



Universidade Federal de Pernambuco
Centro de Informática
Pós-Graduação em Ciência da Computação

“3D Routing with Context Awareness”

Breno Jacinto Duarte da Costa

Advisor: Dr. Djamel Sadok

Co-advisor: Dr. Eduardo Souto

Recife, march 2009

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

BRENO JACINTO DUARTE DA COSTA

“3D routing with context awareness”

THIS DISSERTATION HAS BEEN SUBMITTED TO THE
COMPUTER SCIENCE CENTER OF THE FEDERAL
UNIVERSITY OF PERNAMBUCO AS A PARTIAL
REQUIREMENT TO OBTAIN THE DEGREE OF MASTER
IN COMPUTER SCIENCE.

ADVISOR: DR. DJAMEL SADOK
CO- ADVISOR: DR. EDUARDO SOUTO

RECIFE, MARCH/2009

Costa, Breno Jacinto Duarte da
3D Routing with context awareness / Breno
Jacinto Duarte da Costa - Recife : O Autor, 2009.
ii, 98 folhas : il., fig.

Dissertação (mestrado) – Universidade Federal de
Pernambuco. Cln. Ciência da Computação, 2009.

Inclui bibliografia e apêndice.


1. Ciência da computação. 2. Redes de
computadores. 3. Rede de AD-HOC. I. Título.

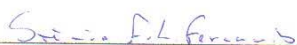
004

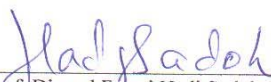
CDD (22. ed.)

MEI2009- 077


Dissertação de Mestrado apresentada por **Breno Jacinto Duarte da Costa** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**3D Routing: Routing with Proximity and Context Awareness**", orientada pelo **Prof. Djamel Fawzi Hadj Sadok** e aprovada pela Banca Examinadora formada pelos professores:


Prof. Manoel Eusébio de Lima
Centro de Informática / UFPE


Prof. Stênio Flávio de Lacerda Fernandes
Departamento de Ensino CEFET/AL


Prof. Djamel Fawzi Hadj Sadok
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 5 de março de 2009.


Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

To my father and mother for
creating me. To their parents for
creating them. And so on.

Acknowledgments

There's a lot to thank for that words can never be enough. I know that there's a tremendous force behind all this – the existence – and I also know that we are all here for a reason. So, in the deepest level, I'm writing this because of this bigger force, for which I can only feel but not fully understand, but that I'm thankful. Thankful for the chance of being here and learn, improving my spirit.

I'm also thankful for everyone else that exists with me: they are a great source of learning as well. My parents are the first ones I should thank, since they created and raised me with care and responsibility. Especially my mother, she was a heroin for raising me and my brothers after my father was gone. Thank you mom, it's a privilege to be your son.

To my older brothers and sister, who stood with me at all times, always supporting me. To Marcel, Giordano, Fábio, Júnior and Amanda, my lovely brothers. To the rest of my family, my nephews and nieces, cousins, uncles and aunts, thank you.

I have a special thank you for my advisor Professor DJamel Sadok, for taking me as his student, as well as giving the opportunity to work with him and doing my master's at the Informatics Center at UFPE. I have learned so much in such a small frame of time. The kind of learning that I'm sure I'll use for the rest of my life. Huge thanks to my co-advisor, Mestre, for helping me out in many difficult situations.

A thank you to my good colleagues at GPRT: Feitosa, Ramide, Fabrício. To Prof. Judith Kelner, for her effectiveness in problem-solving. To all the routing team at GPRT: Josias, Rafael, Rodrigo, Patrícia and Leonardo. And everybody else from the group: it's very tight group where everyone is ready to help each other.

To the friendships I have established during the master's. That was a great surprise to me. I was so convinced that real friendships are impossible to happen after you're an adult. This two-year stage has proven me so wrong, and I'm so glad for that. My new friends Patrícia Endo (Japa), Ana Cristina, Rodrigo Peixoto (Diguinho, El Salvador), Kalil, Rafael Amorim (Fito), Julian, Ermeson, and "C1" people. Thank you all.

Last, but never least, to my good and old friends at Budokan Dojo. From my Sensei to Ig, Marcelo, Lorella, Dayane, Amaro, Alisson, Anderson Senpai, and everyone else who is constantly feeding the "to the death" spirit of living through our Karate training. OSS!

Summary

Abbreviations and Acronyms	8
Abstract	9
Resumo	10
1 Introduction	11
1.1 Motivation.....	11
1.2 Objectives	14
1.3 Methodology	14
1.4 Work Structure.....	14
2 Background and Related Work	16
2.1 Wireless Networking Technologies Arena.....	16
2.1.1 Wi-Fi (IEEE 802.11).....	17
2.1.2 Bluetooth	19
2.2 Related Work.....	21
2.2.1 Heterogeneous MANET Routing Protocols.....	21
2.2.2 Underlay Networking Architectures	23
2.2.2.1 Ana4 Framework.....	24
2.2.2.2 Lilith	27
2.2.3 Discussion.....	29
3 The 3D Routing Protocol	31
3.1 Terminology and Concepts	31
3.1.1 Terminology	31
3.1.2 Concepts	32
3.2 The 3D Routing Protocol.....	34
3.2.1 Messages.....	34
3.2.2 Protocol Functioning	40
3.2.2.1 The Bootstrapping Phase.....	40
3.2.2.2 The Context Awareness and Route Construction Phase	46
3.2.2.3 Role Management Phase	49
3.3 Populating the NIB	50
3.4 Chapter Overview.....	51
4 Implementation	52
4.1 Hardware.....	52
4.1.1 OpenMoko Neo FreeRunner	53
4.1.2 Nokia N810	53
4.1.3 Laptops.....	54
4.1.4 Bluetooth and 802.11 external adapters	54
4.2 Operating Systems and Programming Languages	55
4.2.1 OpenMoko Linux.....	55
4.2.2 Maemo.....	56
4.2.3 BackTrack Linux.....	56
4.2.4 Python	56
4.3 Libraries and Tools.....	57
4.3.1 Linux Wireless Extensions and Tools	57

4.3.2 Linux BlueZ stack tools	58
4.3.3 802.11 Layer 2 communication: Raw Ethernet Sockets	59
4.3.4 Bluetooth Layer 2 communication: L2CAP Sockets.....	59
4.4 3D Routing Implementation.....	60
4.4.1 3D Router Core	60
4.4.2 The Bootstrapper.....	61
4.4.3 The Scanner-Joiner.....	61
4.4.4 The Monitor	62
4.5 Lessons learned from implementing the 3D Routing test-bed.....	63
4.5.1 Experimenting with existing underlay implementations.....	64
4.5.2 The 802.11 Ad Hoc Mode BSSID Cell Splitting Phenomenon	64
4.5.3 Bluetooth Scatternets: a myth?	65
5 Evaluation	67
5.1 Why use a test-bed?.....	67
5.2 Evaluation Methodology	68
5.3 Instrumentation of the 3D Router Core	68
5.4 Evaluations	70
5.4.1 Scenario 1: Network Bootstrap time	70
5.4.2 Scenario 2: Node Bootstrap time	74
5.4.3 Scenario 3: Measuring RTT, Throughput and Jitter.....	78
5.5 Discussion.....	82
6 Conclusions and Future Work	84
6.1 Contributions	84
6.2 Future work	85
7 References	87
Appendix	91
A - 3D Router Core.....	91
B - 3D Routing Measurement Tool.....	96

List of Figures

Figure 1 Typical MANET scenario	12
Figure 2 Conference Scenario with multi-homed heterogeneous devices	13
Figure 3 Detection of alternate wireless routes	14
Figure 5 802.11 versus Bluetooth in terms of band division.....	19
Figure 6 Bluetooth networking topologies: (a) single-slave, (b) multi-slave, (c) scatternet operation	20
Figure 7 HAODV in a Heterogeneous Ad Hoc Network	23
Figure 8 Ana4 three-level network	25
Figure 9 Lilith packet forwarding mechanism	29
Figure 10. Node Information Base	33
Figure 11 Main 3D Routing Message	35
Figure 12 JOIN/LEAVE Message Format	37
Figure 13 HELLO message format.....	37
Figure 14 INFORM message format	38
Figure 15 NEWLEADER message format	39
Figure 16 Node Bootstrap Flowchart	43
Figure 17 MAC-based IP generation.....	46
Figure 18 Default route selection algorithm	48
Figure 19 Relationships between the 3D Routing NIB, Phases and Messages	51
Figure 20 OpenMoko NeoFreeRunner.....	53
Figure 21 Nokia N810 Internet Tablet.....	54
Figure 22 3D Routing Tools Overview	60
Figure 23 The 3D Routing Network Bootstrapper	61
Figure 24 The Scanner-Joiner	62
Figure 25 The Monitor.....	63
Figure 26 Code fragment of the 3D Router Code instrumentation.....	70
Figure 27 Initial scenario: network bootstrapping	71
Figure 28 Statistical summary of 802.11 Network Bootstrapping time.....	72
Figure 29 Statistical summary of the Bluetooth Network Bootstrapping time	73
Figure 30 Bluetooth versus 802.11 Network Bootstrap time	74
Figure 31 802.11 Node Bootstrapping Scenario.....	75
Figure 32 Statistical summary of the 802.11 Node Bootstrapping time	76
Figure 33 Bluetooth Node Bootstrapping Scenario	77
Figure 34 Statistical summary of the Bluetooth Node Bootstrapping time.....	77
Figure 35 Bluetooth versus 802.11 Node Bootstrap time	78
Figure 36 Final evaluation scenario	79
Figure 37 Time series plot of RTT from PNA to PNB (Figure 36).....	80
Figure 38 Statistical summary for the RTT from PNA to PNB	80
Figure 39 Throughput from PNA to PNB.....	81
Figure 40 Jitter from PNA to PNB.....	81

Abbreviations and Acronyms

BNEP	Bluetooth Network Encapsulation Protocol
BSSID	Basic Service Set Identifier
DNS	Domain Name System
GN	Group Ad-Hoc Network
IEEE	Institute of Electronic and Electrical Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
L2CAP	Logical Link Control and Adaptation Protocol
LAN	Local Area Network
MAC	Medium Access Control
MANET	Mobile Ad Hoc Network
NIB	Node Information Base
OSI	Open System Interconnection
PAN	Personal Area Network
PANU	Personal Area Network User
SSID	Service Set Identifier
RTT	Round-Trip Time
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WLAN	Wireless Local Area Network

Abstract

The emergence of low-cost, wireless network interfaces in the market and the ever increasing growth in demand of mobile devices (such as SmartPhones, PDAs, Internet Tablets and Laptops) enabled scenarios where network services for mobile users may exist in an ad hoc fashion, with no pre-existing communications infrastructure. However, the creation of such dynamic, heterogeneous networks is target of current research.

Several researches in the field of wireless ad hoc networks has been focused in a single wireless technology, based on the IEEE 802.11 standard, where nodes are seen in a plain (2D) perspective, that is, as homogeneous elements identified only by IP addresses, not taking into consideration their hardware profiles and networking technologies. This way, researches involving multiple networking technologies are still in early stages. New approaches are necessary for these scenarios, increasingly more common, involving multiple devices with multiple network interfaces (multi-homed).

This work proposes the 3D routing protocol, targeted towards scenarios where heterogeneity of devices and networking technologies is present. The aim of the 3D Routing protocol is to provide mechanisms for the seamless interoperation of these dynamic, heterogeneous ad hoc networks, considering another dimension of information, hereby defined as third dimension, consisting of aggregating more information, such as context information, hardware resources and network interfaces, to the routing process. Hence, the protocol considers the following fundamental aspects: the bootstrapping process of the network and nodes, the building and spreading of context awareness information between the nodes, and the assignment of roles to particular nodes.

The evaluation of the protocol is made through experiments in a real test-bed, running a prototype implementation of the protocol, consisting of mobile devices such as OpenMoko Smartphones, Nokia N810 Internet Tablets and Laptops, with Bluetooth and 802.11 network interfaces, running embedded versions of the Linux operating system.

Keywords: Ad Hoc Networks, Heterogeneous networks, Routing, Seamless Autoconfiguration.

Resumo

O surgimento de interfaces de rede sem-fio de baixo custo no mercado e o crescimento na demanda por dispositivos móveis (como Smartphones, PDAs, Internet Tablets e Laptops) permitiram a criação de cenários onde serviços de rede para usuários móveis possam existir sem nenhuma infra-estrutura pré-configurada. No entanto, a interoperabilidade entre tais redes, que são dinâmicas e heterogêneas, é atualmente objeto de pesquisa.

Várias pesquisas na área de redes ad hoc sem-fio tem focado em uma única tecnologia sem-fio, baseada no padrão IEEE 802.11, onde os nós da rede são vistos de maneira plana (2D), ou seja, como elementos homogêneos, identificados apenas por endereços IP, não levando em consideração seus perfis de hardware e tecnologias de rede. Desta forma, pesquisas envolvendo mais de uma tecnologia de rede encontram-se em estágios iniciais. Novas propostas são necessárias para estes cenários, que são cada vez mais comuns, envolvendo múltiplos dispositivos com múltiplas interfaces de rede (*multi-homed*).

Este trabalho propõe o protocolo de roteamento 3D, direcionado a cenários onde há heterogeneidade de dispositivos e tecnologias de rede. O objetivo do protocolo de roteamento proposto é prover mecanismos para a interoperabilidade de redes ad hoc heterogêneas, considerando outra dimensão de informações, aqui denominada de terceira dimensão (3D), que consiste em agregar mais informações, como informações de contexto, recursos dos dispositivos e interfaces de rede, ao processo de roteamento. Para isto, o protocolo considera os seguintes aspectos fundamentais: o processo de *bootstrapping* da rede heterogênea e dos nós, a construção e disseminação de informações de ciência de contexto entre os nós, e a atribuição de papéis específicos para determinados nós da rede.

A avaliação do protocolo é feita através de experimentos em um test-bed real, utilizando um protótipo da implementação do protocolo, num cenário composto de dispositivos móveis como Smartphones OpenMoko, Internet Tablets N810 da Nokia e Laptops, possuindo tecnologias Bluetooth e 802.11, executando versões embarcadas do sistema operacional Linux.

Palavras-chave: Redes Ad Hoc Móveis; Redes Heterogêneas; Autoconfiguração; Roteamento.

1 Introduction

“We think in generalities, but we live in details.”

Alfred North Whitehead

1.1 Motivation

In the last decade, with the emergence of low-cost, wireless network interfaces in the consumer electronics market targeted to civilian applications (e.g., GSM/GPRS, Infrared, Bluetooth[52] and WiFi[1]), the Mobile Ad Hoc Network (MANET) [8][9] paradigm gained momentum, even outside the military field. The vision of network services for mobile users in areas with no pre-existing communications infrastructure or with an infrastructure that requires wireless extension is very appealing. In this paradigm, mobile devices self-organize to create a network by exploiting their wireless network interfaces, without a requirement for a pre-deployed infrastructure. The simplest ad hoc network is a peer-to-peer network formed by a set of stations within range of each other that dynamically configure themselves to set up a temporary single-hop ad hoc network. This type of network provides an effective solution to home and office automation by dynamically interconnecting devices in short range. Two cell phones exchanging files via Bluetooth are probably the most widespread example of a single-hop ad hoc network. However, these networks are limited to devices that use the same transmission medium, e.g., Bluetooth nodes are only able to communicate with other Bluetooth nodes. This limitation can be overcome by exploiting heterogeneous devices consisting of two or more technologies, who may act as a gateway between the technologies it supports, forming a heterogeneous ad hoc network. In addition, these heterogeneous networks naturally enable multi-hop communication as well.

In a multi-hop network, the packets are forwarded in an ad hoc fashion by the network nodes from the source to the destination. Nearby nodes can communicate directly by exploiting a single-hop wireless technology. Devices that are not directly connected can communicate by forwarding their traffic via a sequence of intermediate devices. Thus, the network nodes (i.e., the users' mobile devices) must cooperatively provide the functionality usually provided by the network infrastructure (e.g., routers, switches, or servers) [54].

Specifically, for more than ten years research has been focused on efforts to design and evaluate algorithms and protocols to implement efficient communications in a scenario similar to

the one illustrated in **Figure 1**. Here, no infrastructure exists and a number of users' devices that are wireless enabled, usually equipped with a single network technology, work together in order to build routes to each other.

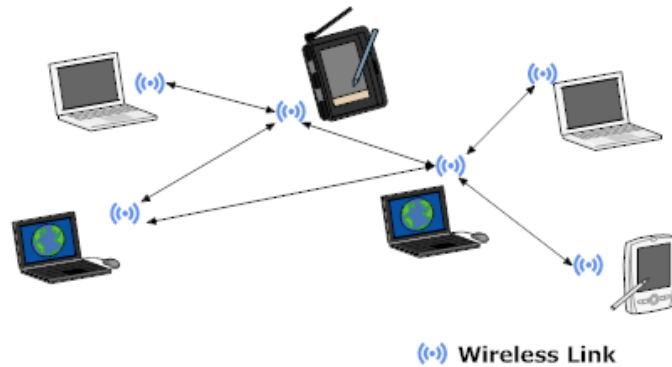


Figure 1 Typical MANET scenario

However, most of the research has been focused on a single wireless technology, namely, the 802.11 standard [1]. Yet, research on heterogeneous scenarios where multiple technologies are involved is still far more incipient [26]. New approaches are necessary for the ever more common scenarios where multiple devices, heterogeneous in their hardware capabilities as well as their communication capabilities can communicate and form heterogeneous ad hoc networks dynamically.

In addition, the need for forming dynamic, heterogeneous ad hoc networks is prevalent. Nonetheless, there are some aspects that need to be considered in order to accomplish such goals.

First, while MANET research has been focused on routing protocols to provide basic multi-hop connectivity, it is assumed that the network is already formed and all nodes have proper addressing. An important step is not considered, which is the initial bootstrapping of the network, and the posterior entrance of new nodes in the network. In addition, as already stated before, there's no consideration for wireless technologies other than WiFi [14].

Second, nodes are treated as equal, even though their hardware capabilities, operating systems and network interfaces may be different. Each node forms a different profile which is in no way taken into account. The network node is just an IP address [8].

Third, although MANETs are decentralized in nature, that does not imply in lack of organization or policies in the network. Some roles may be assigned for particular nodes, which become responsible for setting up and adjusting configurations on behalf of that network [59].

A number of scenarios demand such a solution. Consider a disaster response event where multiple forces are required to assist and provide help to those involved. Information exchange between the forces and the workers involved is a must, and normally the cellular network could be used, as it is almost ubiquitous. But in a disaster event, the infrastructure may have been destroyed in the area and may not be working at all. The only possibility of communication is to form a dynamic network which is potentially formed by multiple devices and multiple networks. To coordinate information exchange among the forces it is necessary to provide mechanisms to inform nodes of each other's presence, to provide mechanisms of joining and leaving a given network, to define policies for what one node is capable of doing or not, etc.

Other scenarios which involve multi-interfaced devices are depicted in **Figure 2**. In this scenario, a number of users may be in a particular place, such as a conference, and need to exchange information among them. However, since devices may have incompatible interfaces, their ability to communicate may be limited, unless an intermediate node, with heterogeneous interfaces, becomes a translator between devices with homogeneous, incompatible technologies. Thus, a heterogeneous network is born composed by homogeneous nodes capable of talking a single technology, as well as composed by multi-homed nodes capable of talking several network technologies at once.



Figure 2 Conference Scenario with multi-homed heterogeneous devices

Another interesting scenario involves optimization through the perception of a given host in heterogeneous networks. For example, a user is trying to communicate with another user through its GPRS interface, but at the same time it detects that it is able to reach this same node through its 802.11 interface. Thus, instead of using a potentially more costly interface (3G or GPRS network), which also offers limited throughput, the node may follow on to establish a connection using its 802.11 interface. This way, the node gains in throughput and economy (**Figure 3**).

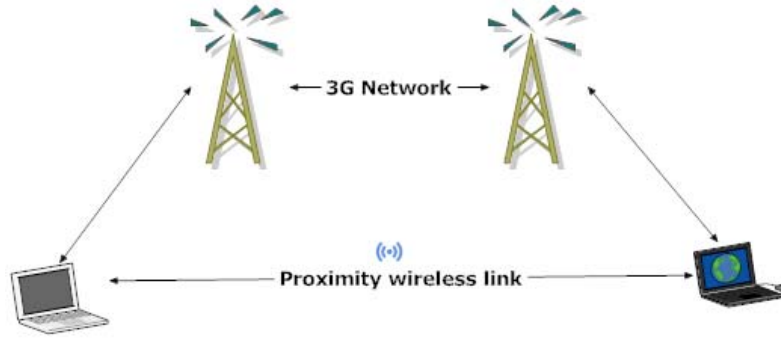


Figure 3 Detection of alternate wireless routes

Such scenarios motivate a new unified approach addressing the challenges involved to accomplish these tasks, with the ultimate goal of providing the creation of seamless, dynamic heterogeneous ad hoc networks. Thus, this work attempts to give a contribution focusing on these challenges.

1.2 Objectives

The overall objective of this work is to elaborate and evaluate a new routing protocol, named 3D Routing, targeted at heterogeneous ad hoc networks.

Specifically, this protocol is divided into three phases, consisting of the bootstrapping process of the network and nodes, the building and spreading of context information of nodes into the network and the assignment and management of roles of specific network nodes. As a consequence, nodes become aware of the context of each other, their hardware, communication capabilities and neighborhood.

Finally, the proposed protocol is evaluated with a functional prototype implementation.

1.3 Methodology

The evaluation methodology for this work will be made through experiments running a prototype implementation in a real test-bed prototype, consisting of mobile devices equipped with multiple heterogeneous network interfaces. Some metrics are analyzed in the experiments: network bootstrap time, node bootstrap time, round-trip time and throughput of the network.

1.4 Work Structure

This dissertation has been organized as follows. In Chapter 2 is given some brief background information and review of current related work.

Chapter 3 presents the 3D Routing protocol, the concepts and definitions behind it, and the actual protocol functioning.

Chapter 4 describes the implementation of the protocol, the hardware and software used in the assembly of the test-bed, as well as the set of tools developed to implement the concepts and ideas behind the protocol.

In Chapter 5, some evaluations of the protocol are performed considering a set of relevant metrics.

Finally, the conclusions, including the enumeration of the contributions and the future works are presented in the Chapter 6.

2 Background and Related Work

“The important thing is never to stop questioning.”

Albert Einstein

In this chapter the main concepts about ad hoc networking and their enabling technologies are presented. Section 2.1 starts with a description on the relevant technologies used to form ad hoc networks today, with a special focus on Bluetooth and 802.11 networks. After that, it presents related work found in literature regarding the interconnection of heterogeneous ad hoc networks. Finally, a discussion of the related work is made.

2.1 Wireless Networking Technologies Arena

The wireless arena has been experiencing exponential growth in the past decade. Great advances have been seen in network infrastructures, growing availability of wireless applications, and the emergence of omnipresent wireless devices such as portable or handheld computers, PDAs, and cell phones, all getting more powerful in their capabilities. These devices are now playing an ever-increasingly important role in the everyday lives of millions of people. To mention only a few examples, mobile users can rely on their cellular phone to check e-mail and browse the Internet; travelers with portable computers can surf the internet from airports, railway stations, cafes, and other public locations; tourists can use GPS terminals installed inside rental cars to view driving maps and locate tourist attractions; files or other information can be exchanged by connecting portable computers via wireless LANs while attending conferences; and at home, a family can synchronize data and transfer files between portable devices and desktops.

In general, wireless networking refers to the use of infrared or radio frequency signals to share information and resources between devices. Many types of wireless devices are available today; for example, mobile terminals, pocket size PCs, hand-held PCs, laptops, cellular phones, smart phones, PDAs, wireless sensors, and satellite receivers, among others. Each of these devices is usually equipped with several heterogeneous networking technologies, allowing them to access several networks at once.

One of the most popular wireless networking technologies today is the 802.11 standard, also known as WiFi [1]. Targeted towards the wireless LAN scope, WiFi aims to be a wireless replacement of the wired Ethernet. WiFi adapters are fairly common in today's laptops, cell phones and PDAs, making it probability the most pervasive wireless LAN networking technology available today. Bluetooth is another popular technology targeted at the wireless PAN scope, in the range of 10m.

Other emerging technologies are Ultra-Wideband (UWB) [61], whose aim is transmitting information spread over a large bandwidth (>500 MHz) that should, in theory and under the right circumstances, be able to share spectrum with other users. This is intended to provide an efficient use of scarce radio bandwidth while enabling both high data rate personal area network (PAN) wireless connectivity and longer-range, low data rate applications as well as radar and imaging systems. Another emerging technology is ZigBee [62], a suite of high level communication protocols using small, low-power digital radios based on the IEEE 802.15.4 standard for wireless personal area networks (WPANs). ZigBee is targeted at RF applications that require a low data rate, long battery life, and secure networking. ZigBee's current focus is to define a general-purpose, inexpensive, self-organizing, mesh network that can be used for industrial control, embedded sensing, medical data collection, smoke and intruder warning, building automation, home automation, domotics, etc.

In this work we consider two technologies: 802.11 and Bluetooth, due to their wide availability in off-the-shelf devices, as well as their capability of forming ad hoc networks. Next, we explore some details of both technologies.

2.1.1 Wi-Fi (IEEE 802.11)

IEEE 802.11 [1] is a family of Wireless LAN (WLAN) protocols. This family includes 802.11a, which operates at 5GHz and provides data rates up to 54 Mbps; and 802.11b/g, which operates at 2.4 GHz. 802.11b operates in the 2.4 GHz frequency band and offers raw bit rates of up to 11Mbps and 802.11g offers up to 54Mbps rates. 802.11b/g networks usually employ a wireless LAN access point (AP) to connect wireless devices between themselves, allowing them to reach outer networks. The IEEE 802.11b standard places specifications on the parameters of both the physical (PHY) and medium access control (MAC) layers of the network. The MAC is a set of rules to determine how to access the broadcast medium and send data, but the details of transmission and reception are left to the PHY.

There are four major components of the WLAN described by IEEE 802.11: stations, access points, distribution system, and wireless medium. The fundamental building block of the

802.11 network is the Basic Service Set (BSS), which is simply a set of stations that communicate with one another. IEEE 802.11 standard defines two types of BSS as shown in **Figure 4**: the infrastructure BSS and the independent BSS (ad-hoc network). The infrastructure network is meant to extend the range of the wired LAN to wireless cells. This mode uses access points with whom mobile stations can communicate using the 802.11 wireless MAC protocol. In the ad-hoc network, two or more stations are brought together to form a network "on the fly" without a central control; and usually every station is able to communicate with every other station.

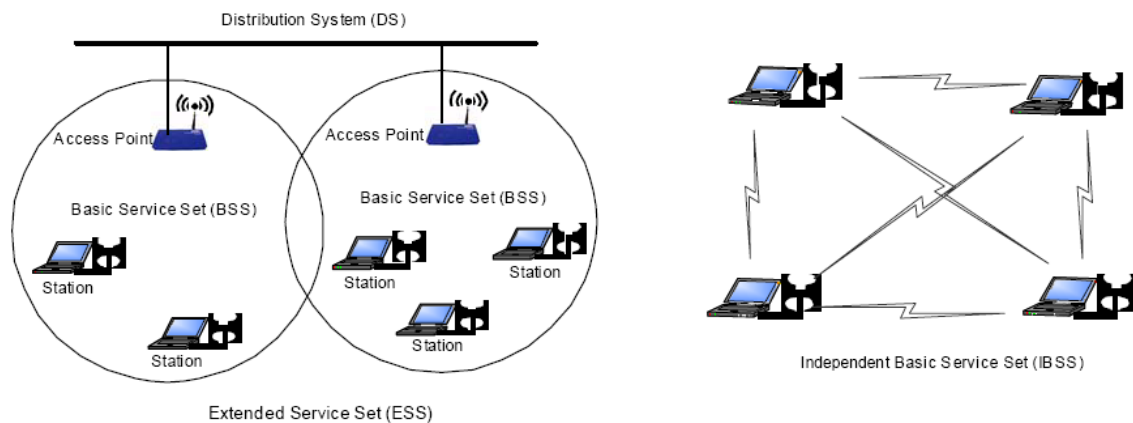


Figure 4 Infrastructure BSS and Independent BSS (Ad-Hoc Mode)

The 802.11 MAC layer is responsible for how a station associates with an access point. One technique for selecting an access point is called active scanning and involves the following steps:

- The station sends a probe request frame to determine which access points are within range;
- All access points within reach reply with a probe response frame, containing capability information, supported data rates, etc;
- The station selects one of the access points, and sends that access point an association request frame;
- The access point replies with an association response frame.

With passive scanning, an access point periodically sends a beacon frame to announce its presence and relay information, such as transmission rate supported by the access point, timestamp, SSID, and other parameters regarding the access point to stations that are within range. Stations

continually scan all 802.11 radio channels and listen to beacons as the basis for choosing which access point is best to associate with.

2.1.2 Bluetooth

Bluetooth (BT) [52] is a short-range, low-cost radio link thought to be a cable replacement between portable and/or fixed electronic devices. Bluetooth operates in the unlicensed industrial, scientific, and medical (ISM) frequency band at 2.4GHz. Unlike 802.11, where fixed channels are established, usually in the range 1 to 14 (in the United States, 1 to 11 in Brazil), Bluetooth employs a fast frequency hopping spread spectrum technique in order to combat interference and fading (**Figure 5**). It divides the band into 79 channels, and keeps changing them frequently.

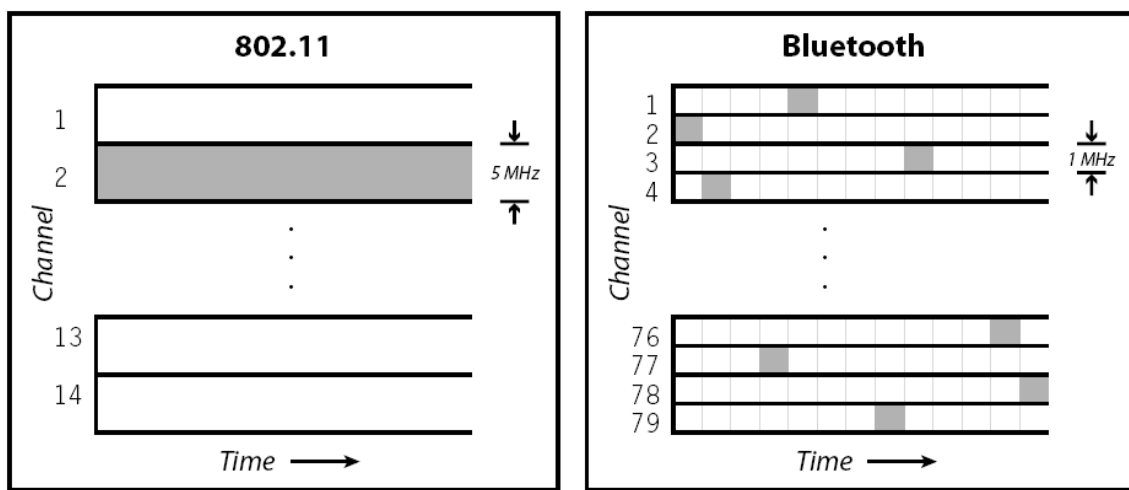


Figure 5 802.11 versus Bluetooth in terms of band division

The BT system provides point to point or point to multi-point connections, a connection of two or more BT units sharing the same channel is called piconet. One BT unit acts as master (M) of the piconet, while the other(s) act as slave(s) (S). Multiple piconets with overlapping coverage form a scatternet. Each piconet can only have a single master, but slaves can operate in different piconets (by means of time division multiplex). A master in one piconet can be slave in others piconets (**Figure 6**).

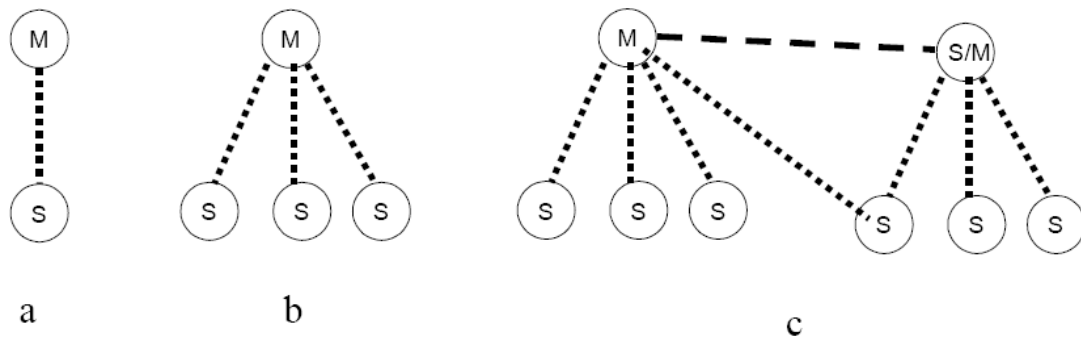


Figure 6 Bluetooth networking topologies: (a) single-slave, (b) multi-slave, (c) scatternet operation

The actual Bluetooth specification is in version 2.1. Since version 2.0, Bluetooth is able to support data rates of up to 3 Mbits/s with the Enhanced Data Rate (EDR) support, reaching distances varying from 1, 10 or 100m (adapter class 3, 2 and 1, respectively)[52].

The Bluetooth specification defines a set of profiles that must be implemented on the devices in order to enable certain functionalities. The PAN (Personal Area Networking) profile defines how to organize Bluetooth ad hoc networks such as the ones shown in **Figure 6**. This profile is detailed in the next item.

Bluetooth PAN profile

The Bluetooth PAN profile lays out the rules for carrying IPv4 or IPv6 traffic over Bluetooth connections. This is done using Ethernet like packets encapsulated in L2CAP packet payloads using the Bluetooth Network Encapsulation Protocol (BNEP) [51]. The PAN profile defines personal area network support in the following situations:

- Ad-hoc networking by two or more Bluetooth devices in a single piconet;
- Network access for one or more Bluetooth devices.

The current version doesn't cover the following topics:

- Automatic network formation;
- General ad-hoc networking where more than one piconet is involved (scatternet);
- Quality of Service.

The PAN profile defines the three following roles:

- Network Access Point (NAP) and NAP service: a network access point is a device that contains one or more Bluetooth radio devices and acts as a bridge, proxy, or a router between

a network (10baseT, GSM, etc) and the Bluetooth network. Each Bluetooth device will have access to all of the LAN's shared resources. The device with the NAP service also forwards Ethernet packets between all of the connected Bluetooth devices.

- Group Ad-hoc Network (GN) and GN service: group ad-hoc networking is a collection of Bluetooth mobile hosts that cooperatively create an ad-hoc wireless network without the use of additional networking hardware or infrastructure. The device with the GN service forwards Ethernet packets between all of the connected Bluetooth devices.
- PAN User (PANU): a PAN user is a Bluetooth device that uses either the NAP or the GN service.

The NAP/GN service performs Ethernet Bridge functions to forward packets from one PANU to another PANU, using a subset of the IEEE 802.1D standard[48].

2.2 Related Work

Research involving heterogeneous ad hoc wireless networks is fairly recent. Most of the focus in this research has been on interoperation of heterogeneous technologies, in order to provide seamless connectivity between them. As a consequence, several works emerged in the form of protocol and architectures. Thus, we divide this section in two. In section 2.2.1, it is reviewed the main protocols focusing on heterogeneous ad hoc networks. In section 2.2.2, novel underlay architectures targeting heterogeneous and hybrid networks are covered.

2.2.1 Heterogeneous MANET Routing Protocols

There are number of routing protocols developed for the last ten years of research on ad hoc network, such as Ad Hoc On-Demand Distance Vector (AODV)[16], OLSR[7] and DSR[13]. Some of these protocols became IETF standards, and implementations are widely available. Recently, some works have focused on adapting such protocols, which originally only work for 802.11 ad hoc networks, to interoperate in the context of heterogeneous MANETs. In this section we cover some of the most relevant works. Here, we highlight only routing-related solutions in heterogeneous MANET.

Traditional Routing algorithms in MANETs focused on the 802.11 standards, being a widely available technology for ad hoc networking. In this context, it is assumed that nodes

have a unique identifier, the IP address, assigned to a single interface. AODV [16] is a well-known reactive protocol capable of discovering routes on-demand.

One of the pioneering work on network heterogeneity focusing on the network interface level was presented in [4], a scheme that supports interoperability between heterogeneous interfaces in MANETs. This scheme uses a unique IP address as a node identifier, a unique interface index for each node interface, and the DSR protocol to support interoperability. In DSR, the sending node first discovers the complete route to reach the destination. The path is an ordered sequence of network hops (i.e., node identifiers) between the source and the destination. Then each packet sent carries this path in its header. The scheme proposed consists of defining the path using not only the node identifier but also the interface(s) that shall be used at each node. This approach has the drawback of increasing the size of each packet's header, as both the source-route and the network interface are included in every packet.

More recent work is described in [5], where AODV was also extended to support heterogeneous networks in a way that is biased to the most capable nodes. A routing algorithm named Heterogeneous Biased Route Discovery (HBDR) was introduced as a method to avoid choosing resource-limited nodes along the built path. HBDR makes use of the delay value which is based on the heterogeneous properties of the node to favor choosing it, by categorizing the nodes in a network as powerful nodes, limited nodes, and weak nodes to force the route discovery towards the most powerful nodes. However, the underlying scheme for supporting heterogeneity in MANETs is related to the hardware capability of nodes, considering nodes with homogeneous 802.11 interfaces.

Another work extending AODV for Heterogeneous MANETs is proposed in [20][21]. Heterogeneous AODV (HAODV) is an extension of the AODV algorithm to operate on heterogeneous networks, in this case a Bluetooth and 802.11 network in ad hoc mode. An interoperability model, the Communication Abstraction Model (CAM)[21] was proposed in a previous paper to handle packet conversion between the different networking technologies. They extend the Enhanced AODV protocol to support heterogeneity via the CAM.

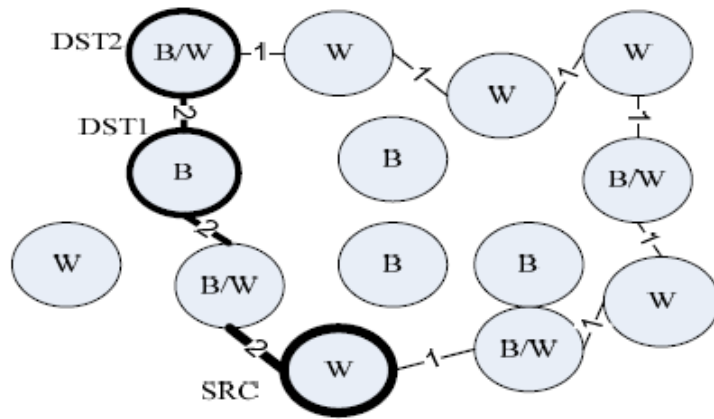


Figure 7 HAODV in a Heterogeneous Ad Hoc Network

The proposed protocol selects the shortest stable routes which may be composed of nodes equipped with heterogeneous communication interfaces. For example, consider the network topology shown in **Figure 7** in which nodes are enabled with different technologies: Bluetooth, WIFI, or both. In a typical AODV implementation, the optimal route from node SRC to node DEST2 will be route path 1. Although a shorter route might exist through path 2, AODV, which does not support interoperability, will not consider this shorter path since it will only follow the nodes with a WIFI interface. HAODV evaluates all the possible routes and selects the optimal one, such as path 2, in this case. Thus, an optimization as well as interoperation is achieved in HAODV.

Finally, the solutions considering adaptations of existing protocol have not made available any implementations for testing on real systems. Mostly these works rely on simulation tools for evaluation, which is considered today in the MANET community as an unreliable research tool [25].

2.2.2 Underlay Networking Architectures

According to the OSI Layer Model and the TCP/IP Model, the routing functionality is located at Layer 3, the networking layer, for which in the TCP/IP protocol suite refers to the Internet Protocol (IP) layer. While most of the focus on MANET research has been on IP routing approaches, there has been considerable efforts to develop packet forwarding mechanisms at the underlay level (below IP).

The reasoning behind underlay networking for MANETs consists on offering a multi-hop abstraction layer for layer 3 and above. This way, a single subnet illusion is created for layer 3 and above protocols.

Next, we review relevant works on underlay networking for heterogeneous ad hoc networks.

2.2.2.1 Ana4 Framework

The Ana4 Framework [3] was designed with the goal of allowing communication between devices of a hybrid and heterogeneous network environment, such as for Intranet connectivity scenarios. The idea is to provide end-to-end communication independently of the number of network interfaces in each node, as well as supporting TCP/IP applications, aiming to keep interoperable inter-networking capability over a heterogeneous networking infrastructure. In addition, Internet connectivity should be provided with other nodes or services which are not localized within the ad hoc network.

To achieve this goal, Ana4 operates at layer 2.5 to provide solutions for ad hoc issues such as address auto-configuration, integration with TCP/IP network protocol stack, and vertical handover¹.

Ana4 Architecture

The Ana4 architecture is divided in three levels of network abstraction: Hardware level, Ad hoc level and the IP level (**Figure 8**).

- **Hardware level:** represents all available hardware networks (physical interfaces). At this level, the notion of communication ability is related to link layer device compatibility and there's no effective communication possibility. The MAC address identifies each hardware interface.
- **Ad hoc level:** represents the ad hoc network. It is a combination of the existing hardware networks in a node, independently whether the interfaces are wired or wireless. At this level, each node is seen through its virtual ad hoc interface and a unique ad hoc address (called Ana4 address) is used to identify it. The role of a virtual

¹ Vertical handover is the ability to switch between different network interfaces within the same node without interrupting its sessions or having to deal with any IP changing mechanism.

ad hoc interface is to hide the different physical devices and hardware networks and to provide the illusion of a single virtual network [6].

- **IP level:** represents the IP abstracted network. In other words, it is an abstraction of the network as a switched Ethernet link, where the hardware address of every node is the ad hoc level address. This way, all nodes have a single Ethernet interface.

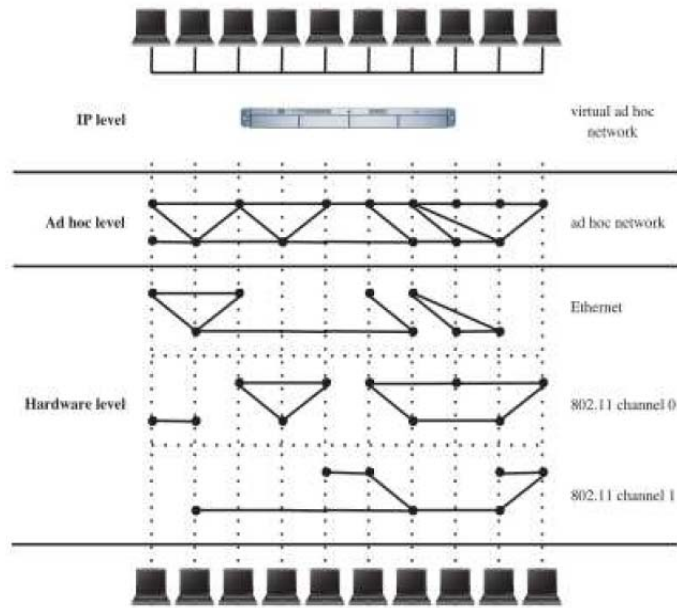


Figure 8 Ana4 three-level network

Ana4 Addressing

As stated before, the addressing scheme used by Ana4 is based on the virtual ad hoc interface, present in all ad hoc nodes. Virtual interfaces are addressed using ad hoc addresses.

An ad hoc address is composed of two fields: Net Id (network identifier) and Node Id (node identifier). Each node independently deduces its own Node Id based on one of its MAC address. At the ad hoc level, data packets are sent and received over the different available interfaces. To reach its destinations, the packets are given to the virtual interface. At the IP layer, the virtual interface acts as a classical interface. This characteristic allows a node to use a device simultaneously in ad hoc mode and in a classical way, with any modification neither in device drivers nor in the TCP/IP stack.

The effective communication through virtual interfaces depends on the address translation, as it internally receives an IP packet; it has to translate a destination IP address

into an ad hoc address. This is solved using a table to record the mappings between IP addresses and ad hoc addresses, called the ARP table. Filling this table is a mechanism deeply bound to the one of creating ad hoc routes and performed by the routing protocol. Moreover, the virtual interface also maintains the ATP table, similar ARP table, to maintain only direct neighbors. ATP table stores Ana4 address, MAC address, output device, and metric (1 for wired and 2 for wireless). Its fields are filled by 1-hop data packets or control packets of the ad hoc routing protocol.

Ana4 Routing

Routing in Ana4 is based on information about the neighborhood and the network topology stored in three tables: the ARP table, ATP table, and COM table. The COM table contains Ana4 node address, Ana4 gateway address, metric (equal to ATP table), and weight, i.e., routing information. The COM table is filled up by routing protocols like OLSR [18], AODV [16] or DSR [17] since the routing protocol is independent of Ana4 architecture.

In cases where no ad hoc routing protocol is used or if no route is found for a specific node, Ana4 employs multi hop communications using a default routing procedure. In this case, unicast packets are broadcasted in the network using the default unicast address. This destination address is used to broadcast in the ad hoc network an ad hoc packet that is destined to a unicast upper-layer address (i.e. IPv4 unicast address). Upon reception of such a packet, it is the responsibility of a node to decide whether the packet is intended to itself based on upper layer information such as the IP destination address.

Additional features

As an additional feature, Ana4 has a Software Suite composed by Ana4 Routing Daemon and Ana4 Network Topology visualization Tools; and there is an Ana4 implementation for Linux and Windows XP (although no code was released for the Windows version).

Considerations: the Ana4 addressing mechanism can be interesting because of its simplicity. Moreover, Ana4 supports interoperable inter-networking capability over a heterogeneous networking infrastructure. The virtual ad hoc addressing allows hiding the heterogeneity hardware and networks, and complies with its connectivity requirements. Furthermore, Ana4 does not need changing neither in devices nor in the TCP/IP stack implementation.

On the other hand, it is necessary that each device stores some extra tables, such as the ARP table to translate IP address into ad hoc address, the ATP table to store direct neighbors information, as well an additional overhead of the layer 2.5 headers for each packet.

Finally, Ana4 has ceased its development a little after its release, and has not built any active user-base or even documentation. During the development of the 3D Routing Protocol, Ana4 was considered as the communication substrate for which the protocol could operate on, but during our initial experiments on introducing Bluetooth support for it, we noticed several stability and compatibility problems with the BNEP [51] emulation used. Given the lack of documentation and developer support to address those issues, we gave up on using Ana4 for our prototyping work and started an implementation from scratch.

2.2.2.2 Lilith

In Lilith [28] it is proposed an interconnection architecture for spontaneous edge networks. By spontaneous they mean the hybrid, heterogeneous networks that are formed on the edge, such as home networks. Similar to Ana4's layer 2.5 approach, Lilith is based on Multi Protocol Label Switching (MPLS) [19] to provide a connection-oriented solution for spontaneous networking scenarios [29].

The solution can be denoted as a hybrid approach, characterized as reactive and proactive. Lilith is reactive, because finds the best path at a given instant on demand and it is also proactive, because it maintains existing paths and tries to find alternate paths that can be immediately used as a back up after a link failure.

Lilith Architecture

In the Lilith architecture, routes and data paths are dissociated so that different protocols can be used for finding a route and establishing a data path. Lilith is composed of the following elements:

- **Interception module:** this module intercepts unicast packets for which no Label Switched Packets (LSPs) exists and broadcast these packets. Then the packets are passed to the Routing protocol module. When Routing protocol module receives unicast packets, it starts the search for a route, which in turn will trigger the establishment of a LSP. Otherwise, when the Routing protocol module receives

broadcast packets, it uses the underlying route discovery mechanism to propagate them to all hosts in the network.

- **Routing protocol module:** this module implements a reactive ad hoc routing protocol, similar to AODV. A ROUTE REQUEST message is generated when a route is needed. This kind of message contains a record of visited nodes of the subnetwork. Once the request reaches the destination node, it generates a ROUTE REPLY message. This message comes back on the reverse path. A BROADCAST message is used when the protocol needs to propagate a broadcast or a scope multicast to all nodes of the subnetwork.
- **LSP establishment module:** this module uses a LSP establishment protocol and makes use of two messages: PATH REQUEST and PATH REPLY. The PATH REQUEST propagates to the destination that replies with PATH REPLY carrying the label bindings for LSP establishment. The Label Information Base (LIB) modification, achieved at each node on the reverse path LSP establishment module, creates a LSP path given intermediate node or a host (Membership Announcement and Management).
- **LSP maintenance module:** this module is responsible to acknowledge traffic received over established LSPs. It is made by sending I'M ALIVE (IMA) message to all immediate neighbors. This information can be used to decide if a given link is broken, or to estimate link quality (Membership Management).
- **LSP optimization module:** this module tries to find alternate routes for existing active destinations using the Routing protocol module. LSP optimization module evaluates the quality of candidate alternate routes and, if relevant, establishes paths using the LSP establishment module.

Lilith Routing

The global idea is to organize a spontaneous network as a single IP subnetwork, to interconnect links with MPLS and to manage Label Switched Path (LSP) paths on demand with a reactive ad hoc routing protocol. Interconnecting links with MPLS and managing flows with dynamic establishment of LSP presents some advantages: LSP paths are explicitly created between end point hosts, avoiding transient loops; multiple paths can be provide for

load balancing or traffic isolation for different QoS requirements; and nodes receive periodically information on the reachability of established LSPs.

By being MPLS-based, Lilith makes possible flow-based forwarding of packets to multiple paths. Since they assume that spontaneous edge networks form a multi-hop ad hoc network, applying MPLS in this context is innovative. Their key advantage is to support network autoconfiguration and support for TCP/IP applications in a usable, practical way for the user. This is possible since they are working at Layer 2.5, as show in **Figure 9**.

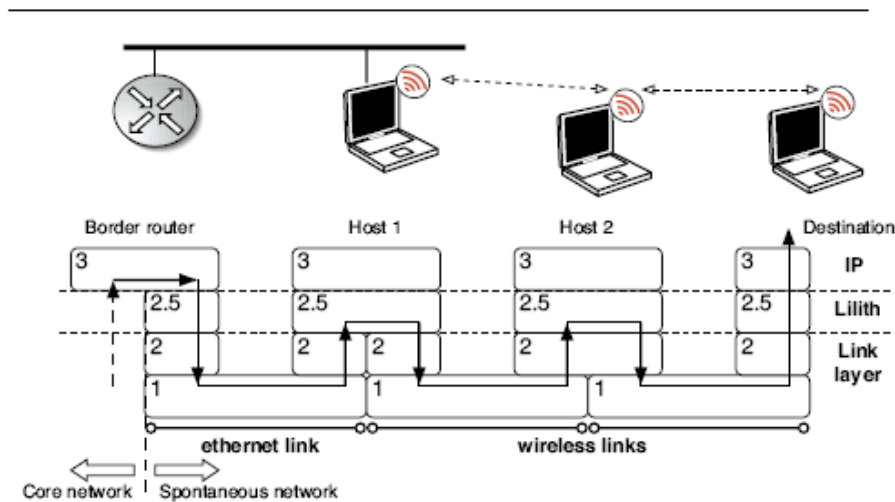


Figure 9 Lilith packet forwarding mechanism

A flow follows a LSP established on demand by an ad hoc routing protocol similar to AODV. Flows with different QoS requirements may use different QoS paths. A Lilith node receives messages with statistics metrics periodically.

Considerations: the peculiarity of the Lilith architecture is in its use of MPLS to forward packets in a multi-hop ad hoc network context. Moreover, Lilith can choose different paths according QoS requirements, by managing LSP paths. Lilith also allows interconnection between heterogeneous networks. However, as far as we are concerned, this solution is purely academic and has not released any public source code or managed to perform tests on real test-beds.

2.2.3 Discussion

In this section we have covered most of the relevant related work where heterogeneity of nodes and/or network interfaces is considered. Thus, it is important to highlight that these solutions focus

either on the adaption of an existing protocol for homogeneous networks, such as the case of HAODV, while others focus on developing new architectures, for which an existing or new protocol may operate, such as the case of Ana4 and Lilith. Still, there's a strong focus only on routing and interconnection issues.

Nonetheless, it's noticeable that in all the works presented above, none of them mention the process of bootstrapping the network, the very beginning. Even Lilith, that coined the word spontaneous, does not provide a clear mechanism of how these networks are born. Who is responsible or able to start a Lilith network? The same can be questioned for Ana4. They mention that their architecture is able to support auto-configuration of nodes, but how is this achieved? There isn't, again, a clear mechanism established.

In order to have dynamic, seamless networking, not only routing should be taken into consideration, but the very birth of the network is as important, as well as their entrance (how nodes join an Ana4 network?). Also, there's no distinction, for example in Lilith, if the node is a sensor or not. Nodes are unaware of each other's capabilities and remain so during the existence of these networks.

Thus, our new approach, the 3D Routing Protocol, is covered in detail in the next chapter, where these and other questions are appropriately taken into consideration and a solution is devised.

3 The 3D Routing Protocol

“Every really new idea looks crazy at first.”

Alfred North Whitehead

The objective of this chapter is to present the 3D Routing protocol. In Section 3.1, the main concepts and terminologies related to the protocol are shown. Section 3.2 details the 3D Routing Protocol, its messages, phases and the Node Information Base (NIB). Section 3.3 presents the relationship between the messages, phases and the NIB. Finally, section 3.4 gives a chapter overview with the most important topics covered.

3.1 Terminology and Concepts

In this section we explain the terminology and concepts used in this work. Although some of these terms may be better defined elsewhere (and we will refer the reader to those definitions in some cases) we include some discussions concerning these terms, as they relate specifically to the tasks involved in 3D routing protocol.

3.1.1 Terminology

- **Neighbor Node:** A node N is a neighbor node M if node M is able to reach directly (one-hop) node N through a given interface.
- **Role:** A given device may have a specific role associated with it.
- **Gateway:** Any node in a 3D Routing Network that has two or more heterogeneous interfaces, and thus is able to reach several networks.
- **Node ID:** A unique identifier for a given device, usually a MAC address from one of the network interfaces a device has. Node IDs are associated with one or more Node Addresses.
- **Node Address:** Addresses associated to a given interface. Each interface usually has a MAC and IP address associated with it. One or more Node addresses are associated with a device's Node ID.

3.1.2 Concepts

- **Context Awareness**

We refer to context as to what capabilities the devices in the network have, in terms of Hardware Capabilities, Connectivity Capabilities and Service Capabilities. In addition, our definition of context encompasses what networks the node may reach, as well as what neighbors are in direct communication with a given node. By exchanging such information, each node in the network becomes aware of each other. That's why we coin the term context awareness.

In addition, **context awareness** refers to the ability of the 3D Routing Protocol to explore the building of routes to a given destination, considering that the node may be in heterogeneous networks, and then make a route decision considering criteria such as hardware capabilities of intermediate nodes or less number of hops to the destination. In the heterogeneous networking scenarios where 3D Routing operates, this may imply on avoiding a route taking a network spanning the globe (such as the Internet) in exchange of a one-hop route (the source and destination may be physically close to each other) discovered by the protocol itself. It is the task of the protocol to sense shorter routes and take an optimal decision in an opportunistic fashion, although this may also be subject to user-defined policies.

Another important aspect of 3D Routing is that nodes may have one or more roles associated with it. This is important to organize the network with nodes capable of performing certain tasks and being responsible for them. A **Leader** is responsible for actions on behalf of the whole network, and is the node who registers all others nodes into the network. **Participants** are the ordinary members of a given network, which may or may not relay data on behalf of other nodes. Other roles include **Gateways**, who act as relay of information for other nodes in a 3D Routing Network.

Context awareness information resides in the Node Information Base (NIB), which consists of a rich repository of information present in every node. The NIB is detailed and its internal structure is shown according to the kind of information it represents.

- **The Node Information Base**

As we previously stated, the NIB is the information repository present in every node in a 3D Routing network. A NIB can be seen as the structure depicted in figure:

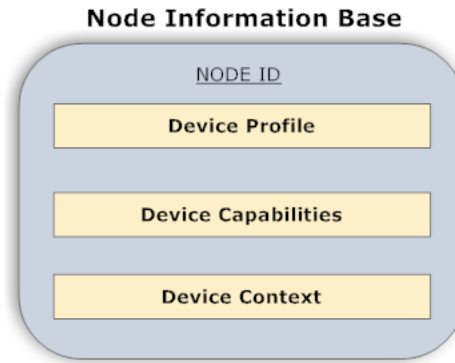


Figure 10. Node Information Base

The first entry in the NIB is about the node itself, and other entries are added as the node builds knowledge of the network topology and participating nodes. It's important to highlight that each node creates an information base about other nodes following this structure, in a similar fashion to a database, with each node being uniquely identified by its Node ID. Thus, the NIB of a node is an entity representing the set of data (Device Profile, Capabilities and Context) for a given Node ID, including the node itself. This set of data is described below:

- **Device Profile (DP):** consists of the network membership list of a node and its respective role in that network. A DP may be initially defined by a simple static policy, or dynamically, through an election process started by the network Leader, based on the device capabilities.
- **Device Capabilities (DCap):** consists of data about the hardware of the device, including network interfaces, processing, storage, and Internet capabilities, and its energy level. In addition, services offered by the device are listed here, in the form of a map of offered services like e-mail, web, ftp and a map of specific shared services as voice, camera, and video processing. To summarize, the DCap represents the capabilities of a node in terms of hardware, connectivity and services.
- **Device Context (DC):** stores knowledge of any nearby (neighboring) devices via each of the interfaces available on the node.

Next, the detail specification of each part of the NIB:

Device Profile (NetworkMembership (NetworkID, Role))

Where:

- NetworkMembership is a list of networks the device is part of, as well as its role in that network. It is in the format of a tuple (NetworkID, Role)

Device Capabilities (ProcessingCapabilities, StorageCapabilities, TransmissonCapabilities, InternetLink, OfferedServices, PowerSupplyType, Energy Level)

Where:

- ProcessingCapabilities is an integer describing CPU processing power in Ghz
- StorageCapabilities is an integer describing storage capacity of the node, in GB
- TransmissionCapabilities describes the network interfaces present on the Device, and their respective MAC and IP addresses, in the form of a tuple: (InterfaceType, MAC Address, IP address).
- InternetLink, may be True (1) or False (0), if the device is able to reach the Internet.
- OfferedServices is a tuple containing services the device itself offers, with types specified in the Service Type item below.
- PowerSupplyType maybe PERMANENT (1) or BATTERY_BASED (0).
- EnergyLevel is an integer representing the percentage of energy, varying from 1 to 100

Device Context (NeighborsBT, Neighbors80211)

Where:

- NeighborsBT is the set of Bluetooth neighbors the device knows of.
- Neighbors80211 is the set of neighbor nodes in an 802.11 network the node may be participating at.

NIB data is encapsulated on the INFORM messages of the protocol. This and all other protocol messages are specified in section 3.2.1.

3.2 The 3D Routing Protocol

3D Routing defines a set of messages necessary for the adequate protocol operation. Each of the messages serves a specific purpose, relating to the three main phases of the protocol: The Bootstrapping Process, Context Awareness and Route Construction Phase, and Role Management phase.

Next, we show the 3D Routing Protocol Messages and follow on to detail the phases above and the messages related to each of them.

3.2.1 Messages

As previously stated, messages in the 3D Routing Protocol are related to the protocol phases. Thus, prior to detailing each phase and message of the protocol, a general overview of the protocol

messages and their basic functionality is shown in order to demonstrate the motivation and reasoning behind them, and their use in the protocol phases, described in the following sections.

- **HELLO:** this message is similar to a beacon that a node keeps sending so that neighbor nodes may hear each other.
- **INFORM:** this is a message is used to disseminate NIB information.
- **JOIN:** it is used by a node to request joining a network.
- **LEAVE:** used by a node to properly leave a given network.
- **NEWLEADER:** this message may only be sent by a leader to pass on his leadership to another node.

Each of the messages above is encapsulated into one main message, containing the control fields necessary for every message transmission in the network.

Packet Format

The basic layout of any packet in 3D routing is as follows:

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	47									
Version								Hop Count								MHC								Length								Message Type															
Node ID																																															
Source Node Address																								Sequence Number																							
Destination Node Address																								Reserved																							
MESSAGE																																															

Figure 11 Main 3D Routing Message

Fields:

- **Version, 8 bits:** The version field specifies the 3D Routing protocol version of this Message.
- **HopCount, 8 bits :** This field contains the number of hops a message has attained. Before a message is retransmitted, the Hop Count must be incremented by 1. Initially, it is set to zero by the originator of the message
- **MaxHopCount (MHC), 8 bits :** This field contains the maximum number of hops a message will be transmitted. Before a message is retransmitted, the MaxHopCount must be

decremented by 1. When a node receives a message with a MaxHopCount equal to 0 or 1, the message must not be retransmitted under any circumstances. Normally, a node would not receive a message with a MaxHopCount of zero. Thus, by setting this field, the originator of a message can limit the flooding radius.

- **Node ID, 48 bits:** This field refers to the unique identifier of a given node, assigned as the MAC address of one of the node's interfaces.
- **Message Type, 8 bits:** This field indicates which type of message is to be found in the MESSAGE field below. Message types are specified in the following subsections.
- **Length, 16 bits:** This is the length of the messages, in bytes.
- **Sequence Number, 16 bits:** This number is important to identify the packets transmitted for a message, so that the order of packets can be checked, and the occurrence of lost or duplicated packets can be detected. This number must be incremented by one every time a new message is transmitted.
- **Source address, 32 bits:** This is the address of the sending node, associated with a given interface.
- **Destination Address, 32 bits:** This is the address of the destination node, associated to a given interface.
- **Reserved, 16 bits:** This is reserved for future use.
- **MESSAGE:** This field encapsulates the 3D Routing messages specified below.

JOIN and LEAVE messages

Here, we describe the message format a JOIN and LEAVE message.

Leaving a 3D Routing network requires a LEAVE request to be emitted, which will again be directed to the network Leader for processing. The network leader, in turn, replies and updates its NIB, as well as spreads this information to all members in the network, which will in turn update their NIBs.

JOIN/LEAVE message format

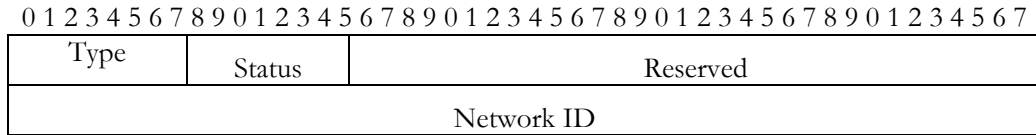


Figure 12 JOIN/LEAVE Message Format

Fields:

- **NetworkID, 48 bits:** The Network ID being announced, to which the node wishes to join.
- **Type, 8 bits:** The JOIN/LEAVE message type, which can be:
 - **REQUEST:** Used by the node to request joining a given network
 - **REPLY:** Used by the Leader to answer for a requesting node. The reply may indicate if the request is successful or not.
- **Status, 8 bits:** The status field indicates the result of the a given request:
 - **POSITIVE:** The Leader accepted the JOIN / LEAVE request
 - **NEGATIVE:** The Leader did not accept the node to JOIN / LEAVE the network.
- **Reserved, 16 bits:** This is reserved for future use.

HELLO and INFORM Messages

HELLO and INFORM are related to sensing neighbor nodes and spreading context information into the network, respectively. Their use is further clarified when the protocol phases are discussed.

HELLO message format

This message acts as a beacon so that neighbor nodes may be aware that its one-hop nodes are reachable. Information gathered from HELLO message exchange is useful to keep the nodes informed of the each other roles and if they are still participating in the network.

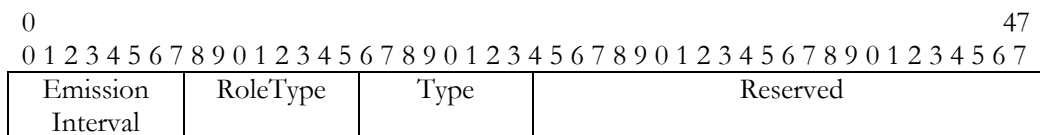


Figure 13 HELLO message format

Fields:

- **Emission Interval, 8 bits:** This field specifies the interval in which the HELLO message should be sent in order to check the reachability of neighbor nodes.

- **Type, 8 bits:** Set to check if the node received the HELLO and replied. This is important to check symmetric / asymmetric links.
 - **REQUEST(0):** Used when sending the message
 - **REPLY(1):** Used when receiving the HELLO, and replying back.
- **RoleType, 8 bits:** This field is used so that other nodes may be aware of the role a given node has at the moment.
- **Reserved, 24 bits:** This is reserved for future use.

The following operations must be performed when transmitting a HELLO message.

- MaxHopCount should be set to 1, since HELLO messages should only reach neighbor nodes
- Emission Interval is set to the default value of 5 seconds.

Upon the reception of HELLO messages, nodes should update their NIB for that neighbor. This way, if a node is not already included in the NIB, the Node should then send an INFORM message containing the NIB. The receiving node should now add this node to its own NIB.

INFORM message format

The task of this message is to populate and update the NIB, which will then be used in the building of routes and context awareness. This way, INFORM messages are sent by a node either periodically. This message may contain the complete Node Information Base or just some fragments of it. In addition, Leader nodes may use this message to announce any event for all network nodes, such as a warning or replying of some negotiation.

0																																						47	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7		
Type								NIB FT								Emission Interval								Reserved															
Data (payload)																																							

Figure 14 INFORM message format

Fields:

- **Type, 8 bits:** This is the type of the INFORM message, which is used by other nodes to perform general informational messaging (for replying, to warn of a failure etc). The following types are possible:
 - **REPLY(0):** For replying a given Message request
 - **WARN(1):** To warn of a Failure in a given negotiation

- **NIB (2):** This type indicates that the Data field contains NIB information, as specified in the NIB FragmentType field.
- **NIB FragmentType (FT), 8 bits:** This field specifies the fragment of the NIB that should be sent in the Data field. The following fragment types are possible:
 - **NONE(0):** If the message is not about NIB Data
 - **FULL(1):** That includes the complete NIB of a given Node ID
 - **DP(2):** Includes the Device Profile only.
 - **DCap(3):** Includes the Device Capability only.
 - **DC(4):** Includes the Device Context only.
- **Data (payload):** This field may include any type of information relevant to the INFORM message. For example, the complete NIB of a node or just some fragments of it, relevant for a certain type of message output.
- **Reserved, 24 bits:** This is reserved for future use.

The firing of an INFORM message is subject to some conditions, such as:

- The entrance of a new node in the Network. This is described in section 3.2.2.
- After joining, every node should flood the network with an INFORM message so that other nodes update their NIBs. This is done after the sensing of the neighboring nodes through HELLO messages.
- When a change occurs in the Device Context, Profile or Capabilities of a given node, it must warn the network members about it. An INFORM message is flooded in the network with the new NIB or the fragment that was changed.

NEWLEADER message

This is the message used by a Leader node to pass on the leadership of the network to a new node.

NEWLEADER Message Format

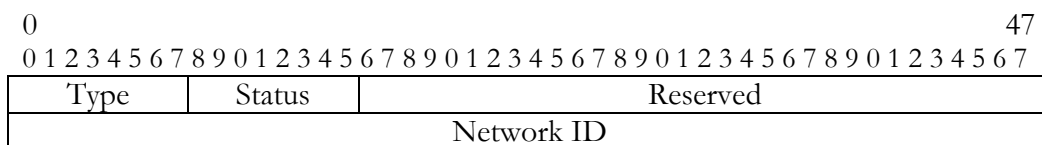


Figure 15 NEWLEADER message format

Fields:

- **NetworkID, 48 bits:** The Network ID to be leaded.
- **Type, 8 bits:** The NEWLEADER message type, which can be:
 - **REQUEST(0):** Used by the Leader to express its will to elect another leadership of a respective Network ID.
 - **REPLY(1):** Used by the Node to answer for a requesting Leader. The reply may indicate if the request is successful or not.
- **Status, 8 bits:** The status field indicates the result of the a given request:
 - **NULL(0):** This value is set when sending a NEWLEADER request.
 - **POSITIVE(1):** The Node accepted the request to become a Leader. It may now receive an INFORM message containing the Leader NIB and start the role of a Network Leader.
 - **NEGATIVE(2):** The Node did not accept the Leadership request from the Leader. Other status messages may be used to indicate multiple results of a NEWLEADER negotiation.
- **Reserved, 32 bits:** This is reserved for future use.

3.2.2 Protocol Functioning

This section outlines the overall protocol functioning. 3D routing is modularized into three functionalities, which are always required for the protocol to operate. Next, we provide details about each one of them.

3.2.2.1 The Bootstrapping Phase

The purpose of the Bootstrapping Phase is providing proper mechanisms for two main processes in a 3D Routing Network:

- **Network bootstrapping:** this is the process of starting a new 3D Routing Network. This is done only by the first node of the network, the node that is giving birth to the network. This peculiar node, the first node, is named **the network bootstrap node**.
- **Node bootstrapping:** The process of a new node joining an existing 3D Routing network through a participant node is called node bootstrapping. This process may be done by any participating node of the network, with the consent of the network Leader.

Thus, it's important to notice that, while the network bootstrapping phase should be done once, the node bootstrapping phase occurs continually, as nodes start participating in the network and should be properly introduced in the network by any in-range participating node.

In addition, the bootstrapping phase is a core part of the 3D Routing Protocol, as it is responsible for the birth and entrance of nodes in the network. Thus, the mechanisms behind the bootstrapping phase focus on providing seamless configuration of nodes using the 3D Routing Protocol.

The following subsections discuss in detail the two main processes of the bootstrapping phase in detail.

The Network Bootstrapping Process

In this process, the network bootstrap node proceeds with the following steps:

- Sets the Network ID (NetID) of the network;
- Define a set of policies that should be followed by every node in the network;
 - This is described in section 3.2.2.3;
- Finally, the network bootstrap node employs the configuration mechanisms for the network interface it is creating the network. This includes how the NetID is announced so that new nodes are able to hear for that network and request for joining the network. Since this is a technology-dependent procedure, it is specified separately for each technology in the item below.

Technology-dependent mechanisms

An additional goal of the network bootstrapping process is to provide a seamless solution for the creation of a 3D Routing Network. As 3D Routing aims at operating with heterogeneous link-layers, it's expected that each media has its own peculiarities and necessary configuration adjustments. Here, we describe the general aspects of such configuration mechanisms, for each of the technologies being considered in this work: Bluetooth and 802.11 links. In the next chapter, we describe how such configurations are performed in an actual operating system.

Bluetooth PAN (piconet)

As seen in section 2.1.2, according to the Bluetooth PAN specification, a node must assume the role of network master, called Group Ad-Hoc Network Controller (GN) node, while others assume the PAN User (PANU) role. This is to comply with the master-slave topology common to Bluetooth

PANs. In the 3D Routing Protocol, the master (GN) role is mapped to the network bootstrap node. Thus, the following steps are necessary for Bluetooth network bootstrapping:

- **Setting up the Network ID:** The NetID in a Bluetooth link is defined by the Bluetooth Device Name. By default, a common prefix, “3DR”, is used for nodes to signal each other of hearing a 3D Routing Network;
- **Configure Network Bootstrap node as the piconet Master:** As pointed out previously, the bootstrap node is mapped as the Bluetooth master;
 - This implies in setting up the Linux Ethernet Bridge [2][48] software, so that the master node is able to receive connections from multiple slave nodes as well as forwarding packets between all other slaves (PANUs);
- **Define an IP range for the network.** At the end of the network bootstrap phase, the network must be formed and layer 3 addresses should be defined;
- **Assign itself an IP address.** This is the IP address of the network bootstrap node, to be used as the layer 3 address of the interface selected for network bootstrapping. This is selected based on the MAC address of the node, according to the algorithm defined in the item “MAC-based IP address generation algorithm” at the end of this subsection.

802.11 Ad Hoc Networks

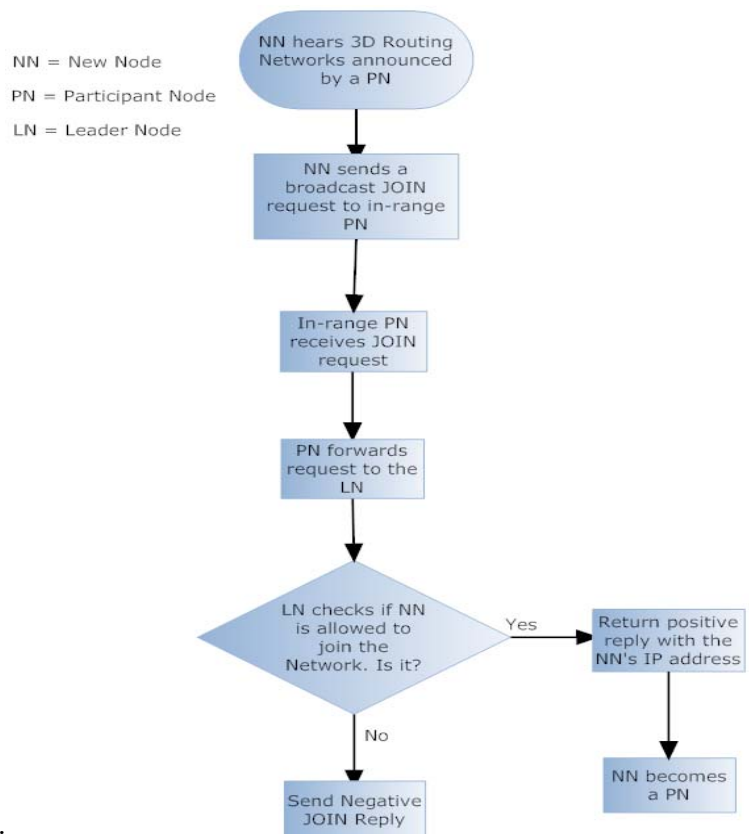
The 802.11 standard extends on the Ethernet standard, and thus inherit its traditional broadcast capabilities. Furthermore, unlike Bluetooth, 802.11 is a connectionless network. Also, there’s no notion of roles or any kind of master-slave relationship as in Bluetooth PANs. In addition, unlike Bluetooth, 802.11 requires manual configuration of the radio channel for operation. 802.11 nodes must all agree on the same channel, as well as on the same ESSID (Extended Service Set ID) in order to communicate. The Network ID in an 802.11 link is mapped to 802.11 ESSID. Similar to Bluetooth, a common prefix, “3DR”, is used for nodes to signal each other that the network is operated by the 3D Routing Protocol. To summarize, the following steps should be done by the network bootstrap node of an 802.11 ad-hoc network:

- **Physical and MAC Layer parameters setup:**
 - Setting up the network interface to operate in Ad-Hoc Mode;
 - Set up the ESSID, which is a string representing the Network ID;
 - Define a channel of operation;

- **Define an IP range for the network.** At the end of the network bootstrap phase, the network must be formed at layer 3. An address range should be defined to be used for all other nodes in network.
- **Assign itself an IP address and bring the interface up.** This is the IP address of the bootstrap node, to be used as the layer 3 address for the interface selected for network bootstrapping. This is selected based on the MAC address of the node, according to the algorithm defined in the item “MAC-based IP address generation algorithm” at the end of this subsection.

The Node Bootstrapping Process

This part of the Bootstrapping phase deals with the entrance of new nodes in an existing 3D Routing network. This is a continuous process, which starts with the first node, the network bootstrap node, and goes on with all other nodes participating in the network: once they are part of the network, they also become able to introduce new nodes in the network, by forwarding requests to the Leader node and back to the new node, being an intermediate node in the process. This



process is depicted in **Figure 16** below:

Figure 16 Node Bootstrap Flowchart

- A new node hears the NetID announcement from a given in-range participant node, and sends a JOIN request to the PN expressing its willingness to be part of the Network.
- The participant node forwards the request to the Leader node, which in turn decides to accept or not the new node into the network based on policies established. Then, it sends a join reply to the requesting participant node with the status of this decision.
- The Participant Node replies to the new node, and if the reply is positive it contains the new node's IP address. The new node then proceeds to become a participant node of the network.

Technology-dependent mechanisms

The technology-dependent mechanisms employed by the nodes in the node bootstrapping process are similar to the network bootstrapping process, with an addition to the negotiation process. Again, here we cover the steps necessary for each technology, Bluetooth and 802.11, in order to achieve the desired effects of seamless configuration of the nodes in the bootstrapping phase.

Bluetooth

- New nodes willing to participate in a 3D Routing Network perform a Bluetooth Inquiry operation, searching for a master node that it is able to answer for Join requests from the protocol;
- Next, since Bluetooth is a connection-oriented technology, it is necessary to establish a connection before exchanging messages. This is the pairing process, done between the PANU and GN. But first the JOIN negotiation must be done. Bluetooth allows layer 2 communication and a JOIN message is sent in unicast to the in-range participating node (piconet master), which will then reply with a valid IP address or a failure message; the details of how this is done is covered in chapter 4.
- After receiving the IP address, the new node now proceeds to a pairing process with the participating node. This process will then bring a network interface up on both devices;
- Finally, the new node assigns itself the IP address received from the master node and is now part of the network.

802.11 Ad Hoc Networks

- New nodes scan for available networks. Since these nodes are supposed to be in-range of a Participant Node, they are able to acquire the channel and ESSID they should use.

- Set the network interface to Ad Hoc Mode;
- Set the channel and ESSID to be exactly the same as the participant node;
- Send a JOIN request in broadcast; it is important to notice that this is a layer 2 message, since the new node is not yet with a valid IP address of the 3D Routing Network. This is further detailed in chapter 4.
- Now, the participating node receives the JOIN request and forwards it to the Leader;
- Finally, the node receives a valid IP address and assigns it for itself.

The next phase is the Context Awareness and Route Construction Phase, which is done immediately done after the Bootstrapping phase. We detail this phase in section 3.2.2.3.

- **MAC-based IP address generation algorithm**

Operating at layer 2, nodes must exchange negotiation messages necessary for the configuration of IP addresses class and netmask, as shown in the General Mechanisms section. Also, to avoid duplicate address assignments, the Network Leader is responsible to provide a unique, unassigned IP address to all Participant Nodes.

- The IP address assignment operation is not a trivial issue and it's the target of current networking research [37]. Nonetheless, in 3D Routing, the bootstrapping phase is responsible for the seamless configuration adjustments comprising not only IP address assignments, but also other parameters specified above (specific to each link-layer). Thus, the final mechanism in the node bootstrap process runs on the Leader node and runs an algorithm for generating a unique IP address based on the MAC address of the new node. The goal is simple: to reduce the probability of duplicate addresses, while creating an association between the MAC address of the interface and the IP address generated. Of course, the effectiveness of such approach relies widely in the selection of a large IP range (such as 10.0.0.0/8). The algorithm is described below (Python code):

```
Generate_Random_IP(mac, iprange):
    ret = "%s.%s.%s.%s"
    iprange = iprange.strip()
    ip_base, cidr = iprange.split("/")
    i3, i2, i1, i0 = ip_base.split(".")
    mac = [int(x, 16) for x in mac.split(":")]
    for i, val in enumerate(mac):
```

```

        if val >= 255:
            mac[i] = random.randrange(254)

    m1, m2, a, b, c, d = mac

    if cidr == "8":
        ret = ret % (i3, b, c, d)

    elif cidr == "16":
        ret = ret % (i3, i2, a ^ b, c ^ d)

    elif cidr == "24":
        ret = ret % (i3, i2, i1, b ^ c ^ d)

    else:
        raise ValueError("CIDR must be '8', '16' or '24' ")

    return ret

```

Figure 17 MAC-based IP generation

- Lastly, even if it's the Leader that is providing the participants IP addresses, there's a possibility of duplicate addresses in the network. At the moment this not part of the protocol scope, but it is an important issue that should be considered in the future [37].

Outcome from the Bootstrapping phase

At the end of this phase, the network should be formed with the following elements:

- **A main network leader:** and one or more network leaders in standby. The active network leader is responsible for determining policies, acting on behalf of the whole network.
- **Participants:** Any node participating in a 3D Routing Network;
- **Gateways:** Nodes with two or more interfaces able to forward messages on behalf of nodes with single interfaces. These nodes are responsible for addressing the existing heterogeneity in a 3D Routing Network.

In addition, the roles of nodes may change over time. The Role Management phase is responsible for dealing with the nomination of roles in the network during the course of its existence. This phase is described in section 3.2.2.3.

3.2.2.2 The Context Awareness and Route Construction Phase

In this phase, nodes start to spread their context information to other network members, in order to build enough knowledge among them. In this phase, two messages are involved: INFORM and HELLO.

At the end of the node bootstrap process, when the node becomes a member of the network, it starts flooding the network with INFORM messages containing its NIB so that other nodes become aware of it.

HELLO messages are sent in parallel, but unlike INFORM messages, which are flooded into the network, HELLO messages are sent to one-hop neighbors, in order to check their connectivity. Any change or sudden disconnections are detected by HELLO messages, which in turn cause updates in the Device Context of the node. This change is then updated to other nodes, through the periodic flooding of INFORM messages in the network.

Default Route Construction Heuristic

A simple algorithm is run on every node in the network, which scans its own NIB to discover information about nodes in the network. The focus for route construction is initially on the analysis of the Device Context (DC) of each node in the network, which may provide useful topological information for the building of routes between nodes. The algorithm is specified as follows:

For a given Node ID, a route is built using the following heuristic:

1. Search at its own Device Context about that Node ID. If that Node ID is found, then:
 - a. For that Node ID, it is added in the routing table its respective Node ID, Destination Address (IP), NextHop Address (IP), Output Interface and Priority. In case there's more than one route to a given Node ID, the Priority field is used to prioritize what route to use.
2. If the Node ID is not found, then now it scans the NIB looking for the device context of other nodes. If this Node ID is found, then a route is added with the respective Node ID, Destination Address (IP), NextHop Address (IP), Output Interface and Priority as well.
3. If the Node ID is not found, then the route selection process is over and no route to that Node ID exists yet.

Such heuristic is shown as the algorithm below:

```
# add a given route to destination (dest) through an intermediate node (gw)
def add_route(self, dest, gw):
```

```

try:

    if dest in [x[0] for x in self.routes]:

        if dest in self.get_iface_80211_neighbors() or\
            dest in self.get_iface_bluetooth_neighbors():

            for i, val in enumerate(self.routes):

                if val[0] == dest:

                    self.del_route(*val)

                    self.routes.pop(i)

                    break

    elif dest in self.get_iface_80211_neighbors():

        print "This neighbor is already known.."

    elif dest in self.get_iface_bluetooth_neighbors():

        print "This neighbor is already known.."

    elif dest == self.get_iface_80211_ip():

        pass

    elif dest == self.get_iface_bluetooth_ip():

        pass

    else:

        bash.route("add", "-host", dest, "gw", gw)

        self.routes.append((dest, gw))

        print "Adding route from %s to %s..." \
            % (dest, gw)

except ShellError, err:

    print "ERROR: Trying to add route. Msg:", err

```

Figure 18 Default route selection algorithm

The route entries are recorded in the routing table in the following format:

1	NODE_ID	DST_ADDR	NEXTHOP_ADDR	OUT_IF	PRIORITY
---	---------	----------	--------------	--------	----------

Table 1 3D Routing Table format

- Node ID (NODE_ID): It's the unique node identifier;

- **Destination Node Address (DST_ADDR):** This is the address of the network interfaces of the destination device;
- **Next-Hop Address (NEXTHOP_ADDR):** This is the Next-Hop towards a destination;
- **Output Interface (OUT_INTERFACE):** The interface for which the packet should be sent;
- **Route Priority (PRIORITY):** In case there are two or more routes to the same destination, the use of a priority is necessary to choose one or the other interface.

The discoveries of routes in 3D routing follows the pro-active paradigm [7] and nodes keep exchanging INFORM messages to keep the NIB, and Routing tables up-to-date. Any changes in the NIB implies in routing recalculations.

It's important to notice that other heuristics may be added to the protocol, such as proximity heuristics. Since at this stage of the protocol we are only considering Bluetooth and 802.11 technologies, such a default heuristic is suitable for simple scenarios with a limited number of nodes.

Outcome from the Context Awareness and Route Construction phase

Although this is a continuous phase, its expected outcome is the following:

- **Updated NIB:** As INFORM messages are being flooded into the network, it's expected that nodes are now aware of each other in terms of their profiles, capabilities and context information.
- **Routing tables:** By employing the heuristics explained above, nodes are expected to have established paths to each other. If multiple routes are available, then simple policies can be employed targeting specific goals, such as battery saving and proximity.

3.2.2.3 Role Management Phase

The assignment of roles in a 3D Routing network is a function of parameters such as policies defined by the leader, hardware capabilities of devices, and user-defined policies, such as a node declaring itself as a gateway. In addition, a Leader node may elect another node as a Leader too, and leave them in standby in case there's a necessity to assume the leadership of a network (in case of a

failure, for example). This way, the network gains fault tolerance and may be able to resist disruptions caused by sudden departures of the actual Leader.

At this stage of the protocol, the role management phase consists of two simple well-defined policies:

- A Leader may nominate another node as a standby Leader by sending a NEWLEADER message;
- Nodes consisting of two or more interfaces become self-nominated gateways;
- The network bootstrap node is the Leader;
- All other nodes are participants.

3.3 Populating the NIB

The information stored in the NIB is collected during the phases of the protocol, as well as during the exchange of messages among the nodes. Specifically, each node builds its own NIB consisting of its Device Profile(DP), Capabilities(DCap) and Context(DC), where:

- DP is defined by the role management phase, where the node assumes a given role and stores the network memberships it is part of. NEWLEADER messages are related to the DP.
- DCap is retrieved from the operating system itself, given that this information consists of hardware, network interfaces and services running on the node.
- DC is fed by the constant updates of neighbors in the network, which are joining and leaving over time. Thus, the messages involved are HELLO, JOIN and LEAVE, and the phase involved is the bootstrapping phase. In addition, while the building of routes is mostly based on DC data, the context awareness and route construction phase is also involved.

INFORM messages are related to all the NIB parts, since this is the message used to spread NIB information throughout the network. Furthermore, this is part of the context awareness and route construction phase.

Figure 19 below shows the relationship between 3D Routing Protocol Phases, Messages and the Node Information Base:

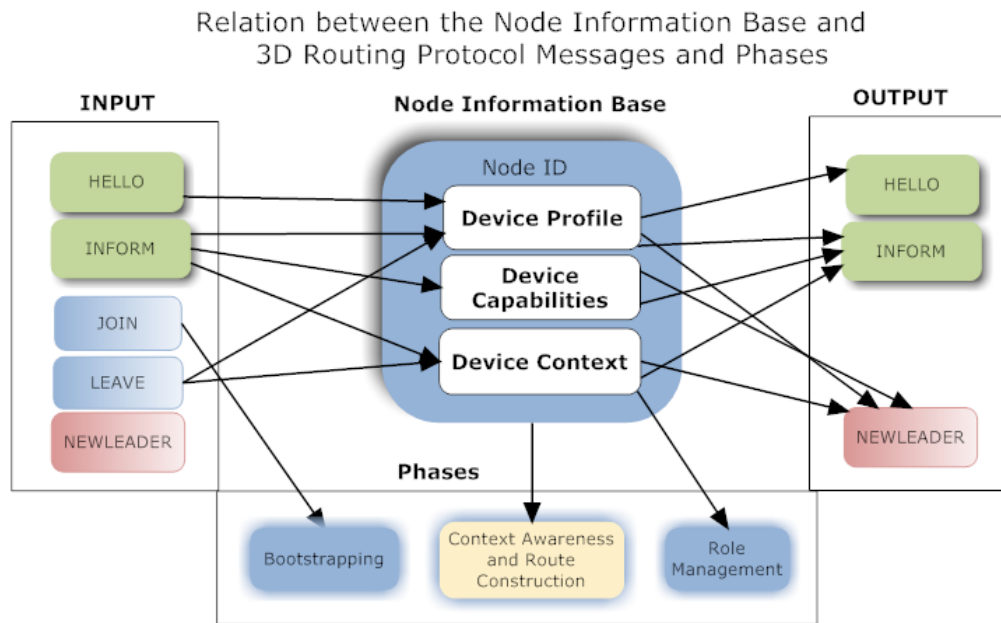


Figure 19 Relationships between the 3D Routing NIB, Phases and Messages

3.4 Chapter Overview

This chapter explained the 3D Routing Protocol in detail. We have seen that it can be divided in three main phases: The Bootstrapping Phase, The Context Awareness and Route Construction Phase and The Role Management Phase. In addition, it was presented the information repository consisting of the Device Profile, Capabilities and Context present in every node, called the Node Information Base (NIB), as well as how 3D routing messages are used during the 3D Routing phases.

4 Implementation

“I hear and I forget. I see and I remember. I do and I understand.”

Chinese Proverb

Building a test-bed for the implementation of the proposed protocol has proved to be a harder task than we expected. Using off-the-shelf mobile devices, we tried to simulate real-life scenarios where multiple devices, running multiple platforms, different hardware and different connectivity capabilities are available to form a network. In this chapter we present in detail the hardware, operating systems, system libraries and software tools used when building the 3D Routing test-bed. In Section 4.1 we present the hardware used in the test-bed, consisting of multiple mobile devices; after that, in Section 4.2, we describe the software tools used in the implementation, from operating systems to programming languages; Section 4.3 covers the libraries and tools used to support our implementation; then, in Section 4.4 we present the actual prototype implementation of the protocol; Finally, in Section 4.5 we discuss some of the lessons learned during our implementation phase.

4.1 Hardware

The choice of the hardware to be used in the test-bed followed a simple criterion: open platforms, running open source software. This is of extreme importance, since open source software allows tighter integration to the hardware, has no kind of limitation regarding the use of tools or accessing any hardware piece via software and so on.

Nevertheless, it was far from trivial finding such open platforms. Although today there are plenty of cell phones, PDAs, laptops, tablets etc, most of them run proprietary software, which are harder or even impossible to customize and develop on. Fortunately, there are two devices that followed our criterion fairly well: OpenMoko Smartphones and Nokia N810 Internet Tablets. They are covered in the next subsection.

Finally, it was necessary to develop and test our software on ordinary laptops, which usually come equipped with an internal 802.11 adapter. On the other hand, driver stability and compatibility with Linux, the operating system used in the devices, is of major importance. In addition, external USB dongles were chosen considering their driver compatibility to Linux.

4.1.1 OpenMoko Neo FreeRunner

Openmoko[57] is a commercial and community driven effort with a mission to create open mobile products that empower developers and consumers to personalize their devices, much like a computer, in any way they see fit. Openmoko is dedicated to helping innovators bring freedom and flexibility to consumer electronics and vertical market devices.

This smartphone comes equipped with multiple network interfaces: GSM/GPRS, Bluetooth, 802.11 and GPS. Besides, it runs a customized version of Linux, allowing complete control of the user to the device. In **Figure 20** below it is shown OpenMoko with their embedded Linux version loaded.



Figure 20 OpenMoko NeoFreeRunner

4.1.2 Nokia N810

The N810[58] is an Internet Tablet device running an operating system based on Linux platform Maemo 4.0, covered in the next section. Similar to OpenMoko, this device supports Bluetooth, 802.11 and GPS connectivity, but does not have any phone functionality and thus is not able to connect to a GSM/GPRS network. Nonetheless, it's a fairly mature product with good stability and available software, capable of running desktop applications with minor or no change.

In **Figure 21** below it is shown the N810 with its attached keyboard and Linux version loaded.



Figure 21 Nokia N810 Internet Tablet

4.1.3 Laptops

Off-the-shelf laptops were used in the test-bed during the software development process as well testing, using either their internal adapters or external USB dongles, which are covered in the next subsection.

4.1.4 Bluetooth and 802.11 external adapters

The choice of proper network hardware was crucial to the success of the test-bed. The external adapters used had a major impact because their chipsets were supported on Linux, and not only this: the ad-hoc mode support, in the case of 802.11 adapters did matter. Unstable implementations of the ad-hoc mode can lead the network to a severe state of instability, making it nonoperational. In sections 4.5, it is covered the main issues faced regarding hardware and driver support.

This is also valid for Bluetooth adapters. Some Bluetooth adapters are known to come with a generic and invalid MAC address, such as 11:11:11:11:11:11, making networking them impossible. Apparently, such adapters are manufactured in places without the proper regulations, which demand that each factory provides a unique IEEE compliant MAC address, as listed in the IEEE OUI and Company_id Assignments [38]. In order to avoid these, the choice of the Bluetooth adapters had to take into account well-known manufacturers as well.

For 802.11 the best adapters complying to our criterion, as far as we have researched, were two: Edimax EW-7318USg [41], which contains a Ralink Corporation chipset and Alfa Networks AWUS036H [40], which contain a Realtek chip. Both companies provide open source drivers, which explain their better compatibility with the Linux OS. As for Bluetooth adapters, Trendnet TBW-105UB, a class 2 implementing the 2.0 EDR specifications was chosen. This adapter has a Broadcom Chipset and is fully compatible with Linux.

In addition, Nokia N810, Openmoko and the laptops had their own internal adapters. As for N810, their internal adapter had a Conexant's 802.11 b/g chipset. Openmoko has an Atheros AR6001 chipset [44]. Unfortunately, both adapters have presented several problems, from simple instabilities to major problems such as BSSID Cell Splits. In section 4.5 it is covered in detail the problems regarding this and other issues. Bluetooth adapters in both, OpenMoko and Nokia N810 have the Cambridge Silicon Radio chipsets, a class 2 device implementing the 2.0 EDR version of the protocol [22].

4.2 Operating Systems and Programming Languages

As stated before, we have chosen open source solutions with flexibility in mind. Today, the Linux operating system is widely used in servers and desktops. It is also available for embedded devices, which is why OpenMoko and Nokia N810 run a customized version of it.

In addition, since the test-bed built consists of multiple heterogeneous devices, running different Linux versions, the choice of a multi-platform language was crucial. Yet, we needed tight integration to the operating system and hardware components. To solve this tradeoff, the Python programming language was chosen. In the following subsections, it is further explained the features of Python and Linux on embedded platforms.

4.2.1 OpenMoko Linux

Openmoko Linux [57] is an operating system for smartphones developed by FIC. It is based on the Ångström distribution, comprising various other open source softwares as well.

Unlike most other mobile phone platforms, the phones on which Openmoko Linux runs are designed to provide end users with the ability to modify the operating system and software stack. The platform is also supported by other mobile phones.

Openmoko Linux uses the Linux kernel, GNU libc, the X.Org server plus their own graphical user environment built using the EFL toolkit, GTK+ toolkit, Qt toolkit and the illume window manager. The OpenEmbedded build framework and a modified version of ipkg package system, called opkg, are used to create and maintain software packages.

OpenMoko has a very active community of users and developers, and several channels where information is available. The main channel is the Wiki [57], which contains extensive documentation on OpenMoko, from hardware specifications to detailed setup, development, networking and so on. These resources were extremely useful during the development of the test-bed.

4.2.2 Maemo

Maemo [58] is an operating system for the Nokia Internet Tablet line of handheld computers. It is similar to many handheld operating systems, but unlike OpenMoko Linux, it is based on Debian GNU/Linux and draws much of its GUI, frameworks, and libraries from the GNOME project. It uses the Matchbox window manager, and the GTK-based Hildon as its GUI and application framework.

Similar to OpenMoko, Maemo has a very active user-base and good amount of documentation available. Again, this was very useful during the development and test process.

4.2.3 BackTrack Linux

BackTrack [56] is a customized Linux distribution targeted at security auditing and forensic tools, including for wireless networks. While this is not of concern to this work, this distribution contains the latest drivers for the wireless adapters chosen above, which made it an ideal choice for the Linux distribution running on the laptops.

4.2.4 Python

Python² is a general-purpose high-level programming language. Its design philosophy emphasizes programmer productivity and code readability. Python's core syntax and semantics are minimalistic, while the standard library is large and comprehensive.

Python supports multiple programming paradigms (primarily object oriented, imperative, and functional) as well as running on multiple platforms, such as Windows, Linux and Mac OS X.

The language has an open, community-based development model managed by the non-profit Python Software Foundation, which maintains the de facto standard definition of the language in CPython, the reference implementation.

One of Python's strongest attributes is the integration with the C language. The C language is still in most of systems programming today, in such a way the Linux, for example is written in C. Device drivers are written in C. As Python provides a lot of binding to C

² <http://www.python.org>

libraries, using Python to access low-level information, such as network interfaces, MAC addresses, etc. proved to be simpler than expected. In addition, the use of a high-level language instead of a low-level language provided a better level of abstraction without compromising the functionality needed to implement the 3D Routing Protocol prototype.

4.3 Libraries and Tools

Linux and Python combined provide a very rich set of tools and libraries. Although some of these tools and libraries lack good documentation, with the appropriate time and testing most of the functionality available was used in our prototype.

Next, we cover the tools and libraries used in the prototype implementation of the 3D Routing Protocol.

4.3.1 Linux Wireless Extensions and Tools

The Linux Wireless Extensions³ is a generic API allowing a driver to expose to the user space (where applications run) information such as configuration and statistics specific to common Wireless LANs. This way, a single set of tools can support all the variations of Wireless LANs, regardless of their type (as long as the driver supports Wireless Extension). Another advantage is that these parameters may be changed on the fly without restarting the driver (or Linux). All the drivers in the test-bed complied with Wireless Extension, and were able to use the Wireless Tools seamlessly.

The Wireless Tools (WT) is a set of tools allowing the manipulation of the Wireless Extensions. They use a textual interface and are rather crude, but aim to support the full Wireless Extension. These tools act like reference implementation and are present in all Linux distributions, as listed below:

- **iwconfig** manipulates the basic wireless parameters, such as ESSID, channel and mode (Ad-Hoc, Managed, etc.);
- **iwlist** allows to initiate scanning and list frequencies, bit-rates, encryption keys, etc;
- **iwspy** allows to get per node link quality;
- **iwpriv** allows to manipulate the Wireless Extensions specific to a driver (private);

³ http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html

In our prototype, some of the tools above were used to configure the network interfaces. Ideally, an implementation of Python bindings for the Wireless Extensions would be better, but for reasons of time we chose to use the tools directly from our code.

4.3.2 Linux BlueZ stack tools

BlueZ⁴ provides an implementation of the core Bluetooth layers and protocols. It is flexible, efficient and uses a modular implementation. Some of the features supported by BlueZ are: Support for multiple Bluetooth devices, hardware abstraction, standard socket interface to all layers and devices and service level security support. The BlueZ implementation is divided into several modules: Bluetooth kernel subsystem core, L2CAP and SCO audio kernel layers, RFCOMM, BNEP, CMTP and HIDP kernel implementations, HCI UART, USB, PCMCIA and virtual device drivers, general Bluetooth and SDP libraries and daemons, configuration and testing utilities and protocol decoding and analysis tools. From all these modules, the most useful in our test-bed were the BNEP implementation, which allowed the creation of piconets, and the set of tools for configuration and analysis of the protocol.

The tools that were used in our prototype are: `pand` and `hciconfig`. `Hciconfig` is similar to “`ifconfig`”, the traditional command-line tool for setting up network interfaces on Linux, but works for Bluetooth interfaces, which follow the prefix “`hciX`”, where `X` is the interface number starting at zero. The `pand` (short for “Personal Area Networking daemon”) tool is explained below.

The PAN daemon

The PAN daemon[47] allows BNEP connections using Bluetooth. This daemon creates a virtual `bnepX` interface which is directly related to an `hciX` Bluetooth interface, which is the physical interface name. This new `bnepX` interface encapsulates Ethernet frames in a Logical Link Control and Adaptation Protocol (L2CAP) session. PAN is the Bluetooth profile using BNEP and defining how to do networking. It includes the definition of the Service Discovery Protocol (SDP) attributes and three node roles. In a typical Ad-Hoc scenario, the server will have the role GN (Group Ad-Hoc Network), and up to 7 clients with role PANU (Pan User) can be connected to it, which is the actual limit of a Bluetooth piconet[23].

⁴ <http://www.bluez.org>

4.3.3 802.11 Layer 2 communication: Raw Ethernet Sockets

During the Bootstrapping phase, explained in section 3.2.2.1, new nodes willing to participate in a 3D Routing Network should send a JOIN message in order to communicate with an in-range participating node. It's important to notice that participating nodes exchange messages and information using layer 3 and above protocols, such as IP, TCP, UDP etc. New nodes are not yet with an IP address, but need to communicate with the participant node request its entrance in the network. This is possible through a mechanism known as Raw Ethernet Sockets.

Raw Ethernet Sockets⁵ allows communication at layer 2, without the use the TCP/IP stack, as it is usually done. This possibility is ideal for the bootstrapping phase, since nodes are willing to participate in the network, and one of the steps is to acquire an IP address. On the other hand, they do not have an IP address, but are requesting it. The JOIN message of the protocol was implemented with raw Ethernet sockets, which use Ethernet frames for transmission and MAC addresses for identification of nodes.

4.3.4 Bluetooth Layer 2 communication: L2CAP Sockets

The Logical Link Control and Adaption Protocol (L2CAP) [52] is a packet-based protocol that can be configured with varying levels of reliability. The default maximum packet size is 672 bytes, but this can be negotiated up to 65,535 bytes after a connection is established.

L2CAP can be compared with UDP, which is a best-effort packet-based protocol, but there are enough differences that the use cases for L2CAP are much broader than the use cases for UDP. Both are packet-based protocols, but L2CAP enforces delivery order. In UDP, it is possible for a computer to transmit two packets and have the second packet arrive before the first, due to quirks in Internet routing. This never happens in L2CAP, and packets are always delivered in the order they were sent. Additionally, UDP is restricted to best-effort guarantees, whereas L2CAP can be configured for varying levels of reliability. These differences mean that L2CAP can often be used where UDP would be used, for simple best-effort packet-based communication, but it can also be used for applications where UDP would not be suitable.

⁵ http://aschauflandshut.org/fh/linux/udp_vs_raw/

Another notable difference between UDP and L2CAP is that L2CAP operates at layer 2, and thus uses MAC addresses for identifying nodes. Again, this is an ideal match for the bootstrapping phase, and it is the resource used in the node bootstrapping process for Bluetooth nodes.

4.4 3D Routing Implementation

3D Routing Implementation can be divided into a set of tools corresponding to the phases, messages and the NIB representation. As such, four tools were developed, a core daemon and three graphical user interfaces (GUI), covered in the next subsections. The relation between the tools can be seen in **Figure 22**.

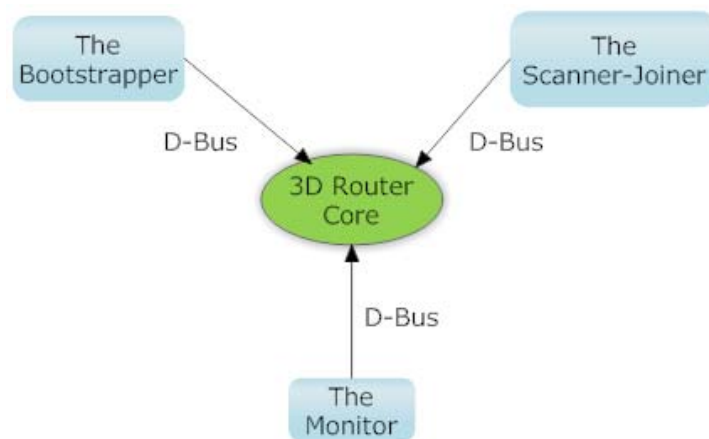


Figure 22 3D Routing Tools Overview

4.4.1 3D Router Core

The 3D Router Core is the main 3D Routing application and it is responsible for implementing the messages, phases and the NIB. In addition, this tool offers inter-process communication to all other tools, through the use of the D-Bus API, making possible for the interfaces to access information acquired by the 3D Router daemon, such as the NIB of other nodes in the network.

All nodes are required to run the 3D Router Core in order to be able to join any 3D Routing network.

4.4.2 The Bootstrapper

This tool is responsible for performing the network bootstrapping process, discussed in section 3.2.2.1. The tool is illustrated in **Figure 23**.

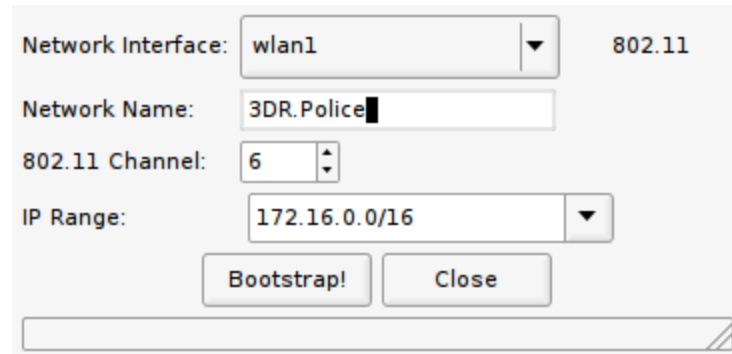


Figure 23 The 3D Routing Network Bootstrapper

The input necessary for the bootstrapping of a network are:

- **802.11:** Network Interface (list taken from the operating system), Network ID, Channel and IP range of the network;
- **Bluetooth:** Network Interface (list taken from the operating system), Network ID, and IP range of the network.

The difference basically is the channel selection. As explained in section 2.1.2, while in 802.11 the channel must previously be set, in Bluetooth this is an automatic process.

4.4.3 The Scanner-Joiner

This is the tool responsible for providing an interface for new nodes to join into the network. As such, the interface provides the scanning of networking surrounding the node, and an option for connecting to the selected network, as shown in **Figure 24**.

When clicking the “Connect” button, the interface communicates via D-Bus with the 3D Router Core in order to send a JOIN message. Then, the 3D Router waits for a response from a participant node. When the reply is received, it assigns itself the IP address received and sends the results of the negotiation to the Scanner-Joiner again, which updates the interface, showing a status message to the user (success or failure).

To put things into perspective, this is the end of the node bootstrapping phase, and the beginning the context awareness and route construction phase. From now on, the node is

participating into the network. First it sends and receives HELLO messages from neighbor nodes. Then, it builds its own NIB and starts flooding INFORM messages into the network, as well as receiving INFORM messages from other nodes in the network. This way, it is able to populate its NIB repository, and build routes by using the algorithm shown in section 3.2.2.2.

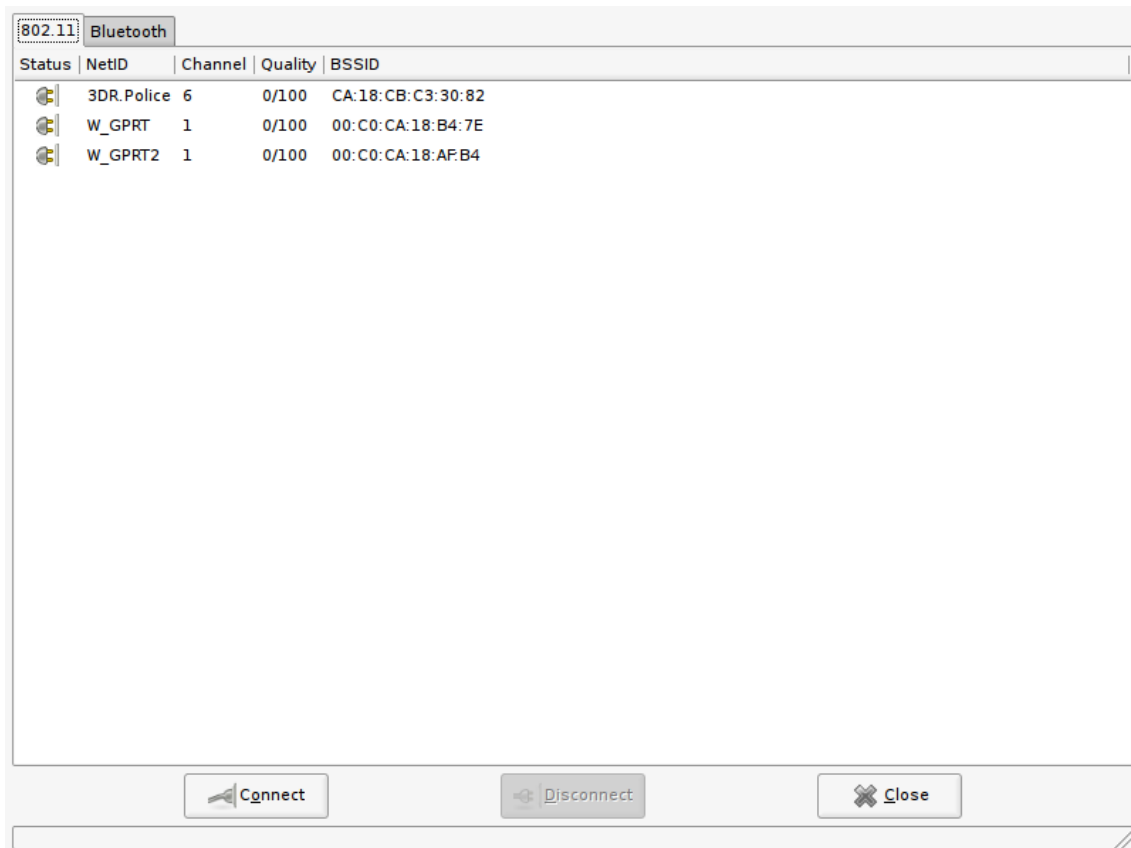


Figure 24 The Scanner-Joiner

4.4.4 The Monitor

The Monitor is able to show all information being acquired from the network, such as the NIB repository, routes built and services available on the network. It's an important tool for monitoring and debugging the network.

In **Figure 25**, it is shown the main screen of The Monitor, listing the NIB of a node whose ID is 00:0C:55:FB:C4:30, under the name Police-05, and another neighbor node, Police-07. Notice that for every node, it is listed its Profile, Capabilities and Context.

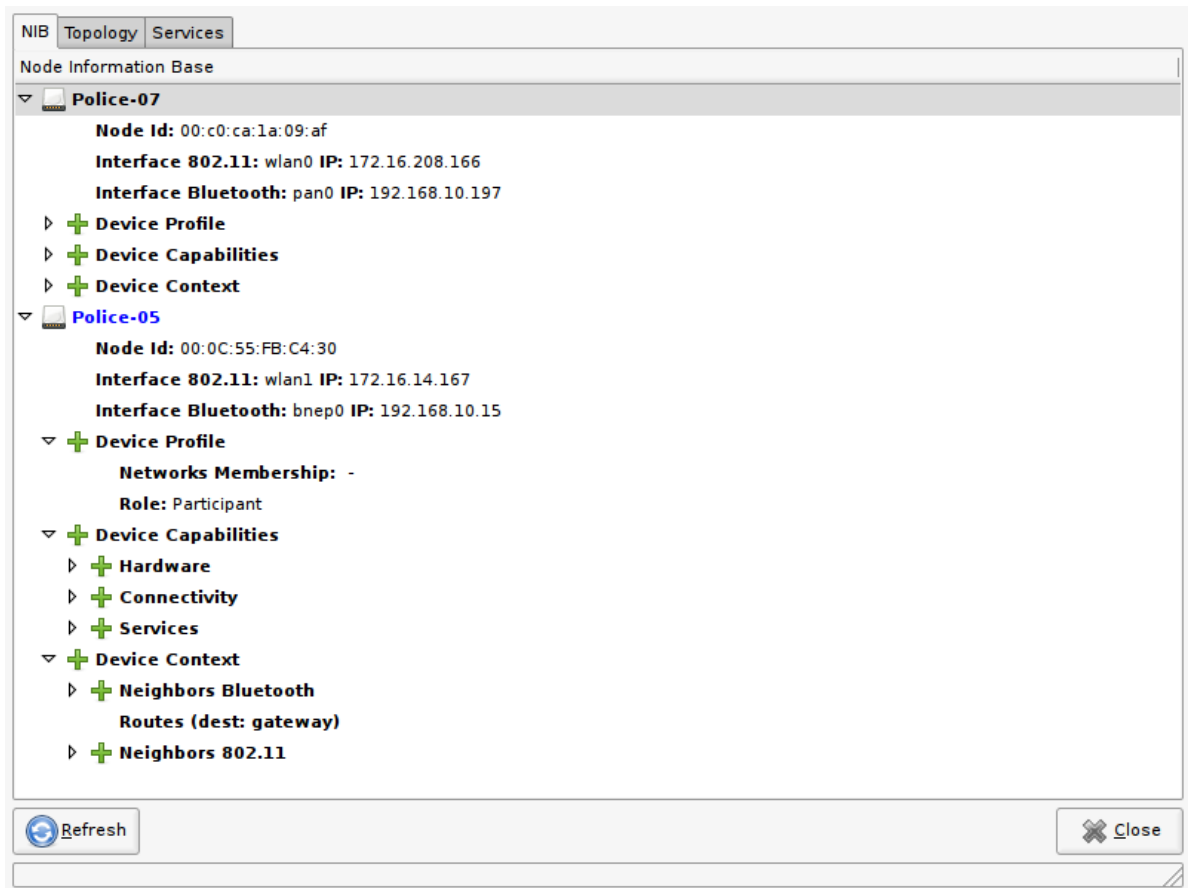


Figure 25 The Monitor

4.5 Lessons learned from implementing the 3D Routing test-bed

Building a real test-bed of mobile devices in order to study the behavior of heterogeneous ad hoc networks is far uncommon than traditional research in ad hoc networking, based on simulations. Today, there's an effort to experiment with off-the-shelf equipment. Most of this effort comes from the Mesh Community Networks [35], where ordinary routers, such as WRT54GL from Linksys [55] are used for building a wireless infrastructure capable of spanning several kilometers, offering Internet access to everyone who is on the Mesh. Such networks have been a great source of research for ad hoc routing protocols, as it is a real scenario under real-world conditions.

We have experienced a number of problems during the assembly and implementation of our test-bed. In this section, we expose some of several important lessons have emerged from experience to deal with a real test-bed. First, we describe our initial attempts at reusing the Ana4 Framework[3], as its source code is available. Next, we describe the phenomenon of BSSID cell splits experienced when forming 802.11 ad hoc networks. Finally, we discuss on Bluetooth scatternets and their practical inexistence today.

4.5.1 Experimenting with existing underlay implementations

The very first attempt to implement 3D Routing took into consideration the existing solutions that had an implementation available, notably the Ana4 Framework [3]. When dealing at the underlay level, in practice it means to operate at the kernel level, which is very bug prone and much harder to debug. Ana4 was developed during a PhD thesis, and since then its development has been stopped and it never reached any acceptable state of practice nor has established an active user base. In addition, the only technologies supported in the latest version available were 802.11 and Ethernet, which are fairly compatible.

One initial test was made with adding Bluetooth support to Ana4, but the first results proved that this would be a very difficult test. BNEP[51] support the sending of Ethernet frames over the Bluetooth L2CAP layer, but in general those frames from the IP layer, whose protocol type is 0x800. Operating at the underlay level implies the creation of raw Ethernet frames, such as the ones used in the JOIN messages, with a different type other than 0x800. Apparently, neither in Ana4 nor in a simple Bluetooth piconet we were able to send those raw frames: they seem to be filtered and never leave the network interfaces towards the destination. We experienced this problem at first when trying to adapt Ana4's implementation, but then reproduced it during our tests without the Ana4 framework. In fact, the use of L2CAP sockets was also a solution for this problem. We contacted again the BlueZ developers on this issue, but they never replied. The main suspect comes from the BNEP specification [51]. BNEP implements protocol filters, for which only 0x800 (coming from the IP layer) frames are allowed. Finding a way to disable such filters may solve the problem.

Other underlay implementations available today is in a very initial stage, implementing a protocol named "Better Approach To Mobile Ad-hoc Networks" (B.A.T.M.A.N)[36]. As this is a very active project, with a very active user base, the use of such underlay substrate may be of interest, especially if it is made to work on Bluetooth piconets. As a consequence, 3D Routing could be run in such underlay substrate and take advantage of the underlay level, such as transparent handover and better support for mobility in general. In addition, adapting the actual 3D Routing implementation, which uses raw Ethernet sockets, to such underlay, is fairly simple.

4.5.2 The 802.11 Ad Hoc Mode BSSID Cell Splitting Phenomenon

The ad-hoc mode has been widely neglected by chipset manufacturers and driver developers. Implementing ad-hoc mode is much more complicated than implementing the managed mode (access-point client). Ad-hoc mode means the capability that everyone in range can talk to everyone (multipoint to multipoint) while access point mode means everyone can only talk to the access point - only the access-point can talk to everyone (point to multipoint).

As mentioned before, driver support for the ad-hoc mode presents multiple problems. Generally, our experiments have shown that it does not work at all or is unreliable. Furthermore, even if most nodes of the network are operating properly according to the specification of the 802.11 standard, it takes a single problematic device to destabilize the whole network. We faced such instability when turning on the N810 devices in our network.

A problematic device can do wrong actions which confuse other nodes in the network, such as:

- Send wrong time stamps after merging to the Cell-ID;
- Not merging at all.
- Send the right time stamp but with the wrong Cell-ID.

The results of false timestamps are IBSS-ID cell-splits, WiFi card lock-ups, and intermittent operation. That is the situation when beacons with different timestamps and the same Cell-ID are in the air, which results in timestamps that are jumping forth and back. Or there are different Cell-IDs that carry the same time stamp. Or the time stamp tells the card that the Cell is now running since 500000 years – which results in an overflow in the counter and the MAC timer starts from zero. The consequences are clear and multiple:

- Multiple little ad-hoc cells that don't talk to each other instead of a single one. Very common in our test-bed;
- Intermittent connectivity between nodes.

Taking a closer look at other researchers who are doing real mesh test-beds, there are mesh community networks such as Freifunk[35] and MIT RoofNet[39]. They present some workarounds for this issue. The most widely used is to set the BSSID statically, and disable the algorithm used for automatic BSSID selection. Unfortunately, this is only available for some drivers, such as Atheros and Ralink.

4.5.3 Bluetooth Scatternets: a myth?

As seen in chapter two, Bluetooth ad hoc networks may be formed as a piconet, consisting of one master node and up to seven slave nodes. In addition, a scatternet may exist by the interconnection of several piconets. There are a number of papers on scatternet formation protocols [60], but we were unable to find a single, functional implementation of a real-world scatternet implementation. Furthermore, the Linux BlueZ implementation does not mention any support for any of the existing scatternet protocols in the literature.

In fact, the lack of documentation in the current implementations was a major problem during our research. In [47], it is apparently the only document available describing how to form a piconet. Yet, the document seems outdated, and newer software may be used for piconets, but there's no documentation available. The authors of the BlueZ stack [ref] were contacted several times in order to clarify this, but unfortunately we never got any answers. In addition, it is known that up to version 2.0 of the protocol, Bluetooth is not supporting scatternets formation. This may justify this limitation. It is known from [52] that version 2.1 will support automatic topology management, allowing the existence of real scatternets.

The bottom-line is that even if we are using free and open source software, lack of documentation may be a barrier, or at least add a steep learning curve, since it will be required to dig into the code and find out how things work. This lack of documentation is especially valid for tiny details we need to work with – drivers, kernel support, specific software unavailable for the embedded versions, etc.

5 Evaluation

“It is a capital mistake to theorize before one has data.”

Sir Arthur Conan Doyle

The previous chapter presented the implementation of the 3D Routing Protocol prototype and the test-bed used for evaluating it. In this chapter, we evaluate some of the concepts of 3D Routing Protocol using a real prototype and extract some useful metrics for measurement.

This chapter is organized as follows. Section 5.1 justifies the use of a real test-bed instead of simulations. In Section 5.2 explains how we instrumented our implementation in order to extract the metrics of interest. In addition, we explain the evaluation methodology used for collecting and processing the results for statistical analysis. Section 5.3 evaluates 3D Routing in some scenarios, collecting metrics of interest. Finally, section 5.4 covers some discussion about the results of the experiments.

5.1 Why use a test-bed?

Most of ad hoc networking research has been based on simulations of ideal situations, ideal obstacles, propagation models, etc. Over time, the use of such ideal scenarios have been proven false in practice, and research towards more reliable results[49] highlighted the importance of wireless test-bed facilities for the research community in view of the limitations of available simulation methodologies. It was concluded that these test-beds enable a critical stage in the process of understanding the relationship between research and reality and help clarify which simplifications in theoretical work are valid.

The criticisms about the credibility of simulations in ad hoc networking research has taken a number of papers [25][32][33]. For example, in [25] the authors perform a survey of the 2000-2005 proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc). They conclude that the four areas of credibility considered (repeatable, unbiased, rigorous and statistically sound), they found out less than 15% of the published MobiHoc papers are repeatable. The reason for this is the difficulty, or even impossibility to repeat a simulation study when the version of a publicly available simulator is unknown, and only seven of the 58 MobiHoc simulation papers that use a public simulator (12.1%) mention the simulator version used.

Furthermore, it seems difficult, or impossible, as well, to repeat a simulation study when the simulator is self-developed and the code is unavailable. In addition, only eight of the 114 simulation papers (7.0%) addressed initialization bias and none of the 84 simulation papers addressed random number generator issues. Thus, their concern is that over 90% of the MobiHoc published simulation results may include bias. With regard to compromising statistical soundness, 70 of the 109 MANET protocol simulations papers (64.2%) did not identify the number of simulation iterations used, and 98 of the 112 papers that used plots to present simulation results (87.5%) did not include confidence intervals. Hence, only approximately 12% of the MobiHoc simulation results appear to be based on sound statistical techniques.

Such results are a strong motivation for performing practical, real-life experimentation with ad hoc networks. In chapter 4, it was explained in detail how we built the test-bed used to evaluate the 3D Routing protocol. Next, it is explained how the software was instrumented in order to extract useful metrics for evaluation.

5.2 Evaluation Methodology

The evaluation methodology consists on instrumenting part of the 3D Router code, with the aim of measuring the time of a particular event. In addition, with the data we extracted from the results, the Bootstrap resampling method [49] is applied in order to have a better estimate of the confidence intervals for the mean of a particular metric.

The bootstrapping method is a statistical method for estimating the sampling distribution of an estimator by sampling with replacement from the original sample, most often with the purpose of deriving robust estimates of standard errors and confidence intervals of a population parameter like a mean, median, proportion, odds ratio, correlation coefficient or regression coefficient. It may also be used for constructing hypothesis tests. It is often used as a robust alternative to inference based on parametric assumptions when those assumptions are in doubt, or where parametric inference is impossible or requires very complicated formulas for the calculation of standard errors.

In our case, some of the measurement involves stochastic factors such as radio waves interference. The application of the bootstrapping method brings more reliable estimates for confidence intervals of the mean for the values collected.

5.3 Instrumentation of the 3D Router Core

The source code for the 3D Router Core was adapted in order to collect information from the mechanisms being evaluated. A logging subsystem was introduced in order to capture the output generated by the tests, which was then used to perform the statistical analysis.

As an example, the following code lists part of the code instrumented:

```
logger.info("Joiner 802.11 starting")

t0 = time.time()

utils.utils.setIface(IFACE_WLAN,

    "Ad-Hoc",

    get_channel(selected_row["channel"]),

    selected_row["netid"],

    "0.0.0.0")

logger.info("Iface set up")

tmp = frame_handler.FrameHandler(IFACE_WLAN, None)

while self.is_running:

    print "Send JOIN message, waiting IP..."

    tmp_frame = tmp.send_recv_frame("JOIN")

    new_ip = tmp_frame["data"]

    if match(r"^\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}$",new_ip):

        print "type of new_ip =>", type(new_ip)

        print "Receive ip (%s) from %r" % (new_ip, tmp_frame["src_mac"])

        tmp.stop()

        del tmp

        break

    else:

        print "Discarted invalid information sent from %r" %

tmp_frame["src_mac"]

logger.info("Ip returned")

utils.utils.setIface(IFACE_WLAN,

    "Ad-Hoc",

    get_channel(selected_row["channel"]),

    selected_row["netid"],

    str(new_ip))

logger.info("Iface set up again")

status =

ROUTER3D_SERVICE_INTERFACE.alien_bootstrap(selected_row["netid"],

    IFACE_WLAN,

    str(new_ip),

    selected_row["channel"],
```

```

51423,
5,
0)

t1 = time.time()

logger.info("Joiner 802.11 finished")

logger.info("Joiner 802.11 time = %7.5f" % (t1 - t0))

connected = not bool(status)

```

Figure 26 Code fragment of the 3D Router Code instrumentation

A complete version of the code is attached in Appendix B.

5.4 Evaluations

In the next subsections we explore scenarios where the tools were used during the tests. In fact, a separate module was implemented with slight modifications to the original code, with the aim of repeatability of the experiments. The following metrics were collected in a couple of scenarios, explained in each subsection:

- **Network Bootstrap Time:** This is the amount of time it takes for a single node, the first node in the network, to create it. The implementation followed the steps described in section 3.2.2.1. We measured each technology individually, and then compared the results;
- **Node Bootstrapping Time:** This is the time it takes for a single node to join a 3D Routing network, from the initial JOIN request to the receiving of an IP address. Again, we measured each technology separately, and then compared the results;
- **Round Trip-Time(RTT):** The RTT was taken for a scenario involving both technologies, with packets passing by an intermediate gateway node;
- **Throughput and Jitter:** Within the same scenario for RTT measurement, we run the *iperf* [53] traffic simulation application to check the throughput and jitter in such a heterogeneous network scenario.

5.4.1 Scenario 1: Network Bootstrap time

The following scenario illustrates the process of network bootstrapping we are evaluating:



Figure 27 Initial scenario: network bootstrapping

A single node starts the network bootstrapping process, using The Bootstrapper tool. Now we measure how long it takes for this node to bootstrap a Bluetooth and an 802.11 network. In this experiment, we collected 50 network bootstrapping iterations repeatedly, for each interface. Then, we resampled it 1000 times and calculated the mean value within a 95% confidence interval. This process is shown graphically as a time series plot containing the network bootstrap time for every technology, during the course of 50 iterations, and a statistical summary of the resample. Then, we compare the results of both technologies in one time series plot.

802.11 Network Bootstrapping Time

Next, we present the measurement of the bootstrapping of a 3D Routing network via an 802.11 network interface. Then, a statistical analysis of the resample is shown, where the mean is calculated and confidence intervals presented.

- **Average 802.11 Network Bootstrap time, with 95% confidence interval**

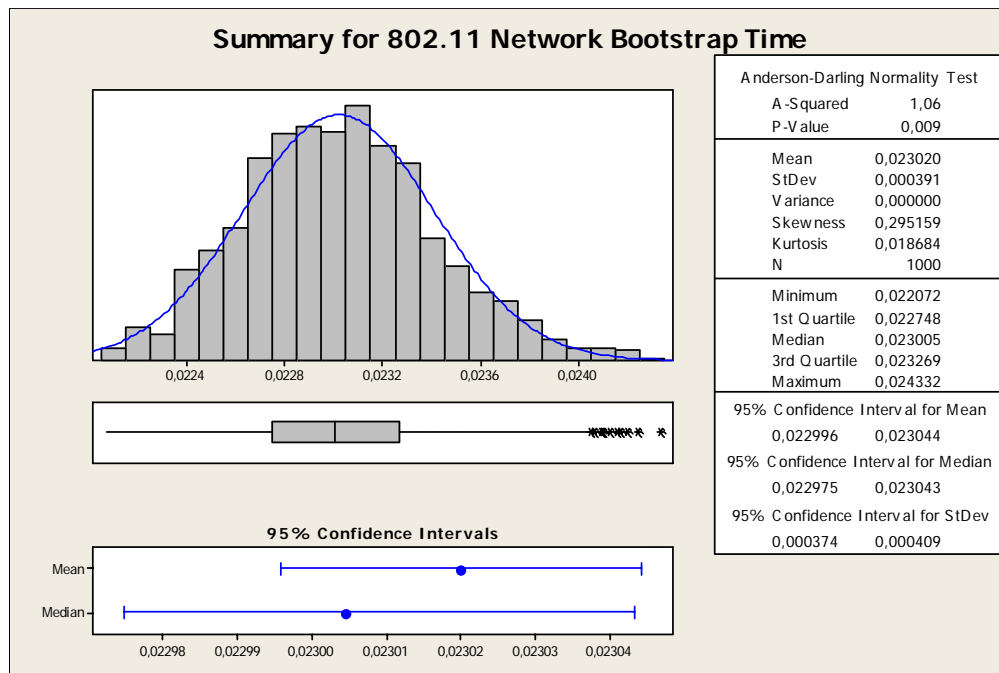


Figure 28 Statistical summary of 802.11 Network Bootstrapping time

The network bootstrapping process is mostly a local process, where the steps involved were described in the network bootstrapping steps mentioned in section 3.2.2.1. The estimated value for the mean can be seen in the summary above as being in the interval $[0,022, 0,023]$ seconds. This is a very small time taken, and while there's no former study comparing the time taken to execute the same steps manually, it's fairly intuitive to conclude that such manual process would not take a similar time interval.

Bluetooth Network Bootstrapping

- Average Bluetooth Network Bootstrap time, with 95% confidence interval

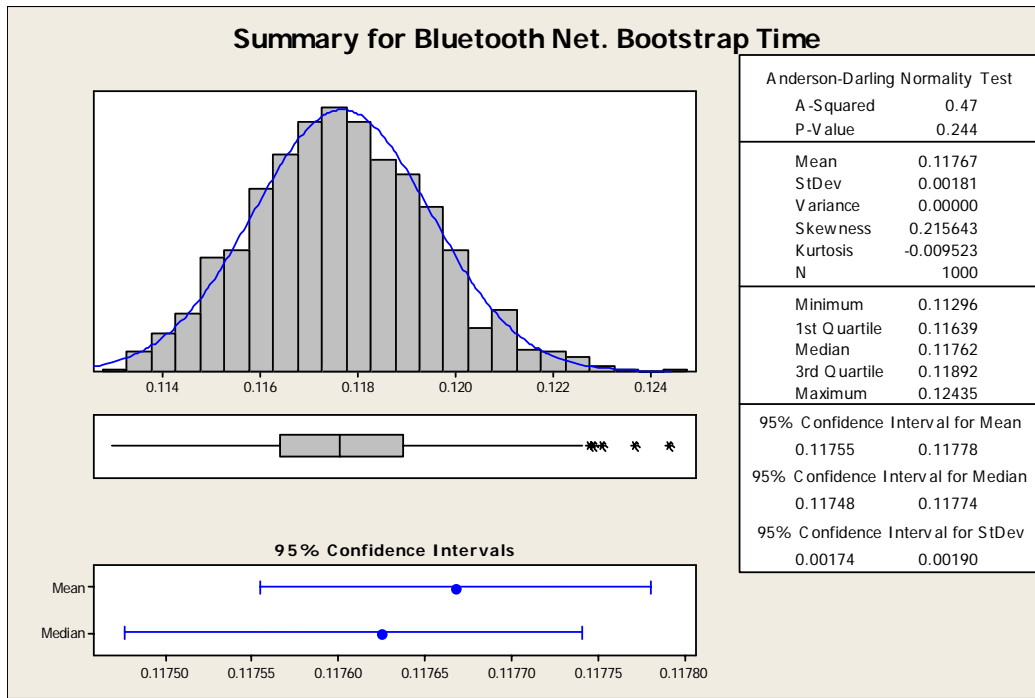


Figure 29 Statistical summary of the Bluetooth Network Bootstrapping time

As for Bluetooth network bootstrapping, the mean time is on the interval of [0.1175, 0.1177] seconds. The reason for taking more time than the 802.11 network bootstrap time is explained by the technology-dependent mechanisms of Bluetooth network bootstrap node. As seen in section 2.1.2, the network bootstrap node is the GN node, and as such must run the bridging software to forward packets for other PANUs (participant nodes). This way, it takes extra time to setup this bridge, taking more steps not present in the 802.11 case. Yet, this is still very fast if compared to the time taken to proceed with the manual steps.

Comparing Bluetooth and 802.11 Network Bootstrapping Time

Finally, the time series below show a comparison between Bluetooth and 802.11 network bootstrap time, considering the original sample.

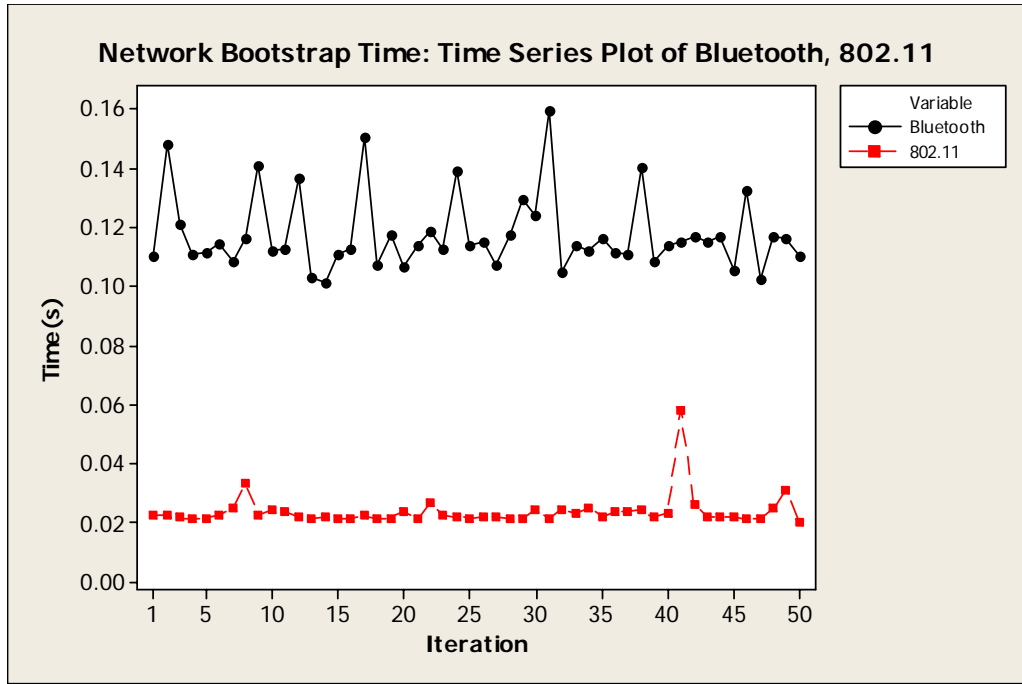


Figure 30 Bluetooth versus 802.11 Network Bootstrap time

The results show that while Bluetooth network bootstrapping takes more time, due to the overheads associated to the technology-dependent mechanisms, both network bootstrapping processes perform under a very acceptable time. It's still possible to tune such processes by accessing the operating APIs directly from our code, instead of the installed command-line tools, eliminating overheads associated to the firing of new process in the operating system.

5.4.2 Scenario 2: Node Bootstrap time

Now, it is evaluated the scenario where a new 802.11 and a Bluetooth node joins the 3D routing network. Again, we present the average time taken to accomplish this process, as well as present a time series plot of both technologies, comparing their values collected along the 50 samples iterations.

802.11 Node Bootstrapping

The scenario in **Figure 31** is fairly simple: New node “A” hears the announcement of the “3DR.Police” using The Scanner tool. Next, it tries to connect to that network. Time is measured for each attempt and successful negotiation.

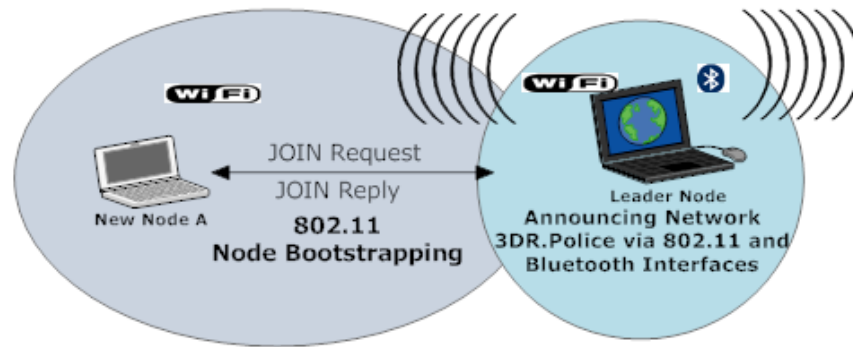


Figure 31 802.11 Node Bootstrapping Scenario

- Average 802.11 Node Bootstrap time, with 95% confidence interval

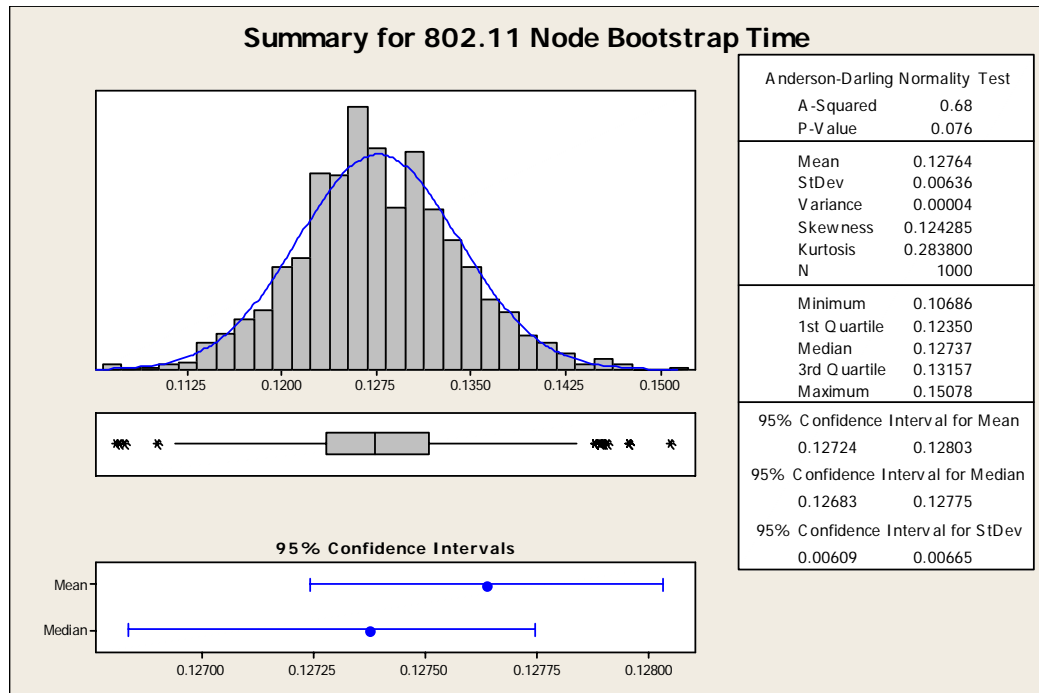


Figure 32 Statistical summary of the 802.11 Node Bootstrapping time

The summary above presents the mean value in the range [0.127, 0.128] seconds. Now, this time is taken considering the sending of the JOIN message in broadcast, as well as the receiving of the unicast JOIN reply message. Although the negotiation itself involves some overhead due to the radio medium, the average time taken to accomplish this is very optimal. Also, it's important to notice that JOIN requests and replies use raw ethernet sockets, as seen in section 4.3.3, which do not contain the overhead associated with other higher-layer protocols, such as IP and UDP, thus optimizing the negotiation.

Bluetooth Node Bootstrapping

Next, the same operation is repeated for the Bluetooth technology, illustrated in **Figure 33**. A new node B hears the network announcement via its Bluetooth interface and sends a join message to negotiate its entrance into the network. Notice that now the network is already formed with two nodes, the Leader node and Participant node A.

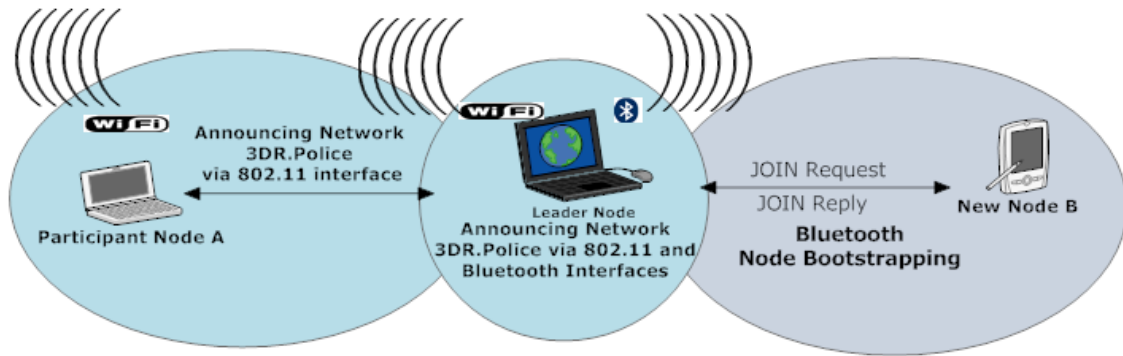


Figure 33 Bluetooth Node Bootstrapping Scenario

- **Average Bluetooth Node Bootstrap time, with 95% confidence interval**

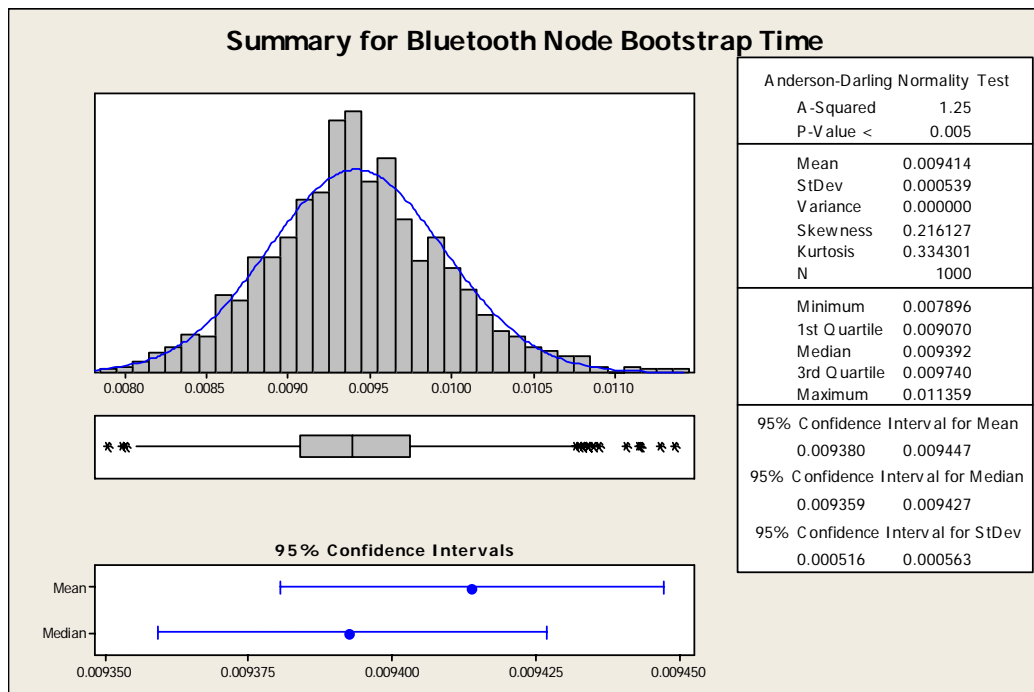


Figure 34 Statistical summary of the Bluetooth Node Bootstrapping time

The mean value for the Bluetooth node bootstrapping can be seen in the interval [0.0093, 0.0094]. This is far faster than the 802.11 node bootstrapping phase, and the cause for this is that JOIN requests for the Bluetooth case is sent in unicast, instead of a broadcast. Since Bluetooth is connection-oriented, the communication to neighbor nodes, such as the Leader node and the new node B is done in unicast via L2CAP sockets communication.

Comparing Bluetooth and 802.11 Node Bootstrapping Time

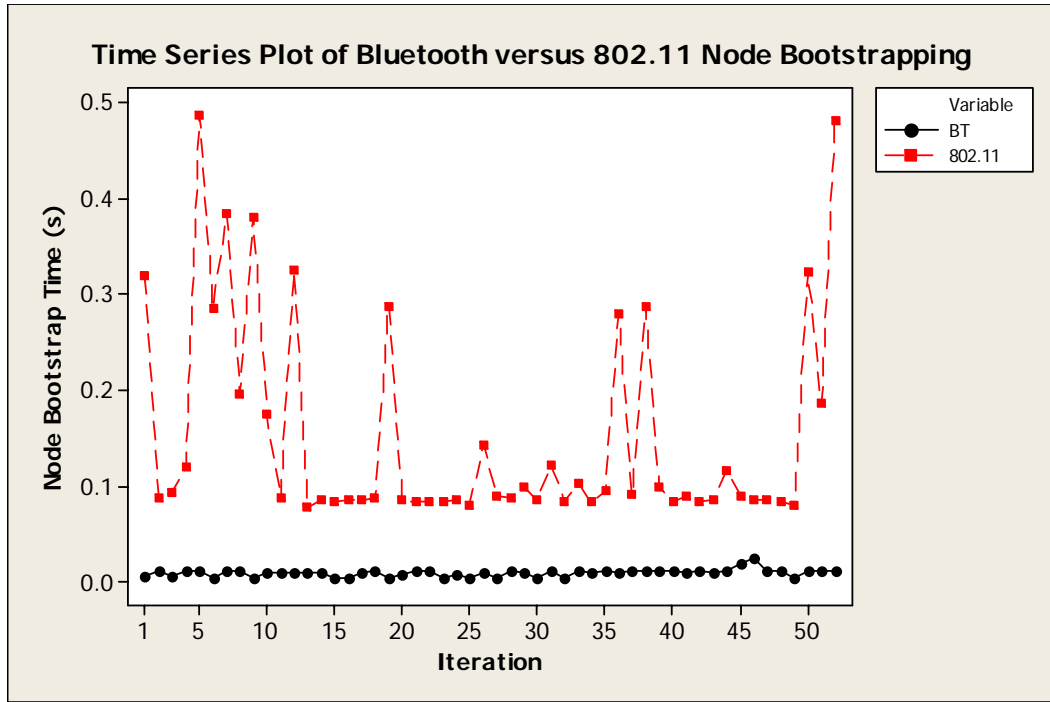


Figure 35 Bluetooth versus 802.11 Node Bootstrap time

Comparing both values it is noticed that the Bluetooth node bootstrapping process is taking a lot less time to conclude. Yet, the overall time taken for the negotiation is very acceptable. During our tests with the user interfaces of the Scanner tool, the overheads associated with this negotiation were unnoticeable to the user.

5.4.3 Scenario 3: Measuring RTT, Throughput and Jitter

After the network is formed and routes built, nodes may now communicate to each other. Now, we are interested in taking some measurements when Participant Nodes A (PNA), with 802.11 technology, communicates to Participant Node B (PNB), with Bluetooth technology. They are able to do so through the intermediate Leader node, acting as a gateway, as shown in **Figure 36**.

Two metrics of interest are measured in this experiment:

- **Round-Trip Time:** This is the time taken for a message to travel from PNA to PNB and back. This is measured in milliseconds;
- **Throughput:** This metric allows the estimation of how much data can be transferred from PNA to PNB in a given instant. This is measured in Kbits/second;

- **Jitter:** This metric represents the variation in the time between packets arriving, in milliseconds.

It's important to highlight that the nominal throughput for each of the technologies considered are 3 Mbits/s for Bluetooth 2.0+EDR (enhanced data-rate) and 11Mbits/second, for 802.11b/g in Ad-Hoc mode.

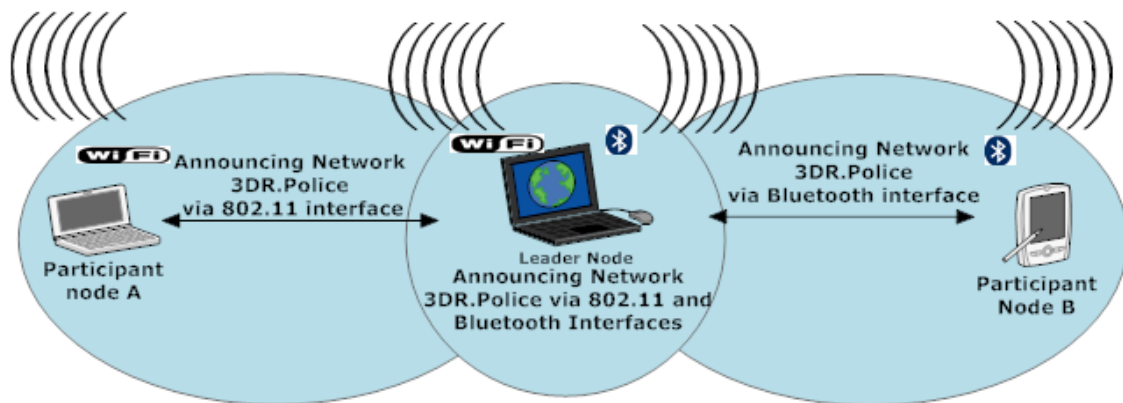


Figure 36 Final evaluation scenario

Round Trip Time (RTT)

The time series plot below draws the sample of 225 iterations of the “ping” program, running from PNA to PNB. In order to obtain a mean RTT from this sample, a resampling was also performed using the bootstrap method.

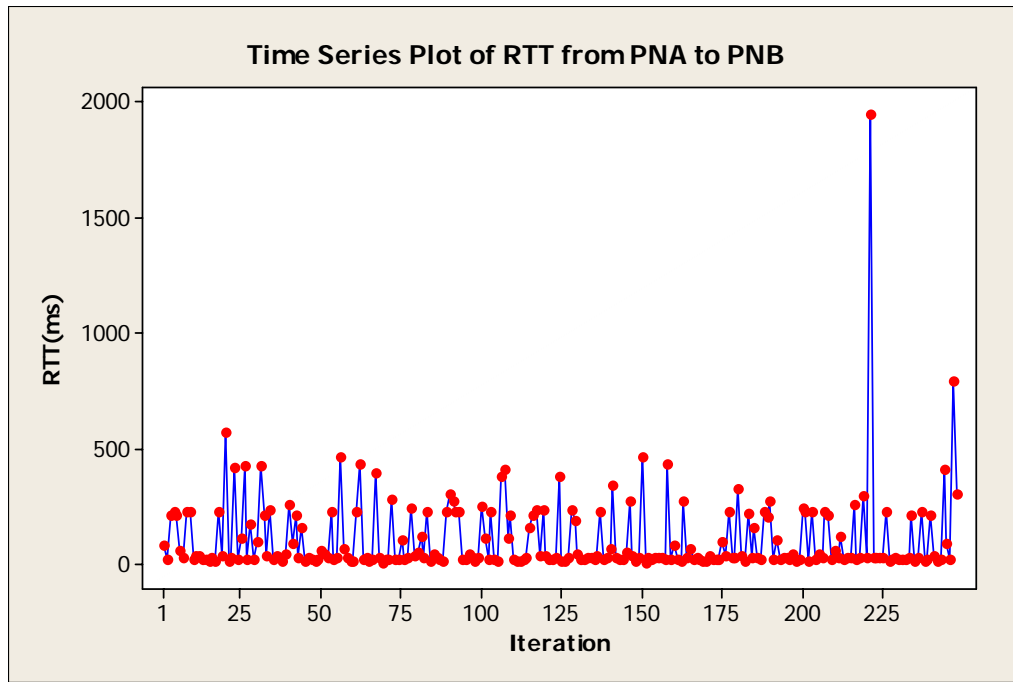


Figure 37 Time series plot of RTT from PNA to PNB (**Figure 36**)

- **Average RTT with 95% confidence interval**

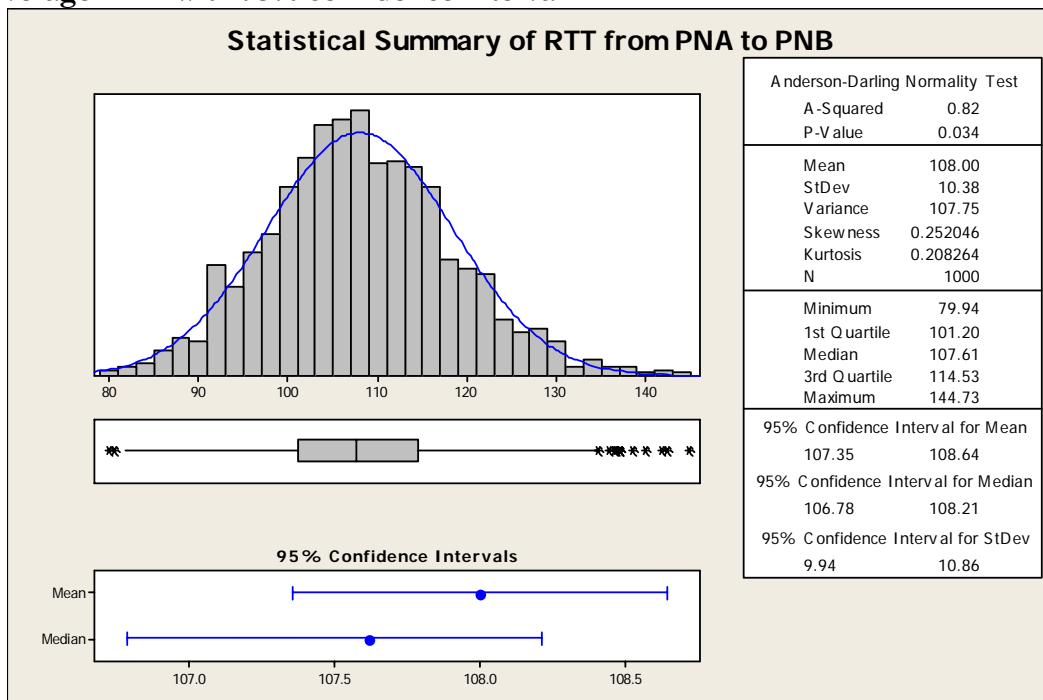


Figure 38 Statistical summary for the RTT from PNA to PNB

The summary in **Figure 38** shows that the interval for the mean RTT is [107.45, 108.64] milliseconds. Such values indicate, at first, that such network is able to support traffic which demands a low delay value, such as VoIP. On the other hand, to confirm this hypothesis it is necessary to evaluate the throughput and jitter experienced in this same scenario.

Throughput and Jitter

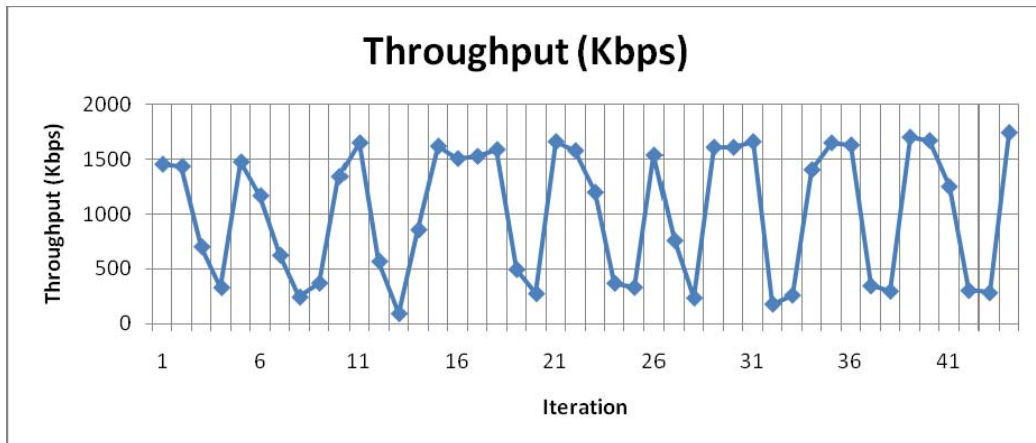


Figure 39 Throughput from PNA to PNB

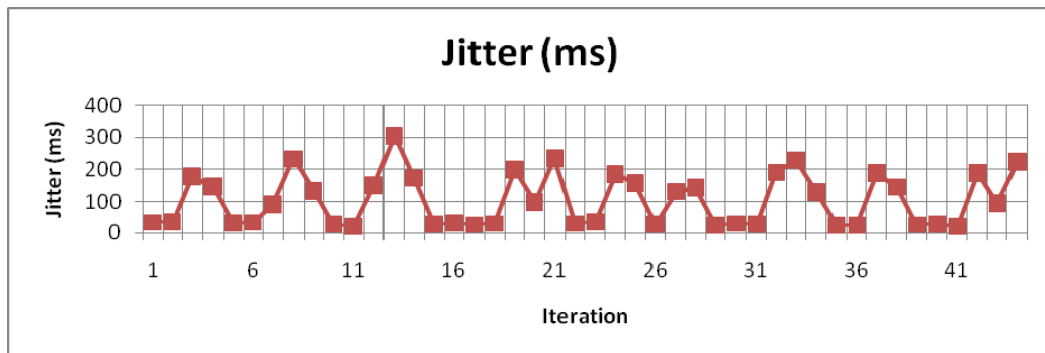


Figure 40 Jitter from PNA to PNB

The most surprising results come from the graphic above, generated by the *iperf* [53] tool, which is a client/server tools for measuring bandwidth and jitter of a given link, from PNA to PNB. The traffic simulated consisted of 250KB UDP packets, which is a kind of traffic similar to VoIP, running 46 times and measuring the data rate. The number of repetitions is necessary to extract a meaningful average of the metric being evaluated.

The jitter experienced was fairly high, varying from 21.24 ms to 304.77 ms. Bandwidth also decreased considerably, given the nominal capacities of both links (11Mb/s and 3Mb/s), the minimum and maximum experienced throughput was 92.6Kb/s and 1740.8Kb/s. Such variations may have several influencing factors:

- Interference between Bluetooth and 802.11, since they share the same spectrum; although Bluetooth employs FHSS (switching randomly over multiple channels) to avoid interference, this can not be guaranteed.

- Interface between the 3DR.Police 802.11 network and other WiFi networks around. This may be possible, but since we were unable to isolate the experiment in a spectrum-free environment, or at least use a spectrum analyzer, this remains to be confirmed.
- Control traffic from the protocol itself; although not intense, this traffic consumes bandwidth as well.
- Overhead generated by BNEP emulation. This is one more layer of overhead, and should be taken into consideration.

5.5 Discussion

The birth of a 3D Routing network begins at the bootstrapping phase, where the first node of the network decides to create it. The first scenario evaluated the time taken for a node to perform the network bootstrapping process, using each of the technologies considered. As seen in the results, the Bluetooth interface takes more time due to its technology-dependent mechanisms, requiring the network bootstrap node to run bridging software, which does not occur in the 802.11 case. Yet, both technologies can be even more optimized if a tighter integration to the operating system is made.

Afterwards, we measured the time taken for a node to be part of the 3D Routing network. The time taken by the 802.11 node was longer than the Bluetooth node, due to the nature of the transmission. While in Bluetooth a unicast JOIN message is sent, in 802.11 the same JOIN message is sent as a broadcast. This is the different nature of both technologies and is hardly changeable. Yet, both time frames are very acceptable.

Furthermore, after the network is formed and routes are built, we performed evaluations on the RTT, Throughput and Jitter of a transmission from one node with an 802.11 link to another with a Bluetooth link, passing by the intermediate node with both technologies. This has shown the performance of both technologies, which coexist and share the same spectrum, by transmitting traffic simulating a VoIP connection. The results weren't the best: with such traffic, there were considerable jitter and also throughput degradation over time. The causes for this can be multiple. As 802.11 and Bluetooth share the same spectrum, they are not free of interference, although Bluetooth's FHSS technology is always attempting to vary channels in order to avoid interferences. Also, there are other overheads in the experiment, such as the BNEP headers used on top of the Bluetooth stack for Ethernet emulation, and the background control traffic of the protocol itself. Yet, future work involving performance evaluation of the protocol may bring more insight on the behavior of such heterogeneous network.

Finally, through these tests we verified that the implementation prototype works in accordance with the specification. Although not all the messages and phases of the protocol were implemented, we could verify how the core messages, the NIB and the phases of the protocol work together.

6 Conclusions and Future Work

“Using no way as way, having no limitation as limitation.”

Bruce Lee

This work studied the conception and implementation of the 3D Routing Protocol, a protocol targeted towards the dynamic formation of heterogeneous ad hoc networks. While a recent research field, works involving heterogeneous ad hoc networks focus primarily on the routing protocols and architectures, ignoring important points such as the creation of a network, as well as the profiling of nodes in terms of their hardware and connectivity capabilities.

3D Routing defines a set of messages, phases and a local repository of information providing not only routes to nodes in the network, but awareness of the context of each other in the network. The three phases of the protocol aim at encompassing the process of creation of the network, the process of nodes building routes and populating context information about each other, and finally the process of organizing the nodes into well-defined roles.

The protocol was specified and implemented in a test-bed consisting of multiple mobile devices. The results of such implementation consisted in a series of tools and graphical user interfaces where the protocol functioning is encapsulated.

This work also evaluated the protocol running on a test-bed, taking measurements of some of the relevant metrics such as network and node bootstrap time, round-trip time and bandwidth of the heterogeneous network.

Thus, this work has shown that the 3D Routing Protocol design addresses fundamental issues of seamless formation of dynamic heterogeneous ad hoc networks, taking into consideration the process of creation of the network, context awareness of nodes and interconnection between heterogeneous networks.

6.1 Contributions

The main contributions of this work can be resumed as follows:

- A new a routing protocol operating over a heterogeneous ad hoc network consisting of Bluetooth and 802.11 nodes, with context awareness of nodes.

- The proposal a network and node bootstrapping mechanism. This mechanism was proposed and evaluated in this work. Its mechanisms and configurations were explained as well. 3D Routing uses this mechanisms for its own network, but it could be easily adapted to work with other protocols;
- Experimentation in a test-bed with real mobile devices. Such experience provided new insights on real-life problems not found on simulations.

6.2 Future work

Some interesting future works include:

- **Generalization of the Bootstrapping phase into a Bootstrapping Framework.** While 3D Routing operates with Bluetooth and 802.11 technologies, new technologies are emerging with time. In addition, 3D Routing could be made to operate in different operating systems, and a framework would establish an abstraction of such platform specific details, regarding hardware, software and networking technology tweaks.
- **Integration to the Internet.** While 3D Routing was evaluated in a local scenario, integration to the Internet is interesting, and with the actual codebase this integration would not take too long. On the other hand, the implications of such integration may open more possibilities, such as the interoperation of distant 3D Routing networks separated by the Internet.
- **Improve Context Awareness Heuristics.** Many heuristics could be applied once context awareness information is spread into the 3D Routing Network. For example, a route could be decided based on the physical proximity of a node, or the hardware capabilities of the intermediate nodes.
- **Policy-based management:** Maturing the Role Management Phase by employing policies considering the roles of nodes in the network, introduce other elements to let a higher level of organization take place.
- **Increase number of numbers, more performance evaluations:** As long as ad hoc drivers become more stable, an interesting study would consist of increasing the number of nodes and running several performance tests, such as the ones in chapter 5.
- **Secure Bootstrapping:** One simple idea would be to incorporate some form of distributed Public-Key cryptography into the bootstrapping process, making the process of authenticating a new node more effective.

7 References

- [1] IEEE standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. Technical report, 2007.
- [2] The linux ethernet bridge, 2008.
- [3] N. Boulicault, G. Chelius, and E. Fleury. Ana4: a 2.5 framework for deploying real multi-hop ad hoc and mesh networks. *Ad Hoc & Sensor Wireless Networks: an International Journal (AHSWN)*, to be published, 2005.
- [4] J. Broch, J. Broch, D. Maltz, and D. Johnson. Supporting hierarchy and heterogeneous interfaces in multi-hop wireless ad hoc networks. In D. Maltz, editor, *Proc. Fourth International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN '99)*, pages 370–375, 1999.
- [5] I. Chakeres, I. Chakeres, and E. Belding-Royer. Transparent influence of path selection in heterogeneous ad hoc networks. In E. Belding-Royer, editor, *Proc. 15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications PIMRC 2004*, volume 2, pages 885–889 Vol.2, 2004.
- [6] G. Chelius, G. Chelius, and E. Fleury. Ananas: a local area ad hoc network architectural scheme. In E. Fleury, editor, *Proc. 4th International Workshop on Mobile and Wireless Communications Network*, pages 130–134, 2002.
- [7] T. Clausen and P. Jacquet. *Optimized link state routing protocol (olsr)*, 2003.
- [8] M. Conti and S. Giordano. Multihop ad hoc networking: The reality. In *IEEE Communications Magazine*, 2007.
- [9] M. Conti and S. Giordano. Multihop ad hoc networking: The theory. In *IEEE Communications Magazine*, 2007.
- [10] G. Dtnr. Delay tolerant networking research group. <http://www.dtnrg.org/wiki>.
- [11] S. Farrell, V. Cahill, D. Geraghty, I. Humphreys, and P. McDonald. When tcp breaks: Delay- and disruption-tolerant networking. *Internet Computing, IEEE*, 10(4):72–78, 2006.
- [12] L. Gong. Jxta: a network programming environment. *Internet Computing, IEEE*, 5(3):88–95, 2001.
- [13] D. B. Johnson, D. A. Maltz, and Y. C. Hu. The dynamic source routing protocol for mobile ad hoc networks (dsr). Technical report, IETF MANET Working Group, February 2007.
- [14] C. Liu and J. Kaiser. A survey of mobile ad hoc network routing protocols. In *University of Ulm Technical Report Series*, 2005.
- [15] NS-2. Network simulator - <http://www.isi.edu/nsnam/ns/>, 2007.
- [16] C. Perkins, E. Royer, and S. Das. Rfc 3561 ad hoc on-demand distance vector (AODV) routing, 2003.
- [17] J. Puttonen and G. Fekete. Interface selection for multihomed mobile hosts. In *Proc. IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–6, Sept. 2006.
- [18] J. Reyes, E. Burgoa, C. Calafate, J.-C. Cano, and P. Manzoni. A manet autoconfiguration system based on bluetooth technology. In *Proc. 3rd International Symposium on Wireless Communication Systems ISWCS '06*, pages 674–678, 6–8 Sept. 2006.

- [19] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture, January 2001.
- [20] H. Safa, H. Safa, H. Artail, M. Karam, H. Ollaic, and R. Abdallah. Haodv: a new routing protocol to support interoperability in heterogeneous manet. In H. Artail, editor, Proc. IEEE/ACS International Conference on Computer Systems and Applications AICCSA '07, pages 893–900, 2007.
- [21] H. Safa, H. Safa, H. Artail, and R. Shibli. An abstract model for supporting interoperability in mobile ad-hoc networks. In H. Artail, editor, Proc. (WiMob'2006) Wireless and Mobile Computing, Networking and Communications IEEE International Conference on, pages 45–52, 2006.
- [22] K. Sethom, K. Sethom, M. Sabeur, B. Jouaber, H. Afifi, and D. Zeghlache. Distributed virtual network interfaces to support intra-pan and pan-to-infrastructure connectivity. In M. Sabeur, editor, Proc. IEEE Global Telecommunications Conference GLOBECOM '05, volume 6, pages 5 pp.–, 2005.
- [23] Bluetooth SIG, Specification of the Bluetooth Protocol (version 2.0), November 2004
- [24] M. Sowmia Devi and P. Agrawal. Dynamic interface selection in portable multi-interface terminals. In Proc. PORTABLE07 Portable Information Devices IEEE International Conference on, pages 1–5, 25–29 May 2007.
- [25] T. C. Stuart Kurkowski and M. Colagrosso. Manet simulation studies: The incredibles. In ACM SIGMOBILE Mobile Computing and Communications Review, pages 50–61, 2005.
- [26] P. Stuedi and G. Alonso. Transparent heterogeneous mobile ad hoc networks. In The Second Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous), pages 237–246, 17-21 July 2005.
- [27] J. Su, J. Scott, P. Hui, J. Crowcroft, E. de Lara, C. Diot, A. Goel, M. Lim, and E. Upton. Haggie: Seamless networking for mobile applications. In J. Krumm, G. D. Abowd, A. Seneviratne, and T. Strang, editors, Ubicomp, volume 4717 of Lecture Notes in Computer Science, pages 391–408. Springer, 2007.
- [28] V. Untz, M. Heusse, F. Rousseau, and A. Duda. On demand label switching for spontaneous edge networks. In FDNA '04: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture, pages 35–42, New York, NY, USA, 2004. ACM.
- [29] V. Untz, V. Untz, M. Heusse, F. Rousseau, and A. Duda. Lilith: an interconnection architecture based on label switching for spontaneous edge networks. In M. Heusse, editor, Proc. First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services MOBIQUITOUS 2004, pages 146–151, 2004.
- [30] M. Weiser. The computer for the twenty-first century. In Scientific American. 1991.
- [31] S. Yi, J. Meegan, and J. H. Kim. Network autoconfiguration for mobile ad hoc networks. In Proc. IEEE Military Communications Conference MILCOM 2007, pages 1–7, 29–31 Oct. 2007.
- [32] R. Todd and A. Yasinsac. “On the credibility of MANET simulations”, In IEEE Computer Society, pages 48 – 54, 2006.
- [33] D. Cavin, Y. Sasson, and A. Schiper. “On the accuracy of MANET simulators”, In Proceedings of the second ACM international workshop on Principles of mobile computing, pages 38 – 43, 2002.
- [34] Y. Zhang, J. Zuo and H. Hu. “Wireless Mesh Networking: Architectures, Protocols and Standards”. Auerbach Publications, 2006.

- [35] Freifunk - German Community Network Project. Available at: <http://www.freifunk.net>. (Last checked 10/01/09)
- [36] B.A.T.M.A.N. Project. Available at: <http://tools.ietf.org/id/draft-wunderlich-openmesh-manet-routing-00.txt>. (Last checked 10/01/09)
- [37] Wangi, N. I. C.; Prasad, R. V.; Jacobsson, M. & Niemegeers, I. "Address autoconfiguration in wireless ad hoc networks: protocols and techniques". *IEEE_M_WC*, 2008, 15, 70-80.
- [38] IEEE OUI and Company_id Assignments. Available at: <http://standards.ieee.org/regauth/oui/index.shtml>. (Last checked 21/01/09)
- [39] MIT RoofNet Project. Available at: <http://pdos.csail.mit.edu/roofnet/>. (Last checked 21/01/09)
- [40] Alfa Networks AWUS036H USB adapter. Available at: http://www.oiw.com.br/produtos/alfa/descricao_awus036h.shtml. (Last checked 16/02/09)
- [41] Edimax EW-7318USg USB adapter. Available at: http://www.edimax.com/en/produce_detail.php?pd_id=8&pl1_id=1&pl2_id=44. (Last checked 16/02/09)
- [42] OpenMoko Smartphone. Available at: <http://openmoko.com/>. (Last checked 16/02/09)
- [43] Nokia N810 Internet Tablet. Available at: <http://www.nokia.com.br/A4886384>. (Last checked 16/02/09)
- [44] Atheros AR6001 internal adapter. Available at: <http://www.atheros.com/pt/AR6001GL.htm>. (Last checked 18/02/09)
- [45] Conexant's 802.11 b/g embedded chipset. Available at: <https://garage.maemo.org/projects/cx3110x/>. (Last checked 05/02/09)
- [46] Trendnet TBW-105UB Bluetooth USB dongle. Available at: http://trendnet.com/langpo/products/proddetail.asp?prod=165_TBW-105UB. (Last checked 05/02/09)
- [47] M. Schmidt, HowTo set up common PAN scenarios with BlueZ's integrated PAN support, <http://bluez.sourceforge.net/contrib/HOWTO-PAN>. (Last checked 05/02/09)
- [48] IEEE Workgroup 802.1D, IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges, June 2004
- [49] Efron, B.; Tibshirani, R.. An Introduction to the Bootstrap. Chapman & Hall/CRC, 1993.
- [50] NSF workshop on network research testbeds, October 2002. Available at: http://www-net.cs.umass.edu/testbed_workshop/ (Last checked 07/02/09)
- [51] Bluetooth SIG, Bluetooth Network Encapsulation Protocol (BNEP) Specification, v. 1.0, tech. specification, Feb. 2003.
- [52] Bluetooth SIG. BLUETOOTH SPECIFICATION Version 2.0 + EDR. November 2004
- [53] Iperf performance evaluation tool. Available at: <http://sourceforge.net/projects/iperf>. (Last checked 07/01/09)
- [54] I. Chlamtac, M. Conti, and J. Liu, "Mobile Ad Hoc Networking: Imperatives and Challenges," *Ad Hoc Networks J.*, vol. 1, no. 1, Jan.–Mar., 2003.
- [55] WRT54GL Routers. Available at: http://en.wikipedia.org/wiki/Linksys_WRT54G_series (Last checked 08/01/09)
- [56] BackTrack Linux Distribution. Available at: <http://en.wikipedia.org/wiki/BackTrack>. (Last checked 09/01/09)

- [57] OpenMoko Neo FreeRunner. Available at: http://wiki.openmoko.org/wiki/Neo_FreeRunner. (Last checked 08/01/09)
- [58] Maemo Linux. Available at: <http://maemo.org/>. (Last checked 08/01/09)
- [59] S. Toner and D. O'Mahony. Self-organising node address management in ad-hoc networks. In Proc. PWC, Lecture Notes in Computer Science 2775, 2003.
- [60] R.M. Whitaker, L. Hodge, I. Chlamtac. "Bluetooth scatternet formation: a survey, Ad Hoc Networks", 2004.
- [61] D. Porcino and W. Hirt, "Ultra-wideband radio technology: Potential and challenges ahead," *IEEE Commun. Mag.*, vol. 41, pp. 66–74, Jul. 2003.
- [62] P. Kinney. ZigBee Technology: Wireless control that simply works. IEEE 802.15.4 Task Group. Available at: <http://www.zigbee.org/resources>.

Appendix

A - 3D Router Core

```
from bluetooth import BluetoothSocket, L2CAP
from node_information_base.device_capability import *
from node_information_base.device_context import *
from node_information_base.device_profile import *
from node_information_base.nib import *
from node_information_base.nib_container import NIBContainer
from protocol_messages import frame_handler
from protocol_messages import inform
from protocol_messages import message_handler
from protocol_messages.announce import *
from utils import utils
from utils.context_awareness import *
from utils.pybash import *
from utils.utils import Loopable
from utils.utils import threaded
import Queue
import cPickle
import dbus
import dbus.glib
import dbus.service
import gobject
import gtk
import gtk.glade
import logging
import socket
import threading
import time

logger = utils.get_logger(os.path.splitext(os.path.split(__file__)[1])[0])
logger.info("===== Starting Router3D =====")

sha_bang = Bash()

CGF_FILE = utils.load_config_file()
IFACE_WLAN = CGF_FILE["IFACE_WLAN"]
IFACE_BT = CGF_FILE["IFACE_BT"]
MY_IFACE = IFACE_WLAN

class Router3D(dbus.service.Object, Loopable):
    def __init__(self, bus_name, object_path="/Router3D/Service"):
        dbus.service.Object.__init__(self, bus_name, object_path)
        self.__nib_container = NIBContainer()
        Loopable.__init__(self)
        # GUI =====
        glade_file = gtk.glade.XML(os.path.join(os.path.dirname(__file__),
                                                "gui_files",
                                                "router3d.glade"))

        self.__main_win = glade_file.get_widget("window")
        self.__main_win.show_all()
        glade_file.signal_autoconnect(self)
        #=====

        deviceCapability = DeviceCapability()
        deviceProfile = DeviceProfile()#deviceCapability.transmissionCapabilities)
        deviceContext = DeviceContext()
        main_nib = NIB(CGF_FILE["ALIAS"],
                       deviceProfile,
```

```

        deviceCapability,
        deviceContext,
        utils.getMACAddress(IFACE_WLAN or IFACE_BT))
self.__nib_container.set_main_nib_id(main_nib.nodeid)

self.__nib_container.add_nib(main_nib)
self.iface_list = utils.getAllIfaces()
print self.iface_list
self.__active_ifaces = []

self.__msg_handler = message_handler.MessageHandler()
self.__frame_handler = None
logger.info("Upping ifaces")
for i in self.iface_list:
    utils.ifaceUp(i)

logger.info("Set IP forwarding")
utils.setIPForwarding(True) # sets IPforwardin to every node
self.__ctx_awa = ContextAwareness()

#==GUI=====
def on_b_stop_clicked(self, *args):
    self.stop_services()

def on_b_bootstrapper_clicked(self, *args):
    #sha_bang.python(os.path.join(os.path.dirname(__file__),"the_bootstrapper.py"),"&")
    sha_bang.thebootstrapper("&")
    def on_b_scanner_clicked(self, *args):
        #sha_bang.python(os.path.join(os.path.dirname(__file__),"the_scanner.py"),"&")
        sha_bang.thescanner("&")
    def on_b_monitor_clicked(self, *args):
        #sha_bang.python(os.path.join(os.path.dirname(__file__),"the_monitor.py"),
"&")
        sha_bang.themonitor("&")
    #=====

@threaded
@dbus.service.method("org.router3d.serviceInterface")
def start_services_layer3(self):
    print "Listening forever for 3D Routing Messages"
    while self.is_running():
        new_nib = self.__msg_handler.receive_msg()
        if new_nib:
            self.__nib_container.add_nib(new_nib)
        else:
            print "No Nib found..."

@threaded
def start_context_awareness_service(self):
    while self.is_running():
        main_nib_ = self.__nib_container.get_main_nib_obj()
        if_80211, if_bt = main_nib_.get_device_context().get_ifaces_ip()
        self.__ctx_awa.change_net_output_packet(if_80211, if_bt,

self.__nib_container.get_nib_with_2_ifaces())

@threaded
@dbus.service.method("org.router3d.serviceInterface")
def start_services_layer2(self):
    print "starting layer2 802.11 services"
    main_dev_ctx = self.__nib_container.get_main_nib_obj().get_device_context()
    self.__frame_handler
frame_handler.FrameHandler(main_dev_ctx.get_iface_80211_name(), None)
    #self.__frame_handler.startup()
    while self.__frame_handler.is_running():
        ret = self.__frame_handler.recv_frame()
        if ret:

```

```

        if ret["data"] == "JOIN":
            try:
                ip_range = main_dev_ctx.get_iface_80211_ip_range()
                mac
            self.__frame_handler.get_str_mac(self.__frame_handler.eth_ntoa(ret["src_mac"]))
                ip = utils.get_mac_based_ip(mac, ip_range)
                self.__frame_handler.send_frame(ip, ret["src_mac"])
            except socket.timeout:
                pass

    @threaded
    def listen_client_socket(self, client_sock, mac, main_dev_ctx):
        while self.is_running():
            ret = client_sock.recv(16)
            if ret:
                if ret == "JOIN":
                    ip_range = main_dev_ctx.get_iface_bluetooth_ip_range()
                    ip = utils.get_mac_based_ip(mac, ip_range)
                    client_sock.send(ip)
                    break
            client_sock.close()

    @threaded
    @dbus.service.method("org.router3d.serviceInterface")
    def start_services_layer2_bluetooth(self):
        print "starting layer2 bluetooth services"
        main_dev_ctx = self.__nib_container.get_main_nib_obj().get_device_context()
        self.__frame_handler = frame_handler.FrameHandler(None,
main_dev_ctx.get_iface_bluetooth_name())
        server_sock = BluetoothSocket(L2CAP)
        server_sock.bind("", 0x1001)
        server_sock.listen(1)
        while self.is_running():
            print "Accept bluetooth layer2 sockets...[#####]"
            client_sock, address = server_sock.accept()
            main_dev_ctx
self.__nib_container.get_main_nib_obj().get_device_context()
            self.listen_client_socket(client_sock, address[0], main_dev_ctx)
            server_sock.close()

    @dbus.service.method("org.router3d.serviceInterface")
    def stop_services(self):
        if self.__msg_handler:
            self.__msg_handler.stop()
        if self.__frame_handler:
            self.__frame_handler.stop()
        self.stop()
        try:
            gtk.main_quit()
        except:
            pass
        self.__ctx_awa.stop_context_awareness()
        raise SystemExit(0)

    @dbus.service.method("org.router3d.serviceInterface")
    def get_iface_list(self, iface):
        return self.iface_list

    @dbus.service.method("org.router3d.serviceInterface")
    def bootstrap(self, netid, iface, iprange, channel, port, interval, leader):
        ret = 1
        try:
            is_80211 = True
            ifaces = ["bnep0", "pan0"]
            if iface == "hci0":
                iface = ifaces[leader]
                is_80211 = False
            #self.start_services_layer2()

```

```

        logger.info("Start bootstrapping iface... %s" % iface)
        t0 = time.time()
        self.start_services_layer3()
        self.activate_iface(iface)
        ip = utils.getIPAddress(iface)
        mask = utils.getNetmask(iface)
        bcast = utils.getBroadcastAddress(iface)
        if iface in utils.getWirelessIfaceList():
            main_nib_ = self.__nib_container.get_main_nib_obj()
            mn_dev_ctx = main_nib_.get_device_context()
            mn_dev_ctx.iface_80211 = IFace80211(str(iface),
                                                ip,
                                                mask,
                                                bcast,
                                                str(iprange))

        elif iface in ifaces:
            main_nib_ = self.__nib_container.get_main_nib_obj()
            mn_dev_ctx = main_nib_.get_device_context()
            mn_dev_ctx.iface_bt = IFaceBluetooth("hci0",
                                                ip,
                                                mask,
                                                bcast,
                                                str(iprange),
                                                iface)

        #self.send_periodic_announce_frame(iface, netid, interval)
        t1 = time.time()
        logger.info("End of bootstrapping iface... %s" % iface)
        logger.info("Bootstrapping time = %7.5f" % (t1 - t0))
        if is_80211:
            self.start_services_layer2()
        else:
            self.start_services_layer2_bluetooth()
            self.send_periodic_inform_msg(iface, port, interval)
            self.start_context_awareness_service()
        ret = 0
    except:
        print "BOOTSTRAP ERROR: %s" % str(sys.exc_info())
    return ret

@dbus.service.method("org.router3d.serviceInterface")
def alien_bootstrap(self, netid, iface, iprange, channel, port, interval, leader):
    ret = 1
    try:
        is_80211 = True
        ifaces = ["bnep0", "pan0"]
        if iface == "hci0":
            iface = ifaces[leader]
            is_80211 = False
        #self.start_services_layer2()
        logger.info("Start bootstrapping iface... %s" % iface)
        t0 = time.time()
        #self.start_services_layer3()
        self.activate_iface(iface)
        ip = utils.getIPAddress(iface)
        mask = utils.getNetmask(iface)
        bcast = utils.getBroadcastAddress(iface)
        if iface in utils.getWirelessIfaceList():
            main_nib_ = self.__nib_container.get_main_nib_obj()
            mn_dev_ctx = main_nib_.get_device_context()
            mn_dev_ctx.iface_80211 = IFace80211(str(iface),
                                                ip,
                                                mask,
                                                bcast,
                                                str(iprange))

        elif iface in ifaces:
            main_nib_ = self.__nib_container.get_main_nib_obj()
            mn_dev_ctx = main_nib_.get_device_context()
            mn_dev_ctx.iface_bt = IFaceBluetooth("hci0",

```



```

        ip,
        mask,
        bcast,
        str(iprange),
        iface)
    #self.send_periodic_announce_frame(iface, netid, interval)
    t1 = time.time()
    logger.info("End of bootstrapping iface... %s" % iface)
    logger.info("Bootstrapping time = %7.5f" % (t1 - t0))
    if is_80211:
        pass
        #self.start_services_layer2()
    else:
        #self.start_services_layer2_bluetooth()
        pass
    #self.send_periodic_inform_msg(iface, port, interval)
    ret = 0
except:
    print "BOOTSTRAP ERROR: %s" % str(sys.exc_info())
return ret

def activate_iface(self, iface):
    if iface not in self.__active_ifaces and iface in self.iface_list:
        self.__active_ifaces.append(iface)

@dbus.service.method("org.router3d.serviceInterface")
def deactivate_iface(self, iface):
    if iface in self.__active_ifaces:
        self.__active_ifaces.remove(iface)

@dbus.service.method("org.router3d.serviceInterface")
def get_ip_class(self):
    pass

@threaded
@dbus.service.method("org.router3d.serviceInterface")
def send_periodic_inform_msg(self, iface, port, interval):
    while self.is_running():
        self.__msg_handler.sendBroadcast(str(iface),
                                         int(port),

cPickle.dumps(inform.InformMsg(self.__nib_container.get_main_nib_obj()))
    time.sleep(interval)

@threaded
@dbus.service.method("org.router3d.serviceInterface", in_signature='ssu',
out_signature='')
def send_periodic_announce_frame(self, iface, netid, interval):
    srcMAC = utils.getMACAddress(iface) #gets interface source MAC
    dstMAC = "\xff\xff\xff\xff\xff\xff"
    frameType = 0x55aa
    data = announce.AnnounceMsg(str(netid), str(srcMAC))
    msg = cPickle.dumps(data)
    #####self.__frame_handler.buildFrame(srcMAC, dstMAC, frameType,
cPickle.dumps(data))
    #self.__frame_handler.sendPeriodicFrame(iface, dstMAC, frameType, msg,
interval)

@dbus.service.method("org.router3d.serviceInterface")
def get_nibs_from_container(self):
    return self.__nib_container.get_container()

@dbus.service.method("org.router3d.serviceInterface")
def get__nib_container(self):
    return self.__nib_container

@dbus.service.method("org.router3d.serviceInterface")

```

```

def get_nib_by_nodeid(self, nodeid):
    return self.__nib_container.get_nib_by_nodeid(nodeid)

@dbus.service.method("org.router3d.serviceInterface")
def get_main_nib(self):
    return self.__nib_container.get_main_nib()

#=====
# MESSAGE HANDLER METHODS =
#=====
@dbus.service.method("org.router3d.serviceInterface")
def sendBroadcast(self, iface, port, msg):
    self.__msg_handler.sendBroadcast (iface, port, msg)

@dbus.service.method("org.router3d.serviceInterface")
def sendPeriodicBroadcastMsg(self, iface, port, msg, interval):
    self.__msg_handler.sendPeriodicBroadcastMsg(iface, port, msg, interval)
@dbus.service.method("org.router3d.serviceInterface")
def sendUnicastmsg(self, destAddr, msg, port):
    self.__msg_handler.sendUnicastmsg(destAddr, msg, port)

@dbus.service.method("org.router3d.serviceInterface")
def sendPeriodicUnicastMsg(self, destAddr, msg, port, interval):
    self.__msg_handler.sendPeriodicUnicastMsg(destAddr, msg, port, interval)

if __name__ == "__main__":
    session_bus = dbus.SessionBus()
    name = dbus.service.BusName("org.router3d.service", bus=session_bus)
    print "Starting 3DRouting Message Listeners"
    core = Router3D(name)
    #core.start_services_layer3()
    ##core.sendPeriodicFrame("eth0", dstMAC, frameType, data, interval)
    #core.send_periodic_announce_frame(MY_IFACE, "Fireman", 5)
    #core.send_periodic_inform_msg(MY_IFACE, 51423, 5)
    ##_test(core.get__nib_container())
    try:
        print "======"
        print "  Startup the 3DRouter service daemon...ok  "
        print "======"
        gtk.gdk.threads_init()
        gtk.gdk.threads_enter()
        gtk.main()
        gtk.gdk.threads_leave()
    except KeyboardInterrupt:
        core.stop_services()
    finally:
        print "======"
        print "  Shutting down 3DRouter service daemon...ok  "
        print "======"

```

B - 3D Routing Measurement Tool

```

from the_scanner import *
from the_bootstrapper import *
import gtk
import time
import os
import utils
import sys

```

```

logger = utils.utils.get_logger(os.path.splitext(os.path.split(__file__)[1])[0])
logger.info("===== Starting stress =====")
bash = utils.pybash.Bash()

class AlienTheScanner(TheScanner):
    def __init__(self, test_iface, count):
        self.times = int(count)
        self.test_iface = test_iface=="bt"
        TheScanner.__init__(self)
        if WLAN_IFACES:
            self.scan_80211_devices()
        if BT_IFACES:
            self.scan_bluetooth_devices()
        if ETH_IFACES:
            pass

    def on_b_join_clicked(self, *args):
        for i in range(self.times):
            self.stress(self.test_iface)
            #time.sleep(1)

    def stress(self, tech=0):
        try:
            bash.ifconfig("bnep0", "down")
        except utils.pybash.ShellError:
            pass
        if tech == 0: # 802.11
            selected_row = dict(zip(["status", "netid", "channel", "quality",
                                     "bssid"],
                                     [0,"3DRW", "4","",""] ))
            logger.info("Joiner 802.11 starting")
            t0 = time.time()
            utils.utils.setIface(IFACE_WLAN,
                                "Ad-Hoc",
                                get_channel(selected_row["channel"]),
                                selected_row["netid"],
                                "0.0.0.0")
            logger.info("Iface set up")
            tmp = frame_handler.FrameHandler(IFACE_WLAN, None)
            while self.is_running:
                print "Send JOIN message, waiting IP..."
                tmp_frame = tmp.send_recv_frame("JOIN")
                new_ip = tmp_frame["data"]
                if match(r"^\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}$",new_ip):
                    print "type of new_ip =>", type(new_ip)
                    print "Receive ip (%s) from %r" % (new_ip, tmp_frame["src_mac"])
                    tmp.stop()
                    del tmp
                    break
                else:
                    print "Discarted invalid information sent from %r" %
tmp_frame["src_mac"]
                    logger.info("Ip returned")
                    utils.utils.setIface(IFACE_WLAN,
                                        "Ad-Hoc",
                                        get_channel(selected_row["channel"]),
                                        selected_row["netid"],
                                        str(new_ip))
                    logger.info("Iface set up again")
                    status = ROUTER3D_SERVICE_INTERFACE.alien_bootstrap(selected_row["netid"],
                                IFACE_WLAN,
                                str(new_ip),
                                selected_row["channel"],
                                51423,
                                5,
                                0)

```

```

        t1 = time.time()
        logger.info("Joiner 802.11 finished")
        logger.info("Joiner 802.11 time = %7.5f" % (t1 - t0))
        connected = not bool(status)
    elif tech == 1: # bluetooth
        selected_row = dict(zip(["status", "netid", "address"],
                                [0, "3DRW", "00:18:E7:36:38:CB"]))
        logger.info("Joiner bluetooth starting")
        t0 = time.time()
        new_ip = ""
        tmp = frame_handler.FrameHandler(None, IFACE_BT)
        while self.is_running:
            tmp_frame = ""
            try:
                print "Send JOIN message, waiting IP..."
                tmp_frame = tmp.send_recv_frame_bluetooth("JOIN",
selected_row["address"])
            except:
                print sys.exc_info()
                print "Timeout error, cannot get ip. Try again."
            else:
                new_ip = tmp_frame
                if match(r"^\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}$", new_ip):
                    print "type of new_ip =>", type(new_ip)
                    print "Receive ip (%s) from %r" % (new_ip,
selected_row["address"])
                    tmp.stop_layer2()
                    break
                else:
                    print "Discarded invalid information sent from %r" %
selected_row["address"]
                    time.sleep(0.5)
                utils.utils.setbnepSlave(selected_row["address"], new_ip)
                status = ROUTER3D_SERVICE_INTERFACE.alien_bootstrap(selected_row["netid"],
                                                                    IFACE_BT,
                                                                    new_ip,
                                                                    1000,
                                                                    51423,
                                                                    5,
                                                                    0)

            t1 = time.time()
            os.system("ifconfig bnep0 down")

            logger.info("Joiner bluetooth finished")
            logger.info("Joiner bluetooth time = %7.5f" % (t1 - t0))
            connected = not bool(status)
            #time.sleep(3)
        else:
            print "WARNING: There is no third page!!!"

class AlienTheBootstrapper(TheBootstrapper):
    def __init__(self, count):
        TheBootstrapper.__init__(self)
        self.count = count

    def on_btboot_clicked (self, widget):

        IFACE = "hci0"
        IP_RANGE = "172.16.0.0/16"
        for i in range(self.count):
            randomip = utils.utils.get_mac_based_ip(utils.utils.getMACAddress(IFACE),
IP_RANGE)
            self.stress(netid="STRESSTEST",
                        iface=IFACE,
                        iprange=IP_RANGE,
                        channel=4,

randomip=utils.utils.get_mac_based_ip(utils.utils.getMACAddress(IFACE), IP_RANGE))

```

```

def stress(self, netid, iface, iprange, channel, randomip):
    logger.info("Starting network bootstrapper time")
    t0 = time.time()
    utils.utils.setIface (iface, "Ad-Hoc", channel, netid, randomip)
    logger.info("Set up iface")
    ROUTER3D_SERVICE_INTERFACE.alien_bootstrap(netid,
                                                iface,
                                                iprange,
                                                channel,
                                                51423,
                                                5,
                                                1)

    t1 = time.time()
    logger.info("End of network bootstrapper time")
    logger.info("Network bootstrapper time = %7.5f" % (t1 - t0))

if __name__ == "__main__":
    """
    usage:

        Test TheScanner:
        python the_stress_test.py 1 <iface = bt|80211> <count>

        Test TheBootStrapper:
        python the_stress_test.py 0 <count>
    """

    test = None
    if int(sys.argv[1]):
        test = AlienTheScanner(sys.argv[2], sys.argv[3])
    else:
        test = AlienTheBootstrapper(int(sys.argv[2]))
    try:
        gtk.gdk.threads_init()
        gtk.gdk.threads_enter()
        gtk.main()
        gtk.gdk.threads_leave()
    except KeyboardInterrupt:
        test.stop()
        raise SystemExit(0)

```

