overlay. The assessment of consensus mechanisms based on probabilistic consensus algorithms in this case is one important goal of the thesis.

For the experimental assessment, a simulation environment has been used. For this purpose, the WiSeNET simulation environment [13] is considered. This simulation environment has been developed and used in the scope of the SITAN project [10]. However and because of the characteristics of these networks (including the huge number of sensors, usually in the magnitude of thousands), it's hard to know if the simulation results are reliable (ie., if in a simulated environment the output is equal to an hypothetical output obtained from a real network configured in the same way). The results of this thesis will help to obtain real outputs from a real network, in order to calibrate the simulation environment (concerning the consensus mechanisms), as a future work; this way the obtained results from the simulator will be more reliable as they are based in real values. With this approach, this thesis will drive an hybrid simulation environment in which certain nodes of the WiSeNet simulator will be implemented as virtual nodes representing Base Stations. The evaluation of the consensus protocols running at this level is considered by implementing these algorithms in real nodes, materialized by Rabsperry Pi [14] platforms.

1.6 Thesis Contributions

The contributions of this thesis are:

- Design of the dependable routing service for large scale WSNs, based on a multi-path tree-based routing protocol (called MINSENS++) and its integration with the consensus overlay composed by multiple sync nodes;
- Design, implementation and assessment of consensus mechanisms in the WSN context, as discussed above in the introduction of the system and adversary models;
- Assessment approach by using an hybrid simulation environment, where the Base Stations are implemented with real nodes (Raspberry PI), represented in the WiSeNet simulator as virtual nodes; In this way, BSs or Syncnodes will act as real nodes.

1.7 Organization and document's structure

The next chapters are organized as follows:

- In chapter 2, WSN are explained in a more detailed way. This includes WSN applications, related software, routing and security;
- In chapter 3 is presented the related work already done and the state of the art. This includes the simulation of WSN and specific routing protocols developed recently for WSN and inspiring the desired purposes for this thesis;
- In chapter 4 a System Overview is done, including the contextualisation of the previous contributions.
- In chapter 5 the MINSENS++ protocol is presented and specified.
- In chapter 6 the Multi-Valued Consensus (MVC) and Turquois protocols are presented and specified.
- In chapter 7 the results of this thesis are presented.
- In chapter 8 the conclusions of this thesis are discussed.

2

Wireless Sensor Networks

In this chapter, Wireless Sensor Networks (WSN) will be discussed, focusing on its applications, supporting software, topologies, routing and security.

2.1 Wireless Sensor Networks (WSN)

Wireless Sensor Networks (WSN) are radio frequency communications' networks using the standard IEEE 802.15.4, ZigBee [12] as better explained in 2.2. These networks are deployed by using devices (called sensors) that are cheap (expected to cost about 1 US$) and small (from tens of cubic millimetres to tens of cubic centimetres). These sensors are very limited in computational, energetic and communication capabilities [15]. They are deployed along some geographic area, forming a communication network (that can be more dense or sparse) taking as advantages the price and the easiness of deployment of the sensors. The involved costs of deployment are usually insignificant and for it contributes the fact that the network must be able to self-organize and don't need a supervision during its work [16].

On a WSN, sensors cooperate among them in order they can produce the desired work. This work can be (as already stated in 1.1) the result (in form of events) of monitoring of/interaction with physical environment (measuring the involved physical variables).

A sensor node has a microprocessor, a radio communications circuit (IEEE 802.15.4) and one or more sensors (environment variables measurement units) with capabilities to measure and process some kind of external events: temperature, humidity, pressure, noise, light, etc. These sensors can also be applied to medicine, with the external events being the physiological indicators or vital signs of the human body.

Events are detected and measured by sensors and then the information is disseminated over the network (optionally, before being sent, data can be preprocessed). The network usually has a multi-hop topology (single-hop in some cases [17]) and along the network can exist some intermediate nodes that process and aggregate data sent by source sensors. In the end, data is collected by Base Stations (BSs) that are responsible for the aggregation of all the data. Base Stations are special nodes that usually have less limitations than the sensor nodes; this includes better computational, energetic, storage and communication capabilities. These BSs are usually connected in some way (for example a TCP/IP wired or wireless network, or over Internet) between them and to systems where applications are running and waiting for the data collected by sensors.

2.2 Software Architecture in WSN

The Software Architectures in WSN are usually structured considering the specific requirements of the software being developed. It can even happen that different layers could merge, in order to achieve a better performance meeting specific requirements.

However it is possible to present a generic reference model as the one presented in figure 2.1, representing the communications stack supported in a sensor node platform [18, 19, 20, 21, 11, 22]. It can be decomposed in:

- **Application —** Refers to the specific application software running on WSN nodes;
- **Network —** Refers to the routing protocols used by sensors;
- **MAC (Media Access Control) —** Refers to the media access control implemented by the hardware present in the sensors;
- **Physical —** Refers to the physical used medium (in the case of WSN, the air), managing the emitter/receiver of radio waves and selecting appropriate communication channels;

The **security** layer is transversal to all the other layers, since we're interested in having security in all the above layers.

IEEE 802.15.4 [11] norm specifies the physical and MAC layers, intending to make nodes communicate in dozens of meters; because of its short ranges, this communications are called Wireless Personal Area Networks (WPAN). This norm tries to satisfy the requirements of this kind of networks, being energy-aware and obtaining minimal operation costs with technological simplicity.

Considering the WSN security stack as represented in the Figure 2.1, on top of the MAC-DataLink Layer can be found a base secure wireless communication layer, with different solutions from the industry (e.g. IEEE 802.15.4 security services [23] or TinySec [24] or from research implementations like MiniSec [25]). These base secure sensor network communication protocols need to provide three basic properties: data secrecy,

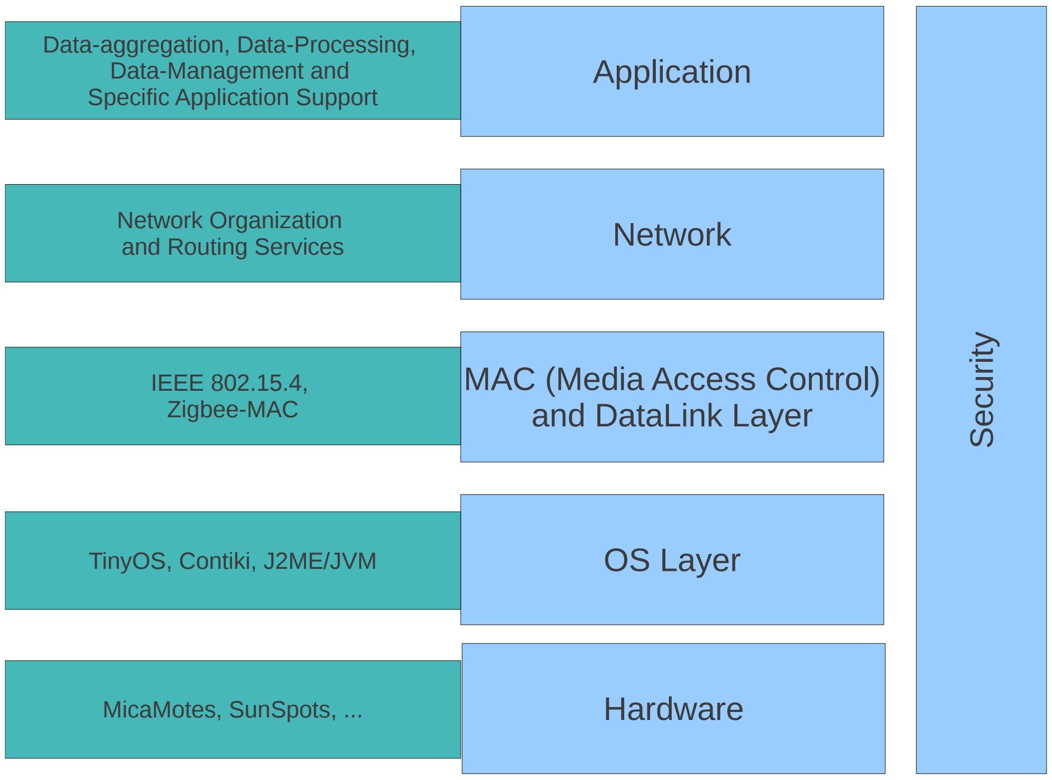

Data-aggregation, Data-Processing, Data-Management and Specific Application Support
Application
Network Organization and Routing Services
Network
IEEE 802.15.4, Zigbee-MAC
MAC (Media Access Control) and DataLink Layer
TinyOS, Contiki, J2ME/JVM
OS Layer
MicaMotes, SunSpots, ...
Hardware
Security

Figure 2.1: Stack for WSN Software Support

authentication, and replay protection. Secure sensor network link layer protocols such as TinySec and ZigBee benefit from a significant attention in the research community. However, TinySec achieves low energy consumption by reducing the level of security provided. In contrast, ZigBee provides high security, but suffers from high energy consumption. MiniSec is a secure network layer that obtains the best of both worlds: low energy consumption and high security. MiniSec has two operating modes, one tailored for single-source communication, and another tailored for multi-source broadcast communication. The latter does not require per-sender state for replay protection and thus scales to large networks.

In the context of this thesis, these security layers are not in the focus of our expected contributions and we will reuse a specific solution implementing MiniSec primitives, as the main base security abstraction level to support the routing services. In the context of the SITAN project, a publicly available implementation of MiniSec for the Telos B platforms has been used. Experimental results [26] demonstrate that it is a good solution (compared with the other solutions above), for the balance optimization between security and energy savings.

For this thesis, the developed work will be focused on Network layer. It is expected that the MAC layer already offers secure communications mechanisms (including the cryptographic keys management and distribution). The idea of this project is to propose, implement and assess an intrusion tolerant routing service for WSN inspired by existing routing protocols [27, 28, 29, 30, 31]. An analysis to these protocols is made in section 3.2. General considerations about routing protocols for WSN are made in 2.4.

2.3 WSN for large scale topologies and environments

Wireless Sensor Networks are supposed to be used in large-scale environments (thousands of nodes) in order to cover large geographic areas. Obviously some properties must be verified, like the density of nodes and their distribution. The number of nodes must be sufficient to cover the desired area and must be properly distributed along the field in order to comply with the communications level limitations.

In such environments (large geographic extensions or locations hard to reach), it is fundamental that the network can organize itself (at start or at each reset, by establishing connections and routes between nodes accordingly to some criteria of discovery and self-organization) and operate in an autonomous way, without any kind of supervision. During the organisation process and the normal operation, the network must also be able to adapt itself to the physical environment where is placed (for example, deal with signal reflections, environmental conditions, hidden terminal problem, etc.).

The communication model consists in the multi-hop routing of the data generated by nodes, through various nodes of the network until the data aggregation nodes (Base Stations). Although the number of nodes generating data is of the order of thousands of nodes (as already stated), the Base Stations are in small number (usually not more than from one to some dozens). So that the energy consumption can be reduced (increasing the network lifetime), some filtering to the sent data shall be applied; this filtering can be done in the source node (through some preprocessing, like sending the mean of some measurement instead of all reads) or in intermediate nodes (through some kind of processing by these nodes).

2.4 Routing Protocols

An ad-hoc network consists of a collection of wireless nodes, each node communicating directly with other nodes within its transmission range. Communications between out-of-range nodes have to be routed through one or multiple intermediate nodes (hops). Since nodes may be mobile, the routing protocol should be able to handle rapid topology changes.

Wireless Sensor Networks are considered as a particular case of ad-hoc networks, where the nodes are extended with sensing capabilities. This leads to a network composed by one or multiple Base Stations or syncnodes (acting as sinks) and many tiny and low-powered sensor nodes. Both networks share in common many properties, such as the self-organisation, energy efficiency, wireless multi-hop communications and the usage of CSMA-CA technique for MAC level.

A Wireless Sensor Network differs however from an ad-hoc network in several aspects:

- **Equipment —** A node in an ad-hoc network is usually a more powerful device in terms of available resources (like laptops or PDAs), while in WSN a node is typically a smaller device with a low-speed processor, limited memory and a short-range transceiver;

- **Software stack —** An ad-hoc network offers an uniform software stack for different applications, possibly with multiple applications running over the same stack. In a WSN the software stack offers the more low-level common abstractions (physical and data-link IEEE 802.15.4) and all the rest is left to application-specific requirements;

- **Energy —** WSN have tighter requirements in terms of energy. This is usually a finite resource in many large-scale outdoor environments;

- **Mobility —** In ad-hoc networks the nodes are usually mobile while in WSN the sensors are not (or have a limited mobility);

- **Environment interaction —** Although more recent applications for ad-hoc networks can interact with the environment (like participatory sensing applications), this interaction is usually absent in these networks. In the other hand, the interaction with the environment is the core of WSN.

- **Communication throughputs —** Theoretical throughputs are 250 Kbps for WSN (IEEE 802.15.4) while in ad-hoc networks tens to hundreds of Mbps can be achieved (IEEE 802.11).

Considering the previous points, the differenced characteristics of the WSN add strong constrains to the routing protocols, namely in terms of correctness, evolving and performance, as well as the supported security properties.

Routing protocols (network layer in Figure 2.1) have thus an essential role in Wireless Sensor Networks. In fact, they have the responsibility to make all the nodes in the network able to communicate with all others (obviously in a non-partitioned network) through multi-hop in a simple, efficient and energy-aware manner. In Section 3.2, WSN specific protocols are analysed in a detailed way; in this section, we'll talk about general concerns of these protocols.

In the scope of the ad-hoc networks, routing protocols are usually divided into two large categories (as studied in the following sections):

- Proactive Routing (2.4.1);

- Reactive Routing. (2.4.2).

2.4.1 Proactive Routing

Proactive routing protocols (also called as table-driven protocols) use pre-established routing tables in order to forward the packets. In these protocols, nodes constantly exchange information in order to maintain the routing tables always updated (even when there's no communication requests). DSDV (Destination-Sequenced Distance-Vector) [32] and OLSR (Optimized Link State Routing) [33] are examples of table-driven routing protocols used in ad-hoc networks.

This kind of protocols are usually divided into the following main steps:

1. Nodes discovery and self-organisation;
2. Routes discovery;
3. Routes selection;
4. Routes maintenance.

Step 1 refers to each node discovering the existence of other nodes in its physical neighbourhood and organising itself in order to establish, for example, cryptographic keys with each of the neighbours. Each node sends HELLO messages to the others, in order to introduce himself in the network.

Step 2 refers to the process of (for each node) discovering which possible routes exist between the node and the Base Station(s).

Step 3 is the selection of the output produced by step 2; the previous step usually finds a large set of possible routes, and the node is usually interested in just a subset of them, accordingly with defined priorities (for example, the n routes with less hops). In the end of this step, the network is operating normally.

Step 4 refers to a step that is always present during the normal operation of the network. Nodes can fail, a pair of nodes belonging to a route can no more be able to communicate with each other (for example, due to change of weather conditions) or even after a certain amount of time operating, cryptographic keys can be cracked. This way, the routes should be constantly monitored/maintained and managed. One possibility is to start the process from the beginning (reset and start from step 1) that is specially suitable for cases like the crack of cryptographic keys.

The overhead created by all these steps and by the routing tables management compensates in applications where data is sent by nodes in a regular or permanent way. Some examples are applications for permanent monitoring of events or regular measurement of physical phenomenons.

2.4.2 Reactive Routing

Reactive routing protocols (also called as on-demand protocols) are, as the name says, protocols that are reactive, acting just when necessary in order to forward the packets.

For the support of multiple applications, based in non-regular events dissemination, on-demand protocols can be better as the nodes just exchange informations when there are communications to establish. This way, routing tables are not necessary and routes selection processes are started when there is data to send over the network. This avoids the overhead of routing tables management and maintenance.

AODV (Ad-hoc On-demand Distance Vector) [34] is an on-demand protocol and DSR (Dynamic Source Routing) [35] is a variant, with source-routing policies.

On-demand protocols are widely used in ad-hoc networks but mostly in mobile networks, whose nodes can move difficulting the maintenance of links states. In our case study (WSN), sensors are statically placed and are sending regularly measured data. In this environment, there is no sense in implementing on-demand routing protocols as routes reorganization, discovery or selection and evaluation of new routes will have no advantages, considering energy consumption and latency in the network.

In order to evaluate the most appropriated routing category, it is important to take into account the context of the secure routing service for WSN presented on this dissertation. It is important to consider the particularities of the ad-hoc networks in our context (considering WSN as a particular case of them), namely the autonomous operation and the non-mobility of the nodes. In the scope of this work, we are particularly interested in routing protocols with proactive characteristics, for plain topologies (as will be discussed in 2.4.3).

2.4.3 Routing topology

Apart of being table-driven or on-demand, there are another typologies to consider.

In terms of centralization, a routing protocol can discover the routes in a centralized or distributed manner. On the centralized discovering, nodes ask Base Stations (or other special nodes) about routes. This way, Base Stations are the nodes responsible for routes discovery and computation, based in each node neighbourhood's information. On the distributed discovery, each node is responsible for create, compute and maintain (update) its routing table.

In terms of organization of the routes in the network, there are three modes:

- Geographical routing [36];
- Cluster-based routing [37, 38];
- Plain routing.

Geographical routing requires that each node knows its own location as well as the location of the destination nodes where to it will send data. The route (and thus the next hop) of some packet is determined in the moment that the packet is received and depends just on the geographical information of the destiny. This way, nodes do not need to store any information about routes. Examples of protocols of this group are the SIGF [36] protocols family.

Cluster-based routing aggregates the nodes in clusters, having each cluster a representative node called cluster-head. Nodes of each cluster send the packets to their representatives (cluster-heads) which in turn forward the packets between cluster-heads until their destinations. Destiny is usually a Base Station or another node in the network - regular node or another cluster-head. This kind of protocols can have energy-aware properties, as they can define as cluster-heads those nodes with more energy at a given moment; after a certain amount of time, the cluster-heads can be exchanged with other nodes that have then more energy. Cluster hierarchies are allowed, meaning that some cluster-head can itself be a member of a cluster with another cluster-head of a superior level. Some examples of this kind of protocols are LEACH [37] and PEGASIS [38].

Plain routing protocols do not require any kind of structure in the network; who develops the protocol is totally free to define the organisation of nodes and how routes are generated. This way, some protocols choose to structure the network in some way while others choose to use the network without any specific structure. While in a non-structured network is not necessary, in a structured network some rules must be followed in order to construct the network. For example, in a tree organized network, the sink node (usually a Base Station) is placed as the root of the network; each of the other nodes must be organized in such way that at least one root exists above it. The tree organization leads to a faster wastage of energy in the topper levels nodes.

This dissertation addresses the secure routing problem with intrusion tolerance characteristics in ad-hoc WSN; in this setting, we are particularly concerned on routing protocols following the randomly generated plain topology routing model. Within this idea, we are particularly interested in protocols with the following characteristics:

- Exploit of redundancy in order to tolerate intrusions without any need for detecting the nodes where intrusions have occurred. This will allow the network to operate correctly in the presence of undetected intrusions;

- Perform all the heavy computations at the BS level, minimising the role of sensor nodes in building routing tables or dealing with security and intrusion-tolerance issues. This allows to minimise the need for computation, storage, bandwidth and energy capabilities at the sensor nodes;

- Limit the scope of damage done by undetected intruders, limiting the flooding and using appropriate authentication mechanisms (like symmetric-key cryptography).

As related examples of the approach to the goals of this dissertation, protocols with these characteristics are presented in Chapter 3.

2.5 Security and Reliability

Security services, techniques and mechanisms usually used in conventional networks (or even in general purpose ad-hoc mobile networks) can not be used in WSN due to their characteristics and limitations. Considering all the physical characteristics, wireless communications properties and application scenarios, security mechanisms must be present at all stack protocol's layers.

WSN are usually deployed in geographical areas without any kind of supervision; this factor leads us to the risk of physical attacks to the sensors. Plus, the interaction of these kind of networks with the environment increases vulnerability, as sensors are exposed to all kind of adversities from the environment.

For a network to be secure, the degradation of the operation must be at most proportional to the number of attacks (this is called graceful degradation). Obviously, a network will be as better as how much it can resist to the fail or attack situations; the more immune the network is to affected nodes (maintaining global connectivity), the better service it will provide. A secure routing service should therefore guarantee the network connectivity, reliability and preferably that the dissemination is made in optimal conditions (relative to latency and energy consumption). In order to reach this purpose, various security services from different stack's layers should complement each others; it is also important to be conscious that security services must have adequate communications and computational complexity for WSN.

The security focus of this thesis will be held in the network layer. It is important however to understand what guarantees are offered by lower layers, as MAC. This topic is studied in 2.5.2.

2.5.1 Attacker's model

Attacker's model defines the (possible) behaviour of an attacker, in such a given system. Attacker's model for WSN is defined by Karlof-Wagner [39] and the following description is based in that definition.

In a WSN an attacker can be active or passive. A passive attacker (similar to what is defined by Dolev-Yao [40]) just intercepts communications in order to know the information that is being transmitted. On the other hand, an active attacker (with some characteristics from Dolev-Yao and from OSI X.800 Framework [41]) is able to change the normal behaviour of the network for example by changing messages' data, posing as various other nodes, dropping all the messages he receives, sending a node's information to other not in its neighbourhood, etc. Passive attacks are more difficult to detect, as such attacks do not destroy the operations of routing protocols. A passive attack typically involves only eavesdropping[1] the routing traffic, in order to discover valuable information; two possible solutions to restrain eavesdropping can be the adoption of encryption in the application layer [42] or to transmit parts of a message over multiple disjoint paths and reassemble them at the destination [43].

An attacker can also be internal or external. An internal attacker attacks from inside

[1]Eavesdropping is the act of secretly listening to the private conversation of others without their consent.

of the network, while an external attacker acts from outside of the network. An internal attacker is able to capture any node from the network and obtain all the data within the node (communications history and cryptographic keys). The captured node(s) still belongs to the network (as it still has its credentials) but can have a behaviour defined by the attacker. This way, the infected node can have the same behaviour of an external attacker, with the privilege that is trusted by the other nodes. By changing the behaviour of a node, this intruder can attack accordingly to the Byzantine model. Security mechanisms relying on authentication or encryption may not handle internal attacks, since these compromised nodes also have the keys (and thus are treated as authorised parties in the network).

From outside of the network, an external attacker can see the traffic or block communications through jamming[2]. The external adversary can belong to one of two classes: sensor-class or laptop-class. A sensor-class attacker uses one or more sensors equals to those from the WSN to attack the network; that means he can only reach the neighbourhood of each attacker's sensor. A laptop-class attacker is equipped with a laptop computer, meaning that he has more computational power, memory storage, communication and energy than WSN's nodes; this way, this kind of attacker can reach a big part or even all the network to do more sophisticated attacks.

As an internal attacker is trusted by the network, it is the most dangerous attacker in WSN and that's why WSN need security services with intrusion tolerance.

Several possible attacks are presented in the following sections.

2.5.2 MAC Layer

The attacks made to MAC layer can follow the MAC protocol or not.

If following the protocol, the attacker acts as a legitimate member of the network. One example of possible attacks is DoS (Denial of Service) attack (through the flooding of maximum sized packets), causing an energy draining in correct nodes and decreasing the available bandwidth. Another example is the configuration of a node to act as if working without battery (infinite energy), what makes the pause time between retransmissions to be reduced, causing CSMA/CA mechanism to monopolize the access to the communication medium. These two attacks make the delivered packets ratio to be reduced [44].

If not following the protocol, the attacker can make a node not to use the CSMA/CA mechanism correctly. This way, collisions will exist in the communications and all the transmissions will be blocked. This attacker can also resend, modify or send false information (for example, false ACKs[3]).

[2]Radio jamming is the (usually deliberate) transmission of radio signals that disrupt communications by decreasing the signal to noise ratio.

[3]In data networking, an acknowledgement (ACK) is a signal passed between communicating processes to signify acknowledgement, or receipt of response, as part of a communications protocol.

2.5.3 Network Layer and Routing Attacks

An analysis to the network layer comprises the study of the network organisation and the data dissemination models (routing) (2.4). Another important aspect is the security concerns of the layer. Cryptographic keys distribution mechanisms are also considered; however, in this thesis context, it is assumed that a secure cryptographic keys distribution mechanism is already available.

The earliest developed routing protocols used to assume that all the nodes in the network had a correct behaviour, without any security concerns. More recent routing protocols are now aware of the attacker's model for WSN (2.5.1). One of the main concerns is to make intrusion tolerant routing protocols; this can be done in a preventive way or in a detection/correction way.

Preventive protocols use mechanisms in order to work in the best possible way in the presence of intruders. The detection/correction protocols use intruders' detectors and try to kick the intruder(s) out of the network. Concerning the network's organisation model and data routing, in 3.2 some routing protocols are analysed.

Accordingly with the attacker's model (2.5.1), we introduce below several possible attacks; some of them intend to modify the exchanged information while others intend to modify the network's topology. We divide these attacks into three classes: attacks on route discovery process, attacks on route selection process, and attacks after establishing routing paths.

Before the analysis to some of the existing attacks, the analysed ones are summarised in Figure 2.2. As we can see from the figure, some of these attacks are directed just to sensor networks, just to ad-hoc networks (composed by powerful devices) and others are directed to both [45]. In the scope of this thesis, we are interested just in the attacks directed to sensor networks (just to sensor networks or both ad-hoc and sensor networks). Although Base Stations are powerful devices when compared with sensor nodes, in our solution they are working in infrastructure mode (instead of an ad-hoc operation). Attacks directed just to ad-hoc networks (Rushing Attacks and RREQ[4] Flood Attacks) are not considered as their main purpose is to break the route discovery process of a pure ad-hoc network (composed by powerful devices) using an on-demand routing protocol.

2.5.3.1 Attacks on Route Discovery Process

The attacks directed to the route discovery process attempt to prevent legitimate nodes from establishing routing paths among them, for example by sending fake routing information. This kind of attacks will lead to a denial of service (DoS).

Fake Routing Information

An efficient attack against a routing protocol is to provide fake routing information, during its route discovery process.

[4]RREQ is an acronym used for *Route Request Messages*.

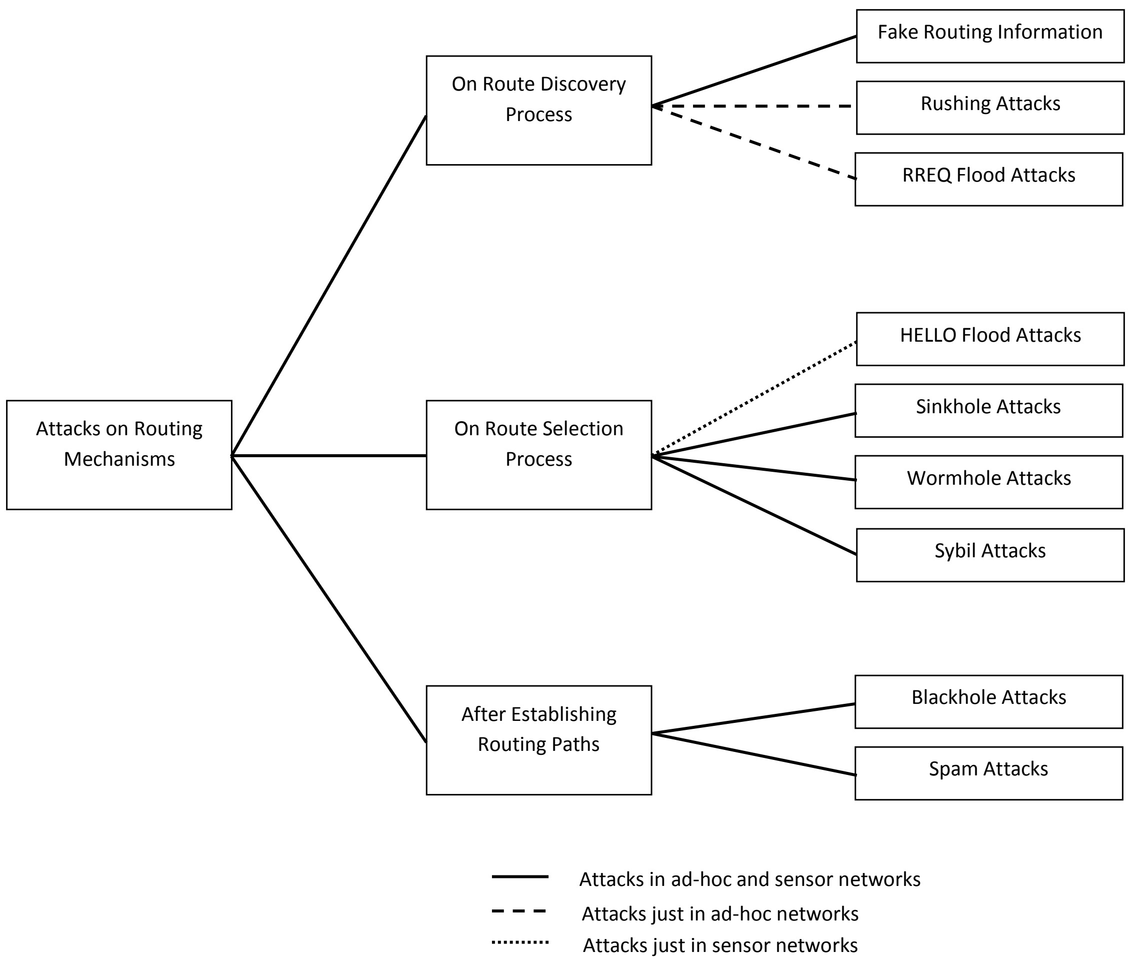

Attacks on Routing Mechanisms
On Route Discovery Process
On Route Selection Process
After Establishing Routing Paths
Fake Routing Information
Rushing Attacks
RREQ Flood Attacks
HELLO Flood Attacks
Sinkhole Attacks
Wormhole Attacks
Sybil Attacks
Blackhole Attacks
Spam Attacks
Attacks in ad-hoc and sensor networks
Attacks just in ad-hoc networks
Attacks just in sensor networks

Figure 2.2: Summary of the attacks analysed in the following sections

For table-driven routing protocols (the protocols family we are interested in the scope of this thesis), a malicious node can interfere with legitimate nodes by announcing incorrect routing information [46]. Such behaviour can invalidate the routing tables of the legitimate nodes, leading to the impossibility of communication among them. Some possible results are the partition of the network or the creation of routing cycles.

2.5.3.2 Attacks on Route Selection Process

The attacks directed to the route selection process attempt to increase the chance that malicious nodes are selected as part of the routes, by legitimate nodes.

By establishing routes through malicious nodes, they can overhear transmitted messages or improve the attacks presented in 2.5.3.3, leading to a disruption of the network operation.

HELLO Flood Attacks

Many of the routing protocols developed to WSN require nodes to broadcast HELLO messages, in order to announce themselves to their neighbours. The nodes receiving such messages assume that the senders are their neighbours, with a one-hop distance.

A laptop-class external attacker (2.5.1) may however violate this assumption. Such an attacker has a greater transmission power to broadcast HELLO messages, thus being able to cover a larger range of nodes. The receiving (legitimate) sensors will be convicted that the attacker is one of their one-hop distance neighbours.

Protocols relying on localised information exchange between neighbours for topology maintenance (or flow control) are vulnerable to this attack. Furthermore, the attacker can advertise higher quality or shorter routes, causing the nodes to follow these routes.

With this attack [15], most of the messages (from legitimate nodes) will not reach the attacker, since sensor nodes have a much smaller transmission power than the attacker and the network will be driven to a chaotic and confused state. An example of this attack is shown in Figure 2.3.

Sinkhole Attacks

The goal of a sinkhole attacker [15] is to attract all the messages from its neighbours to be routed through it.

With this attack, all the traffic from the area near the adversary will flow through it, creating a (metaphorical) sinkhole centred in the malicious node. These attacks are performed by making the attacker look especially attractive to its neighbours (respecting to the routing algorithm). An attacker can, for example, offer an extremely high quality route to a Base Station; alternatively, the malicious node may even announce itself as a neighbour of a Base Station, spoofing the legitimate nodes.

While in the HELLO flood attack the attacker uses a greater transmission power, the sinkhole attacker usually uses a normal transmission power, thus affecting just a certain

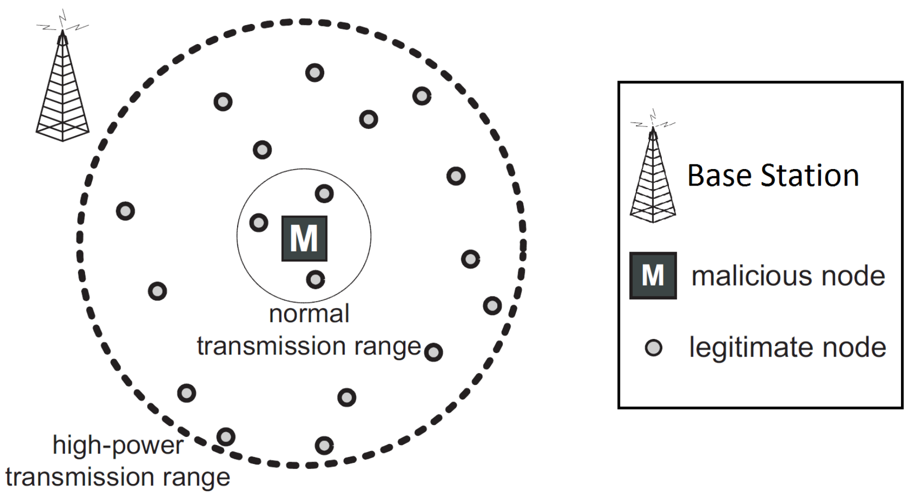


Figure 2.3: HELLO flood attack.

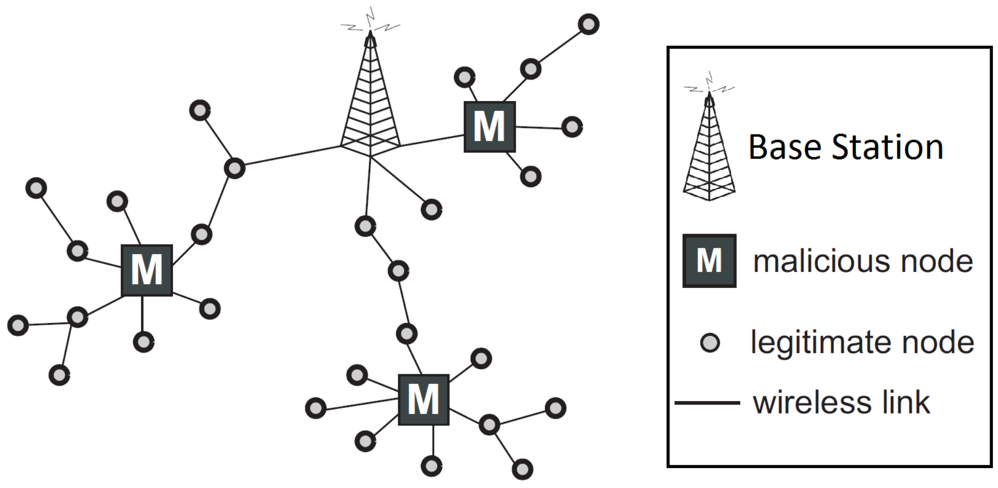


Figure 2.4: Sinkhole attack.

subset of the network. This attack is shown in Figure 2.4 and can be performed both by an internal or by an external adversary.

Wormhole Attacks

Unlike HELLO flood and sinkhole attacks, multiple malicious nodes can cooperate to generate attacks against the network during its route selection process; one of these attacks is the wormhole attack.

In a wormhole attack [47], two distant malicious nodes use an autonomous communication channel (available only to the attackers) in order to tunnel received messages from one side to the other. Packets transmitted through this wormhole tunnel usually have lower latency than those sent between the same pair of nodes over normal multi-hop routing (due to the number of hops). This causes the false appearance that routing

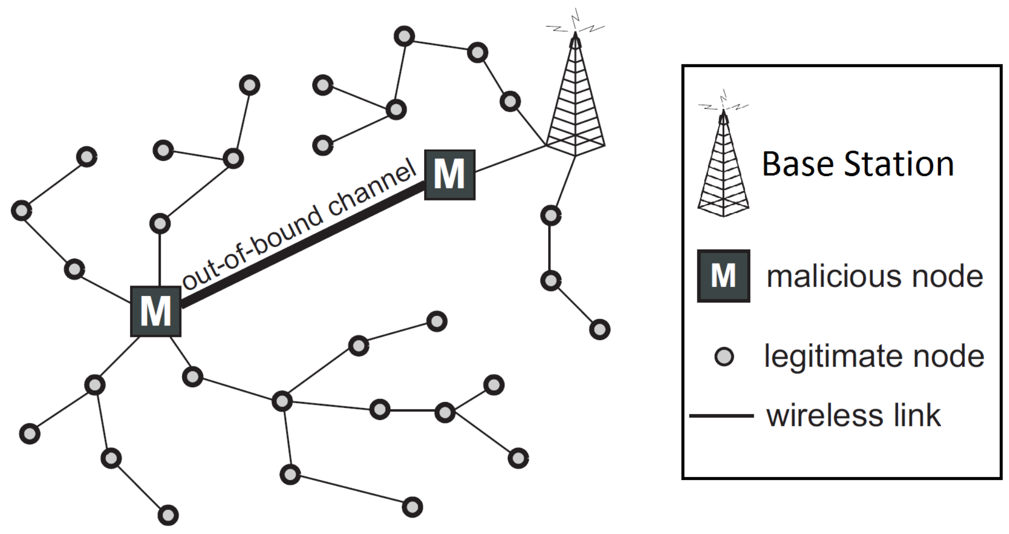


Figure 2.5: Wormhole attack.

through the malicious nodes is a better choice and the legitimate nodes will select the malicious nodes as intermediate nodes in their routes. Furthermore, wormholes can interfere in the network's topology, by relaying packets between two distant nodes and thus causing them to consider themselves as neighbours in a fake network topology.

WSN are very vulnerable to the wormhole attack due to two reasons. First, because external laptop-class attackers can use communication technologies not available to the sensors (like IEEE 802.11 or IEEE 802.3), offering low latency and high bandwidth tunnels. Second, because an adversary geographically close to a Base Station can control a lot of the routing, by creating a well-placed wormhole. Figure 2.5 exemplifies the situation in which the adversary is located close to a Base Station, and thus more than half of the sensor nodes guide their traffic through the wormhole tunnel.

Sybil Attacks

In a Sybil attack [48], an adversary disguises itself with multiple identities.

Such an attacker acts as multiple different nodes, by advertising to its neighbours a set of multiple identities. Furthermore, the adversary can create many fake (virtual) nodes and thus increase the probability of the malicious (real) node being selected by the other nodes as part of their routing paths. Another perverse effect of this attack is that it can also significantly reduce the effectiveness of fault-tolerance mechanisms, such as multiple disjoint paths routing [49, 50], making the adversary belong to two different routes due to its multiple identities. Other nodes will then treat the fake nodes as different nodes and establish different routes through the same infected node.

2.5.3.3 Attacks after Establishing Routing Paths

Once an attacker successfully inserts an infected node in the network (using for example techniques explained in 2.5.3.2), several attacks after the establishment of routing paths can be made. At this stage, source nodes have already established routes through the infected node.

Such a malicious node can unscrupulously drop all the packets passing through it, or modify their contents (if no encryption is used). Another common attack is to act as a source node (establishing routes to other nodes) and just send dummy messages, exhausting other nodes' energy and network's bandwidth.

Blackhole Attacks

The operation of networks based on multi-hop communications (such as WSN) must rely on the assumption that the participating nodes will cooperate on the forward of received messages.

In a blackhole attack [51], the malicious nodes violate such assumption. In fact, they drop all the received messages, preventing them from being propagated until their destinations (creating a metaphorical blackhole).

To prevent these attacks, many routing protocols have a *route maintenance mechanism* that removes from the network the nodes that are not propagating the messages. Such infected nodes are detected when a legitimate node finds that its next-hop neighbour is not routing the messages and warns the source to create another routing path. With such mechanism, a blackhole attack is trivial to detect; a trickier and more sophisticated attacker selectively forwards some of the packets [15], in order to cheat the source that the route is still alive and forwarding.

Spam Attacks

Spam attacks[5] [52] frequently generate a large number of unsolicited and useless messages to the network. The only purpose of this attack is to waste the network's bandwidth and the energy of nodes receiving or forwarding such messages.

Spam attacks are more harmful to sensor networks as in these networks the energy is (most of the times) a finite resource. A well planned attack (possibly with more than one infected node) will make the infected node(s) transmit dummy messages to the Base Stations, consuming energy of the forwarding sensors (specially those closer to the Base Stations). Once the energy of these sensors is exhausted, Base Stations will no more be able to receive data from the sensor network, causing the collapse of all the WSN (although with most of the sensors alive).

2.5.4 Defences of Routing Attacks for WSN

In this section, we make an overview of some possible defences against the attacks specified in the previous section (2.5.3).

[5]Spam is the use of electronic messaging systems to send unsolicited bulk messages indiscriminately. While the most widely recognized form of spam is e-mail spam, the term is applied to similar abuses in other media.

The routing service presented in this dissertation does not rely on detecting intrusions, but rather tolerates intrusions by bypassing the malicious nodes.

In Wireless Sensor Networks, the wireless nature of the communications among the sensor nodes increase the vulnerability of the network to a wide range of attacks (like eavesdropping, unauthorised access, spoofing and replay attacks). Also, the presented highly resource-constrained nature of the sensors limit the degree of encryption, decryption and authentication that can be implemented on individual sensor nodes; this calls into question the suitability of traditional security mechanisms (such as compute-intensive public-key cryptography) for such resource-constrained sensors. Furthermore, WSN face the added physical security risk of sensor nodes deployed across the field being captured by an intruder; with such successful attack, the intruder can use the compromised sensor node to instigate malicious actions (as advertising fake routing information) from within the sensor network.

The combination of these threats motivate the following design philosophy in order to achieve secure WSN: concede that a well-equipped intruder can compromise individual sensor nodes, but secure the overall design of the WSN so that these intrusions can be tolerated and the network as a whole remains functioning despite of such localised intrusions. The objective of this dissertation is thus the design of intrusion-tolerant secure WSN that have the property that a single compromised node can only disrupt a localised portion of the network, and can not shut down the entire WSN.

2.5.4.1 Defences against Fake Routing Information

One possible solution to prevent an external attacker from generating fake routing information on the network, is to apply security mechanisms based on encryption; with the usage of such mechanisms, authentication may be required by the routing protocol.

In order to use this defence, nodes in the network share keys to authenticate their data packets and routing control messages. Within this vision and since an external attacker does not have the keys to authenticate its packets, all the fake routing information injected in the network will not be accepted by the other legitimate nodes. With such mechanism, these attacks can be defended.

2.5.4.2 Defences against HELLO Flood Attacks

As already studied, HELLO Flood Attacks are usually caused by an external laptop-class attacker, using a large transmission power and creating asymmetric links between it and the legitimate nodes. One possible and intuitive defence for this attack is to check the bi-directionality of a link between two neighbours. With this defence, a node discovering neighbours broadcasts an HELLO message and waits for each neighbour to answer with its identity; the first node will only consider as neighbours the nodes who from it receives correct responses.

Another defence to this attack is to use a Base Station as a trusted third-party to help

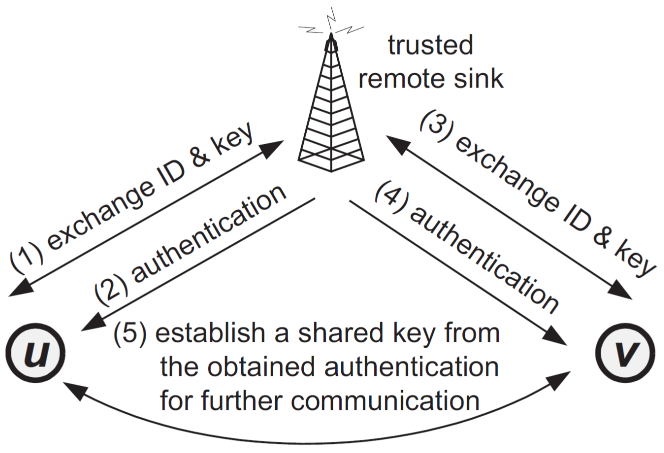


Figure 2.6: Base Station authentication.

two sensors verify each other; thus, each node in the network must share a unique symmetric key with the BS. Two sensors (u and v) are then able to verify each other's identity and establish a shared key through the BS, that will be used for them to communicate. To prevent an attacker from trying to establish too many connections (with HELLO Flood Attack), the Base Station can reasonably limit the maximum number of neighbours for each node. This mechanism is represented in Figure 2.6.

2.5.4.3 Defences against Sinkhole Attacks

In this attack, the intruder tries to attract the traffic to be routed through it.

A proposed solution to on-demand protocols (more specifically to *Dynamic Source Routing* protocol) [53] uses three indicators to determine whether sinkhole attackers exist in the network:

1. **Discontinuity of sequence numbers —** Sequence numbers of packets originated from a node should strictly increase in DSR. However, a sinkhole attacker attempts to use a very large sequence number, in order to update the route caches contents of other nodes. To prevent the attack, a node must then monitor the sequence numbers of received RREQ packets and pay attention to those not strictly increasing or unusually large;

2. **Ratio of verified RREQ packets —** The source address of a node initiating a RREQ packet shall be its own address. A sinkhole attacker initiates, however, RREQ packets with different sources (and periodically broadcasts them). Anyway, the RREQ can be verified by the source's neighbours; this means that a lower ratio of verified RREQ packets in the overall network may reveal the presence of a sinkhole attacker.

3. **Ratio of routes through a particular node —** Legitimate nodes of the network may determine the existence of a sinkhole attacker by checking their routing caches. If

a given (legitimate) node finds that most of its cached routes are through a particular node, it will suspect that this node is a potential attacker, since the goal of the attacker is to exactly have a big amount of routes passing through it.

2.5.4.4 Defences against Wormholes Attacks

A possible solution to prevent two distant malicious nodes from using an out-of-bound channel to tunnel packets (a wormhole attack) is to introduce the concept of *packet leash* [47]. A *leash* is the information added to a packet, defining the maximum allowed transmission distance; such distance can be geographical or temporal. While the geographical guarantees that the receiver of the packet is within a certain distance from the sender, the temporal guarantees that the packet has a lifetime, which also restricts its maximum travelling distance.

With temporal leashes all the nodes in the network must have tight time synchronization (the maximum allowed difference in clocks between any two nodes is Δ, known by all of them). When a node u transmits a packet, the claimed transmission time (t_u) and a expiration time ($t_{expire} = t_u + \frac{L}{v_c} - \Delta$, where L is the maximum distance that the packet is allowed to travel and v_c is the propagation speed) are included and protected with cryptographic authentication. When a node v receives the packet at time t_v, it can determine if there are any wormhole attacks in its route. If $t_v < t_{expire}$, v will accept the packet; otherwise, the packet is discarded and the wormhole attack is found, with earlier RREQ packets having been tunnelled previously.

2.5.4.5 Defences against Sybil Attacks

In the Sybil Attacks, the intruder presents itself with many different identities to its neighbours. One possible defence is called *radio resource testing* [48] and is based in the assumption that each physical node (including the attacker) has only one radio device; furthermore, a radio is incapable of simultaneously sending or receiving on more than one channel. With this idea, a node searching for Sybil attackers in its neighbourhood assigns to each of its neighbours a different channel to broadcast messages and then randomly selects a channel to listen. If a message can be received, the neighbour is legitimate; otherwise, the node is considered a Sybil attacker.

Another defence is the *random key pre-distribution*. Although this scheme is intended to be a key distribution mechanism, it can also prevent the Sybil attacker. It consists in each node having a set of keys assigned, from which its personal keys are those whose indexes are determined by the hash value of its identity. Since the hash function is very hard to inverse (and obtain the original identity), a Sybil attacker cannot just collect a random subset of keys and claim fake identities from these keys.

2.5.4.6 Defences against Blackhole Attacks

A proposed solution [54] to defend against blackhole attacks is to use a *watchdog* scheme to identify potential malicious nodes, and provide data to a *pathrater* scheme.

The watchdog scheme is based in the assumption that a node can overhear the packets transmitted by its neighbours. The idea for a protocol like DSR is then that when a node u transmits a data packet to its next-hop neighbour v, u will overhear the transmission from v to check if v is really forwarding the packet to its neighbours or not. At each node, the watchdog maintains a counter to record bad behaviours of each of its next-hop neighbours; if the counter exceeds a given threshold for a given neighbour, the watchdog will infer that its neighbour may be a malicious node and reports it to the source.

Combined with the watchdog, a pathrater scheme helps to avoid these kind of attacks. With the data provided by the watchdog, each node will eventually assign a rating to every other nodes of the network (a non-negative value to every legitimate node and a highly negative value to each malicious node). The overall rating of a routing path is the average rating of nodes on that route. The source is then able to select the most reliable route, by using the one with the highest ranking value to forward its packets. A routing path with an overall negative ranking value implies the presence of malicious nodes in that path.

Another solution [55] for a protocol like DSR uses the following mechanism:

1. **Source routing —** The source specifies in each data packet the sequence of nodes that the packet has to traverse;

2. **Destinations acknowledgements —** The destination sends an acknowledgement (*ACK*) to the source through the same route (in the inverse path) when it receives a data packet;

3. **Timeouts —** The source and each intermediate node set a timer for each data packet, during which they expect to receive an ACK or a *fault announcement (FA)*;

4. **Fault announcements —** When the timer expires, the node generates a FA and propagates it to the source.

All the exchanged data within this mechanism (data, ACK and FA messages) are authenticated, in order they can not be modified or fabricated by an attacker. When the source receives a FA message, the source can detect the presence of a potential blackhole attacker, and thus select another route to forward its packets.

2.5.4.7 Defences against Spam Attacks

Detect and defend spam (DADS) scheme [52] proposes quarantine regions to isolate spam attackers and thus defend from spam attacks. Base Stations are responsible for detecting the attacks in the network, with three methods available. The first method consists

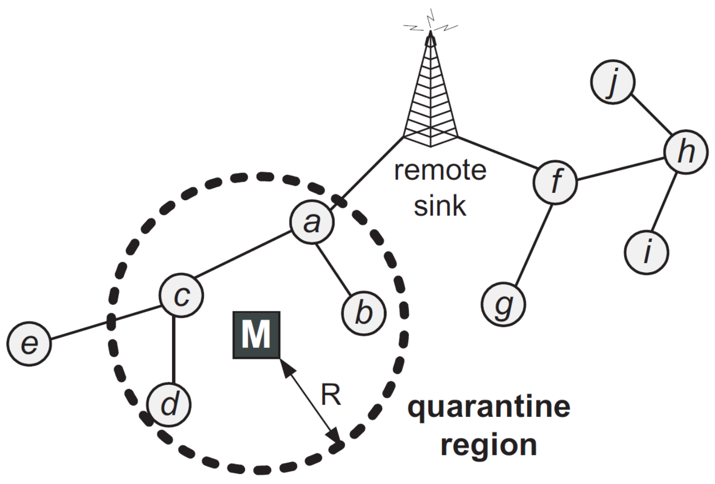


Figure 2.7: DADS example, with a quarantine region.

in filtering incoming messages accordingly with their contents, and detect the nodes frequently sending faulty messages; the second method consists in analysing the arrival rate of messages from a region of the network; finally, the third method consists in analysing the packet generation rate of the overall network. The third method is the one suggested by DADS, because an attacker can possibly move or change its identity, spoofing the BS.

When the number of data packets arriving at the Base Station exceeds a reasonable level, a *defend against spam (DAS) message* is sent by the BS to the network. The purpose of such messages is to quarantine a spam attacker by its one-hop neighbours. When a node a receives a DAS message, it starts a timer (t_a). While this timer does not expire, a only forwards authenticated messages; if an unauthenticated message is received from b, a will ask b to retransmit an authenticated message. If this authentication fails, a considers itself inside a quarantine region and will just forward authenticated messages (including their own). To save the authentication overhead, if t_a expires and a does not detect any failed authentication during that time, a switches back to the normal mode, cancelling the quarantine.

Figure 2.7 shows a situation where M failed the authentication; note that all the nodes outside of the quarantine region are able to send unauthenticated messages, except e because c will only accept authenticated messages.

2.5.5 Critical Analysis

In this section, Security and Reliability in WSN was discussed.

An attacker in a WSN can be internal (as an intrusion attack, through the capture of legitimate nodes and consequent modification of their behaviours) or external (as an attack to the communications); if external, an attacker can belong to a sensor-class (where the attacker uses similar devices as nodes) or to a laptop-class (where the attacker uses more powerful devices, like laptops). An internal attacker is observed as an apparently legitimate and trustable node, that knows the cryptographic keys in use by the network. In the other hand, an external attacker has the advantages of possibly being distributed with various nodes across the network (if a sensor-class attacker) or to have more resourceful devices with greater capabilities (if a laptop-class attacker).

An intruder can inject a malicious behaviour in a Wireless Sensor Network at many levels. However, in the scope of this thesis we are particularly interested in the attacks directed to the routing layer of the network, with the previously described topology; attacks to other layers (like DoS attacks to the MAC layer) are out of the scope of this thesis.

Concerning the attacks to the routing layer in WSN, they can be grouped into *attacks during the route discovery process*, *attacks during the route selection process* and *attacks after the establishment of the routing paths*. Many possible attacks within these groups were presented, as well as possible defences for each. A summary of the discussed attacks and some possible preventions is presented in table 2.1.

Routing Phase	**Attack**	**Defences**	**Intrusion-level adversary defences**
Route Discovery Process	Fake Routing Information	Techniques based on encryption.	X
Route Selection Process	HELLO Flood Attacks	Check of bi-directionality on links; Base Station authentication	X
	Sinkhole Attacks	Sequence numbers on messages; Verification of RREQ source address by neighbours; Analysis of routing caches	✓
	Wormhole Attacks	Packet leashes, protected by encryption	X
	Sybil Attacks	Radio Resource Testing; Random key pre-distribution	✓
After Establishing Route Paths	Blackhole Attacks	Watchdog + pathrater; ACK / Fault announcements scheme	✓
	Spam Attacks	Detect and defend spam (DADS) scheme	X

Table 2.1: Summary of the discussed attacks and some possible preventions

3

Related Work

The goals for this thesis are the development of a secure, reliable and multi-hop routing service for a large-scale WSN, the experimental assessment of such routing mechanisms, assessment of consensus mechanisms and the approach to an hybrid solution within the simulation environment. The focus is thus in the Network layer as presented in 2.2. The important related work for such goals is analysed below. In 3.1 simulation environments will be studied, including an address to emulation environments. In 3.2 an approach to routing algorithms will be done. The algorithms we are interested in are those offering security and reliability services. In 3.3 the problem of the distributed consensus will be discussed, including protocols for solving it.

3.1 WSN Simulation

In the development of software for a Wireless Sensor Network, it is mandatory to have some way to test it (as in the development of software in general). The developed software has a lot of requirements; thereby, the need for reliable and dependable implementations grows, making almost infeasible to deploy a real WSN with thousands of nodes for test purposes.

The operation of the protocols is driven by environment variables (measured by the sensors) and is able to affect even the topology of the network; the energy consumption depends on the behaviour of the nodes. These constraints make unfeasible [56] to analytically model a WSN, predicting its behaviour and performance of the implemented protocols.

Instead, this is leading the simulation tools to become more and more important. In the presence of appropriated testing suites (with good models and tools), the application software (developed for the WSN) can be previously tested by running it in PC-environments before the deploy to sensor nodes of a physical WSN.

3.1.1 Simulation with Emulation

As stated in 3.1, it is almost infeasible to deploy a real network to test in development software for WSN, bringing simulation testing suits a good approach to testing. However, testing suites can not only simulate but also emulate. These two concepts can be defined as [57]:

- **Simulation —** The properties of an existing or planned network are computationally simulated in order to assess performance and predict behaviours that occur in the network. Is useful for quickly try new ideas and to evaluate the behaviours of the implemented protocols; however, there is in simulation a lack of fidelity, making unrealistic to simulate at instruction-level and with high-fidelity physical models (radio, power-consumption), that can be relevant for environments where there is a lack of resources and incidental physical characteristics.

- **Emulation —** The actions of the nodes are similar to real nodes, running the same code that would be ran in real sensor nodes, making a similar behaviour occur (processor, sensors and radio). This solves the flaws from simulation and can be done, for example, in a simulator supporting both virtual nodes and physical nodes working on the same network as neighbours [58]. Emulation concept is therefore situated between a Physical Network and a Simulated Network.

3.1.2 Hybrids Environments for Simulation and Calibration

As already stated, simulation in WSN development is a valuable test environment (for example, to understand the behaviour of the network in large-scale ad-hoc networks in order to evaluate the protocols).

However, simulation tests inevitably take assumptions and simplifies mathematical models like assuming bi-dimensional topologies, perfectly circular transmission ranges, non-existence of environmental noise, fading effects, etc. [59]. With all these flaws, the credibility of the obtained measurements and of the results of studies can be questioned because of this lack of accuracy, comparing with the real world. This conducts to inconsistent or misleading results, what requires to calibrate the simulators in order to achieve realistic results.

Realistic results in WSN Simulators can be achieved by two similar ways:

(a) Implementing a real (small-scale) WSN, obtaining measurements and calibrating the simulation model of the simulator by reintegration of the measured real-world parameters [59];

(b) Using an Hybrid Simulator that can, autonomously calibrate itself with measurements obtained from real sensors [60].

An Hybrid Simulator is a simulator that, while providing the simulation capabilities, enables the interaction of virtual nodes with real ones. While the simulation capabilities allow the generation of easily scalable scenarios (and all the already referred advantages), the interaction with real deployed nodes in a real sensor field allow the generation of realistic data models. With these obtained data models, the calibration of the simulator [60] is possible.

3.1.3 WSN Simulators

The key properties to select a good and suitable simulation environment for WSN are [56]:

1. Availability and reusability;
2. Performance and scalability;
3. Support for rich-semantics scripting languages to define experiments and process results;
4. Graphical, debug and trace support.

Item 1 refers to the existence of implementations of common models in the simulation tools and the easiness in modifying such existing models, or adding new ones. The former property is important in order a researcher can compare, for example, the performance of a new technique against existing ones, and the latter is important in order a researcher can reuse or modify code from previous implementations.

Item 2 refers to a major concern in simulation. Simulators should have a nice performance in order to consume a reasonable amount of time and allow the simulation of large-scale environments.

Item 3 refers to the support of a scripting language that can be used as input to the simulator, with high-level semantics. This supports the definition of the simulation environment, as for example but not exclusively, the number of nodes, where is each node placed, how nodes generate events, etc. This item also refers to an output scripting language, allowing the output results to be quickly and precisely analysed.

Item 4 refers to the Graphical User Interface (GUI) being important as a debugging aid (allowing to watch the behaviour of the simulation), as a visual modelling and composition tool (in small and basic experiences) and as a results visualiser (showing quick results of the execution).

Below are presented some simulators with its main characteristics.

3.1.3.1 TOSSIM/PowerTOSSIM

Tossim [61] is a simulator for sensors equipped with TinyOS. This simulator compiles and executes the same code that a physical sensor can execute. However, in terms of emulation, it just emulates a limited set of hardware in a too simplistic way; this leads to results very different from those obtained from a real physical deployment. Other problem is that all the sensors in the simulator must run the same code.

PowerTOSSIM [62] appeared as an extension to TOSSIM. It introduces energy consumption emulation mechanisms in the nodes. For each hardware component of the sensor, energy consumption measurements are made. This is an important extension that allows the programmer to study the behaviour of one of the most important limitations in WSN: the energy consumption.

3.1.3.2 Freemote

Freemote [58] is a Java-based emulation environment. This platform emulates nodes running Java code, trough optimized JVMs (Java Virtual Machines) for sensors.

The software architecture is divided into three distinct layers: Application Layer, Routing and Communications Layer and Hardware Layer. Real nodes can be any devices based in communications standard IEEE 802.15.4 (MICAz, JMotes, etc).

3.1.3.3 Avrora

Avrora [63] simulator was born from a research project called *AVR Simulation and Analysis Framework Platform*. It is used as a set of simulation and analysis tools for software developed for AVR micro-controller (used by Mica2 sensors). This simulator offers an almost complete and realistic implementation of Mica2 sensors' hardware.

AvroraZ [64] appeared as an extension to Avrora and allows the emulation of nodes with AVR micro-controller and radio communications over IEEE 802.15.4 norm. The main goal of this extension is to provide an accurate emulation of radio communications, without any changes in running code from real to simulated nodes.

3.1.3.4 VMNet

VMNet [65] is a WSN emulator that aims to provide applications realistic performance evaluation. In this platform, a WSN is emulated as a Virtual Mote Network (VMN). The developed software for real nodes can be executed in the emulator; this emulates the operation of the real node's hardware, allowing a realistic evaluation of the response times and of the energy consumption. However, VMNet supports only Crossbow Mica2 sensors.

3.1.3.5 NS-2 and NS-3

NS-2 and NS-3 [66] are discrete events simulators, focused to support the computer networks' researches. They are two of the most used simulators that support WSN simulation. They include a large set of protocols, traffic generators and tools to simulate many routing protocols over wired and wireless networks, local or by satellite.

The main focus is the OSI model simulation, including the random generation of phenomenons at hardware level and energy consumption models. For WSN, they include sensors simulation models, battery models, protocols stacks for sensors and have tools for generating statistics of the network behaviour.

However, the detail level in simulation makes infeasible to simulate large networks, with thousands of nodes [67] (like the ones we are interested in).

3.1.3.6 SENSE

SENSE [68] is a specific simulator for WSN. It offers battery, routing layer and application layer models. SENSE is able to simulate networks with about 5000 nodes, but this number can decrease depending on the communication's patterns used. However, the radio communications are limited to IEEE 802.11 norm.

3.1.3.7 JProwler

JProwler [69] is a discrete events simulator implemented in Java. It simulates the radio transmission, propagation and reception, including collisions and the MAC layer operation. The radio definitions are implemented with plug-ins, that allows a great extensibility to the simulator. The discrete events simulator can be set to work in a deterministic way (reproducing replicable results) or in a probabilistic way (that simulates the non-determinism of the communication channels and of the low level communication protocols).

3.1.3.8 WiSeNet

Wisenet [13] is a simulator based in JProwler (3.1.3.7), adding attacks to the WSN simulation mechanism. Its main focus in the design is the easiness of networks topologies' creation and configuration.

Some interesting points of this simulator are its graphical user interface (GUI) for network visualisation and configuration and the modules. GUI has general information about the network as well as information for each node in particular; it has also a network topologies' generator, allowing the generation of random, grid or structured topologies. The modules allow the extraction of certain metrics as energy consumption, latency, reliability, etc.

3.1.4 Critical Analysis

In order to evaluate the secure routing service developed in this thesis, it is mandatory to observe its behaviour during attacks. WiSeNet is the chosen simulator, as is the only one allowing to easily introduce attacks in the network. It also provides information about metrics (like energy consumption, reliability, latency, etc.) that are fundamental to assess the network operation.

3.2 Secure Routing in WSN

In the past some routing protocols were developed for WSN, although they were not thought to deal with intrusions. However, these protocols had characteristics and functionalities that inspired the developed protocol; in this way, next we will see the most important ones.

3.2.1 Clean-Slate

Clean-Slate protocol [27] is a routing protocol for WSN that was developed thinking in prevention, detection/recovery and resiliency.

This protocol assumes that a Certifying Authority (CA) exists in the network (with its public and private key). That CA delivers to each node, at start, a Certified Identity (CI) as well as an unique address and a set of random challenges that should be solved by the nodes.

After having the CI, the nodes start finding neighbours through a secure protocol (using their CIs) where a given node presents itself to its direct neighbours. After this procedure no more sensors can join the network, avoiding the entrance of intruder nodes.

Then, in order to configure the routing, it is started a recursively group creation algorithm based in the idea that the network can be seen as groups of nodes. This algorithm starts by creating a group for each node (ie. each node is alone on its own group); each group sends a joining proposal to the smaller group (with less nodes) and if the proposal occurs in both ways, the two groups (G0 and G1) merge creating a new and unique group. The process repeats until all the network converge to just one group with all the nodes. For each merge, nodes' addresses are revised and routing tables are updated, in order that each node from G0 has at least a next-hop to reach the group G1 and vice-versa. For security reasons, at each stage of the algorithm, groups are authenticated using a Group-Verification Tree (GVT).

With the previously found addresses and routing tables, it is possible to do the routing of the messages. However, the resiliency routing mechanism (that consists in maintaining multiple paths from the source to the destination) offers better guarantees in the messages delivery. The multiple paths are obtained from the Groups Merging Algorithm that keeps in the routing tables multiple next-hops for each destination group; this way, the sender can select the desired path. This selection can be done, for example, considering the next-hop as one of those nearer from the destination group (ie, based in the distance).

Detection and recovery of attacks are made in the following ways:

- For the inconsistencies in the Groups Merging Algorithm, a Group-Verification Tree (GVT) is used;
- Detection of replicas/duplicated nodes, that does not allow the nodes to have different identities (using, for instance, copies of CIs from other nodes) or trying to be members of more than one group simultaneously;
- Remotion of detected malicious nodes through the Honeybee technique. This technique consists in removing both the node that detected and reported the intruder as well as the intruder itself. This double remotion prevents from the case in where a malicious node is reporting innocent nodes as being malicious; however, this recovery technique can also be a problem if used intentionally by an attacker, as he can use it to shutdown correct nodes.

3.2.2 H-SPREAD

H-SPREAD [30] is a routing protocol that uses multiple disjoint routes from the nodes to a Base Station (BS). In order to enhance the network reliability, the BS sends the messages through the multiple routes using a threshold secret sharing scheme.

The threshold secret sharing scheme consists in dividing a secret into smaller parts called shares; then, for someone to read the secret, it is necessary to collect at least a certain (defined) number of shares.

The routes discovery is done in a distributed way by all the nodes of the network by using a branch-aware flooding algorithm; this consists in first calculating a spanning tree of the network and then using another algorithm to create new connections between the nodes, in order to create the multiple routes to the BS.

When a sensor in the network wants to send a message to the BS, it starts by dividing the message into smaller parts as already stated with the threshold secret scheme algorithm; then, each share of the message is sent across a different route of the set of multiple routes that connect the sensor and the BS. After receiving the minimum (defined) set of shares, the BS can decode the message. To read the message, an intruder must at least infect a minimum set of routes from the sender to the BS, corresponding to the minimum number of shares to decode and read the message.

3.2.3 SeRINS

SeRINS (Secure alternate path Routing IN Sensor networks) protocol [31] is a routing protocol that uses multiple routes from each node to the Base Station (BS). The network structure is based in a tree where the root is the BS and the other nodes are the sensors; each of the other nodes can have multiple parents.

The routes discovery process is started by the Base Station, which broadcasts a route update message announcing its distance (hops number between the BS and the node) as being 0 (zero). Every node receiving the message adds the sender as first parent if it hadn't a parent yet or if the new parent has a lower hop count. The distance of the message (hop count) is then incremented and the message is forwarded at most once.

During this process, an attacker could easily announce a false distance and compromise the protocol; however, this protocol implements a intrusion detection system that detects intruders that are changing the routing data, based on the neighbours' reports to the Base Station (belonging to the BS the final decision). Once the BS decided that the reported node is in fact an intruder, all the nodes from the network having it as parent remove it from the parents list.

A message sent by one node to the BS is always forwarded by each intermediate node to one of its parents. The parent where to the message is forwarded is chosen randomly, inserting non-determinism in the routing process and thus avoiding possible attacks.

3.2.4 INSENS

INSENS (INtrusion-tolerant routing protocol for wireless SEnsor NetworkS) [28] is a secure routing protocol that is intrusion tolerant in a network with an asymmetrical topology.

The intrusions in this protocol are tolerated by the existence of multiple disjoint paths, whose messages are sent through, as well as the guarantee that just one (infected) node can not compromise all the network (affecting just a little portion without compromising all the network operation).

In this protocol, just the Base Station (BS) is able to broadcast messages through the network; every node that wants to send a message to another nodes (even if in an unicast manner), must always communicate first with the BS. This way, BS will work as a filter avoiding a node from flooding all the network with messages. This mechanism avoids DoS attacks.

The routes discovery in INSENS is done in the following way:

1. Route Request - The BS starts a limited flooding, requiring for information about all the reachable nodes in the network. This mechanism makes nodes to meet their neighbours;

2. Route Feedback - All the nodes send their local information/topology to the BS as a response to the previous request. Messages are sent to the BS through the inverse path of the received message;

3. Routing Table Propagation - The BS computes the routing tables for each node and sends them to each one.

Each node shares a symmetric key (due to resources limitations) with the BS, in order that the messages from the previous steps can be authenticated. This way, the fake routing information attack is avoided.

The asymmetrical topology of the network is used as a solution for the problem of the energy consumption. As the BS has greater computational and energetic power than sensors, the BS aggregates the data sent by the nodes and computes the routing tables for all sensors (as previously stated).

3.2.5 MINSENS

MINSENS [29] is a protocol developed as an improvement of INSENS protocol. MINSENS allows the coexistence of multiple base-stations (BSs), providing BS redundancy improving reliability and balancing the energy consumption over the WSN.

In terms of routing each BS works as an INSENS' BS, creating its own dissemination tree; thus, each node will have $\#B$ routing tables, where $\#B$ is the number of BSs in the WSN. Each message shall include a route id (unique for all the sets of routes) in order that the nodes can understand to which route the message belongs, therefore searching the next-hop in the appropriate routing table.

In this protocol all the routes from all the BSs are disjoint, meaning that a single node can belong at most to one route of the network. This property guarantees that an infected node will affect just one route. At start, each BS computes its own disjoint routes that could not be disjoint from all the routes of the network; then all the BSs share among them the computed routes, agreeing in the routes used by each one in order to make all of the routes disjoint.

A node can send a message in MINSENS protocol to a Base Station (like in INSENS, it is not able to send directly to another node) by one or more than one routes; it can even send the same message to one or more Base Stations. If the message is sent to more than one BS, then all the BSs that received the message shall agree in the value of the received message. This protocol assumes that the Base Stations are not however under attacks.

3.2.6 MINSENS++

MINSENS++ [70] is an improved protocol, based on MINSENS.

This protocol works like MINSENS on terms of routing: various disjoint routes from each node to each Base Station are discovered like in the previous protocol.

However and in contrast with MINSENS, MINSENS++ Protocol extends the adversary model to the Base Stations, considering that they are not trustable and can also be under attack. This fact leads us to the necessity of consensus mechanisms among the existing Base Stations, in order to recover from attacks at Base Station level. For this reason, it is very important to study the problematic of consensus; therefore, this problematic and the distributed consensus in WSN are discussed in section 3.3 (page 41).

The secure routing service presented in this dissertation tries to protect a WSN against the attacks presented in 2.5.3 by using different techniques. As discussed in 2.5.4, we concede that an intruder can compromise individual sensor nodes; the network can thus tolerate these intrusions while remaining functioning properly as a whole without being shut down. MINSENS++ protocol uses the combination of nodes redundancy and the existence of multiple disjoint routes in order to achieve such property; many attacks (such as *HELLO flood attack*, *Wormhole attacks*, *Sybil attacks* and *SPAM attacks*) are circumvented in this way.

In addition to this redundancy, some other countermeasures are taken into account for some specific attacks. To protect the network against the *fake routing information attack*, MINSENS++ protocol uses encryption in the exchanged messages; the protocol assumes that a keys' distribution mechanism is available, and thus all the nodes have their keys at start.

In the other hand, the possible defences presented against *Sinkhole attacks* (2.5.4.3) and *Blackhole attacks* (2.5.4.6) do not apply to our specific routing algorithm; due to the use of a table-driven philosophy in our protocol (as already discussed in 2.4.1 and 2.4.2), the routing path is not completely decided by the source and thereby MINSENS++ must deal with these attacks by using multiple routes and redundancy. Also the proposed solutions for *Wormhole attacks* can not be used by our solution because we consider an asynchronous system, without a geographical awareness by the nodes.

The usage of multiple disjoint routes and redundancy in order to defend against the attacks creates a whole new problem: *If different BS receive from different routes different values, which one is the correct? And if some of the BS are under attack, and just try to propose on fake values?* To solve this, the presented secure routing service will include a data consensus component, in order to perform a consensus over the received values.

The encryption used by our routing service is also an advantage against a passive attacker that could be eavesdropping the packets through an attack to the communications.

3.2.7 Critical Analysis

Several protocols for secure routing were analysed. The following table (Table 3.1) presents a summary of each analysed protocol, concerning the defences against some of the possible attacks.

	False Routing Information	Selective Routing	Sybil	Sinkhole	HELLO flood	Intrusion on BS / Syncnodes
Clean-Slate	✓	Probabilistic	Remove	N/A	✓	
H-SPREAD	✓	Probabilistic		N/A		
SeRINS	Remove	Probabilistic		Remove		
INSENS	✓	Probabilistic	✓	N/A	✓	
MINSENS	✓	Probabilistic	✓	N/A	✓	
MINSENS++	✓	Probabilistic	✓	N/A	✓	✓

Table 3.1: Routing protocols defences against internal attacks

A common technique of all the analysed algorithms is the multiple routes strategy, present in all of them.

False Routing Information attacks are prevented with cryptography by all protocols, except by SeRINS that uses also the *neighbour report system* and removes the compromised node(s) from the network.

The prevention of Selective Routing attacks is probabilistic, through the use of multiple routes by all the protocols.

Sybil and HELLO flood attacks are prevented by Clean-Slate and INSENS-based protocols through the use of cryptography; however, Clean-Slate uses the Honeybee technique to also remove the intruder in case of a Sybil attack.

Finally, Sinkhole attack only applies to SeRINS protocol due to the routes computation philosophy; once again, this attack is prevented through the *neighbour report system* present in this protocol.

As we can see from the table, the best protocols fulfilling our requirements are the INSENS-based protocols and Clean-Slate; however, Clean-Slate suffers from the described adverse effect of Honeybee technique. Thus, for this thesis we are particularly interested in the INSENS-based protocols.

MINSENS adds the possibility (when compared to INSENS) of the existence of multiple Base Stations. However, the intrusion problem must be solved by the network; if the network does not solve a problem, Base Stations will not be able to solve it (as they are considered as always safe and correct).

MINSENS++ is the most evolved protocol of the INSENS family. As Base Stations are considered as attackable, possible intrusion problems not resolved by the network can be solved by the Base Stations, trough consensus mechanisms. At this point, consensus mechanisms will add the possibility of all the correct Base Stations decide on the correct values. Thus, in this thesis and for the development of the intrusion tolerant secure routing service, the focus will be in the MINSENS++ protocol.

3.3 Consensus Protocols

3.3.1 The Distributed Consensus Problem

Consensus protocols have many applications in distributed systems. A consensus protocol consists in a set of processes, each one proposing at start a value and at the end all of the processes agreeing unanimously in a common value [71] (which was one of the initially proposed values). These kind of protocols comprise the following properties:

- **Termination —** Every correct process eventually decides on a value;
- **Integrity —** A process decides at most once;
- **Agreement —** Two correct processes do not decide differently;
- **Validity —** A process can only decide a value that was previously proposed (at start).

Although the consensus problematic has simple and already developed solutions in the absence of failures [72], in the presence of failures solutions are not trivial; what increases the complexity is that the protocol shall work properly even in the presence of failures, that can be of one of two types:

- **Fail-stop failures —** Occur when a process operating properly suddenly crashes, ceasing all its actions;
- **Byzantine failures —** Occur when it is not possible to make assumptions about the process' behaviour. These are the most difficult type of failures to deal with, as a given process can send messages not supposed to be sent (with erroneous data, partial data or temporally incorrect messages), stop answering for a certain period of time (seeming as a fail-stop failure) and then answering again soon after, or send different messages to different nodes.

In addition to the failures model, there is another assumption about the system model that make the solutions depend of. There are two types of synchronism:

- **Synchronous system —** In this kind of systems, it is assumed that all the operations have a well defined portion of time to be executed (temporal limits are assumed), what means it is reasonable to make a request and wait (without doing nothing more – blocked) for an answer;
- **Asynchronous system —** In this kind of systems, no assumptions are made about the execution time of the operations (no temporal limits assumed). This means that when a request is made, the wait for the answer shall not be blocking.

Because of the WSN properties (as studied in Chapter 2) as for example the communications failures, this thesis will assume a system model with Byzantine failures and asynchronous operation. Consensus will be essential, for example, when the Base Stations need to agree in a received value (from multiple values received from different routes and to different Base Stations).

3.3.2 WSN and the impossibility of data consensus

As viewed in section 3.3.1, will be assumed for the WSN an asynchronous model with byzantine failures.

A key aspect is if the failure of a node (sending an expected message) can or cannot be detected by other nodes; if it can be detected, the receiver gains the role of failures detector. However, this can only be done in a system where there are clocks and time-bounds or in a synchronous model where the processes block until receive an expected message. Although, in asynchronous systems it is impossible to differentiate a crashed node from a node that is running very slowly; thus, this kind of detection (by the receiver) is impossible to implement in asynchronous systems (and consequently in WSN) as a correctly operating node could be waiting for indeterminate time for a message from a node that have one of the referred behaviours.

This impossibility leads us to the Fischer-Lynch-Paterson (FLP) impossibility result [73], stating that consensus problem is impossible to solve by using deterministic protocols in asynchronous systems in the presence of failures.

Also for synchronous systems, there is an analogous impossibility when communications are unreliable. This result is Santoro & Widmayer Impossibility [74], stating that even with strong synchronism, there is no deterministic solution to the consensus problem if $n - 1$ or more messages are lost per communication round, in a system with n processes.

3.3.3 Protocols with Randomness

Historically, many protocols were developed working on a randomisation basis [75]. The reasons for the development of these protocols so many years ago were the low computational power of the computers at that time, the network constraints (bandwidth) and the high packet-error ratios.

Back then, it was very difficult for a computer to, for example, make computations based on asymmetric cryptography algorithms with good keys (more than 1024 bits) and to have reliable communications; this was also a problem for consensus algorithms. Based on this assumptions, randomised algorithms were developed using a non-deterministic approach.

However, with evolution of computers and consequent growth of computational power and improvement of communications, these kind of non-deterministic algorithms lost their importance. From then, networks became better with larger bandwidth, more reliable with lower ratios of packet-error and computers gained stronger computational power.

Nowadays, with the growing popularity of small devices (low computational powered as the wireless sensors (Chapter 2)), this kind of protocols gained again an important role in computer science and particularly in the research area of this thesis.

3.3.4 Probabilistic Failures Detectors

The FLP impossibility can be bypassed by using time limits for messages delivery in the system. In [76] the impact of adding limited synchronism to a (asynchronous) message exchange system was studied. Latter, the partial synchronism model was introduced [77, 78], developing mathematical formulas for the calculus of the byzantine temporal limits (given the considering number of faulty process and some temporal limit values known by processes).

Based on this, fails detectors can be developed; these fails detectors notify the participating processes that another process may have failed. However, these fails detectors are probabilistic, meaning that they can eventually identify a correct process as being faulty (for example, a delayed but still correct one) and *vice versa*. These detectors can be called as unreliable fails detectors and their development was started by Chandra and Toureg [79, 80].

3.3.5 Non-Deterministic Consensus

As stated in 3.3.2 and accordingly to the impossibility of consensus, in this thesis a non-deterministic consensus approach will be used. To achieve this form of consensus, it is necessary to weaken the usual consensus properties, allowing a probabilistic termination property instead of the deterministic termination property. Thus, the new consensus properties are the following:

- **Validity —** If every sensor proposes on a same value x, then all the correct sensors that make a decision, will decide the value x;
- **Agreement —** Two correct processes do not decide differently;
- **Termination —** All the correct sensors eventually decide, with probability $p = 1$.

The next presented algorithms for non-deterministic consensus are based on the coin tossing cryptographic schemes.

3.3.5.1 Coin Tossing Protocols

The coin tossing cryptographic schemes main idea is the consensus and delivery of a binary value: 0 or 1 (or an array of such values), accordingly with a given probabilities distribution [81].

The security guarantees given by these algorithms depend on the considered adversary: an adversary that has access to previous coin-toss extractions is able to compute the coming values.

These protocols are mainly categorized in one of two classifications: Local Coin-toss Protocol (LCP) and Shared Coin-toss Protocol (SCP). LCP protocols ([82, 83]) are computationally lighter (since they use symmetric cryptography) but are expected to end in an exponential number of execution rounds; on the other hand, SCP protocols ([84, 85]) are computationally heavier (since they use asymmetric cryptography) but are expected to end in a constant number of execution rounds.

Since LCP and SCP are opposites in their mode of operation, it is necessary to find a trade-off between them. This trade-off was experimentally assessed, but over common networks (with computers) and over mobile networks (ad-hoc) with devices like PDAs [86, 87].

These algorithms are just thought for the binary consensus problem; however, for some applications (as our) this could not be sufficient as they may need to reach a consensus over a complex value (non-binary) or a set of complex values. On next sections (3.3.5.2 and 3.3.5.3), two protocol's stacks are presented, solving this problem of the non-deterministic and byzantine failures tolerant consensus.

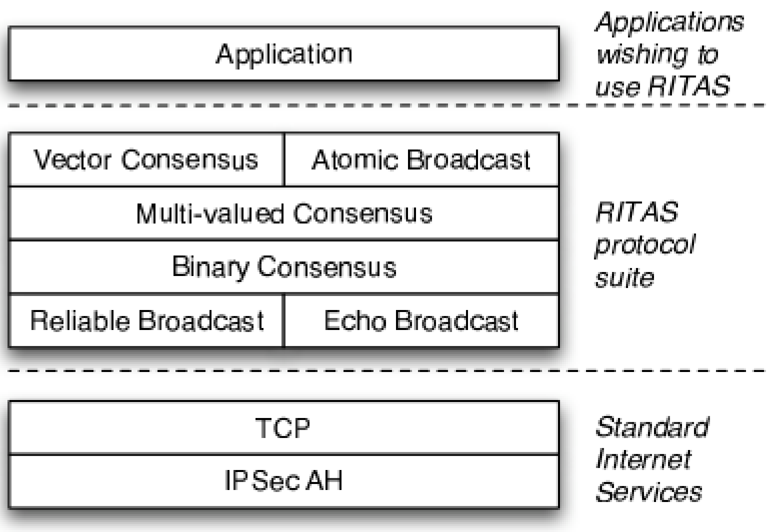


Figure 3.1: RITAS protocols stack

3.3.5.2 RITAS Stack

RITAS (Randomized Intrusion-Tolerant Asynchronous Services) [88] is a protocols stack (all of them are asynchronous) that uses protocols with randomness to solve the problem of the multi-valued (or set) consensus. This implementation shows that the protocols with randomness (as LCP and SCP) are efficient solving the problem of the distributed consensus over LAN's and WAN's.

On the stack (represented in Figure 3.1), TCP layer is used to guarantee reliability and the IPSec layer is used to guarantee integrity of the transmitted data. The set of layers between TCP and the Application are intended to solve the consensus problem, with optimal resilience to $f = \frac{N-1}{3}$ processes with Byzantine failures. The binary consensus layer uses a protocol with randomness based in LCP (developed by Bracha [83] as referred on the previous section 3.3.5.1).

3.3.5.3 Turquois

Turquois [89] is a binary consensus protocol specifically designed for wireless ad-hoc networks that assumes nodes being subject to transitory disconnection (because of unreliable communications) and permanent corruption by a malicious entity.

Turquois is developed for resource-constrained devices and thus maximises the efficiency of the consensus, by making a rational use of the resources provided by the environment while aiming for optimal resilience parameters. Namely, since the network provides a natural broadcasting medium, the cost of transmitting a message to multiple nodes can be just the same of sending it to a single one (assuming they are all within the communication range). This property can have a profound impact on performance.

The model proposed by Turquois derives from the one introduced by Santoro & Widmayer (3.3.2). This means that the model assumes that any communication from one node to another can be faulty at a given moment and be correct at another. The result of such assumption is that any broadcast message may be delivered non-uniformly by the intended recipients (some of them may deliver while others may not). Under particularly harsh conditions (for example, during a jamming attack), all the messages may be lost during a certain period of time.

Turquois' model assumes a set of n ad-hoc nodes and tolerates a subset of f compromised by a malicious adversary nodes (possibly with a Byzantine behaviour) where $f < \frac{n}{3}$. All the communications from that f nodes might potentially be lost or discarded. Additionally, dynamic omission transmission faults can exist, affecting the communications between correct nodes.

This protocol is based in LCP (3.3.5.1) as it is designed for resource-constrained devices.

The Turquois' evaluation results are promising when compared with the ones from other available solutions. The key to its performance is the assumption of unreliable communications, while allowing the protocol to take full advantage of the broadcasting medium. Furthermore, the protocol avoids the use of public-key cryptography during its operation, in order to preserve the limited computational power of the nodes.

3.3.6 Critical Analysis

WSNs have an asynchronous communications model, resource-constrained devices and are subject to intrusions or Byzantine flaws. A consensus solution for such networks must therefore adapt to these characteristics. The non-deterministic consensus and intrusion tolerant solutions are the most adequate to this problem; RITAS and Turquois were the analysed solutions (3.3.5.2 and 3.3.5.3).

RITAS protocols stack assumes the presence of TCP protocol; however, this protocol is not practical for WSNs due to their communication characteristics (in terms of reliability and in terms of the characteristics of the communications medium).

Turquois protocol is the best that suits WSN characteristics. This protocol assumes unreliable communications and has special concerns about the resource-constrained devices and consequent performance. While aiming for optimal resilience parameters, Turquois outperforms other protocols particularly as the number of processes in the system increases. For a possibly large scale network as a WSN, Turquois is then the chosen protocol for binary consensus.

4 System Overview

In the present Chapter, a System Overview is done, including the contextualisation of the previous contributions.

4.1 System Model

The presented secure routing service was developed with large scale networks in mind, from dozens to thousands of nodes (as a reference, let's consider from 1000 to 10000 nodes). Those nodes are randomly deployed in large geographic areas without any kind of supervision and subjected to all kind of adversities from the environment. Furthermore, we assume that sensor nodes have a very limited mobility after their initial deployment; we believe that this is the commonest case in many situations in these networks. However, the network's set up process may rerun periodically in order to accommodate changes in the network (like changes in topology due to faults or limited mobility).

The routing is based on a multi-hop philosophy, where data is disseminated from the sensor nodes until special aggregation nodes called Base Stations. These special aggregation nodes have special processing, storage, energy and communication capabilities; the special communication capabilities mean that BSs are connected among them with a dedicated and more powerful network environment (such as IEEE 802.11 or IEEE 802.3 Ethernet), supporting the TCP/IP stack (this can be viewed as an overlay network). Base Stations are then also connected with data management software that will provide the captured data for the developed applications.

Network's topology is based on a plain topology (see 2.4.3), supported by IEEE 802.15.4

protocol in an ad-hoc mode with (possibly) intermittent connectivity conditions. The network is thereby a graph that must ensure (after the self-organisation process) connectivity and coverage to all nodes, in order that every one can communicate with the Base Stations.

The operation of the WSN assumes that communications are asynchronous and not reliable (but with a best-effort delivery service).

4.2 Adversary's Model

Accordingly with the definitions in 2.5.1, in the scope of this dissertation the considered adversary's model is based on the definitions by Karlof-Wagner [39], Dolev-Yao [40] and OSI X.800 Framework [41].

An attacker to a WSN can change the normal behaviour of the network, by dropping messages, faking route information or modifying messages, and we call him an active attacker. This kind of intrusion corresponds to a possible proactive introduction of Byzantine flaws by the attacker. In the other hand, an attacker can simply intercept the communications in order to capture transmitted information (also called eavesdropping). This kind of attacker is known as a passive attacker.

Besides, an attacker can be internal or external. For an internal attacker, we assume that he can physically capture a limited number of nodes and/or introduce malicious behaviour on them. By network's nodes we intend sensor nodes or Base Stations; however, this attacks must be independent (an attacker can not compromise both Base Stations and sensor nodes simultaneously). This includes capturing data from the captured nodes (namely cryptographic keys), being an apparently legitimate and trustable node (with the stolen identity) while having a behaviour defined by the attacker.

An external attacker acts from outside of the network for example by listening to communications (passive attacker) or jamming (active attacker). Two classes of external attacker exist:

- **Sensor class —** This attacker uses one or more sensors, equivalent to those present in the WSN, with the same limitations (meaning that he can reach just the neighbourhood of each attackers' node);

- **Laptop class —** This attacker uses a laptop computer, which gives him more computational power, memory storage, communication power and energy autonomy when compared with the nodes from the WSN. This attacker is usually able to reach a big part of the WSN.

Respecting the flaws' model, asynchronous, independent and Byzantine flaws are considered. These flaws can be present both in the sensor nodes and in the Base Stations, but in an independent way as already stated.

Denial of Service (DoS) attacks and all those made to other layers than the network layer (for example, attacks to the MAC Layer exploiting the IEEE 802.15.4 stack (2.5.2)) are not considered in this thesis, as well as other possible physical or data-link attacks that can affect the radio communications. For the purposes of the base security communication properties in the WSN level, we consider the use of a well-known implementation of secure MAC-level primitives, as proposed in the TinySec [24] or MiniSec [25] security stack. These security properties, based on symmetric cryptography algorithms, will help to solve the issues of the lower levels namely the attacks to the communications at MAC Layer.

4.3 System Software Components

The system software model adopted in this dissertation is based on the specification studied in section 2.2, being divided in the layers presented below.

For the physical layer, the considered nodes are of MicaMotes' type; this is also the nodes type supported by WiSeNet simulation environment.

Concerning the Operating System layer, an OS supporting Java (with a JVM implementation) is considered. In fact, in the scope of this dissertation we are highly interested in systems supporting Java. Despite of the disadvantages of Java (like in terms of performance and memory consumption), among others Java has the following advantages we are interested in:

- **Platform independence —** Java is platform independent and thus Java programs can easily be moved between heterogeneous systems without significant changes;
- **Object oriented —** Java is object oriented, allowing the creation of modular and reusable code;
- **Distributed —** Java has the network capabilities and design to easily allow the development of distributed applications;
- **Secure and Reliable —** The design of Java considers security and the errors checking of Java compiler provides reliability;
- **Easiness —** Java simplifies the process of code writing, compiling, debugging and running.

For the MAC layer, the IEEE 802.15.4 is the chosen one, with CSMA/CA mechanism support; this thesis assumes that this layer provides secure communications mechanisms as well as secure and intrusion tolerant keys exchange mechanisms.

In this thesis, the layer we are focused in is the Network layer. Our secure routing service is inserted in this layer, offering the secure and intrusion tolerant routing service, with self-organisation of the nodes.

The applications that will consume the data produced by the WSN are developed in the Application layer.

4.3.1 MINSENS++

MINSENS++ is a secure routing protocol specifically designed for WSN, and is an improvement of MINSENS protocol, based on INSENS protocol.

MINSENS ++ protocol extends the adversary model to the Base Stations, assuming that they can be compromised (what didn't happen in MINSENS, whose Base Stations were always trustable).

This protocol offers security and reliability guarantees among the network with the help of route's replication (together with other countermeasures stated in 3.2.6).

This protocol will be further studied in Chapter 5.

4.3.2 MVC and Turquois

MINSENS++ protocol has the need of a consensus mechanism, in order to guarantee the desired security and reliability levels.

This consensus mechanism must be able to perform a consensus over a set of non-binary values, with intrusion and fault tolerance capabilities. Multi-Valued Consensus protocol (MVC) was developed to provide this service. MVC protocol has the need of a binary consensus layer in order to perform the non-binary consensus; as the binary consensus layer, Turquois protocol is used.

MVC and Turquois protocols are studied in Chapter 6. We consider f flaws to n nodes, where f is the number of failing/attacked nodes and n is the total number of nodes involved in consensus.

4.4 Contributions Contextualisation

In Chapter 3, many related works were presented; in the current section, the most important contributions for this thesis are summarised.

A simulation environment is important in order to observe the network's behaviour while being attacked. WiSeNet simulator (3.1.3.8) was the chosen one, because it allows to easily introduce attacks in the network while providing metrics to assess the network operation.

Concerning the routing protocols, MINSENS++ protocol (3.2.6) is the base of this work. MINSENS protocol was developed within the SITAN Project, and is the best fitting the project's purposes. MINSENS++ is an improvement of MINSENS.

About the consensus protocols, MVC and Turquois protocols are used. MVC protocol is inspired by RITAS stack.

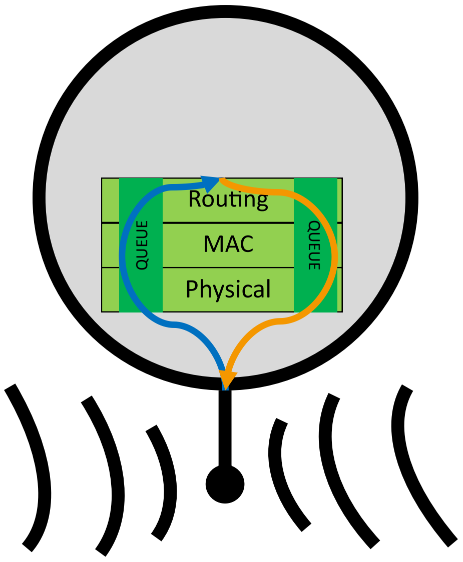


Figure 4.1: Scheme of a sensor forwarding data

4.5 Network Model

Wireless Sensor Networks are networks composed by (possibly thousands of) small devices very weak in terms of resources (computing, storage, memory, battery and communications) called sensor nodes. These devices usually operate in a self-organised ad-hoc way, with no human supervision. These sensors have two tasks: collect and send data from the physical environment and forward the data sent from their neighbours to the data's destination. The destinations of all this data are the Base Stations, that are special nodes (usually less than ten percent in number, when compared to sensor nodes) with special capabilities in terms of resources (more computing power, storage, memory, power and communications). Figure 4.1 represents a sensor forwarding data; the decision to forward data is made in the routing / network layer. Thus the message is not passed to the application level (hidden in the figure). Note that it might happen that the message is not immediately transmitted; message queues must be used, in order to avoid collisions.

For such aim, the WSN must adapt itself to the asymmetric architecture and resource constraints between sensor nodes and Base Stations. The network uses a plain topology; it is organised using a multi-path tree-structured routing topology, whose roots are the Base Stations and whose leafs and intermediary nodes are sensors. The data flows from the sensor nodes to the Base Stations with the help of a secure and intrusion tolerant routing algorithm. The sensors do not communicate with each others, except for routing purposes; this means that peer-to-peer communications among sensors is out of the scope of this dissertation. Figure 4.2 shows simplified[1] examples of possible network topologies, from the point of view of the sensor marked in green.

The main objective of this dissertation is to propose, implement and test an intrusion tolerant routing service for such dependable WSNs.

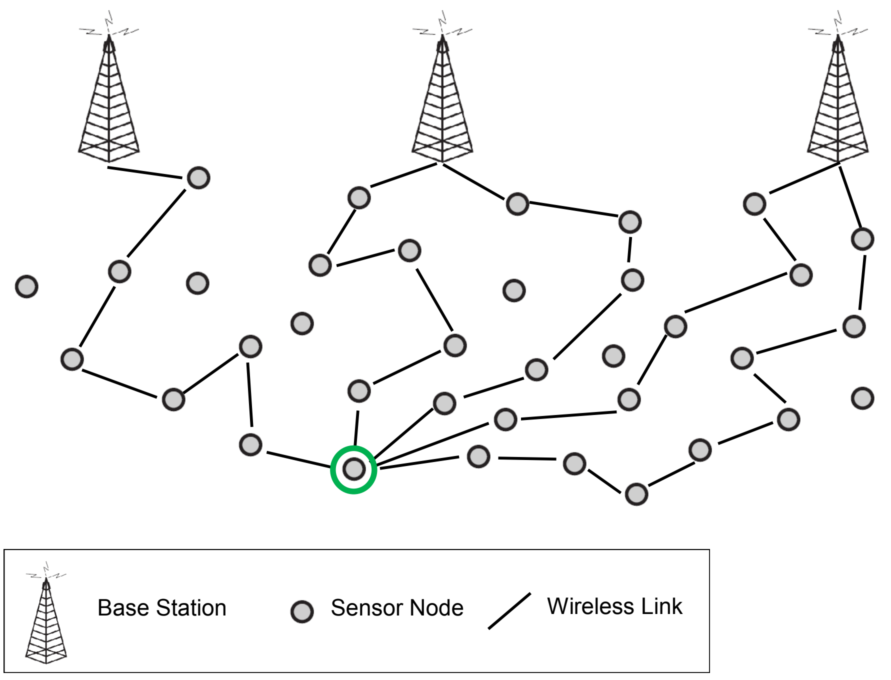
In order to successfully route all the generated data by the sensors, MINSENS++ Protocol is used. This protocol is responsible for organising all the network (including its routes) and forward all the data from each sensor to each Base Station.

When the same message arrives to various Base Stations, they need to reach a consensus about its value. This is important when an attacker is modifying the message's value, trying to persuade the other Base Stations to accept the wrong value as correct. Such consensus between Base Stations is made with the MVC Protocol as we'll see in the next chapters.

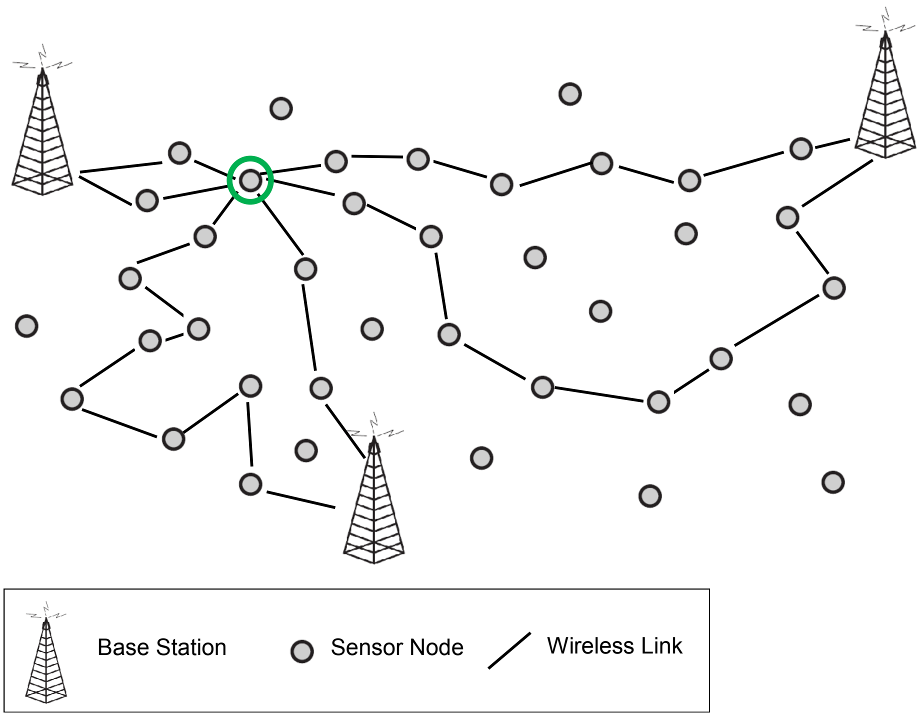
Finally and after all these processes, the messages can be passed to the application layer.

From this point of this document, only the MINSENS++ and MVC protocols overview, implementation and assessment will be addressed.

[1]Note that these examples are simplified for visibility purposes. Firstly, only the multi-path routes from the node in green are showed. Secondly, the number of Base Stations is high when compared with the number of sensors. And lastly, the number of sensors and the wireless connections between them are fairly modest; a real scenario would have more, resulting in more routes to the Base Stations.



(a) Example 1



(b) Example 2

Figure 4.2: Simplified example of a network's topology

5 MINSENS++

In this chapter, the MINSENS++ protocol is presented and specified.

5.1 System Model for MINSENS++

MINSENS++ is a routing protocol designed specifically for WSNs with a multi-hop routing philosophy. This protocol uses multiple disjoint routes from each sensor node to several Base Stations, in order to guarantee security and reliability. The target networks can have thousands of nodes (large scale networks) and are self-organised (operating in an autonomous way).

MINSENS++ is an extension to the MINSENS protocol (see 3.2.5) and its improvements are:

- Improve network's reliability;
- Extend the attacker's model (2.5.1) to the Base Stations.

Network's reliability is improved through the introduction of consensus mechanisms for the received data. For the extension of the attacker's model to the BSs, it is necessary to use consensus protocols specially adapted and developed for WSNs, with some properties as faults and intrusions tolerance.

MINSENS++ protocol has three different phases (based in the general proposal in 2.4.1):

1. Nodes and routes discovery;
2. Routes selection;
3. Data routing.

Each phase is better described below, in the following subsections. A more precise and algorithmic vision is given in the next section (5.2).

5.1.1 Nodes and routes discovery

In the nodes and routes discovery phase, each node presents itself to its neighbourhood. At the same time, it also discovers its neighbours. After that presentation phase, each node transmits its neighbourhood's information to the Base Stations. In the end of this process, each Base Station will have computed multiple disjoint routes from each node in the network to it. This computations are based in the neighbourhood's information provided by the sensor nodes of the network.

5.1.2 Routes selection

The routes selection phase is started immediately after the previous phase (5.1.1), when the Base Stations transmit to each networks' node its own routing table.

Routing tables are sent to the nodes ordered by the distance between the given node and the Base Station. After all the nodes received their routing tables, the network is finally (self-)organised and ready to start data dissemination.

5.1.3 Data routing

During the data routing phase, accordingly with the WSNs philosophy, the sensor nodes (producers) send data to the Base Stations (consumers) through multi-hop paths.

In order to check the correctness of the transmitted data, all the Base Stations receiving a message must reach a consensus for the data received within the message. This consensus process specification is not considered as belonging to the MINSENS++ protocol and thus is studied in its own chapter: Chapter 6 (page 61).

5.2 Algorithmic vision

In this section, an algorithmic vision of the different phases of the MINSENS++ protocol will be given. For a better understand, each subsection corresponds to each phase of the protocol.

5.2.1 Nodes and routes discovery

In the beginning of the protocol, each Base Station bs_{ID} broadcasts a route request message to the network. The message has the $\langle RREQ, bs_{ID}\rangle$ format, and is propagated epidemically.

When a node n_i receives one of that messages, it verifies if it has already received that message ("infected" by bs_{ID}). If the message is new, the emitter is added to the neighbours' set (N_i) and defined as father of i; otherwise, the emitter is just added to N_i.

After a certain amount of time, n_i sends to bs_{ID} its neighbourhood data; this data is sent through the inverse path (ie., each node sends to its father until reach bs_{ID}). This message has the ⟨*FDBK, i, N_i*⟩ format.

Base Stations store the neighbourhoods' data, in order to compute the network's routes. In each BS, disjoint routes are computed like in INSENS (3.2.4); however, it is necessary that all the routes computed by all the BSs are disjoint. To accomplish this, a routes' consensus is done between all BSs as explained in 6.4.1. When the routes are consensual, they are saved in the routing tables that will be sent to each routes' source.

5.2.2 Routes selection

When the routing tables' computation is ended, they must be distributed over the network's nodes. This distribution is ordered by the distance between the Base Station and the node; this way, routing tables from nearer nodes can be used to route the routing tables of the farther ones.

This messages are called *route update* and have the following format: ⟨*RUPD, bs_{ID}, n_i, routing_table*⟩, where bs_{ID} is the (unique) ID of the BS, n_i is the (unique) ID of the node and *routing_table* is the set of disjoint routes from n_i to bs_{ID}. The routing table is ciphered with a symmetric key (shared with the destination sensor node) in order to allow just n_i to access its contents.

A sensor node (n_i) stores in its final routing table the union of all the received routing tables (to different BSs).

However, a sensor node (n_i) can receive a message whose destination is not it. In this case, it verifies if has in its routing table a path from the destination node (n_k) until the emitting BS; if so the message is retransmitted.

When (n_i) has its routing table complete (with routes to all the BSs), it declares itself as stable. This means it is ready to start disseminating and routing data over the network.

This description is summarised in Algorithm 1.

```
Input: ID of the node $nodeId_i$
$routingTable_i \leftarrow \emptyset$
when $m = \langle RUPD, bsId, nodeId, routingTable\rangle$ is received do
    if $nodeId = nodeId_i$ then
        $routingTable_i \leftarrow routingTable_i \cup routingTable$;
    else
        if $routingTable_i$ contains route to $nodeId$ then
            broadcast(m);
        end
    end
end
```

Algorithm 1: RUPD messages processing in node $nodeId_i$

5.2.3 Data routing

In MINSENS++ protocol, data routing is divided in two distinct phases:

1. Routing between sensor nodes;
2. Data consensus in each Base Station.

The first phase deals with the data packets' routing, through the computed routes, from the origin until the BSs.

In the second phase, each Base Station must manage the reception of the different replicas of a given message (received from different routes). Then, the Base Station performs the consensus protocol over the received data. This phase shall not be confused with the consensus referred in the end of 5.1.3. The consensus referred at this point is local to each Base Station; as a different number of replicas of a given message are received by a given Base Station, a local consensus is made in order to determine the correct value of the message. Then, the product of this consensus is used as referred in the end of 5.1.3 to do a distributed consensus within all the Base Stations. We can then say that the final consensual value is the result of the (distributed) consensus of many (local) consensus.

Once more, in this chapter just the local consensus will be discussed. As stated in the end of 5.1.3, the distributed consensus is studied in Chapter 6 (page 61).

5.2.3.1 Routing between sensor nodes

Sensor nodes are regularly measuring physical phenomenons and environmental variables; this measured data must then be sent to the Base Stations. Data messages have the following format: $\langle$*DATA, id,* n_i*,* bs_{ID}*, routes, r, data*$\rangle$, where id is the (unique) ID of the message, n_i is the (unique) ID of the source node, bs_{ID} is the (unique) ID of the BS, $routes$ is the set of routes that will disseminate the message, r is the current route of this copy of the message and $data$ is the ciphered data. The messages are authenticated with MAC, assuring also the integrity of the message.

MINSENS++ provides five routing modes:

1. Routing trough one random route;
2. Routing trough one route chosen accordingly with a round-robin[1] policy;
3. Routing trough k random routes;
4. Routing trough k random routes, balanced by Base Station;
5. Routing trough all the routes.

[1]Round-robin is one of the simplest scheduling algorithms. It consists in selecting each resource in a circular order, handling all without priorities.

When the data is sent just through one route, data arrives just to one BS. In the other hand, if the data is sent through more than one route, two different situations may occur: data can be sent through multiple routes just to the same Base Station or can be sent through multiple routes to different Base Stations (possibly, more than one route to each Base Station).

In Algorithm 2, the routing process is presented. When a sensor node is ready to send a new message m, it broadcasts m. Another sensor node receiving m verifies if any of the routes of the message is present in its routing table; if so, the message is retransmitted; otherwise, it is ignored.

In order to reduce the messages' number in the network, each node has a log of received messages, in order to retransmit just new messages.

```
Input: ID of source node $nodeId_i$; routing table $rt_i$; routing mode $rm_i$

when has data ready to send do
    $msgId \leftarrow$ get new unique message identifier;
    $routes \leftarrow$ get routes for $rm_i$ from $rt_i$;
    foreach $r : r \in routes$ do
        $d \leftarrow$ destination of $r$;
        broadcast($\langle DATA, msgId, nodeId_i, d, routes, r, data \rangle$);
    end
end

when $m = \langle DATA, id, s, d, routes, r, data \rangle$ is received do
    if $nodeId_i = d$ then
        handle reception of $m$;
    else
        if $\exists route \in rt_i$ : route identifier $= r$ then
            broadcast(m);
        end
    end
end
```

Algorithm 2: Data messages routing protocol

5.2.3.2 Data consensus in each Base Station

This phase is started when a message reaches a Base Station, and it depends on the routing mode.

When a message is routed just through one route to one Base Station, the BS validates the message and (when valid) delivers it to the application level. This validation verifies the message's authentication and integrity.

When a message reaches a given BS through many routes, the BS stores all the different replicas of the message. Then a consensus is made and when the BS stored more than half of the replicas ($\frac{message\_routes}{2}$) with the same value, message is validated (with the consensual value). At this point, two possibilities exist: the message is just destined to that BS and after the validation, the message is delivered to application level; or the message was sent to many BSs and it's necessary to start a consensus process between all the BSs with the local validated data (more on this in the next chapter).

It is important to understand that if the BSs agree in a given value (of a message), it is considered a valid message and passed to the application level by all the correct BSs. If a consensus is not reached, the message is simply ignored. When the message is passed to the application level, its contents are deciphered in the BSs.

5.3 MINSENS++ Implementation

In the scope of this project, MINSENS++ protocol was implemented in Java. This implementation was based in the WiSeNet simulation environment that was the chosen simulator.

The Java implementation of MINSENS++ was programmed based on the presented algorithm.

6 Multi-Valued Consensus

In this chapter, Multi-Valued Consensus (MVC) [70] protocol and mechanisms are presented and specified.

6.1 System Model for Multi-Valued Consensus

MVC is an asynchronous and probabilistic protocol designed to reach consensus over a set of non-binary values. This protocol has intrusion and fault tolerance in the presence of f fails or attacks when $f < \frac{n}{3}$. Considering the asynchronism, some communications can fail without compromising all the consensus (complying with the previously mentioned fault tolerance criteria).

MVC protocol allows to consensus over any type of value since they are convertible to a byte array. It is based on the multi-value consensus protocol from RITAS stack (see 3.3.5.2).

This protocol uses also a randomised binary consensus protocol (in our case, Turquois is used) as we'll see in 6.2.2.

Details about the algorithmic vision of these protocols are given in the next section.

6.2 Algorithmic vision

In this section, an algorithmic vision of Turquois and MVC protocols will be given.

6.2.1 Turquois

Turquois is a randomised binary consensus protocol that allow k processes out of n ($k \subseteq n$) reach a binary consensus $v \in \{0, 1\}$. The correctness of the protocol is guaranteed as long as the Byzantine flaws f satisfy the condition $f < \frac{n}{3}$. The pseudo-code for Turquois is presented in Algorithm 3.

In the beginning, each process has an internal state composed by:

- phase $\phi_i \geq 1$;
- proposed value $v_i \in \{0, 1\}$;
- decision status $status_i \in \{decided, undecided\}$.

The protocol is started with values $\phi_i = 1$, $status_i = undecided$ and v_i is the input $proposal_i$.

The protocol runs in cycles, each one with three phases: *CONVERGE* ($\phi_i \bmod 3 = 1$), *LOCK* ($\phi_i \bmod 3 = 2$) and *DECIDE* ($\phi_i \bmod 3 = 0$). During *CONVERGE* phase, the processes try to converge to the most observed value, among all the participants; on *LOCK* phase, the processes try to lock a value $v \in \{0, 1\}$ or $\perp$ if the process is not able to decide (without preference); finally on *DECIDE* phase, the processes try to decide on the locked value on the previous phase. If a consensus is not reached, each process calculates a random value to start a new cycle. This random choose of new values guarantees that eventually all the correct processes will decide in one value.

Sender and Receiver tasks run in parallel. Sender task defines a broadcasting round and is activated periodically upon a local clock tick. A process p_i broadcasts a message with the following format: $\langle i, \phi_i, v_i, status_i \rangle$, where i is its identifier and ϕ, v and $status$ are its local variables comprising its internal state.

Receiver task is activated whenever a message is received. All the received messages are subject to a validation procedure, in order to ignore Byzantine messages; this way it is guaranteed that all the considered messages were sent by correct processes. All the valid messages are stored in a set V_i.

The state of a process changes if:

a. The set V_i holds some message whose phase value ϕ is higher than the current phase ϕ_i;

b. The set V_i holds more than $\frac{n+f}{2}$ messages with equal phase (ϕ_i).

In the first case, the state of the current process is set to match the state of the received message. There is however an exception: if the phase is *CONVERGE* and v is a random value (obtained as result of a coin flip), a local coin flip is done to determine v_i.

The second case is more complex and depends on the value of process' current phase. In *CONVERGE* phase the proposal value is set to the majority value of all messages with phase value $\phi = \phi_i$.

In *LOCK* phase the proposal value v_i is updated as follows: if there are more than $\frac{n+f}{2}$ messages in V_i with the same value v and phase equal between them and the local process ($\phi = \phi_i$), then v_i is set to v; otherwise it is set to $\perp$ meaning lack of preference

```
Input: Initial binary proposal value, $proposal_i \in \{0, 1\}$
Output: Binary decision value, $decision_i \in \{0, 1\}$
$\phi_i \leftarrow 1$;
$v_i \leftarrow proposal_i$;
$status_i \leftarrow undecided$;
$V_i \leftarrow \emptyset$;
Task Sender:
when local clock tick do
    broadcast($\langle i, \phi_i, v_i, status_i \rangle$);
end
Task Receiver:
when $m = \langle j, \phi_j, v_j, status_j \rangle$ is received do
    $V_i \leftarrow V_i \cup \{m : m \text{ is } valid\}$;
    if $\exists \langle *, \phi, v, status \rangle \in V_i : \phi > \phi_i$ then
        $\phi_i \leftarrow \phi$;
        if $\phi$ mod $3 = 1$ and v is the result of a coin flip then
            $v_i \leftarrow$ coin$_i$();
        else
            $v_i \leftarrow v$;
        end
        $status_i \leftarrow status$;
    end
    if $|\{\langle *, \phi, *, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n+f}{2}$ then
        if $\phi_i$ mod $3 = 1$ then /*phase CONVERGE*/
            $v_i \leftarrow$ majority value $v$ in messages with phase $\phi = \phi_i$;
        else if $\phi_i$ mod $3 = 2$ then /*phase LOCK*/
            if $\exists v \in \{0, 1\} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n+f}{2}$ then
                $v_i \leftarrow v$;
            else
                $v_i \leftarrow \perp$;
            end
        else /*phase DECIDE*/
            if $\exists v \in \{0, 1\} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n+f}{2}$ then
                $status_i \leftarrow decided$;
            end
            if $\exists v \in \{0, 1\} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| \geq 1$ then
                $v_i \leftarrow v$;
            else
                $v_i \leftarrow$ coin$_i$();
            end
        end
        $\phi_i \leftarrow \phi_i + 1$;
    end
    if $status_i = decided$ then
        $decision_i \leftarrow v_i$;
    end
end
```

Algorithm 3: Turquois

/ undecided. This step ensures that in the next phase every process proposes the same value $v \in \{0, 1\}$ or $\perp$. Furthermore, if consensus was achieved amongst correct processes at the previous phase, then every process must set its proposal value to the same value v (since messages with a different value are considered invalid). This will imply that in the next phase every process receives the same value $v \in \{0, 1\}$ in all valid messages and decides.

In *DECIDE* phase a process sets its status to *decided* with $v_i = v$ if there are more than $\frac{n+f}{2}$ messages in V_i with the same phase of the process ($\phi = \phi_i$) and the same value $v \in \{0, 1\}$ ($v \neq \perp$). Otherwise, v_i receives the result of a coin flip c ($c \in \{0, 1\}$, each with probability $\frac{1}{2}$). Regardless of the previous steps, the phase is always incremented by one unit.

Concerning the validation of received messages, two different checks are made: authenticity validation and semantic validation.

Authenticity validation guarantees that a given message was actually generated by its declared sender. In original Turquois, this validation requires the share of keys' matrices with all the *phase vs. proposed value* possible combinations. As MINSENS++ protocol requires all the nodes to share symmetric keys, that mechanism can be reused. Therefore Turquois' authentication mechanism was replaced with MAC authentication in the messages; this is equally effective and avoids the round for secure dissemination of keys' matrices.

Semantic validation ensures that the contents of a given message are congruent with the current execution of the algorithm. There are two ways for the congruency of messages to be verified: one is implicit and the other is explicit. The implicit way is based on whenever a process receives a message, it verifies (in V_i) if enough messages have arrived to justify the values carried by the recently received message.

The explicit way is based on broadcasting along with the message the previous messages that justify the values of the state variables. All the sent messages for the explicit semantic validation are also verified. Each state variables carried by a message are validated independently. A message passes this validation if all the following tests are passed:

Phase value — The phase value ϕ of a message requires more than $\frac{n+f}{2}$ messages from the previous phase ($\phi - 1$) to be considered valid;

Proposal value — The validation of the proposal value depends on the phase (ϕ) carried by the message:

Messages with phase $\phi = 1$: These messages do not require validation and are immediately accepted;

Messages with phase CONVERGE: The validity of the proposed value v depends if it was obtained deterministically or randomly. In the first case, it requires more than $\frac{n+f}{2}$ messages with value v and phase $\phi - 2$; in the second case, it requires more than $\frac{n+f}{2}$ messages with value $\perp$ and phase $\phi - 1$;

Messages with phase LOCK: The proposal value v is valid if there are more than $(\frac{n+f}{2})/2$ messages with phase $\phi - 1$ and value v;

Messages with phase DECIDE: If the proposal value is $v \in \{0, 1\}$, then it requires more than $\frac{n+f}{2}$ messages with phase $\phi - 1$ and proposal value v. If the proposal value is $\perp$, then it requires more than $(\frac{n+f}{2})/2$ messages with phase $\phi - 2$ and value 0 and more than $(\frac{n+f}{2})/2$ messages with phase $\phi - 2$ and value 1.

Status value — Concerning the *status* value, any message with phase $\phi < 3$ must necessarily carry the status *undecided*, because no process can decide prior to phase 3. For messages with $\phi \geq 3$, a status *decided* (for a value v) requires more than $\frac{n+f}{2}$ messages with $\phi \bmod 3 = 0$. The *undecided* status requires more than $(\frac{n+f}{2})/2$ messages with phase ϕ' and value 0 and more than $(\frac{n+f}{2})/2$ messages with phase ϕ' and value 1, where ϕ' must be the highest $\phi' \bmod 3 = 2$ lower than ϕ.

Explicit semantic validation increases the amount of exchanged data, by adding a huge amount of messages needed to be transmitted and an extra computation effort; for this reason, its usage is not mandatory.

6.2.2 MVC

Multi-Valued Consensus protocol (MVC) allows a set of processes to reach consensus over a value of any size $v \in \bigvee$; the decided value is one of the proposed values or a predefined value $\perp \ni \bigvee$.

MVC protocol requires a communication layer and a binary consensus layer. These layers are transparent to the MVC protocol. In this thesis, Turquois was used for the binary consensus layer; for the communication layer, a set of protocols were widely tested as specified in Chapter 7, where the main contributions of this thesis are presented.

The pseudo-code for MVC is presented in Algorithm 4.

The protocol starts when each process announces its proposal v_i to the others. The sent message has the following format: $\langle$*INIT, i,* $v_i\rangle$, where i is the (unique) ID of the process and v_i is its proposal. Each process accumulates the received messages in V_i and waits until $(n - f)$ *INIT* messages have arrived. If a process receives $n - 2f$ messages with same value v, it broadcasts the message $\langle$*VECT, i, v*$\rangle$. If v can not be decided, $\perp$ value is sent instead.

In the next step, the process waits until it receives $(n - f)$ *VECT* messages. If it process receives $(n - 2f)$ *VECT* messages with the same proposed value (v), the binary consensus is started with value 1 as proposal. If it is not possible to decide on a v value, the binary consensus is started with value 0 as proposal. Value v is accepted if the result of the binary consensus is 1; otherwise, the decided value is $\perp$.

```
Input: initially proposed value, $proposal_i \in \bigvee$; (unique) ID of the node $i$
Output: consensus value, $decision_i \in \bigvee$ or $\perp$ if no decision reached
$v_i \leftarrow proposal_i$;
$phase_i \leftarrow INIT$;
$status_i \leftarrow undecided$;
$V_i \leftarrow \emptyset$;
Initialization:
broadcast($\langle INIT, i, v_i \rangle$);
Task Receiver:
when $m = \langle msgPhase, j, v_j \rangle$ is received do
    $V_i \leftarrow V_i \cup \{m : m$ is valid $\}$;
    if $phase_i = INIT \wedge |\{\langle INIT, *, * \rangle \in V_i\}| > n - f$ then
        if $|\{\langle INIT, *, v \rangle \in V_i\}| > n - 2f$ then
            $v_i \leftarrow v$;
        else
            $v_i \leftarrow \perp$;
        end
        $phase_i \leftarrow VECT$;
        broadcast($\langle VECT, i, v_i \rangle$);
    else if $phase_i = VECT \wedge |\{\langle VECT, *, * \rangle\}| > n - f$ then
        if $|\{\langle VECT, *, v \rangle\}| > n - 2f$ then
            $binResult \leftarrow$ binary consensus(1);
        else
            $binResult \leftarrow$ binary consensus(0);
        end
        if $binResult = 1$ then
            $decision_i \leftarrow v$;
        else
            $decision_i \leftarrow \perp$;
        end
        $status_i \leftarrow decided$;
    end
end
```

Algorithm 4: MVC algorithm

6.2.2.1 Delayed nodes recovery

Due to the asynchronous communications model of WSN, a message being disseminated through various disjoint routes to various Base Stations can arrive to each of its destinations with a considerable delay (relatively to each others). This leads some processes to start the consensus protocol before some others, causing the agreement rate among all the participants to be very low.

The problem for a delayed process is that it will never receive the messages of the initial phases from the other processes. The reason for some processes start the consensus before others is because just $\frac{n+f}{2}$ processes are required to start the protocol.

In order to solve the problem of the delayed nodes, a mechanism for the recovery of them was developed. Each consensus' participant stores the sent messages in each protocol's phase; when it receives a message with an older phase value than its current phase ($\phi < \phi_i$), it resends the message corresponding to the previous phase (ϕ) in order that the other process can recover from the delay. To prevent cycles, a message is only resent if it is an answer to a new or non-resented message.

6.3 Multi-Valued Consensus Implementation

During the elaboration of this thesis, MVC and Turquois protocols were implemented in Java 7. These implementations were not based in the WiSeNet simulation environment (like previous implementations) as the Base Stations are supposed to have special resources, namely communication capabilities (see 2.1).

The Java implementation of MVC and Turquois was programmed based on the presented algorithms. The Javadoc API was also generated, with the help of the Javadoc Tools.

6.4 Multi-Valued Consensus Usage

As already stated in 5.2.3, in the context of this project, consensus mechanisms are used in many situations. After the study of MINSENS++, MVC and Turquois protocols, a summary of the consensus usage is now made in the following subsections.

6.4.1 Routes Consensus

In MINSENS++, during the routes selection process (5.1.2), it is necessary that all the BSs reach a consensus about each route, in order to guarantee that all the routes are disjoint.

After the generation of a route r_i by a BS bs_i, it is necessary that r_i is accepted by all the others BSs. A message with format $\langle ROUTE, i, r_i \rangle$ is then sent by bs_i; a binary consensus (with Turquois protocol) is started and all the BSs accepting r_i propose 1, while the others propose 0. If the consensus result is 1, r_i is considered valid and accepted; otherwise r_i is discarded.

6.4.2 Local Data Consensus

Local Data Consensus in MINSENS++ is the process described in 5.2.3.2.

When a set of replicas of the same message arrive to a BS through various disjoint routes, a local consensus is done in order to determine the value of the received message.

6.4.3 Distributed Data Consensus

Distributed Data Consensus is the protocol executed between BSs in order to reach a consensus over a value, described in the present Chapter. The set of values (one per Base Station) used in this consensus are, for each BS, the result from the consensus described in the Local Data Consensus (6.4.2).

7 Evaluation

In this Chapter, the assessment process and the obtained results are presented and discussed. Initially we introduce in the section 7.1 the relevant information related with the adopted testbed. This includes the rational for the MINSENS++ implementation base and software dependencies, the simulation environment used to run the WSN routing component and the hardware and software used to implement syncnodes or Base Stations, running the consensus algorithms and mechanisms.

In the section 7.2 we include the first experiments and observations for the implemented MINSENS++ Protocol.

In the section 7.3 the methodology for the consensus assessment is presented.

In the section 7.4, we present the obtained results with the consensus mechanisms. This evaluation allows us to obtain the results that will help to form conclusions relatively to the correctness and expected work of the protocols. The main focus in the assessment of this thesis is concerning the consensus protocol, which results are potentially interesting and more innovative, in the context of the MINSENS++ protocol.

In the section 7.5 we include some complementary results and more observations related with optimisations introduced in the consensus protocols and the testbed for the MINSENS++ proposed solution.

In section 7.6 an assessment to an integrated solution is made and discussed.

Finally in section 7.7 a summarised analysis of the present chapter is made.

7.1 Implementation issues and testbeds

In this section we introduce the adopted testbeds, as well as an introduction to some implementation issues.

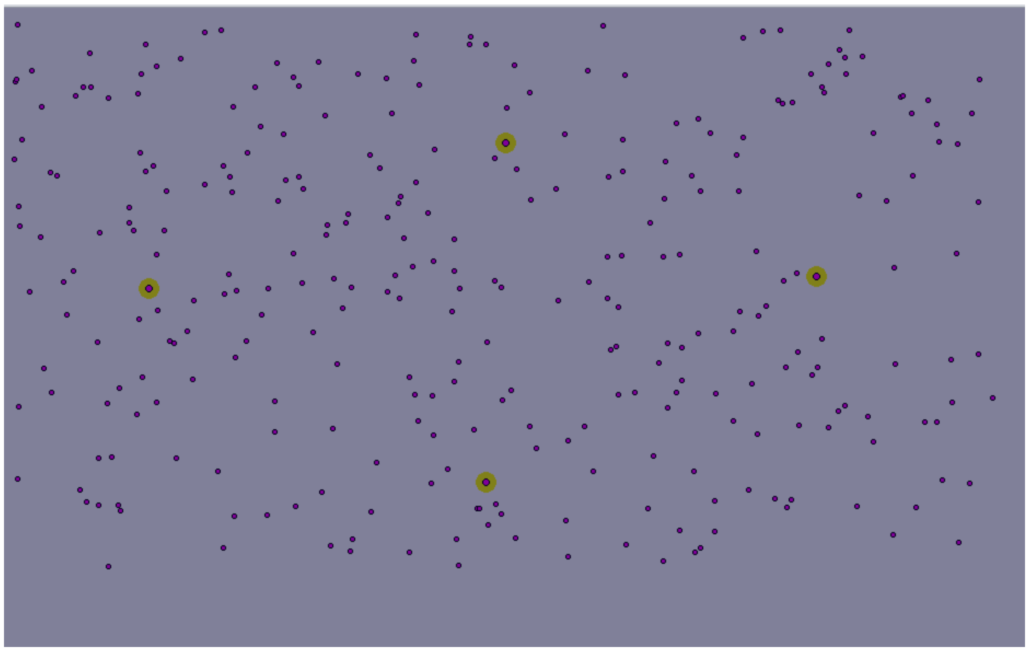


Figure 7.1: Network topology with 300 nodes and 4 base stations

We start by studying the implemented network for the MINSENS++ assessment (7.1.1) followed by the testbed for the consensus protocol (7.1.2). In the end of this section (7.1.3), we introduce Raspberry PI computers, including its specifications and some possible limitations or bottleneck previsions.

7.1.1 MINSENS++ Implementation

The assessment to MINSENS++ protocol (section 7.2) intends to evaluate its performance, as well as its correlation with the performance of the Consensus Mechanisms. The used simulation environment (WiSeNet) allows the study of large scale WSNs, from hundreds to tenths of thousands of nodes. In the presented tests, the used network topology varies in the number of nodes; three settings were used: 300, 500 and 1000 nodes. These topologies represent in the simulator 900*500, 900*800 and 900*1600 metres of terrain, respectively. Figures 7.1, 7.2 and 7.3 are graphical representations of the used topologies.

7.1.2 Consensus Protocol

The assessments' focus within this thesis are mainly on the data consensus mechanisms.

As already stated in 3.1.2, the results obtained from a real deployment are essential in order to validate and calibrate the parameters of existing simulators and protocols. To obtain such measurements, a real deployment with Base Stations was executed.

During the experiments and to represent Base Stations, Raspberry PI devices [14] Model B were used. These were the chosen devices because they are relatively weak in terms of computation, storage and memory; however, they are less limited than the

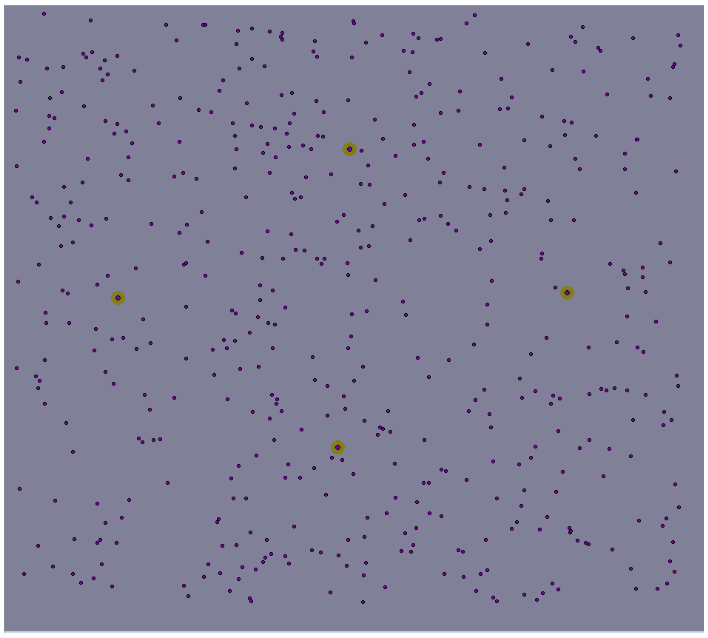


Figure 7.2: Network topology with 500 nodes and 4 base stations

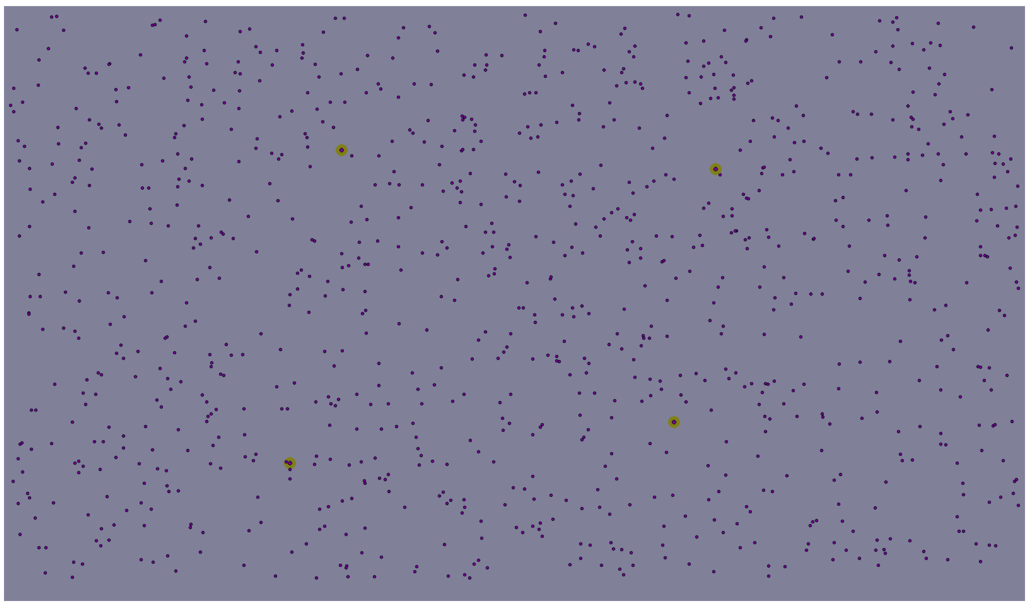


Figure 7.3: Network topology with 1000 nodes and 4 base stations

sensor nodes of the WSN (as referenced in 2.1). Therefore, these devices could easily be the ones found in a real deployed WSN, acting as Base Stations.

During the tests, the Base Stations used IEEE 802.11 communications in infrastructure mode (using a Linksys WRT54GC wireless router). The devices were randomly deployed across the space (a few meters distant from each other), accordingly to the system model for a real WSN.

A total amount of 13 nodes were used in the largest test and the methodology of the tests is presented in section 7.3. The Raspberry PI characteristics are studied further in the following subsection (7.1.3).

7.1.3 Raspberry PI nodes

The Raspberry PI computer was designed as a cheap credit-card-sized single-board computer. It was initially designed and developed in the United Kingdom, by the Raspberry PI foundation [14]. This foundation was created in 2009 as a non-profitable organisation, under the support of the Charity Commission for England and Wales (a non-ministerial government department that regulates different initiatives and projects from charity institutions). The foundation works in the direct coordination with the UK's parliament and government, and is hosted by the University of Cambridge, Computer Laboratory and Broadcom. The main objective of the Raspberry PI foundation is the promotion of basic Computer Science in schools. The Raspberry PI computer project is one of the projects associated to this objective, funded to "promote the study of computer science and related topics, especially at school level, and to put the fun back into learning computing" [14]. The Raspberry PI is manufactured under the license of manufacturing for different companies, namely: Element 14 Ltd [90], Premier Farnell [91], RS Components [92] and Egoman [93]. All these companies sell the Raspberry PI computer online, all over the world, since August 2012; the first alpha model of the system was released in August/2011 for use by the early adopters. The official launch (after a period of Beta versions) to the general consumer was on was 29 February 2012. Although the Foundation's goal was to offer two versions (A priced at US$25.00 and B priced at US$35.00), the initial release included just the B Model; Model A (with the lower cost of US$25.00) was released on 4 February 2013.

Since the model B was released, the platform became very popular and is currently inspiring different projects and initiatives by researching the use of cheap and small on-board credit-card-sized computers as possible solutions to support a range of different appliances. Some possible appliances are: on-board media-centre solutions, IP Phone devices, IP routing boxes, TCP/IP firewall nodes, Ethernet switching boxes, sensor based computing platform or Wireless Local Area Network Access Points. In the context of our dissertation, we followed these initiatives and contributions and we found an interesting motivation in the materialisation of cheap syncnodes or Base Stations, as well as future gateway solutions combined with the available technology for Wireless Sensor Networks.

7.1.3.1 Raspberry PI Architecture

The information related with the Raspberry PI computer and its architectural issues is currently available in a large documentation base in the Internet [94].

Here we include a summarised vision of the Raspberry PI architecture of the Model B - the platform used in the implementation and validation of the consensus protocol in the tests presented in the present chapter.

In the following table (table 7.1) we summarise the main hardware specifications and related characteristics of the Raspberry PI computer (model B) [94], as used in the experimental work of this thesis.

Price per unit	US$38.00 plus US$12.00 for additional power cord, DC Power Supplier (5V to 250V, 1.2A), mini USB connectors and VGA/HDMI cables
System on a Chip	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, and a single USB native port on board)
CPU	700 MHz ARM1176JZF-S core (ARM11 family, ARMv6 instruction set)
GPU	Broadcom VideoCore IV @ 250 MHz, OpenGL ES 2.0 (24 GFLOPS), MPEG-2 and VC-1, 1080p30 h.264/MPEG-4 AVC high-profile decoder and encoder.
Memory (SDRAM)	512 MB, shared with GPU
USB	2 * USB 2.0 ports (via the built-in integrated 3-port USB hub (with only one USB via, multiplexing the USB available ports, supported on in-processing software running in the CPU)
Video Input	A CSI input connector allows for the connection of a RPF designed camera module
Video Output	Composite RCA (PAL and NTSC), HDMI, raw LCD Panels via DSI connection. 14 HDMI resolutions supported, from 640*350 to 1920*1200 plus various PAL and NTSC standards.
Audio Output	3.5 mm jack, HDMI, I^2S audio (potentially used for audio input)
Onboard storage	SD / MMC / SDIO card slot (3,3V card power support only). No expansions used.
Onboard network	10/100 Ethernet (8P8C) USB adapter on the third port of the USB hub
Low-level peripherals	8 * GPIO, UART, I^2C bus, SPI bus with two chip selects, I^2S audio, +3.3 V, +5 V, ground.
Power ratings	700 mA (3.5 W)
Power source	5 volt via MicroUSB or GPIO header
Size	85.60 mm * 53.98 mm
Weight	45 g

Table 7.1: Technical specifications of used Raspberry PI nodes

7.1.3.2 Hardware architectural issues

In the context of our dissertation, it is important to add some additional notes related with the hardware architecture of the used nodes. Such details will have significant impacts in our results, as we will see further in this subsection.

- **Memory management —** In the architecture of the nodes and serial numbers that were acquired for the experimental work in the dissertation and hardware configuration as used, the total memory of 512 MB is available (both to the GPU and for the CPU). Level 2 Cache is configured to 128 KB but is used primarily by the GPU, and not by the CPU;

- **Processor —** All the acquired nodes use the ARM11 processor, designed from the version 6 specification of the ARM architecture (ARMv6)[1] [95], which due to its age is no longer supported by several popular and recent versions of Linux - including Debian, Fedora or Ubuntu which dropped support for processors prior to ARMv7 since 2009;

- **Other devices —** The nodes support a 15-pin MIPI camera interface (CSI) connector (not used in our work) and raw LCD panels available in hardware through DSI connections and standardised by the Mobile Industry Processor Interface (MIPI) Alliance. For the most part of our observations, we do not use this support, since we used ssh to have remote terminals running in the nodes (and only one of the nodes was connected to a LCD monitor - with a 1280*800 WXGA resolution - and an USB keyboard);

- **Wireless Ethernet Support —** We used a Tiny Wireless Ethernet USB EDIMAX (EW 7811 UN model) adapter that was easy to use and set up in an experimental wireless network. Such network interconnected the Raspberry PI nodes implementing the group of Base Stations and running the consensus algorithms. The main features of the EDIMAX dongles used are the following:

  - 2.4 GHz, ISB Band;
  - Low Power (<110mA) with Advanced Power Management;
  - Host Interface: USB 2.0/1.1;
  - LED: Link/Activity control.

  The used EDIMAX dongle is currently one of the smallest USB wireless adapters and supports a data rate of 150 Mbps (IEEE 802.11n) - the latest wireless standard. However the driver used in the Raspberry PI Raspbian OS only allows a stable use of IEEE 802.11b or IEEE 802.11g modes. Power Saving designed to support smart

[1]The ARM architecture describes a family of RISC-based (Reduced Instruction Set Computing) computer processors. Using a RISC based approach, ARM processors require significantly fewer transistors benefiting of lower costs, less heat, and less power usage.

transmit power control and auto-idle state adjustments are not supported by the Raspbian OS at the time of our experiments (and thus was disabled).

The most recently available firmware for the used nodes contains an option to configure five overclock options. This allows to get different performance levels out of the SoC[2], without impairing the lifetime of the hardware. Such adjustment is done by monitoring the core temperature of the processor chip and the CPU load, while dynamically adjusting clock speeds and the core voltage. When there is a low demand on the CPU or it is getting too hot, the performance is automatically throttled down; in the other hand, if the CPU has too much to do and the chip's temperature allows it, the performance is temporarily increased by increasing the clock speeds up to 1 GHz - depending on the individual board and on which of the overclock setting is used. Five settings are available:

- **None —** 700 MHz ARM, 250 MHz core, 400 MHz SDRAM, 0 overvoltage[3];
- **Modest —** 800 MHz ARM, 250 MHz core, 400 MHz SDRAM, 0 overvoltage;
- **Medium —** 900 MHz ARM, 250 MHz core, 450 MHz SDRAM, 2 overvoltage;
- **High —** 950 MHz ARM, 250 MHz core, 450 MHz SDRAM, 6 overvoltage;
- **Turbo —** 1000 MHz ARM, 500 MHz core, 600 MHz SDRAM, 6 overvoltage.

In the highest preset ("Turbo") the SDRAM clock was originally defined to 500 MHz, but this was later changed to 600 MHz because 500 MHz caused sometimes the corruption of the SD Card (which is a critical problem in the use of Raspberry PI nodes). Simultaneously, in the "High" mode the core clock speed was lowered from 450 to 250 MHz, and in "Medium" mode from 333 to 250 MHz. We observed that the processing of protocols and intensive communication patterns imply on possibly considerable load in the processor and temperature increase, so we decided to keep the configuration in the "None" setting. In fact, we observed that there were no real gains in practice in adjusting these settings for our case. Using a significant overclock setting would require more power from the power source; as the available power sources were unable to provide more power and as the CPU would require more power, such power would be diverted from the USB components. The USB is crucial for us (due to the wireless adapter and the Ethernet port) and thus such overclocks would be ineffective or counter-productive.

The referred additional notes (in the start of this subsection) are relevant for the analysis of the performance achieved with the use of the present hardware settings to support and evaluate the consensus protocols and obtain performance metrics. This is particularly important in the identification of limitations and bottlenecks at the hardware level.

[2]SoC is an acronym for System on a Chip. A SoC is an integrated circuit that integrates all the basic components of a computer or other electronic system into a single chip (namely CPU, memory, external interfaces controllers, video interfaces controllers, etc.).

[3]When the voltage in a circuit or part of it is raised above its upper design limit, this is known as overvoltage.

In particular we must emphasize some limitations related with: power supply issues, USB support solution and networking capabilities (including the wired 10-100 Mbps Ethernet natively supported via a UTP RJ45 connector in the main board but physically supported by the integrated USB controller, causing overhead to the CPU) and wireless Ethernet, only available through external USB adaptors. At the same time, the three available USB ports are all multiplexed by an internal USB hub in the same USB bus (with only one native USB port supported). The USB limitations and the power supply issues are important limitations for the performance of inter-networking and operation of the TCP/IP stack. However, this will be discussed in more depth in the chapter dedicated to the evaluation of the consensus performance, as described in the context of the following sections of this chapter.

7.1.3.3 Raspberry PI Operating System

The Raspberry PI can use as operating system Linux kernel-based operating systems, in different variants. The GPU hardware is supported by a firmware image loaded into the GPU at boot time from an SD Card (in the native on board SD card reader/writer). This firmware image is known as binary blob[4] in the initial terminology used by the development community). While the associated ARM coded Linux drivers were initially closed source, nowadays the code is partially released in open source - even that the actual driver-development work is done using the initial closed source GPU code. On 19 February 2012, the Raspberry PI Foundation released the first considered stable proof of concept of an SD Card image that could be loaded onto an SD Card to produce a preliminary operating system. The image was based upon Debian 6.0 (Squeeze), with the LXDE desktop environment and the Midori browser - plus various typical programming tools - as found in a Linux-based desktop environment. The image also runs on QEMU allowing the Raspberry PI to be emulated on many other platforms.

In our implementation we used the Raspbian Operating System, an obtained pre-installed SD Card distribution (15/July/2012 packaging) with a Linux kernel version 3.2.27+ (dc4@dc4-arm-01 release). Raspbian is a Debian-based free operating system optimised for the Raspberry PI hardware. It is the current recommended system, whose first stable version was officially released in July 2012 (although it is still in continuous development, refinement and optimisation). It is a free software although it is not developed by the Raspberry PI Foundation: in fact, the Raspbian OS is only recommended as the more stable boot. The system is based on the ARM hard-float (armhf)-Debian 7 'Wheezy' architecture port and its LXDE desktop environment, initially optimised for the ARMv6 instruction set of the Raspberry PI. It provides some available Debian distribution software packages and pre-compiled software bundles. A minimum size of 2 GB SD card is required for Raspbian, but a 4 GB SD card or above is recommended as

[4]In the context of open source software, a binary blob is a closed source binary-only driver without publicly available source code.

we could confirm. We ended using 8 GB SD Cards in our implementation setup and experiments. The downloaded Raspbian "wheezy" image file had to be unzipped and then written to a suitable SD Card, formatting it for use with efficiency. There is a server based deployment as a specific edition of Raspbian (known as the Raspbian Server Edition (or RSE). RSE is, in fact, a stripped version of the Raspbian LXDE with other software packages bundled. We used also this variant to observe possible implications in the study and analysis of the protocols developed in our dissertation.

7.2 MINSENS++ Protocol Assessment

The obtained results for MINSENS++ are presented during the following subsections. This tests are important in order to understand the correlation of them with the obtained for the Consensus Mechanisms Assessments (section 7.3).

The tests to MINSENS++ protocol were conducted over a set of various metrics (namely energy cost, connectivity, effective reliability and latency conditions), in order to understand its behaviour in a large-scale WSN environment. However and for the purposes of this thesis, we are only interested in the connectivity, effective reliability and latency conditions.

7.2.1 Methodology

In order to achieve this thesis' goals, the performance of MINSENS++ Protocol was evaluated with the following indicators:

- **Network connectivity percentage —** The ability of any node to transmit data to any other node in the network, by using the implemented protocol. It is calculated as the percentage of the relation between the covered nodes (those sending messages successfully delivered) and the total number of sender nodes;

- **Network reliability percentage —** The quality of communication, given the transmission method. It is calculated as the percentage of the relation between the successfully received messages and the total number of sent messages;

- **Latency —** The measurement of the average number of hops that a message travels before arriving to its destination.

The results of the presented tests are calculated as the average values for all the analysed indicators; for each test, 10 observations are made. Each of the observations uses a different set of sender nodes, chosen randomly by the simulator. This randomness simulates the real randomness of a real WSN, where the senders can be any participating node.

Every observation for each test was made under the same network topology and protocol instantiation. The measurements for the tests are made after the start of the protocol (thus after the setup phase), allowing the ad-hoc network organisation to be built without influencing them. This allows also to have the maximum amount of stable nodes[5].

7.2.2 Results

In this section, the observed results (accordingly with the presented Testbed and Methodology) are presented. For the results presentation, a comparison between INSENS and MINSENS++ Protocols is made for each test, allowing to take a glance at both of them.

MINSENS++ Protocol (in opposition to INSENS) is highly customisable, namely regarding to the number of Base Stations or the transmission scheduling policy. For simplicity and accordingly with the needs for this thesis, the presented tests use MINSENS++ with 2 Base Stations and with a round-robin scheduling policy (when choosing the destination Base Station).

[5]A node is considered stable when it is part of the routing process and is able to send and forward messages.

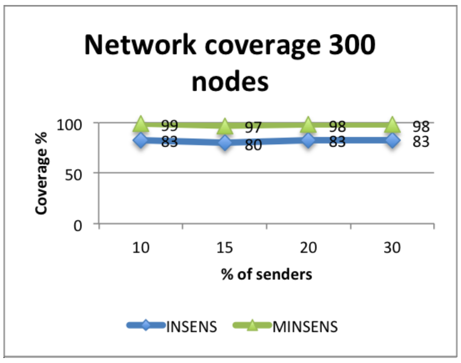
7.2.2.1 Network Connectivity

For the network connectivity test, the following parameters were used:

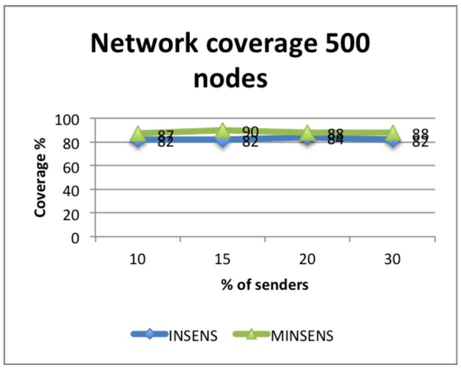
Parameter	Defined Setting
Number of stable nodes	INSENS - 250/426/680 MINSENS++ - 297/490/996
Number of observations	10
Percentage of sender nodes	10% to 30%
Number of Base Stations	INSENS - 1 MINSENS++ - 2

Table 7.2: Parameters used for MINSENS++ connectivity test

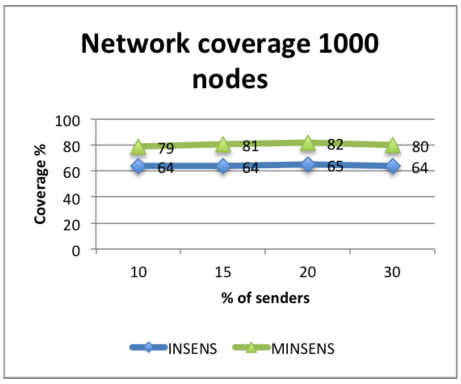
The observed results with the above parameters are the presented in Figure 7.4.



(a) with 300 nodes



(b) with 500 nodes



(c) with 1000 nodes

Figure 7.4: Network coverage with INSENS and MINSENS++ Protocols

Network connectivity is directly related and limited by the number of stable nodes of the network. As we can see in the used parameters, a significantly bigger number of stable nodes was achieved by MINSENS++ (for every number of nodes of the network). This leads MINSENS++ to offer a superior network coverage when compared to INSENS.

MINSENS++ scales thus better, as we can see by Figure 7.4(c) where connectivity is about 33% superior when compared to the INSENS Protocol.

7.2.2.2 Network Reliability

For the network reliability test, the following parameters were used:

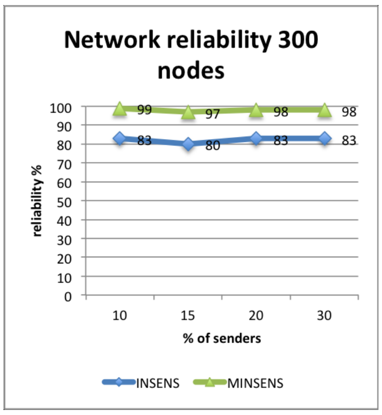
Parameter	Defined Setting
Number of stable nodes	INSENS - 250/426/680 MINSENS++ - 297/490/996
Number of observations	10
Percentage of sender nodes	10% to 30%
Number of Base Stations	INSENS - 1 MINSENS++ - 2

Table 7.3: Parameters used for MINSENS++ reliability test

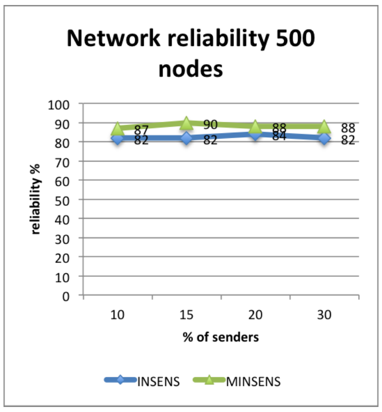
The observed results with the above parameters are the presented in Figure 7.5.

Like in network coverage, the network reliability is directly related and limited by the number of stable nodes of the network, as the source nodes are chosen from all the nodes of the network. Once again, reliability is better in MINSENS++ Protocol for every number of nodes of the network.

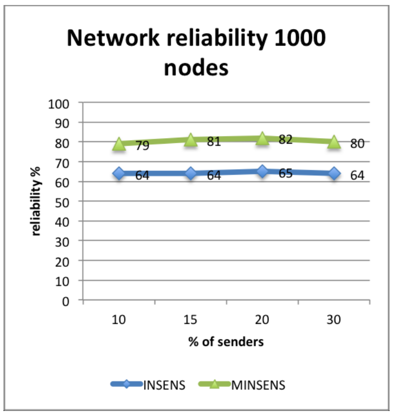
MINSENS++ showed again to scale better then INSENS, with reliability being about 20% better in MINSENS++ (Figure 7.5(c)).



(a) with 300 nodes



(b) with 500 nodes



(c) with 1000 nodes

Figure 7.5: Network reliability with INSENS and MINSENS++ Protocols

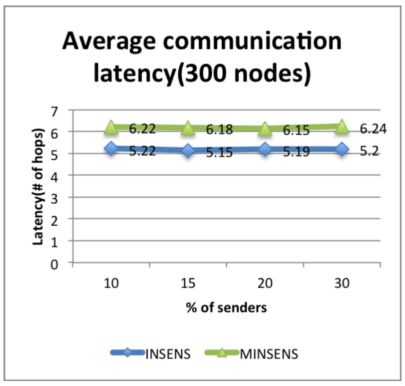
7.2.2.3 Latency

For the latency test, the following parameters were used:

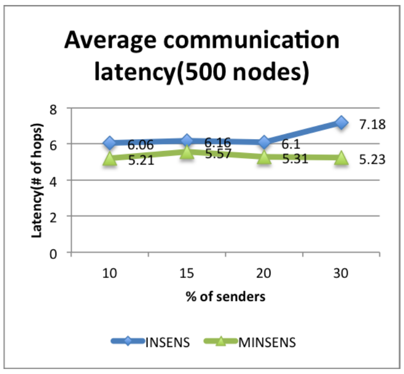
Parameter	Defined Setting
Number of stable nodes	INSENS - 250/426/680 MINSENS++ - 297/490/996
Number of observations	10
Percentage of sender nodes	10% to 30%
Number of Base Stations	INSENS - 1 MINSENS++ - 2

Table 7.4: Parameters used for MINSENS++ latency test

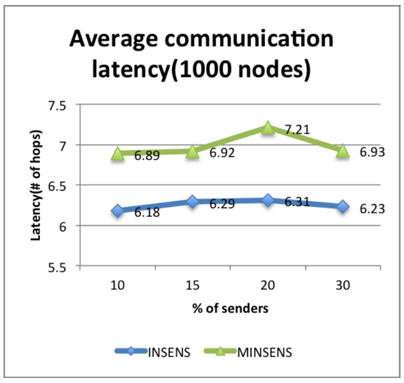
The observed results with the above parameters are the presented in Figure 7.6.



(a) with 300 nodes



(b) with 500 nodes



(c) with 1000 nodes

Figure 7.6: Network latency with INSENS and MINSENS++ Protocols

Although with better results in network coverage and reliability, MINSENS++ showed worse results in the latency tests, when compared to INSENS. However, the obtained results are very close between INSENS and MINSENS++ and are inconsistent across topologies (with the MINSENS++ being better with 500 nodes).

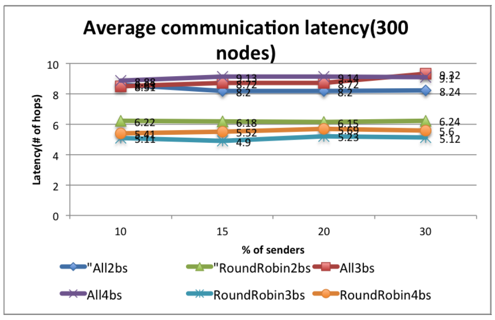
7.2.2.4 Latency (revisited)

For this latency test, various combinations on the number of Base Stations and on the scheduling policy (when choosing the destination Base Station) are made. The following parameters were then used:

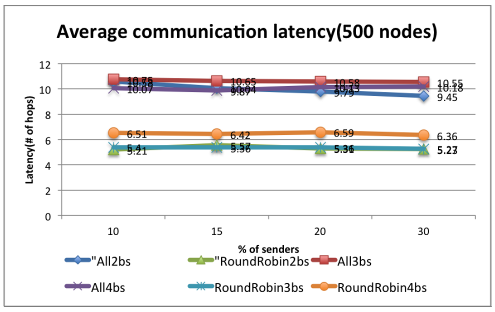
Parameter	Defined Setting
Number of observations	10
Percentage of sender nodes	10% to 30%
Number of Base Stations	2 to 4

Table 7.5: Parameters used for MINSENS++ second latency test

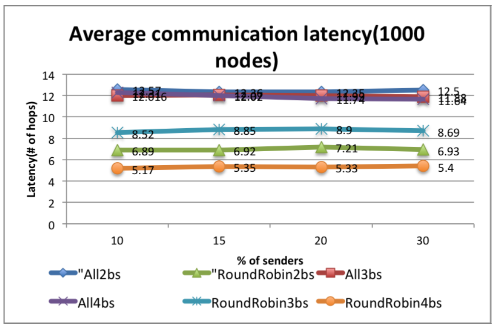
The observed results with the above parameters are the presented in Figure 7.7.



(a) with 300 nodes



(b) with 500 nodes



(c) with 1000 nodes

Figure 7.7: Network latency with various Base Stations

These results show that on the 1000 nodes topology, the bigger the number of available Base Stations, the better the average communication latency. Also, the round robin scheduling policy is better than sending the same message to all the BSs.

For the scope of this thesis, the latency results are the most important as they allow to understand the correlation between the messages arriving rate to Base Stations and the consensus times. These results will thus be recovered later on section 7.6.

7.3 Consensus Protocol Assessment and Methodology

The performance metric used in the experiments is the *latency*. This metric is always relative to a given process p_i, and is denoted as the interval of time between the moment that the process proposes a value to a consensus execution and the moment it decides.

The average latency for the set of processes is obtained in the following way: A signalling machine (that does not participate in the consensus protocol) coordinates the experiment; it starts by sending a virtually measured value (as if it was a sensor node in a WSN) to each process, over a TCP connection. A process receiving such message, assumes the received value as originated in a WSN and considers it as the proposal value; the process stands then on hold. After all the participating processes received the proposal message, the signalling machine sends a broadcast message in order to the processes start the consensus; when a process receives such message, it starts the consensus execution. Each process records the latency value and after the end of the protocol, such value is printed in the console. The average latency is obtained by executing 3 consensus instances, and then averaging the latencies collected by all processes through all the instances.

The experiments were made in various combinations of group size, communication protocol, presence of failures and configuration refinements. The group size defines the number of processes (nodes) in the system; in our case, the values were 4, 7 and 10 nodes. The communication protocol defines the protocol used by the nodes to communicate among them; the implemented communication protocols were UDP, UDP Multicast, Non-Persistent TCP and Persistent TCP[6]. The presence of failures respects to the number of nodes present in the consensus group that are acting as malicious. Within this criteria, two experiments are made: without any fail ($f = 0$) and with $\max f : f < \frac{n}{3}$. Malicious nodes (f nodes) broadcast an erroneous predefined value, simulating a Byzantine failure.

Concerning the configuration refinements, during the implementation of the algorithms some sleep timers were implemented. This allows a given portion of code to run slower (making the process to sleep for a certain amount of time). Although at a first glance setting all the timers with the zero value could seem a pretty good decision, this would increase the messages collisions rate. A more careful approach needs to be done to these configuration refinements, and the results of such refinements are presented following in this Chapter.

7.4 Assessment for Unanimity Conditions

In the following subsections, the observed results for the Consensus Mechanisms are presented.

[6]The difference between Non-Persistent TCP and Persistent TCP concerns the persistence properties of the TCP connection, as exposed in section 7.4.1.5.

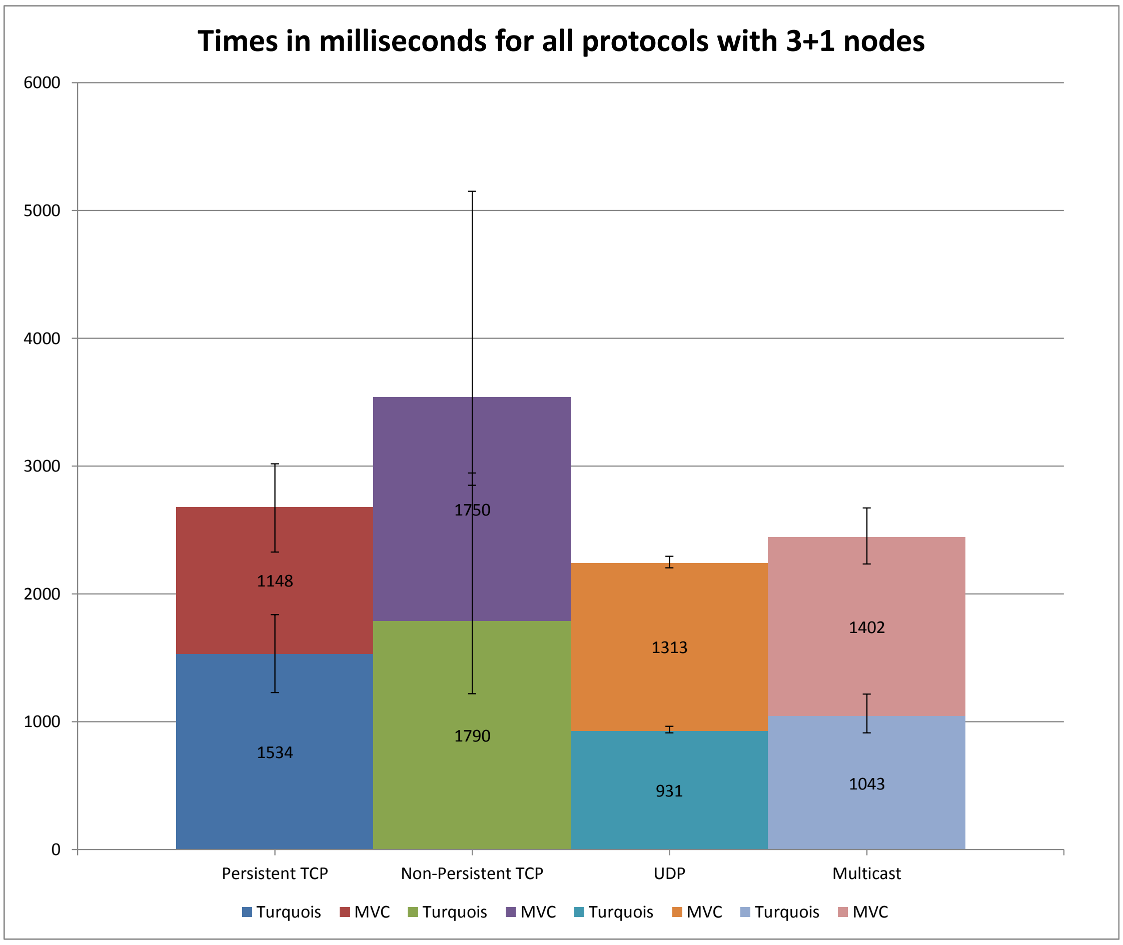


Figure 7.8: Comparison between all protocols for 3+1 nodes

7.4.1 Results

In this section, the observed results are presented. During the tests, when we refer to a group of nodes as being $x+y$ nodes, we mean that in a group of z nodes (with $z = (x+y)$), x of them are working correctly while y are attacked and acting as malicious, introducing Byzantine failures.

Figure 7.8 compares the results for each protocol with 4 nodes. In all the performed tests and as showed by the chart, UDP and UDP Multicast Protocols seem to always be better than TCP (both Persistent and Non-Persistent) Protocols; this means that UDP and UDP Multicast Protocols seem to be the most interesting protocols with the best results for this kind of application. With this result in mind, in the next subsections further results are presented for the tests where UDP and UDP Multicast Protocols were used. However, an analysis to Non-Persistent TCP and Persistent TCP Protocols is made in subsection 7.4.1.5.

The following subsections are organised as follows: results are first divided by the size of the group (amount of nodes) and then by communication protocol. The impact of the presence of failures for the performance of the protocols is discussed in 7.4.1.6.

The configuration refinements specified in the charts use the terminology explained by Table 7.6.

Refinement	a	b	c	d	e	f	g	h
MVC Sleep Time Between Receptions	50	50	50	50	25	25	25	0
MVC Sleep Time Between Checks to Turquois	500	0	0	0	0	0	0	0
MVC Sleep Time Between Sends	500	500	300	200	150	125	100	0
Turquois Sleep Time Between Receptions	50	50	50	50	25	25	25	0
Turquois Sleep Time Between Sends	500	500	300	200	150	125	100	0
Application Sleep Time Between Checks to MVC	500	0	0	0	0	0	0	0

Table 7.6: Configuration refinements terminology (all times in milliseconds)

In the previous table (Table 7.6), the refinement settings have the following meanings in the context of the implementation of the algorithms:

- **MVC Sleep Time Between Receptions —** This time interval corresponds to the time that, in the MVC layer, the protocol waits between two consecutive message receptions. This refinement allows to avoid the overload of the CPU, if suddenly many messages arrive to this layer;

- **MVC Sleep Time Between Checks to Turquois —** This time interval corresponds to the time that the MVC layer waits between two consecutive requests to the Turquois layer, asking about the results of the binary consensus. This refinement allows to avoid the useless usage of the CPU due to busy waiting[7], allowing it to perform other tasks;

- **MVC Sleep Time Between Sends —** This time interval corresponds to the time that, in the MVC layer, the protocol waits between two consecutive message sends. This refinement tries to constrain the overload of the medium, by restricting the sends' frequency;

- **Turquois Sleep Time Between Receptions —** This time interval corresponds to the time that, in the Turquois (binary consensus) layer, the protocol waits between two consecutive message receptions. This refinement allows to avoid the overload of the CPU, if suddenly many messages arrive to this layer;

- **Turquois Sleep Time Between Sends —** This time interval corresponds to the time that, in the Turquois layer, the protocol waits between two consecutive message

[7]Busy waiting is a technique in which a process repeatedly checks to see if a condition is true. Busy waiting itself can be made much less wasteful by using a delay function (e.g., sleep()) found in most operating systems. This puts a thread to sleep for a specified time, during which the thread will waste no CPU time. If the loop is checking something simple then it will spend most of its time asleep and will waste very little CPU time.

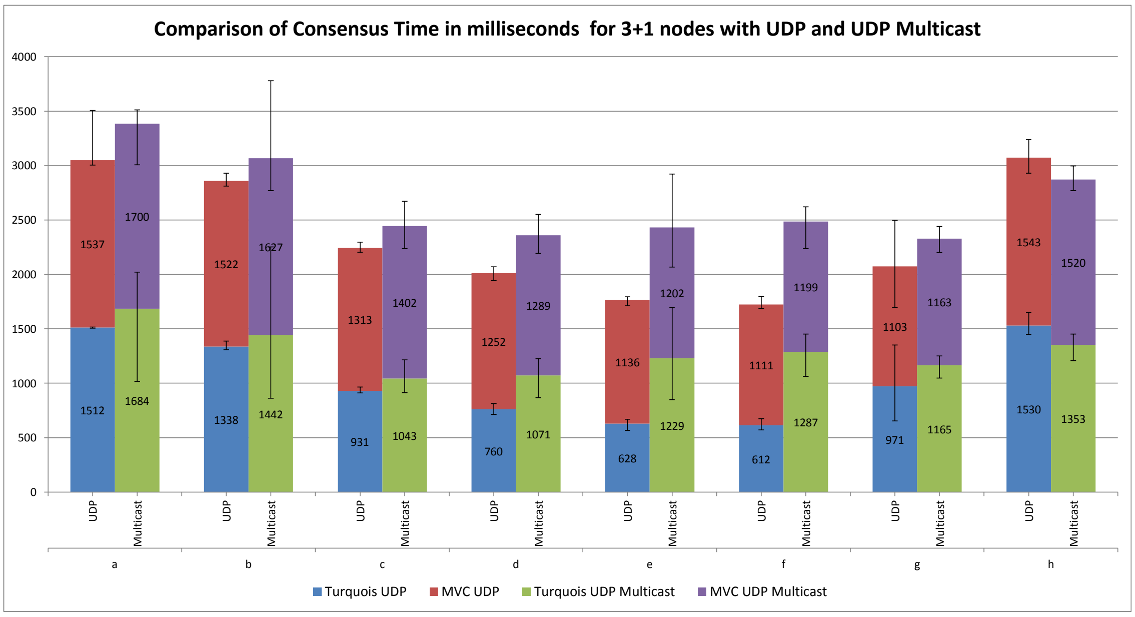


Figure 7.9: Comparison between UDP and Multicast protocols for 3+1 nodes

sends. This refinement tries to constrain the overload of the medium, by restricting the sends' frequency;

- **Application Sleep Time Between Checks to MVC —** This time interval corresponds to the time that the application waits between two consecutive requests to the MVC layer, asking about the results of the multi-valued consensus. This refinement allows to avoid the useless usage of the CPU due to busy waiting, allowing it to perform other tasks.

7.4.1.1 Group size with 4 nodes

In this tests, the group size is of 4 nodes. The obtained results were the presented in Figure 7.9.

For the group with 4 nodes and by using the UDP Protocol, we can verify that the results are always better than those when using UDP Multicast, except for the most *aggressive* case (with no sleeps between sends or receives). Therefore, the UDP Multicast Protocol does not offer any advantage over the UDP Protocol, except for an extremely *aggressive* choice of the refinements (case *h*), when there are no sleep times between sends or receives.

In the first cases, the amount of messages exchanged over the medium is still not significant; this allows the communication mechanisms to perform correctly with UDP. However, in the last case the high rate of exchanged messages causes a lot of collisions; this leads to a high rate of dropped messages. UDP Multicast Protocol reduces the number of exchanged messages (by its multicast nature) and that's the reason for the collisions rate reduction, thus having better results than UDP.

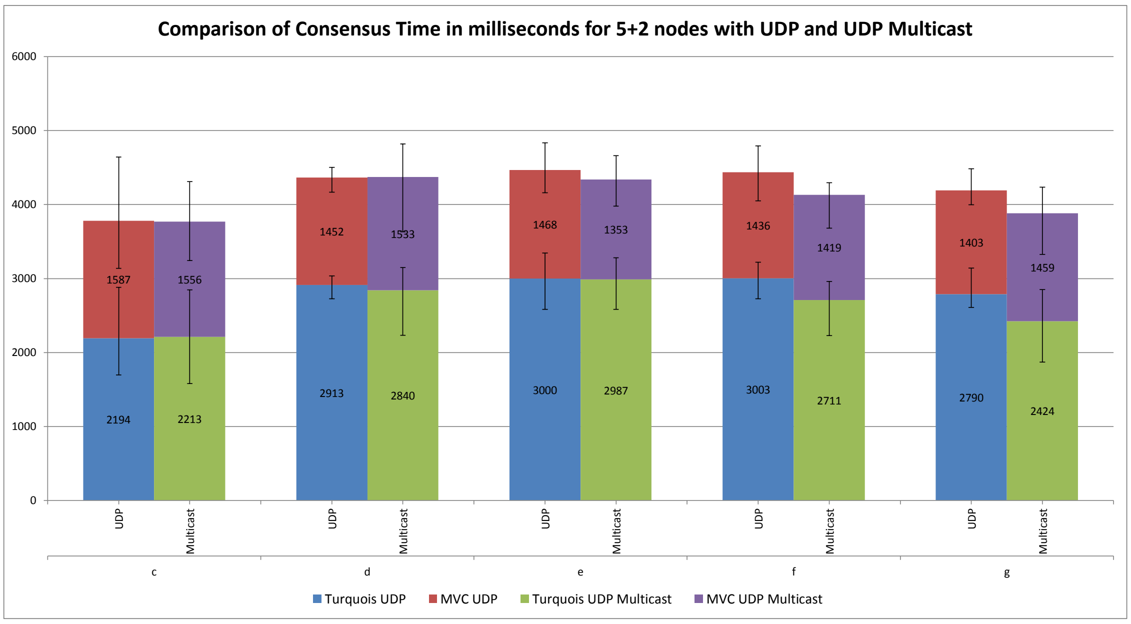


Figure 7.10: Comparison between UDP and Multicast protocols for 5+2 nodes

7.4.1.2 Group size with 7 nodes

In this tests, the group size is of 7 nodes. The configuration refinements set was reduced in order to focus on those with better results whose consensus process succeeds; the 5 best refinements for 4 nodes were then chosen to continue the tests. The obtained results were the presented in Figure 7.10.

In this tests with 7 nodes, an inversion between UDP and UDP Multicast Protocols can be clearly observed. In configurations c and d, the results for both protocols are nearly the same. However, as the sends / receives sleep times are reduced, UDP Multicast Protocol starts to show a better performance than UDP Protocol. UDP Multicast avoids the increase of collisions number that occur in UDP Protocol by reducing the amount of exchanged messages.

7.4.1.3 Group size with 10 nodes

In the following tests, the group size is of 10 nodes. The configuration refinements set was reduced again (to focus on those with better results whose consensus process succeeds) and the 3 best refinements for 7 nodes were chosen to continue the tests. The obtained results were the presented in Figure 7.11.

The obtained results start to evidence that the choice between the UDP Protocol or the UDP Multicast Protocol depends on the user's profile. If the user is more conservative and prefers configurations with greater sends / receives sleep times, then the UDP Protocol shall be chosen; however, if the user is less conservative and prefers configurations with more *aggressive* sends / receives sleep times, then the UDP Multicast Protocol shall the chosen.

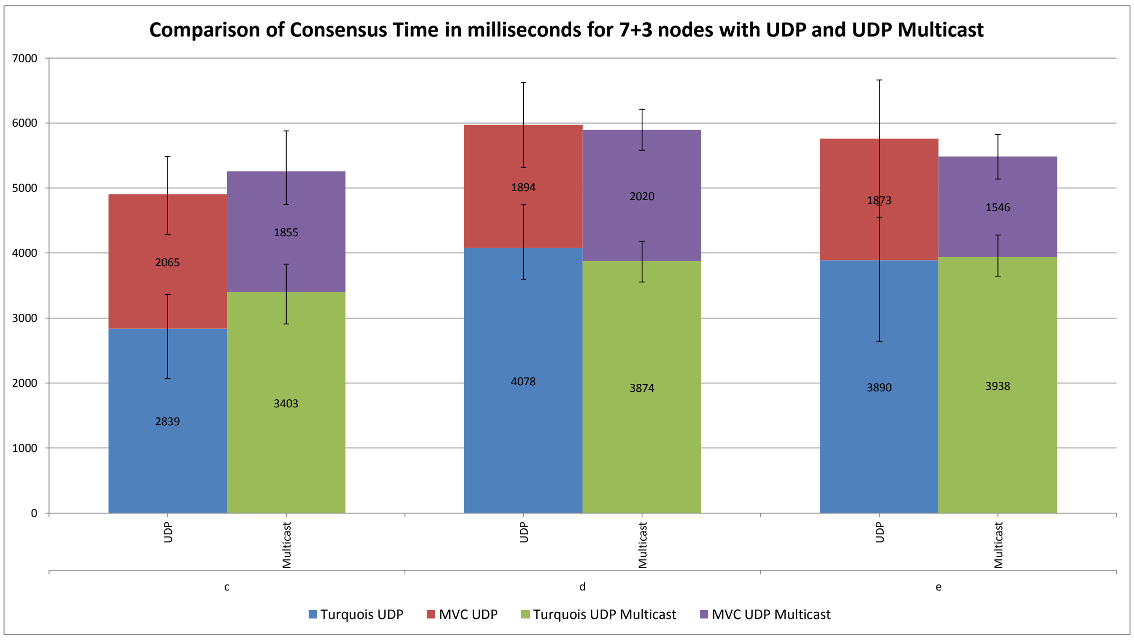


Figure 7.11: Comparison between UDP and Multicast protocols for 7+3 nodes

These results evidence that the UDP Multicast Protocol is a better choice when a lot of collisions occur over the air and that the UDP Protocol is a better choice when there is a small amount of collisions occurring, due to slower communications. UDP Multicast Protocol reduces the collisions by drastically reducing the exchanged messages by its multicast philosophy, as one single message has various destinations.

7.4.1.4 Group size with 13 nodes

Although the tests were supposed to be performed just over a maximum of 10 nodes, we tried to reach more results by increasing the group size to 13 nodes. However, with 13 nodes it was observed that the consensus process was so unstable that it was almost impossible to reach a consensus (thus leading the mechanism to start failing). The best obtained value was a total consensus time of 6750 milliseconds (2305 milliseconds during the MVC Protocol and 4445 milliseconds during the Turquois Protocol).

This result reveals that the chosen combination of Hardware and Network constrain the number of nodes for these protocols.

7.4.1.5 TCP Protocol Considerations

The tests were also performed with two variants of the TCP Protocol. In the first variant (Non-Persistent TCP Protocol), each time a message needs to be transmitted, a new connection is opened; then the message is transmitted and finally the connection is closed.

In the second variant (Persistent TCP Protocol), all the connections among the Base Stations (all to all) are opened before the start of the protocol; whenever bs_i needs to send a message to bs_j, the former checks its connections table and obtains the opened socket to the latter. A *watchdog* process runs in background, recovering any connections that may accidentally go down.

As already stated in 7.4.1 and viewed in Figure 7.8, both the two variants of the TCP Protocol have the worst results, when compared to the variants of the UDP Protocol.

For a big group of nodes, it was observed that in our study the TCP Protocol based consensus do not decide in a reasonable time.

7.4.1.6 Presence of failures

During the execution of the tests for 4, 7 and 10 nodes, tests with no Byzantine failures were also performed. These tests intended to observe if any relation between the presence or not of failures would exist.

These tests were similar to the others, except that all the nodes were (at start) proposing the exactly same value.

With these tests we could observe that there is no direct relation between the presence or not of failures. The obtained results were similar to those from the tests performed with Byzantine failures injection. In both cases the number of exchanged messages is similar, and this causes both tests to have similar results.

7.4.2 Critical analysis for the obtained results

During the assessment phase of the consensus mechanisms, 4 communication protocols were implemented: Non-Persistent TCP, Persistent TCP, UDP and UDP Multicast.

TCP Protocols (both Persistent and Non-Persistent variations) showed to be always worse than the UDP based protocols. For that reason, the major part of the assessments were made by just considering the UDP and UDP Multicast Protocols.

When the number of nodes increases, UDP Multicast revealed to be better than UDP. With a great number of nodes, the number of exchanged messages also increases, thus leading to the increase of the collisions rate; UDP Multicast allows to reduce the number of exchanged messages (and thus reduce the collisions rate) due to its multicast philosophy.

For the same number of nodes, there are also differences between UDP Multicast and UDP accordingly to the configuration refinements. For more conservative configurations (high sends / receives sleep times) UDP Protocol showed the best results while for less conservative configurations (low sends / receives sleep times) the UDP Multicast Protocol is desirable.

The presence or not of failures showed however that it has no influence in the observed results, as (for the same configuration) the number of exchanged messages is similar in both cases.

7.4.3 IPSec Considerations

Additionally to the presented assessment, we studied a possible approach to communications with IPSec[8] in Raspberry PI nodes, in order to anticipate some possible impacts of such protocol in the consensus results.

IPSec protocol offers security guarantees, such as confidentiality, integrity and authentication through the usage of cryptography.

IPSec can operate in two distinct modes. In the transport mode, only the payload of the IP packet is usually encrypted and/or authenticated; in the other hand, in the tunnel mode the entire IP packet is encrypted and encapsulated into a new IP packet with a new IP header (allowing the original packet to securely cross insecure networks).

The testbed for this test is composed by two Raspberry PI nodes (with the previous base configurations), sending files of different sizes through the TFTP protocol[9]. A mechanism for automatic keys distribution was used, as the used kernel version did not support manual keys distribution.

Firstly we measured the transfer latency through IEEE 802.3. The obtained results are presented in table 7.7; note that a zero value on the table means that the transfer time is negligible.

File Size	Without IPSec	With IPSec
1KB	0	0
10KB	0	0.1
100KB	0.24	0.5
256KB	0.58	1.2
512KB	1.14	2.3
1MB	2.38	4.7
10MB	23.26	47.44

Table 7.7: Latency times (in seconds) of a file transfer with and without IPSec in a wired network

After the first test, the Raspberry PI nodes were configured to operate in a wireless network (IEEE 802.11) and the obtained results are presented in table 7.8.

From the observations we made, we can conclude that the usage of the IPSec protocol significantly increases the latency of the network. In the first tests (wired network) the latency duplicates when using IPSec; in the second tests (wireless network) the differences in latency are even more evident.

Although the usage of IPSec eliminates many attacks to a computer network, it introduces an overhead to the participating devices. In the case of low computation power

[8]Internet Protocol Security (IPSec) is a protocol suite for securing Internet Protocol (IP) communications by authenticating and/or encrypting each IP packet of a communication session. IPSec also includes protocols for establishing mutual authentication between agents at the beginning of the session and negotiation of cryptographic keys to be used during the session.

[9]Trivial File Transfer Protocol (TFTP) is a file transfer protocol notable for its simplicity. It is extremely limited and does not provide authentication.

File Size	Without IPSec	With IPSec
1KB	0.22	2
10KB	0.32	10
100KB	12.18	71
256KB	29.9	149
512KB	280.84	328

Table 7.8: Latency times (in seconds) of a file transfer with and without IPSec in a wireless network

devices (our case with Raspberry PI computers), such overhead have a significant impact in the network latencies, and thus IPSec is not the most advised.

7.5 Complementary Assessment under Different Settings

7.5.1 Motivation and Settings

Revisiting the limitations on the Raspberry PI nodes (7.1.3.2), namely the networking capabilities, the following thought has came into account: *Would it be possible that Raspberry PI nodes' architecture is causing a bottleneck on our solution?*. Another possible thought could be *Could we be more optimistic and, in some way, relax the approval conditions of a value in order to decide faster?*. In this section (7.5) we present several complementary assessments under different settings, having in mind the obtained results from the assessment tests of section 7.4.

7.5.1.1 Raspberry PI USB bottleneck

Considering the first thought (and having the Raspberry PI limitations in mind), we decided to perform some assess to the networking support of Raspberry PI nodes with the iperf tool[10]. Such assessment is relevant to understand the severity of the networking limitations on Raspberry PI nodes, and thus understand better the performance of the consensus mechanisms assessed in this dissertation. For such tests, two versions of the Raspberry PI were prepared:

- **Raspberry PI Standard Version (RSV) —** This version corresponds to the normal Raspberry PI node, as already used in all the previous tests;
- **Raspberry PI Optimised Version (ROV) —** This version corresponds to a Raspberry PI with a more recent kernel version and with an available patch applied. This patch, for CPU performance and USB interrupt rate reduction (called *Gordon's*

[10]Iperf is a tool for network performance measurement written in C++. Iperf was developed by the Distributed Applications Support Team (DAST) at the National Laboratory for Applied Network Research (NLANR), a research lab that merged with the University of California San Diego's CAIDA group, United States of America

FIQ Fix), allows to reduce the USB interrupt rate and thus improve the general performance of the CPU by about 10% and improving by 20% the data-losses in the USB bus. This patch was obtained from USB GitHub rpi-update firmware, provided by the Element 14 distributor [90].

To support the tests, we used the following machines:

- **Machine A —** A computer running Mac OS X version 10.7.5 with a 3.1GHz Intel Core i5 CPU, 4GB (1333MHz DDR3) RAM and native Wi-Fi and wired Ethernet connections;
- **Machine B —** A Raspberry PI node with the specified configurations (RSV or ROV).

In our evaluations we used first the Machine A as a server in the iperf tool and the Raspberry PI node (Machine B) as a client; then roles were changed and new observations were made. All the results were observed with 10 measurements, sending IP packets (of 1024 bytes) during 60 seconds. The presented results were calculated as the average of the respective observations. Below we present the observed facts about the Raspberry PI nodes' limitations.

Iperf results

The first test made with iperf tool (as usually used in a unix environment) corresponded to a test running among Machine A and the Raspberry PI through wired Ethernet. The obtained results are the present in table 7.9.

Raspberry PI version	Raspberry PI as client	Raspberry PI as server
RSV	28 Mbps	26 Mbps
ROV	36 Mbps	36 Mbps

Table 7.9: Iperf results with wired Ethernet

The second test consisted on using wireless communications (IEEE 802.11g) structured with a Linksys Broadband Wireless-G Router (Model WRT54GC) with DHCP server. The obtained results are the ones in table 7.10.

Raspberry PI version	Raspberry PI as client	Raspberry PI as server
RSV	24 Mbps	24 Mbps
ROV	32 Mbps	28 Mbps

Table 7.10: Iperf results with structured IEEE 802.11g

Finally, in the last test we used ad-hoc wireless communications (IEEE 802.11g) and fixed IP addresses. The results are presented in table 7.11.

In all the performed tests, CPU usage rate was always nearly the 100%; this illustrates how the I/O operations are CPU-intensive in the Raspberry PI computers. We can observe that in all the cases, the performance was always far from the theoretical 54 Mbps

Raspberry PI version	Raspberry PI as client	Raspberry PI as server
RSV	25 Mbps	24 Mbps
ROV	32 Mbps	26 Mbps

Table 7.11: Iperf results with ad-hoc IEEE 802.11g

of the IEEE 802.11g protocol. The Raspberry PI Optimised Version (ROV) showed however to always have a better performance in all the test cases, when compared with the Raspberry PI Standard Version (RSV). This points to the value of the applied patch; considering such results, in all the following tests the Raspberry PI computers have always the patch applied (corresponding to the ROV).

HTTP test results

After the assessment to the TCP/IP stack performance, we decided to test the application-level performance, namely using HTTP/TCP. An Apache HTTP server[11] was then installed on the Machine A and a 100 MByte data file was deployed to it (in order to test the download times and throughputs by the Raspberry PI computers). The wget tool was used to download the 100 MByte file to Machine B (Raspberry PI Optimised Version) and the average velocity of 2.2 Mbps was achieved. This value was observed when, while downloading the file, the CPU was pushed to 100% load. The download tests ran 10 times consecutively with, however, an interval of 5 minutes between each – in order to allow the device to recover a steady state, in terms of temperature and power consumption.

The following test without the CPU fully available to the USB bus relied on a call to the X Window server[12]. The X Window server was then started in the Raspberry PI, using an ASUS LCD Monitor (with a resolution of 1024*768 pixels) connected via the HDMI port, during the download of the file. We could observe that the download slowed down to about 60 Kbps. When the X Window server was stopped, the download rate increased to about 1.6 Mbps. An interesting fact is that the observations were similar both for the wired or wireless connections and also for the different configurations presented in the previous tests.

After some research work over the Internet, we found several discussions about the performance of the USB controller used in the Raspberry PI's SoC (Broadcom BCM2835). Some possible reasons pointed are buggy drivers (causing the serious problems currently noticed) and the fact that Raspberry PI can only supply an unusually low amount of power to its USB devices (approximately 140 mA). One possible solution to solve the second problem would be the usage of powered hubs; it was however reported that this

[11]The Apache HTTP Server is a web server software program developed and maintained by an open community of developers under the auspices of the Apache Software Foundation.

[12]The X Window System is a computer software system and network protocol that provides a basis for graphical user interfaces (GUIs) and rich input device capability for networked computers. The X.Org Foundation leads the project, with the current reference implementation available as free and open source software under the MIT License and similar permissive licenses.

solution aggravates the issues caused by the USB drivers. The Ethernet connections are also directly affected by this problem because, as already studied, the Ethernet port is connected to the SoC via USB. This causes a significant amount of lost packets, total loss of network connectivity or even crashes (depending on the used power supply). We found several attempts to fix the buggy USB drivers but unfortunately none of them achieved full success until now. Despite, some cases reported possible bandwidths over 80 Mbps, with specific OS optimisations (using other kernel distributions) and specific USB dongle hardware (not used in our testbeds).

Power suppliers issues

The power supplier seems to be pointed as the origin of many issues and malfunctioning cases of Raspberry PI computers by the adopters' community. The overload of the power supply seems to cause different problems with devices connected to the Raspberry PI, namely erratic operation or stop of keyboard, mouse and serious network problems. The action of hot plugging[13] is also reported as a possible origin of crashing other devices or the system. Finally, under significant network usage, the device may crash or have a high packet losses rate (usually when the power supply is limited to 5V and 1A) as the network processing is made by the CPU via the USB bus and driver.

IP Packet Losses Evaluation

We tested the loss of IP packets by measuring the performance of ICMP protocol. To do so, we made several observations by running Ping tests[14] (with 64 bytes ICMP messages) in parallel, from 10 Raspberry PI nodes to other 10 Raspberry PI nodes (in pairs) on a wireless LAN in infrastructure mode. We observed that the round-trip times (RTT) were of 3ms to 5ms, with no relevant difference between different nodes. However, if the CPU load was increased, the packet loss ratios were the presented on table 7.12.

CPU Load	50%	80%	100%
Packet Loss Ratio	30%	50%	80%

Table 7.12: Relation between CPU Load and Packet Loss Ratio on Raspberry PI

In another experiment, the CPU load was set to 20%. Using the iperf tool to regulate the bandwidth, we observed that with the increase of bandwidth the packet losses increase significantly as presented in table 7.13. We also noticed that on the last setting (25 Mbps) the CPU load was about 100%.

A last test consisted in 10 Raspberry PI sending ping packets to the air, using as destination the multicast IP address 224.0.0.1. In this situation we observed a packet loss near 100% when sending the packets at 30 Mbps.

[13]Hot plugging is a term used to describe the functions of replacing computer system components without shutting down the system.

[14]Ping is a computer network administration utility used to test the reachability of a host on an Internet Protocol (IP) network and to measure the round-trip time for messages sent from the originating host to a destination computer.

Bandwidth	5 Mbps	10 Mbps	15 Mbps	20 Mbps	25 Mbps
Packet Loss Ratio	1%	2%	40%	65%	90%

Table 7.13: Relation between network bandwidth and Packet Loss Ratio on Raspberry PI

7.5.1.2 Testing topologies

Concerning the second thought introduced in 7.5.1, we can be more optimistic in the tests' topologies.

The *k-consensus* problem [89, 87] considers a set of n processes where at least k of them have to decide on a common value proposed by one of the processes (in our case, $k = n - f$ and $f = \frac{n-1}{3}$). The remaining non-Byzantine processes (at most $n - k$) do not necessarily have to decide; however, if they decide, they are not allowed to do it on a different value. This means that we can be more optimistic and, instead of requiring that all the nodes decide on the correct value, require just k of them to do it.

The reasons from 7.5.1.1 and 7.5.1.2 lead us to a new assessments set where the Raspberry PI nodes have the USB interrupt rate reduction patch applied and where just k nodes are necessary to reach a consensus. In the new topology, there are three fault types:

- **Failure-Free Setting —** In this setting (section 7.5.2) all the nodes behave correctly;
- **Fail-Stop Setting —** In this setting (section 7.5.3), $f = \frac{n-1}{3}$ nodes crash during the consensus process;
- **Byzantine Setting —** In this setting (section 7.5.4), $f = \frac{n-1}{3}$ nodes try to avoid the correct processes from reaching a decision, by attacking the protocol's execution. A Byzantine node proposes the opposite value that it would propose if it were behaving correctly in phase 1 or 2; in phase 3 it simply proposes the value $\perp$.

Each test ran 10 times and the presented values are the average of the respective observations. The configuration parameters for the protocols on this assessment were the presented on table 7.14.

Refinement	
MVC Sleep Time Between Receptions	50
MVC Sleep Time Between Checks to Turquois	500
MVC Sleep Time Between Sends	125
Turquois Sleep Time Between Receptions	50
Turquois Sleep Time Between Sends	125
Application Sleep Time Between Checks to MVC	500

Table 7.14: Configuration refinements for the new assessment (all times in milliseconds)

The proposal distribution defines the initial values to be proposed by each process, and has the following two variants:

- **Unanimous —** All the processes propose the same initial value;

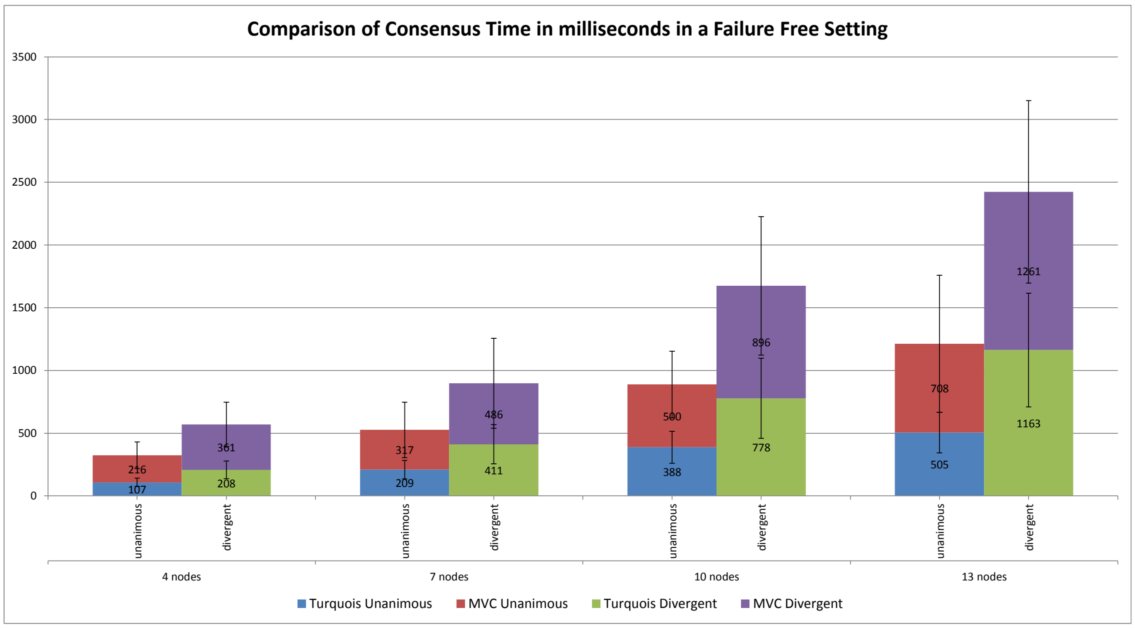


Figure 7.12: Comparison of Consensus Time in milliseconds in a Failure Free Setting

- **Divergent** — Processes with an odd ID propose one value, while the others propose another.

7.5.2 Latency in a Failure-Free Setting

In this section we present the results of the assessment when there are no failures, on figure 7.12 and using a UDP / Multicast IP philosophy for communications.

In this test, all the nodes whose correct decision was mandatory (accordingly with 7.5.1.2) reached a consensus.

It is impossible to compare the results of this test with the tests from the previous tests (section 7.4) as the settings are completely different (in section 7.4 we assumed always Byzantine attacks).

In our measurements the network jitter[15] was averagely 12% for MVC and 4% for Turquois and not much variable, what means that the network is relatively stable.

7.5.3 Latency in a Fail-Stop Setting

In this section we present the observed performance when $f = \frac{n-1}{3}$ nodes crash before the execution of the protocol begins. In such situation, since f nodes crash exactly $n-f = \frac{n+f}{2} + 1$ nodes are left in the system. Results are presented on figure 7.13 and the used communications were based on UDP / Multicast IP.

In this test, all the nodes whose correct decision was mandatory (accordingly with 7.5.1.2) reached a consensus.

[15]On computer networks, jitter is a statistic variation of the data delivery delay. It can be measured as the variation delay between successive data packets.

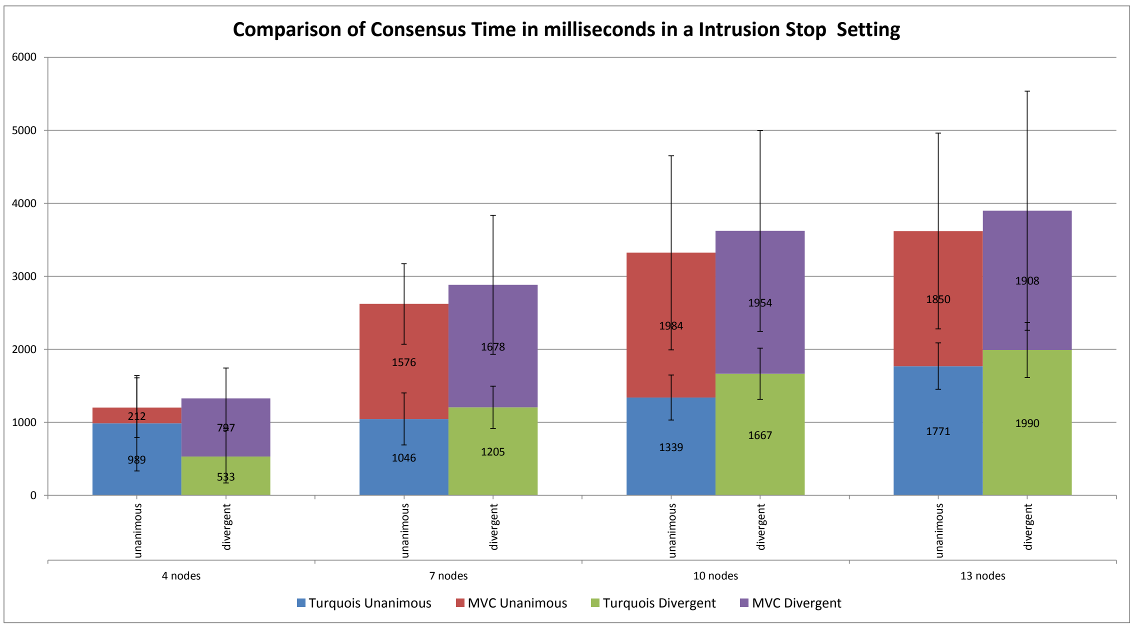


Figure 7.13: Comparison of Consensus Time in milliseconds in a Fail-Stop Setting

Once again, these results are not comparable with the results obtained in section 7.4 as the settings are not the same.

In the case of the Fail-Stop Setting and contrarily to the Failure Free Setting, the network jitter is considerable. It was in average of 19% for the MVC protocol and 48% for the Turquois protocol and a bit variable. This means that there is some instability in the network, mainly during the execution of Turquois.

7.5.4 Latency in a Byzantine Setting

Here we present the observed results when $f = \frac{n-1}{3}$ nodes act accordingly with a malicious strategy, on figure 7.14. The communications' pattern was based on UDP / Multicast IP.

In this test, all the nodes whose correct decision was mandatory (accordingly with 7.5.1.2) reached a consensus. The communications' pattern was, once again, based on UDP / Multicast IP.

In figure 7.15 a simplified version of the best obtained results allows us to compare the results from section 7.4 (Test A) and the results from the current setting (Test B). As we can see, the results obtained after the improvements (improvements in Raspberry PI nodes and new testbed) are always better than those obtained on Test A. Furthermore we can see that as the number of nodes increase, settings from Test A have a tendency to increase consensus times more rapidly than the settings from Test B.

Finally, in the Byzantine Setting the network jitter is significant. In fact, the jitter reached values of 40% for the MVC protocol and 31% for the Turquois protocol. Such instability can be explained by the number of collisions in network that with this test

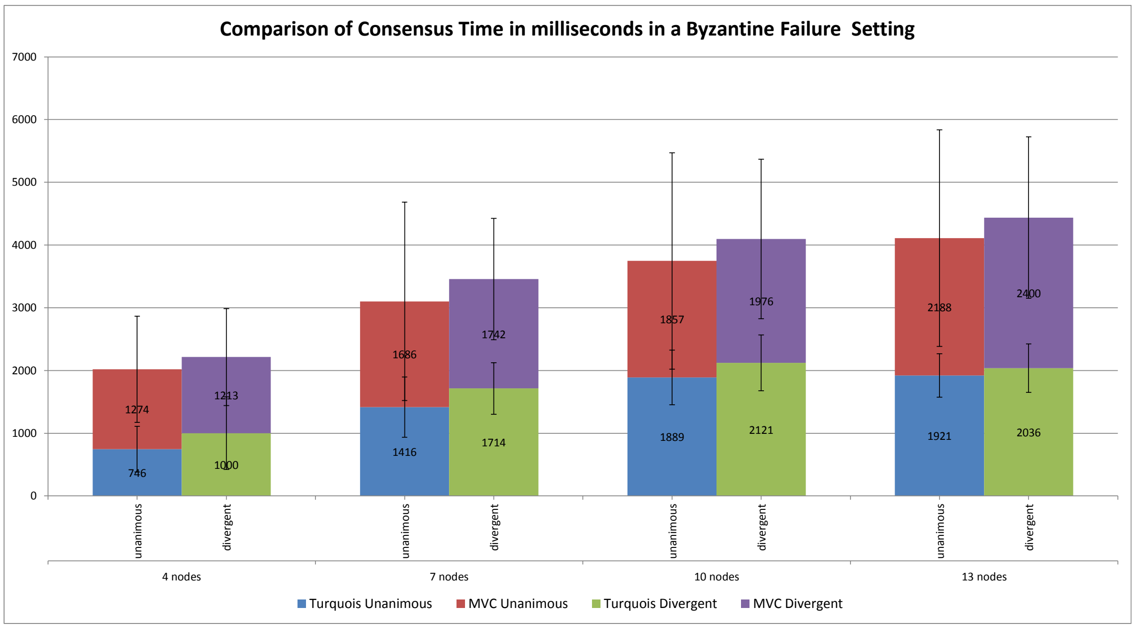


Figure 7.14: Comparison of Consensus Time in milliseconds in a Byzantine Failure Setting

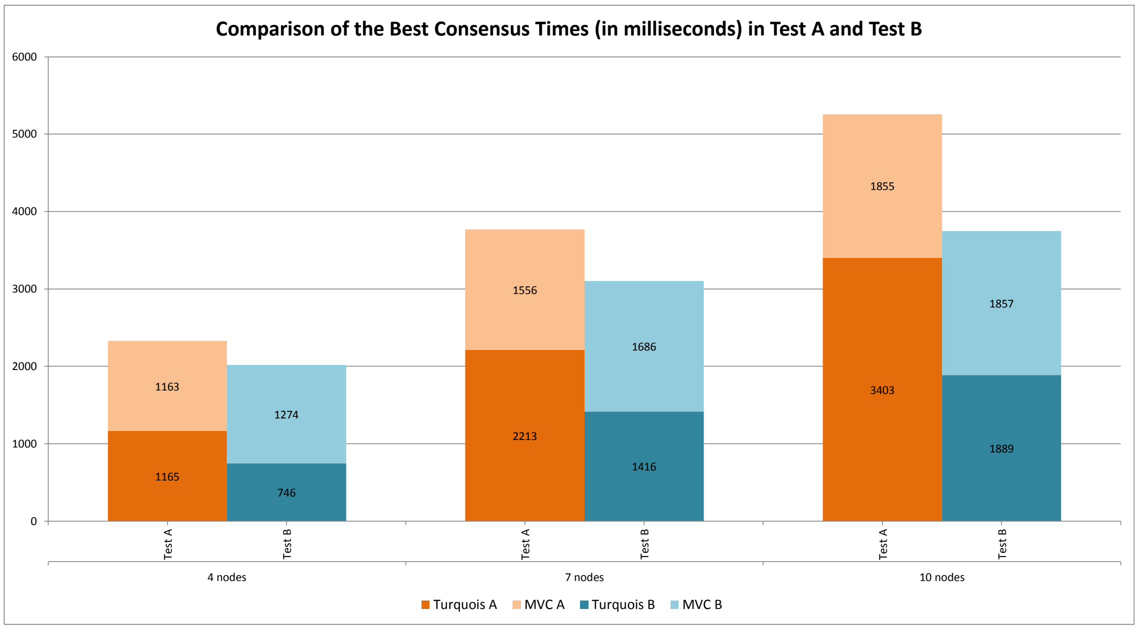


Figure 7.15: Comparison of the Best Consensus Times (in milliseconds) in Test A and Test B

start to be significant.

7.5.5 Critical Analysis

In this section, we tried to improve the results obtained previously on section 7.4.

The research for improvements was made mainly in two directions:

1. Possible optimisations to the Raspberry PI nodes' performance;
2. Possible improvements in terms of the assessment philosophy, trying to be more optimistic about the consensus.

Concerning the optimisations on the Raspberry PI nodes' performance, we started by the study and comprehension of the hardware's architecture. After our suspicions concerning the USB bus and controller, we did a research on the Internet. Such research allowed us to understand that many problems we were suspecting on were already reported and somehow documented.

We installed a new kernel version and a patch for the USB controller issues. Then we made some tests with two objectives in mind:

- Investigate if the Raspberry PI nodes' performance increased with such fixes;
- Get to know better the behaviour of the network of such devices when the CPU was stressed by increasing its load.

Both tests were successful and helped our comprehension in two different directions. With the first we verified that in fact the patches and tips we found with our research were useful; thus all the subsequent tests were performed with these patches applied on the Raspberry PI nodes. With the second test, we confirmed that when the CPU is subjected to an increase on its load the ratio of lost packets is surprisingly high. The hardware architecture (with the networking capabilities multiplexed to the single USB bus) is the main reason for such behaviour, exacerbated with some bugs on the USB drivers and weakness on the energy supply.

Concerning the possible improvements on the assessments' philosophy, we decided to assess the *k-consensus* problem, where we only require k out of n processes to decide on the consensus value, where $k = n - f$ and $f = \frac{n-1}{3}$. This allowed us to be more optimistic and to reach better results. Following the philosophy of [87] in order to allow the comparison of results with the original implementation of Turquois, we decided to have three different settings: one with no failures, other with fail-stop flaws and another with Byzantine flaws.

Having the original implementation of Turquois in mind ([87]), we can compare such results with our measurements. In all the settings, the obtained results were always worse than those from the original implementation; the latency values on the original implementation were in average 12% of the values we assessed (between 4% in the minimum and 35% in the maximum). However, if we consider each results set in its order of magnitude and compare how they evolve, we can verify that both the original and our measurements evolve proportionally. This means that the algorithms' behaviour is correct and evolving accordingly with the expected. Some factors that may contribute for the better results in the original implementation are:

- **Java implementation vs. C implementation —** The fact that we used Java has advantages and disadvantages (as already analysed in section 4.3). We were aware for the fact that one of the biggest disadvantages of Java is its poor performance when compared with lower-level languages like C. This fact leaded our implementation to a worse performance than the original one, written in C;

- **Testbed —** The assessment to the original implementation of Turquois used more powerful devices (namely Pentium III computers with 600 MHz of clock speed and 256MB of RAM) then our Base Stations (Raspberry PI computers). The computation power also contributed for the performance differences;

- **Simulation Environment —** In the original assessment the experiments were carried out on the Emulab simulation environment. Although this simulator attempts to simulate the issues of real wireless networks, it is not possible to completely and reliably simulate the effect of real flaws (like electromagnetic interferences on the frequency spectrum). Thus, collision conditions will not be as aggressive as possibly in a real network, by creating a more aseptic environment.

As in the original experiments and accordingly with the expected, we can observe that in our results the unanimous proposal distribution has always better results than the divergent.

The assessment of the third setting (Byzantine Failure Setting) allowed us to compare the previous results from the Unanimity Conditions (section 7.4) with the new ones, obtained from the refinements to the Raspberry PI nodes and from the change of assessment philosophy. Such comparison showed that the improvements we made were significant, allowing to reduce the time to reach consensus in about 13% for four nodes, 17% for seven nodes and 28% for ten nodes – what is significant. As we can see, as the number of nodes increases, the improvements achieved by the new tests tend to increase, resulting in bigger improvements.

7.6 MINSENS++ Integration Assessment

In this last section, we present the MINSENS++ Integration Assessment. The objective in this testbench was to observe the interoperation issues as well as evaluate the performance of the integrated MINSENS++ solution. The integration and interoperation of our solution has two main components:

- The multi-hop routing protocol running with multiple disjoint routes, established to multiple Base Stations (MINSENS++ WSN Routing Layer), as initially evaluated itself in the first testbench (section 7.2);
- The multi-value consensus protocol (MINSENS++ MVC), performed by the group of Base Stations in order to achieve final data-consistency guarantees from the packets received from the WSN, as initially assessed itself in the observations of the second testbench (section 7.4).

The main motivations and objectives for this integrated evaluation were focused on searching answers for the following questions:

1. Considering the integration of the two main components of the global intrusion tolerant routing service, does the solution work as expected?
2. Can we evaluate *end-to-end* latency metrics under different conditions, in the conjugation of the two main components of the MINSENS++ integrated solution?
3. In the previous overall evaluation, which is the component that may introduce possible bottlenecks in performance and reliability? Is it the latency mainly introduced by the base routing layer service running at the WSN level? Or in the other hand is the bottleneck due to the MINSENS++ MVC latency limitations?
4. From the obtained results, can we argue our validity conclusions with an improved confidence level, to address a future implementation of the proposed solutions in a fully real test environment with available technology?

In the following assessments we will try to find the answers to these questions.

7.6.1 Integration Setting

In order to answer to the previous questions, we created an hybrid testbed environment, by mixing a simulated large scale Wireless Sensor Network supported by the WiSeNet simulation platform (as used in the first testbench in section 7.2) and external real nodes implemented by Raspberry PI computing nodes. External real nodes are connected to virtual Base Stations, supported by the simulation environment.

The virtual Base Stations of simulated WSNs (created in the WiSeNet environment) are selected as Base Station nodes, using the functionality provided by the WiSeNet management tools. By doing this, each virtual node opens a socket to an external specific process implementing a relay point. Such point sends the received IEEE 802.15.4 data packets as payloads of UDP datagram packets by IEEE 802.11 to a specific Raspberry PI computing node. Each relay process routes the received messages to one and only one Raspberry PI node.

In order to coordinate all the participants, a process running in the machine executing WiSeNet is used as a coordination process. All the Raspberry PI receiving a message announces that event to this process. When the number of messages with the same message ID reach a predefined *k-resiliency* threshold, the coordinating process broadcasts a message to all the Raspberry PI nodes in order that they start the consensus process.

When a Raspberry PI terminates the consensus process, it sends the decision value to the coordinating process. Then the end-to-end latency time is measured (real time in the WiSeNet machine), considering the moment when each message is sent by the respective virtual node representation (in the WiSeNet platform) and the moment when the message is received by the coordinating process (after the respective consensus). We also evaluate the latency observed between the virtual node sender (in the WiSeNet environment) and the coordinating process. This time is equivalent to the time observed by the last virtual Base Station that received the message from some disjoint route link, created by the MINSENS++ routing protocol running in the WiSeNet simulation.

The coordinating process only stores the moment correspondent to the last virtual Base Station that observed the message before the *k-resiliency* threshold is reached. To do this, the coordinating process has a counter for each observed message, incremented each time the same message is received by a virtual Base Station through the multiple established routes. With this setting, we can separate and analyse the impact of latencies imposed by the WiSeNet MINSENS++ routing protocol and the latency imposed by the Minsens++/MVC protocol.

Complementarily to the above testbed, SunSpot [19] sensors were also used in order to evaluate real-time indicators of IEEE 802.15.4 data-link conditions. Such indicators can then be used for manual calibration purposes of the WiSeNet simulation environment, namely: IEEE 802.15.4 frame losses, link latency and jitter conditions. The SunSpot testbed was used as follows:

- We measured the effective throughput and packet losses over IEEE 802.15.4 links, with a tool designed and implemented to have the same functionality as the provided by the ICMP ping tool for IP reachability tests. The use of that tool is quite similar to the use of the conventional ping tool between two nodes interconnected by IEEE 802.11 or IEEE 802.3 links. The used test tool runs on top of the SQUAWK Operating System [96, 97]. Squawk is a Java Micro-Edition Virtual Machine for embedded systems and small devices, and is a conventional available operating system for SunSpots [98]. The SunSpots implement the wireless standard IEEE 802.15.4, using it as the data-link communication layer.

- We used a SunSpot kit [19], composed by two SunSpot sensor devices and a gateway device interconnected via a USB connection to the machine running the WiSeNet platform. The specific hardware of the SunSpot kit is available in [19]. The used kit implemented a local WSN, used only to register some indicators in order to calibrate the WiSeNet platform, namely: latency, packet losses and variable jitter conditions. With this calibration, we balanced the latency conditions observed in the WiSeNet simulation platform between two nodes (interconnected by multi-hop

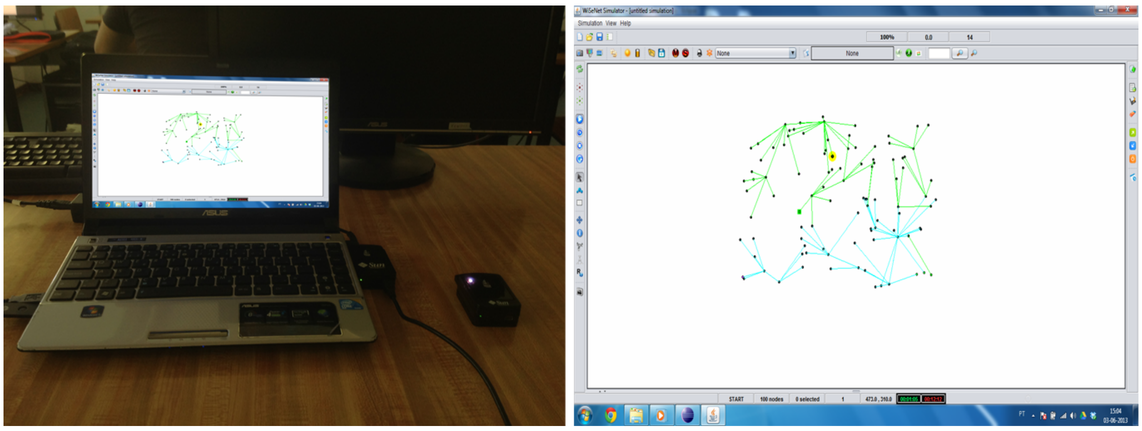


Figure 7.16: Interconnection of external SunSpot sensors (left) with the WiSeNet Simulation Platform (right)

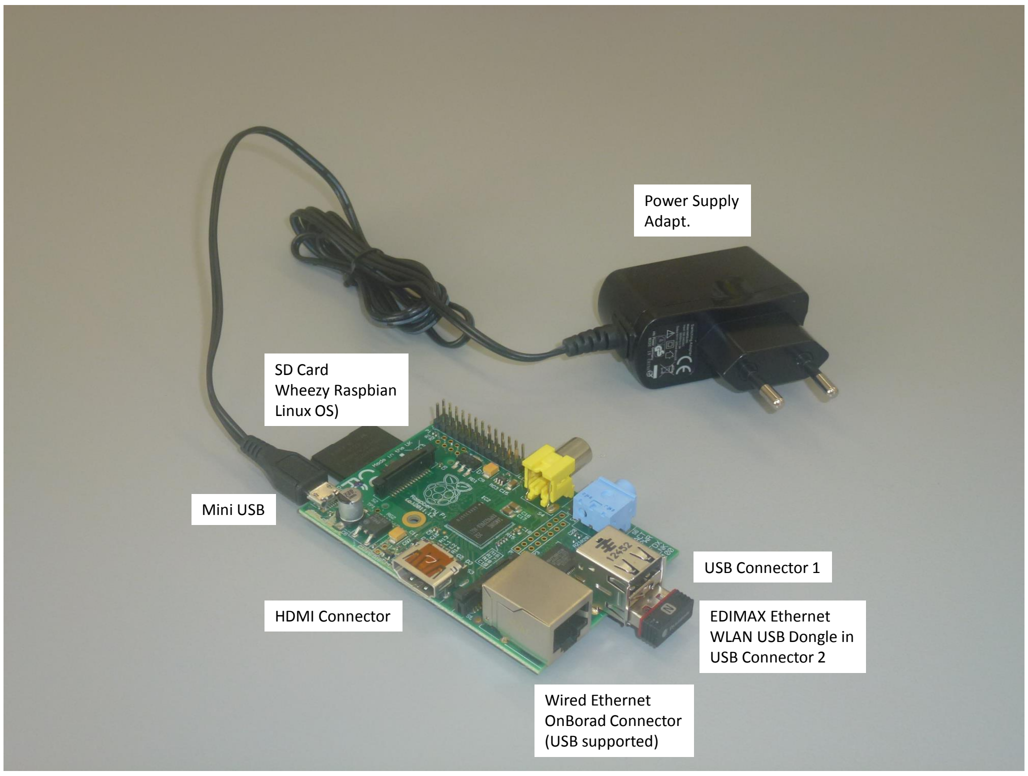
links in a random-based topology), obtaining similar conditions for latency, packet losses and jitter when compared with multiple ping/pong rounds between two real Sun Spots before the injection of data-packets in the WiSeNet simulated WSN.

- We also calibrated *simulation time* and *real-time* conditions as explained before, to evaluate the latency conditions. The calibration of the simulation environment is done by tuning the discrete-event simulation kernel, in order to map real-time conditions as observed with ping/pong messages (sent and received by SunSpot sensors). With this strategy we have a reasonable approach of real-time and simulation-time assumptions in the WiSeNet environment.

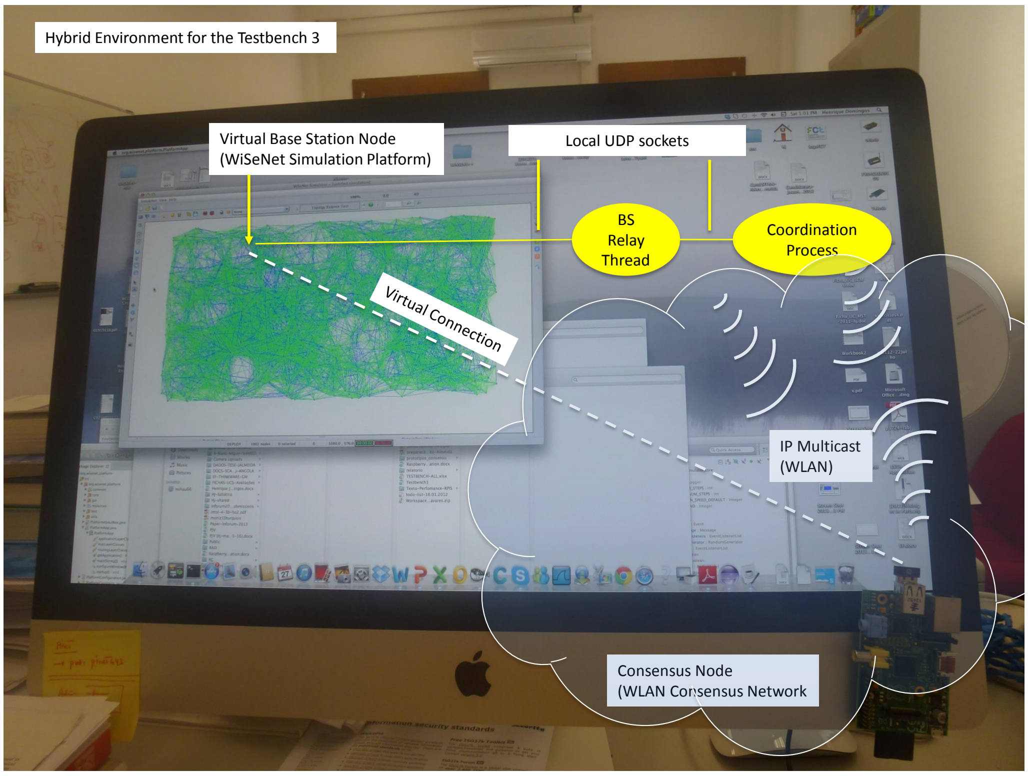
Figure 7.16 shows the interconnection between the external real WSN (implemented by the SunSpot kit) and the simulated WSN. An external real node (on the left) is represented as a virtual node in the visualised graph in the WiSeNet environment (on the right).

Figure 7.17 shows the testbed composed by Raspberry PI Base Stations (figure 7.17(a)), running the Minsens++ MVC protocol, with groups of 4, 7 and 10 nodes in the integrated environment (represented in figure 7.17(b)).

In the following subsection (7.6.2) we present the results of the integration assessment. For such tests, we executed the MINSENS++ protocol in the simulated WSN with the same conditions used in section 7.2. For each observation, a random topology was generated in the WiSeNet environment, selecting the number of base stations for each case.


Power Supply Adapt.
SD Card Wheezy Raspbian Linux OS)
Mini USB
USB Connector 1
HDMI Connector
EDIMAX Ethernet WLAN USB Dongle in USB Connector 2
Wired Ethernet OnBorad Connector (USB supported)

(a) Raspberry PI Base Station


Hybrid Environment for the Testbench 3
Virtual Base Station Node (WiSeNet Simulation Platform)
Local UDP sockets
BS Relay Thread
Coordination Process
Virtual Connection
IP Multicast (WLAN)
Consensus Node (WLAN Consensus Network

(b) Integrated WiSeNet environment

Figure 7.17: Testbench for the consensus component, integrating external real Base Station nodes as virtual nodes in WiSeNet

7.6.2 Integration Results

In our experiments we generated WSN random topologies in the WiSeNet platform for 300, 500 and 1000 nodes.

For each generated WSN, we selected 1% of its nodes as sender nodes and a number of Base Stations as stated in table 7.15.

WSN Size	# of senders	# of Base Stations	# of disjoint routes for each sender to different BS (average)	# of disjoint routes between each sender and each BS (average)
300	3	4	9.2	2.25
500	5	7	23.1	3.29
1000	30	10	35.9	3.60

Table 7.15: Parametrised metrics in the WiSeNet for the randomly generated topologies

For each test topology, we configured a certain number of nodes to have an omission-failure and a Byzantine behaviour during the execution of the MINSENS++ routing protocol. The obtained metrics (following the same criteria as in 7.2) were the results in table 7.16.

WSN Size	Connectivity ratio	Reliability ratio with 10% of Omission Failure Nodes	Reliability ratio with 10% of Byzantine Failure Nodes
300	100%	98%	99%
500	93%	91%	89%
1000	83%	79%	77%

Table 7.16: Obtained connectivity and reliability metrics in the randomly generated topologies

In our experiments each sender sends 10 messages over IEEE 802.15.4, each message having a size of 32 Bytes. The messages can be decomposed in header (28 Bytes) and payload data (4 Bytes). Although the IEEE 802.15.4 standard states that a message can have up to 127 Bytes, we observed that messages with more than 56 Bytes would cause many collisions. Thus we decided to have (besides the header) messages with a payload of 4 Bytes, representing a (32 bits) integer.

Measuring the throughput of IEEE 802.15.4 communications between a pair of SunSpot sensors (with ping tests), a value of 2.2 Kbps was achieved. The WiSeNet simulator was thus tuned to this value in order to obtain reliable results from WiSeNet.

After such calibration, the *end-to-end* latencies for the different network sizes were measured by the coordinating process; the obtained results are represented on table 7.17 and all the proposal distributions in Turquois are always unanimous.

WSN Size	**# of hops per sender (average)**	***k-resiliency* factor**	**MINSENS++ routing latency (milliseconds)**	**MVC consensus latency, Intrusion Free (milliseconds)**	**MVC consensus latency, Intrusion Stop (milliseconds)**	**MVC consensus latency, Byzantine Attack (milliseconds)**	**% of messages with terminated consensus**
300	7.7	3	1405	354	1298	2098	96%
500	10.3	5	1790	666	2586	3134	88%
1000	11.9	7	1968	1441	3526	3852	75%

Table 7.17: Latency times (in milliseconds) measured by the coordinating process

WSN Size	**Attacker Type**	**End-to-end latency (millisen-conds)**	**Effective throughput**	**Time consumed on routing layer (%)**	**Time consumed on consensus layer (%)**
300	Intrusion Free	1759	1.45 Kbps	80%	20%
	Intrusion Stop	2703	0.95 Kbps	52%	48%
	Byzantine Attack	3503	0.73 Kbps	40%	60%
500	Intrusion Free	2456	104 bps	73%	27%
	Intrusion Stop	4376	58.5 bps	41%	59%
	Byzantine Attack	4924	52 bps	36%	64%
1000	Intrusion Free	3409	75 bps	58%	42%
	Intrusion Stop	5494	46.6 bps	36%	64%
	Byzantine Attack	5820	44 bps	34%	66%

Table 7.18: Effective throughput and location of bottlenecks in the network

On table 7.18 we summarise the *end-to-end* latencies observed on the different test settings (failure free, omission stop and Byzantine failures) as well as the bottlenecks for each setting.

With the obtained results in mind, we can now revisit the previous questions that motivated the assessments present in this section.

This experiment allowed us to verify that the integration of the two main components of the intrusion tolerant routing service (routing layer provided by MINSENS++ protocol and consensus layer provided by MVC and Turquois protocols) is possible. The solution works as expected, routing the data generated by the sensors to the Base Stations and then performing a consensus (among Base Stations) over the received values by the multiple Base Stations over the multiple routes.

The *end-to-end* latency metrics were successfully assessed under different settings. The tests comprised different number of nodes in the WSN (300, 500 and 1000 nodes) as well as different types of attacks (Failure Free, Intrusion Stop and Byzantine Attacks).

The observed results showed that the possible bottlenecks in the network are not always present in the same components, considering the different assessment settings. Accordingly with the network's throughput, the bottleneck occurrence depends on the number of Base Stations performing the consensus algorithm. In our tests we observed that around the 7 Base Stations the bottleneck changes from the routing to the consensus component (and vice versa). Thus, as represented on table 7.18 and accordingly with our observations, the location of the bottlenecks are as follows:

- **Bottleneck in the routing layer (MINSENS++) —** The bottleneck occurs on the routing layer when the number of Base Stations is low (below 7 Base Stations, in our testbed). With such a small group of Base Stations, they can rapidly reach consensus decisions over the received values, with a small amount of exchanged messages and in a small number of rounds.

- **Bottleneck in the consensus layer (Base Stations) —** The bottleneck is located in the Base Stations layer (performing the consensus mechanisms) when the number of Base Stations is high (7 or above Base Stations, in our testbed). With such a big group of Base Stations, it is not trivial for them to reach a consensus over the received values. The amount of consensus rounds necessary to decide increase, especially with Byzantine attacks. The significant increase on the exchanged messages (due to the increase on the number of Base Stations and to the number of rounds) causes the Base Stations to take longer to reach the necessary consensus.

After the observations made, we can argue that an implementation of the solution proposed in this dissertation, as a fully real test environment with the currently available technology, would be a very interesting work in the future. Such implementation and assessment would allow to get real assessment metrics, considering all the mechanisms involved in the proposed solution (routing and consensus layers). The integration of these two main components of the intrusion tolerant routing service would be expected to work correctly and accordingly with the observed behaviour in this testbed.

7.7 Critical Analysis

In this chapter many different assessments were made. We measured latencies between sensor nodes in a WSN, consensus latencies in a real deployment of Base Stations, researched possible optimisations and assessed an integrated testbed, comprising both virtual and real sensor nodes and Base Stations.

On section 7.4 we could successfully implement and assess a testbed with real nodes representing Base Stations and performing the consensus mechanisms of our solution. We observed that such consensus mechanisms work as expected, having in mind the throughput / latency limitations of the conditions obtained in such experiment.

On section 7.5 we successfully optimised the previously obtained results. The tuning applied to the Raspberry PI nodes, as well as the flexibility on the consensus conditions proved to have an important and significant impact on the measured latencies.

Finally, the proposed solution including the integration of the two main components of the intrusion tolerant routing service (routing layer and consensus layer) analysed in section 7.6 is viable. We must have however in mind the issue of the tradeoff between the maintenance of the throughput of the WSN and the number of available nodes (Base Stations) with the hardware and software characteristics of Raspberry PI devices. Such tradeoff will be decisive in the location of the bottleneck of the network.

8 Conclusions

In this Chapter, the conclusions of this research work are presented, as well as some research directions for future work.

8.1 Research conclusions

This thesis addresses the design of a dependable routing service for large scale WSNs, based on a multipath tree-based routing protocol (called MINSENS++) integrated with a data consensus protocol. The proposed solution provides an intrusion-tolerant routing service composed by two main components:

- A component to construct a secure and efficient tree-structured routing structure supported by an intrusion tolerant routing algorithm that establishes multiple disjoint routes between sensor nodes in a large scale WSN and multiple Base Stations (or syncnodes);
- A component supporting a consensus protocol for intrusion tolerant agreement of data values received by the multiple Base Stations through the multiple disjoint routes.

The multi-path algorithm (called MINSENS++) is designed to adapt to the asymmetric architecture and resource constraints between typical sensor nodes and Base Stations (or syncnodes) in a WSN, minimising the processing and communication requirements at the WSN level and balancing these requirements by using the processing resources provided by the Base Stations (or syncnodes). The key objective of MINSENS++ is to circumvent (with a preventive intrusion tolerance approach) the possible damage caused by an intruder who has compromised one or more deployed sensor nodes in a specific route. Such an intruder could inject, modify, or block data packets routed through the multi-hop structure of the WSN. The routing service is therefore designed to prevent from these intrusions, limiting the ability of an intruder to cause mischief through a combination of distributed lightweight security mechanisms, where the use of multiple disjoint routes is a key strategy.

The data consensus performed by the Base Stations (or syncnodes) is used with an overlay network approach, being a second intrusion tolerance level over the base WSN (designed as a consensus overlay network). The intrusion tolerance consensus mechanisms used at this level are inspired by probabilistic Byzantine consensus protocols, adapted to the requirements of the designed routing service proposed by the thesis. The solution follows the Turquois [87] approach (section 6.2.1), an intrusion-tolerant binary consensus protocol designed for resource-constrained nodes, in wireless ad hoc networks. The original Turquois protocol allows an efficient utilisation of the IP multicasting medium in a typical TCP/IP stack – as implemented in a typical node belonging to an IEEE 802.11 ad-hoc network. The protocol avoids synchrony assumptions, refraining from public-key cryptography during its normal operation and takes into account the typically constrained resources of typical wireless ad-hoc nodes (such as mobile hand-held computers, PDAs or mobile phones). The protocol also assumes asynchronous communication settings, while aiming for optimal resilience in the case of intrusions and Byzantine attacks or failures.

In the context of this dissertation, the adoption and integration of the Turquois protocol to run as a data-consensus layer of the MINSENS++ routing algorithm required the adaptation of the original protocol in some different directions:

- in adding support to run in different TCP/IP communication settings, namely TCP, UDP/Unicast IP, UDP/Multicast IP;
- in providing a new Java implementation;
- in implementing a stack based in a first layer, by the adaptation of the original Turquois protocol and a complementary multi-value consensus layer to support data consensus related with data encapsulated in IEEE 802.15.4 messages, routed to the Base Stations by the MINSENS++ protocol.

The final solution was designed and implemented in real Raspberry PI computing nodes, communicating in a IEEE 802.11 WLAN. This was the selected technology to materialise the Base Stations or Syncodes, in the context of the proposed intrusion tolerant routing service.

The adapted consensus protocol is safe, despite the arbitrary failure of $f < \frac{n}{3}$ processes running in from a total of n processes supported in n Raspberry PI nodes. It is also safe despite unrestricted message omissions. The implemented solution assumes communication to be inherently unreliable by incorporating a communication failure model presented in this thesis [74] (section 3.3).

The integrated consensus mechanisms support the required safety and liveness properties in the objectives of the dissertation. Safety is maintained despite unrestricted message omissions and under asynchronous communication settings. Liveness is ensured in different rounds where the number of omissions is limited to $(n - k - t) = \frac{n-t}{2} + k - 2$, where k is the number of correct processes required to decide a correct message, and $t \leq f$ is the number of processes that are actually faulty by possible intrusions.

In the designed and implemented solution of the data-consensus mechanisms integrated with the MINSENS++ routing protocol, timing assumptions are weak. It is only required a local timeout on each process running in each Raspberry PI Base Station to ensure these keep sending messages, as well as some other parameters to regulate and optimise the performance of the internetworking support in each Raspberry PI node. These parameters also minimise the effect of possible collisions during the communication rounds required to achieve the consensus termination.

The multi valued consensus protocol supported in the IEEE 802.11 overlay network (over the base IEEE 802.15.4 WSN running the MINSENS++ routing protocol) ensures the following properties:

- **Validity —** If all correct processes running in the base stations propose the same value v, then every correct process that decides a value, decides v;
- **Agreement —** No two correct processes running in two Base Stations decide differently;
- **Termination —** At least k correct processes running in k Base Stations eventually decide, with probability 1.

The key to the performance optimisation of the protocol and to promote termination conditions was the decision to assume unreliable communications. This allowed the protocol to take full advantage of IP multicasting support, even considering the limitations and drawbacks observed in the TCP/IP stack performance of the Raspberry PI nodes. By using IP multicast the cost of transmitting a message to multiple nodes can be just the same as sending it to a single node, since the necessary regulation to minimise collisions is prevented by a set of configuration parameters (provided in the implementation of the consensus protocols).

When a lot of Base Stations exist in the network, or when they are sending a lot of messages rapidly (which is the case when the multi-valued consensus protocol is running over the binary consensus protocol), the adoption of UDP transport using IP Multicast proved to be the best solution as it reduces the amount of messages over the air (and thus the collisions rate). On the other hand, when the number of Base Stations is reduced or when the Base Stations are sending the messages slowly, UDP Protocol using IP Unicast used as the communication backend proved to be the one with best results.

As an important part of the proposed solution, the MINSENS++ algorithm operates correctly in the presence of (undetected) intruders in the base WSN, promoting a preventive intrusion tolerance strategy that minimises computation and communication requirements at the level of IEEE 802.15.4 WSNs. To address these resource constraints, computation on the sensor nodes is offloaded to the more resource-rich Base Stations, even implemented with no expensive and limited hardware/software (such as the case of the used Raspberry PI devices). Following this strategy, Base Stations compute and establish routing tables to set up multiple disjoint routes, while only low-complexity security methods are required at the WSN node level, forming a first baseline of secure communication in the WSN level (for example, symmetric key cryptography for message-confidentiality, one-way hash functions and hashed message authentication codes for integrity checks). By using multiple routes established as disjoint routes over multiple Base Stations, the scope of the possible damage inflicted by (undetected) intruders is further limited, by restricting flooding to the Base Station and by having its packets ordered using one-way sequence numbers. Later, possible intrusion attacks will be discarded by the consensus protocol performed by the group of Base Stations.

An important property of the MINSENS++ component is that while a malicious node may be able to compromise a small number of nodes in its vicinity, it cannot cause widespread damages in the network. Performance measured from a prototype implementation using the WiSeNet simulation tool showed that INSENS tolerates malicious attacks launched by intruder nodes, performing correctly over a variety of simulated random and grid topologies, despite possible intrusions.

The consensus mechanisms combined with the MINSENS++ protocol were validated with an hybrid environment mixing:

- The WSN simulation environment for IEEE 802.15.4 WSN nodes, interconnected with Base Stations (materialised by the WiSeNet simulation platform);

- Real Base Station nodes, materialised by a group of Raspberry PI nodes running the consensus protocols supported in a WLAN IEEE 802.11 environment. The integration of the multi-valued consensus layer with the simulated MINSENS++ protocol shows an interesting potential to help in the development of a new approach and novel direction to propose innovative Intrusion Tolerant Routing Solutions for Scalable WSNs.

The implementation prototypes, testbench installations and assessment results obtained in the validation of the ideas developed in this dissertation, show that the proposed solutions are valid and the interesting achieved results consolidate the potential of innovation that may be explored in future wok directions.

8.2 Future Work

The obtained results were very interesting and innovative. However, many directions for future work are still opened issues. Therefore and accordingly with the conclusions of this research, many interesting directions for future work exist, namely:

- Revise the implementation of the integrated solution in order to support more extensive tests for latency conditions, energy consumption, and resilience metrics under simultaneous and independent intrusion attacks against the Base Stations running the Consensus Layer and the WSN nodes executing the MINSENS++ protocol in the WiSeNet simulation platform;
- Protocols optimisation:
  - Implement and assess the optimised version of Turquois protocol [87];
  - Optimisation at the level of Java Serialization of Messages, in the current implementation of the Binary Consensus and Multi-Valued Consensus Layers;
  - Optimisation by using a dynamic monitoring component, providing autonomous dynamic configurations and adjustments of the Consensus Protocol parameters, in order to dynamically obtain in real time the best settings for the Binary and Multi-Valued Consensus Protocols, optimising latency, minimising collisions and thus ensuring the termination conditions.
- Tuning of the Raspberry PI computers in order to achieve better performance, namely at a low level in the devices (Operating System and drivers);
- Implementation and assessment of the Consensus Layer using other devices as Base Stations, to investigate possible replacement alternatives for Raspberry PI nodes, overcoming the performance problems at the level of LAN and WLAN support and the TCP/IP stack observed bottlenecks for IP Unicast and IP Multicast communications;
- Implementation of the MINSENS++ protocol in real IEEE 802.15.4 sensor nodes, comparing the performance to calibrate the results obtained by simulation;
- Analysis and study of hybridisation facilities to interconnect real WSN nodes in a simulated large scale environment, allowing a dynamic calibration of the simulation environment with the results observed in real nodes;
- Implementation of the proposed solution as a fully real test environment with real WSN nodes and real Base Stations, comprising the integration of the two main components of the intrusion tolerant routing service (routing layer provided by MINSENS++ protocol and consensus layer provided by MVC and Turquois protocols).

Bibliography

[1] M. Kuorilehto, M. Hännikäinen, and T. D. Hämäläinen, "A survey of application distribution in wireless sensor networks," *EURASIP J. Wirel. Commun. Netw.*, vol. 2005, pp. 774–788, Oct. 2005.

[2] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh, "Vigilnet: An integrated sensor network system for energy-efficient surveillance," *ACM Trans. Sen. Netw.*, vol. 2, pp. 1–38, Feb. 2006.

[3] V. C. Gungor and G. P. Hancke, "Industrial Wireless Sensor Networks: Challenges, Design Principles, and Technical Approaches," *IEEE Transactions on Industrial Electronics*, vol. 56, pp. 4258–4265, Oct. 2009.

[4] A. Milenković, C. Otto, and E. Jovanov, "Wireless sensor networks for personal health monitoring: Issues and an implementation," *Comput. Commun.*, vol. 29, pp. 2521–2533, Aug. 2006.

[5] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, (New York, NY, USA), pp. 88–97, ACM, 2002.

[6] J. Burrell, T. Brooke, and R. Beckwith, "Vineyard computing: Sensor networks in agricultural production," *IEEE Pervasive Computing*, vol. 3, pp. 38–45, Jan. 2004.

[7] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing*, vol. 10, pp. 18–25, March 2006.

[8] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health monitoring of civil infrastructures using wireless sensor networks," in *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, (New York, NY, USA), pp. 254–263, ACM, 2007.

[9] L. Evers, M. J. J. Bijl, M. Marin-perianu, R. Marin-perianu, and P. J. M. Havinga, "Wireless sensor networks and beyond: A case study on transport and logistics," in *In International Workshop on Wireless Ad-Hoc Networks (IWWAN 2005*, pp. 1381–3625, 2005.

[10] "Sitan - services for intrusion tolerant ad-hoc networks." http://asc.di.fct.unl.pt/SITAN/.

[11] "Ieee 802.15.4." http://www.ieee802.org/15/pub/TG4.html. (Accessed: 09/July/2012).

[12] P. Baronti, P. Pillai, V. W. C. Chook, S. Chessa, A. Gotta, and Y. F. Hu, "Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards," *Comput. Commun.*, vol. 30, pp. 1655–1695, May 2007.

[13] P. Silva, "Wisenet - wireless sensor networks simulator." http://code.google.com/p/wisenet/.

[14] "Raspberry pi devices." http://www.raspberrypi.org/. (Accessed: 10/July/2013).

[15] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures," in *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pp. 113–127, 2003.

[16] J. P. Walters, Z. Liang, W. Shi, and V. Chaudhary, "Wireless sensor network security: A survey," in book chapter of security," in *in Distributed, Grid, and Pervasive Computing, Yang Xiao (Eds*, pp. 0–849, CRC Press, 2007.

[17] D. Schmidt, M. Berning, and N. Wehn, "Error correction in single-hop wireless sensor networks: a case study," in *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pp. 1296–1301, 2009.

[18] "Mica motes device types." http://www.memsic.com/products/wireless-sensor-networks/wireless-modules.html. (Accessed: 09/July/2012).

[19] "Sunspot." http://www.sunspotworld.com/. (Accessed: 20/July/2013).

[20] "Tinyos." http://www.tinyos.net/. (Accessed: 09/July/2012).

[21] "Contikios." http://www.contiki-os.org/. (Accessed: 09/July/2012).

[22] "Zigbee suite." https://docs.zigbee.org/zigbee-docs/dcn/09-5231.PDF. (Accessed: 09/July/2012).

[23] Z. Alliance, "Zigbee specification. technical report document 053474r06, version 1.0," tech. rep., ZigBee Alliance, June 2005.

[24] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: a link layer security architecture for wireless sensor networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys '04, (New York, NY, USA), pp. 162–175, ACM, 2004.

[25] M. Luk, G. Mezzour, A. Perrig, and V. Gligor, "Minisec: a secure sensor network communication architecture," in *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, (New York, NY, USA), pp. 479–488, ACM, 2007.

[26] J. Borges, "Distribuição e estabelecimento seguro de chaves criptográficas para redes de sensores sem fios," 2008.

[27] B. Parno, M. Luk, E. Gaustad, and A. Perrig, "Secure sensor network routing: a clean-slate approach," in *Proceedings of the 2006 ACM CoNEXT conference*, CoNEXT '06, (New York, NY, USA), pp. 11:1–11:13, ACM, 2006.

[28] J. Deng, R. Han, and S. Mishra, "INSENS: Intrusion-tolerant routing for wireless sensor networks," *Computer Communications*, vol. 29, pp. 216–230, Jan. 2006.

[29] A. Guerreiro, "Intrusion tolerant routing protocols for wireless sensor networks," Master's thesis, Departamento de Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, Sept. 2011.

[30] W. Lou and Y. Kwon, "H-spread: A hybrid multipath scheme for secure and reliable data collection in wireless sensor networks," *IEEE Transactions on Vehicular Technology*, vol. 55, pp. 1320–1330, July 2006.

[31] S.-B. Lee and Y.-H. Choi, "A secure alternate path routing in sensor networks," *Comput. Commun.*, vol. 30, pp. 153–165, December 2006.

[32] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," in *Proceedings of the conference on Communications architectures, protocols and applications*, SIGCOMM '94, pp. 234–244, 1994.

[33] T. Clausen and P. Jacquet, "Optimized link state routing protocol (olsr)," rfc, RFC Editor, 2003.

[34] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *IN PROCEEDINGS OF THE 2ND IEEE WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS*, pp. 90–100, 1997.

[35] D. B. Johnson, D. A. Maltz, and J. Broch, "Dsr: The dynamic source routing protocol for multi-hop wireless ad hoc networks," in *In Ad Hoc Networking, edited by Charles E. Perkins, Chapter 5*, pp. 139–172, 2001.

[36] A. D. Wood, L. Fang, J. A. Stankovic, and T. He, "Sigf: a family of configurable, secure routing protocols for wireless sensor networks," in *Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*, SASN '06, pp. 35–48, 2006.

[37] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8 - Volume 8*, HICSS '00, pp. 8020–, 2000.

[38] S. Lindsey and C. S. Raghavendra, "PEGASIS: Power-efficient gathering in sensor information systems," in *Aerospace Conference Proceedings, 2002. IEEE*, vol. 3, pp. 3–1125–3–1130 vol.3, 2002.

[39] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures," in *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pp. 113–127, 2003.

[40] D. Dolev and A. C. Yao, "On the security of public key protocols," tech. rep., Stanford, CA, USA, 1981.

[41] I. Telegraph and T. C. Committee, *CCITT Recommendation X.800: Data Communication Networks: Open Systems Interconnection (OSI); Security, Structure and Applications : Security Architecture for Open Systems Interconnection for CCITT Applications*. International Telecommunication Union, 1991.

[42] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures," in *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pp. 113–127, 2003.

[43] H. Chan and A. Perrig, "Security and privacy in sensor networks," *Computer*, vol. 36, no. 10, pp. 103–105, 2003.

[44] V. B. Misic, J. Fung, and J. V. Misic, "Mac layer security of 802.15.4-compliant networks.," in *MASS*, IEEE, 2005.

[45] Y.-C. Wang and Y.-C. Tseng, "Attacks and defenses of routing mechanisms in ad hoc and sensor networks," *Security in Sensor Networks*, pp. 4–23, 2006.

[46] A. Mishra, K. Nadkarni, and A. Patcha, "Intrusion detection in wireless ad hoc networks," *Wireless Communications, IEEE*, vol. 11, no. 1, pp. 48–60, 2004.

[47] Y.-C. Hu, A. Perrig, and D. Johnson, "Packet leashes: a defense against wormhole attacks in wireless networks," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 3, pp. 1976–1986 vol.3, 2003.

[48] J. Newsome, E. Shi, D. Song, and A. Perrig, "The sybil attack in sensor networks: analysis & defenses," in *Proceedings of the 3rd international symposium on Information processing in sensor networks*, pp. 259–268, ACM, 2004.

[49] J. Chen, P. Druschel, and D. Subramanian, "An efficient multipath forwarding method," in *INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, pp. 1418–1425, IEEE, 1998.

[50] K. Ishida, Y. Kakuda, and T. Kikuno, "A routing protocol for finding two node-disjoint paths in computer networks," in *Network Protocols, 1995. Proceedings., 1995 International Conference on*, pp. 340–347, IEEE, 1995.

[51] H. Deng, W. Li, and D. P. Agrawal, "Routing security in wireless ad hoc networks," *Communications Magazine, IEEE*, vol. 40, no. 10, pp. 70–75, 2002.

[52] S. Sancak, E. Cayirci, V. Coskun, and A. Levi, "Sensor wars: detecting and defending against spam attacks in wireless sensor networks," in *Communications, 2004 IEEE International Conference on*, vol. 6, pp. 3668–3672, IEEE, 2004.

[53] B. Culpepper and H. Tseng, "Sinkhole intrusion indicators in dsr manets," in *Broadband Networks, 2004. BroadNets 2004. Proceedings. First International Conference on*, pp. 681–688, 2004.

[54] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating routing misbehavior in mobile ad hoc networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, MobiCom '00, (New York, NY, USA), pp. 255–265, ACM, 2000.

[55] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy, "Highly secure and efficient routing," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 1, IEEE, 2004.

[56] E. Egea-Lopez, J. Vales-Alonso, A. Martinez-Sala, P. Pavon-Marino, and J. García-Haro, "Simulation tools for wireless sensor networks," in *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'05)*, 2005.

[57] G. Coulson, B. Porter, I. Chatzigiannakis, C. Koninis, S. Fischer, D. Pfisterer, D. Bimschas, T. Braun, P. Hurni, M. Anwander, G. Wagenknecht, S. P. Fekete, A. Kröller, and T. Baumgartner, "Flexible experimentation in wireless sensor networks," *Commun. ACM*, vol. 55, pp. 82–90, Jan. 2012.

[58] T. Maret, R. Kummer, P. Kropf, and J.-F. Wagen, "Freemote emulator: a lightweight and visual java emulator for wsn," in *Proceedings of the 6th international conference on Wired/wireless internet communications*, WWIC'08, (Berlin, Heidelberg), pp. 92–103, Springer-Verlag, 2008.

[59] P. Hurni and T. Braun, "Calibrating wireless sensor network simulation models with real-world experiments," in *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, NETWORKING '09, (Berlin, Heidelberg), pp. 1–13, Springer-Verlag, 2009.

[60] Z. Y. Song, M. Mostafizur, R. Mozumdar, M. Tranchero, L. Lavagno, R. Tomasi, and S. Olivieri, "Hy-sim: model based hybrid simulation framework for wsn application development," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, (ICST, Brussels, Belgium, Belgium), pp. 87:1–87:8, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010.

[61] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, (New York, NY, USA), pp. 126–137, ACM, 2003.

[62] V. Shnayder, M. Hempstead, B. rong Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *In Sensys*, pp. 188–200, ACM Press, 2004.

[63] B. L. Titzer, D. K. Lee, and J. Palsberg, "Avrora: scalable sensor network simulation with precise timing," in *Proceedings of the 4th international symposium on Information processing in sensor networks*, IPSN '05, (Piscataway, NJ, USA), IEEE Press, 2005.

[64] R. de Paz Alberola and D. Pesch, "Avroraz: extending avrora with an ieee 802.15.4 compliant radio chip model," in *Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks*, PM2HW2N '08, (New York, NY, USA), pp. 43–50, ACM, 2008.

[65] H. Wu, Q. Luo, P. Zheng, and L. M. Ni, "Vmnet: Realistic emulation of wireless sensor networks," tech. rep., 2005.

[66] Nsnam, "Ns-3." `http://www.nsnam.org/`. (Accessed: 09/July/2012).

[67] A. Kröller, D. Pfisterer, C. Buschmann, S. P. Fekete, and S. Fischer, "Shawn: A new approach to simulating wireless sensor networks," *CoRR*, vol. abs/cs/0502003, 2005.

[68] G. Chen, J. Branch, M. J. Pflug, L. Zhu, and K. Szymanski, "Chapter 1 sense : A sensor network simulator," *Components*, pp. 249–267, 2004.

[69] A. Ledeczi, "Jprowler." `http://www.escherinstitute.org/Plone/frameworks/nes/tools/prowler`.

[70] J. Tavares, "Encaminhamento e disseminação de dados tolerantes a instrusões para redes de sensores sem fios," Master's thesis, Departamento de Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2012.

[71] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. ACM*, vol. 27, pp. 228–234, April 1980.

[72] D. Davies and J. F. Wakerly, "Synchronization and matching in redundant systems," *IEEE Trans. Comput.*, vol. 27, pp. 531–539, June 1978.

[73] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, pp. 374–382, April 1985.

[74] N. Santoro and P. Widmayer, "Time is not a healer," in *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science*, (London, UK), pp. 304–313, Springer-Verlag, 1989.

[75] P. Feldman and S. Micali, "Optimal algorithms for byzantine agreement," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC '88, (New York, NY, USA), pp. 148–161, ACM, 1988.

[76] D. Dolev, C. Dwork, and L. Stockmeyer, "On the minimal synchronism needed for distributed consensus," *Journal of the ACM*, vol. 34, pp. 77–97, 1987.

[77] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, pp. 288–323, 1988.

[78] H. Attiya, C. Dwork, N. Lynch, and L. Stockmeyer, "Bounds on the time to reach agreement in the presence of timing uncertainty," *J. ACM*, vol. 41, pp. 122–152, January 1994.

[79] T. D. Chandra, V. Hadzilacos, and S. Toueg, "The weakest failure detector for solving consensus," *J. ACM*, vol. 43, pp. 685–722, July 1996.

[80] T. D. Chandra and S. Toueg, "Unreliable failure detectors for reliable distributed systems," *J. ACM*, vol. 43, pp. 225–267, March 1996.

[81] M. Blum, "Coin flipping by telephone a protocol for solving impossible problems," *SIGACT News*, vol. 15, pp. 23–27, Jan. 1983.

[82] M. Ben-Or, "Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols," in *Proceedings of the second annual ACM symposium on Principles of distributed computing*, PODC '83, pp. 27–30, 1983.

[83] G. Bracha, "An asynchronous [(n - 1)/3]-resilient consensus protocol," in *Proceedings of the third annual ACM symposium on Principles of distributed computing*, PODC '84, pp. 154–162, 1984.

[84] R. Canetti and T. Rabin, "Fast asynchronous byzantine agreement with optimal resilience," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pp. 42–51, 1993.

[85] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: practical asynchronous byzantine agreement using cryptography (extended abstract)," in *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, PODC '00, pp. 123–132, 2000.

[86] H. Moniz, N. F. Neves, M. Correia, and P. Verissimo, "Experimental comparison of local and shared coin randomized consensus protocols," in *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems*, SRDS '06, (Washington, DC, USA), pp. 235–244, IEEE Computer Society, 2006.

[87] H. Moniz, *Byzantine fault-tolerant agreement protocols for wireless Ad hoc networks*. PhD thesis, Faculdade de Ciências da Universidade de Lisboa, 2010.

[88] H. Moniz, N. F. Neves, M. Correia, and P. Verissimo, "Ritas: Services for randomized intrusion tolerance," *IEEE Trans. Dependable Secur. Comput.*, vol. 8, pp. 122–136, January 2011.

[89] H. Moniz, N. F. Neves, and M. Correia, "Turquois: Byzantine consensus in wireless ad hoc networks.," in *DSN*, pp. 537–546, IEEE, 2010.

[90] "Element 14 ltd." `http://www.element14.com`. (Accessed: 19/July/2013).

[91] "Premier farnell." `http://www.premierfarnell.com`. (Accessed: 19/July/2013).

[92] "Rs components." `http://uk-rs.online.com`. (Accessed: 19/July/2013).

[93] "Egoman." `http://www.egoman.com.cn`. (Accessed: 19/July/2013).

[94] "Raspberry pi wikipedia page." `http://en.wikipedia.org/wiki/Raspberry_Pi`. (Accessed: 13/July/2013).

[95] "Arm architecture." `https://en.wikipedia.org/wiki/ARM_architecture`. (Accessed: 19/July/2013).

[96] D. Simon and C. Cifuentes, "The squawk virtual machine: Java&#8482; on the bare metal," in *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, OOPSLA '05, (New York, NY, USA), pp. 150–151, ACM, 2005.

[97] N. Shaylor, D. N. Simon, and W. R. Bush, "A java virtual machine architecture for very small devices," in *Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*, LCTES '03, (New York, NY, USA), pp. 34–41, ACM, 2003.

[98] "The squawk development wiki." `https://java.net/projects/squawk/pages/SquawkDevelopment`. (Accessed: 20/July/2013).