

Multi-purpose Embedded Communication Gateway: System Design and Testbed Implementation

Chi Winth Ea
Morten Sørbø

Lei Jiao

This master's thesis is carried out as a part of the education at the University of Agder and is therefore approved as a part of this education. However, this does not imply that the University answers for the methods that are used or the conclusions that are drawn.

University of Agder, 2014

Faculty of Information and Communication Technology

Department of Engineering and Science

Version Control

<i>Version</i>	<i>Status</i>	<i>Date</i>	<i>Change</i>	<i>Author</i>
0.1	Draft	2014-05-11	Chapters 1, 2 and 3	CWE
0.2	Draft	2014-05-18	All chapters changed	CWE and MS
0.3	Draft	2014-05-24	All chapters changed	CWE and MS
0.4	Draft	2014-05-29	All chapters changed	CWE and MS
0.5	Review	2014-05-30	All chapters changed	CWE and MS
0.6	Review	2014-06-01	All chapters changed	CWE and MS
0.7	Review	2014-06-01	All chapters changed	CWE and MS
1.0	Final	2013-06-02	All chapters changed	CWE and MS

Preface

This dissertation was written by ICT masterstudents at the University of Agder in Grimstad, Norway, from January 2014 to June 2014.

The thesis has been supervised by Postdoctoral researcher Lei Jiao, University of Agder. With weekly meetings, the process has been continuously smooth as there has always been enough work to do.

The equipment used during the thesis is financed by the University of Agder. Some was already available at the University, but most of the devices had to be aquired by the writers. As time was of the essence, it has been important to choose equipment that was easily available to borrow or purchase.

Acknowledgements

We would like to express our gratitude towards Dr. Lei Jiao for his invaluable guidance during this dissertation. It has been a pleasure having him as our supervisor as his input during our time together steered us towards the completion of our work. He has been very gracious with his time, and made us feel prioritized along the whole process.

Grimstad

June, 2014

Chi Winth Ea and Morten Sørbø

Abstract

This dissertation revolves around developing a multi-purpose embedded communication gateway. The gateway is equipped with multiple communication interfaces including Ethernet, Bluetooth, WiFi, Zigbee, LTE, and it can be configured and utilized for many purposes, such as a failover of an Ethernet cable via 4G in order to maintain the network connectivity. Raspberry Pi circuit board and the operating system Raspbian are selected as the hardware and the software platforms respectively. Different communication interfaces are coordinated by the Raspberry Pi and are configured via Linux scripts according to various use cases. Furthermore, a hardware watchdog is adopted to enhance the availability of system. In addition, the system is encapsulated into a box to increase its portability. The system is validated and evaluated through rigorous test-bed experiments. Experiment results indicate that the developed router works smoothly and reliably in environments with little electrical disturbances.

Contents

- 1 Introduction..... 12
 - 1.1 Background..... 13
 - 1.2 Problem Statement 13
 - 1.3 Problem Solution 13
 - 1.4 Organization of the Dissertation 15
- 2 Existing Work..... 16
 - 2.1 Commercial Products 16
 - 2.2 Research Studies 17
 - 2.2.1 Network Failover 17
 - 2.2.2 Data Synchronization Technologies 17
 - 2.2.3 Watchdog as a Reliability Improvement 18
- 3 System Design 20
 - 3.1 System Overview 20
 - 3.2 Overview of Hardware 21
 - 3.2.1 Main Computer Board – Raspberry Pi..... 22
 - 3.2.2 Watchdog – Arduino..... 23
 - 3.3 Overview of Software..... 23
 - 3.3.1 Operating System for Raspberry Pi 24
 - 3.4 Overview of Interfaces 25
 - 3.4.1 Technologies..... 25
 - 3.4.2 Communication Dongles 27
- 4 Implementation..... 28
 - 4.1 Software Installation and Interface Initialization..... 28
 - 4.1.1 Operation System Installation 28
 - 4.1.2 Mobile Broadband Software 31

4.1.3	Wi-Fi Access Point	33
4.1.4	Ethernet.....	35
4.1.5	Routing	35
4.1.6	E-mail.....	39
4.1.7	Bluetooth.....	39
4.1.8	ZigBee	41
4.1.9	Kernel Compilation	43
4.1.10	SD-Card Protection	45
4.1.11	MySQL Installation Proposal.....	47
4.2	Mobile Broadband Failover	49
4.2.1	The Initialization of the Script	51
4.2.2	While Loop When There is Internet Connectivity	52
4.2.3	While Loop When There is Only Local Network Access	54
4.2.4	While Loop When There is No Network Access	55
4.3	The Watchdog	57
4.3.1	Internal Watchdog.....	57
4.3.2	External Watchdog	59
4.4	Other Reliability Measures.....	63
4.5	Physical Implementation.....	63
4.5.1	The Casing.....	63
4.5.2	Cables and Connections	66
4.5.3	ZigBee Module.....	69
5	Testing and Validation.....	71
5.1	Reliability	71
5.1.1	Wi-Fi Access Point	71
5.1.2	Ethernet Routing	71

- 5.1.3 Mobile Broadband (3G/4G)..... 72
- 5.1.4 Network Failover 72
- 5.1.5 Bluetooth..... 73
- 5.1.6 Internal Watchdog..... 73
- 5.1.7 External Watchdog 73
- 5.2 Scenarios 75
 - 5.2.1 Wi-Fi Access Point Range Experiment..... 75
 - 5.2.2 Mobile Broadband Gateway in a Car..... 75
- 5.3 Validation 75
- 6 Discussion 77
- 7 Conclusion 79
- References..... 80
- Appendices 91

List of Figures

Figure 1 - Server synchronization scenario	14
Figure 2 - Car scenario	14
Figure 3 - Netgear DGN2200M [7].....	16
Figure 4 - Overview of the system.....	20
Figure 5 - Raspi-config main menu.....	28
Figure 6 - Raspi-config internationalisation options	29
Figure 7 - Raspi-config overclock warning.....	29
Figure 8 - Raspi-config overclock menu	30
Figure 9 - Raspi-config advanced options.....	30
Figure 10 - Basic flow diagram for the installation script.....	31
Figure 11 - Output from "lsusb"	32
Figure 12 - Output from "lsusb" after reboot	32
Figure 13 - Content of usb_modeswitch.conf	33
Figure 14 - Content of wvdial.conf	33
Figure 15 - The contents of the configuration file for hostapd	34
Figure 16 - The first part of the file "/etc/sysctl.conf"	36
Figure 17 - The content of the file "/etc/network/interfaces"	37
Figure 18 - The start of the file "/etc/dhcp/dhcpd.conf"	38
Figure 19 - The end of the file "/etc/dhcp/dhcp.conf"	38
Figure 20 - The main configuration of Postfix	39
Figure 21 - XCTU initial launch.....	41
Figure 22 - XCTU XBee Configuration	42
Figure 23 - XCTU Range Test	43
Figure 24 - Overview of the network failover script	50

Figure 25 - Basic overview of the states in the network failover script	50
Figure 26 - Loop in the failover script where there is Internet access	52
Figure 27 - Loop in the failover script where there is local network access	55
Figure 28 - Loop in the failover script with no network access.....	56
Figure 29 - Simple flow diagram of the internal watchdog.....	58
Figure 30 - The Watchdog configuration file.....	58
Figure 31 - A general flow diagram of an Arduino script	59
Figure 32 - Arduino main loop overview	60
Figure 33 - A detailed view of the main loop in the Arduino script	62
Figure 34 - The Raspberry Pi with the hub and some adapters	64
Figure 35 - The front of the box	65
Figure 36 - The rear side of the box	65
Figure 37 - The top of the box	66
Figure 38 - The inside of the box, seen from above.....	67
Figure 39 - Illustration of connections [91][92][93][43][94][95][96]	68
Figure 40 - Slice of Pi before assembly [97]	69
Figure 41 - The Slice of Pi after soldering the connectors.....	70
Figure 42 - Our Slice of Pi with XBee	70
Figure 43 - RPi with Slice of Pi	70
Figure 44 - RPi with Slice of Pi	70

List of Tables

Table 1 - Hardware comparison 1 21

Table 2 - Hardware comparison 2 22

Table 3 - Operating systems for ARM processors 24

Table 4 - Interface comparison..... 25

Table 5 - Explanation to Figure 39..... 68

Abbreviations

3G	Third Generation Telecommunication Technology
3DES	Triple DES (Data Encryption Standard)
4G	Fourth Generation Telecommunication Technology
5VDC	5 volt direct current
A	Ampere
ARM	Advanced RISC (Reduced Instruction Set Computer) Machine
armhf	ARM Hard Float
ARMv6	ARM version 6
ARMv7	ARM version 7
BBB	BeagleBone Black
BER	Bit Error Rate
CD	Compact Disc
CPU	Central Processing Unit
DES	Data Encryption Standard
DSL	Digital Subscriber Line
GPIO	General-purpose input/output
HDMI	High-Definition Multimedia Interface
ISP	Internet Service Provider
LAN	Local Area Network
LED	Light Emitting Diode
mA	Milliampere
OS	Operating system
RAM	Random Access Memory
RAP	Remote Access Points
RISC	Reduced Instruction Set Computer
RPi	Raspberry Pi
RTT	Round Trip Time
SD	Secure Digital
SSH	Secure Shell
SSID	Service Set Identifier
TCP	Transmission Control Protocol
USB	Universal Serial Bus
WLAN	Wireless Local Area Network

1 Introduction

The Internet acts as a gateway to knowledge and information, and its usages are unlimited. Connecting to the Internet allows billions of people around the world to communicate with each other in different ways. People all over the world connect to the Internet to receive the services it provides. It is a medium which has a large role in education, media entertainment as well as social interaction between users. The Internet is a resource and it would be hard to argue against it being an indispensable resource in today's society. Devices which connect to the Internet every day are often shipped with other communication interfaces than Wi-Fi and Ethernet, an example would be a smart-phone with bluetooth and infrared support. It does not necessarily have to be a smart-phone, it may be an older phone that does not have Wi-Fi support, or a camera that are able to transfer files, wirelessly, through bluetooth. We would like to take advantage of the fact that there are a lot of devices with support for different communication interfaces. We would like to develop a multi-purpose embedded communication gateway that different devices may connect to and use in different scenarios. The embedded system will act as a gateway for communication devices and provide services that they cannot achieve on their own. An example would be to forward pictures from a digital camera to the Internet by use of mobile broadband.

Speaking of the Internet, one could imagine it would mean a great deal for most people to lose access to the services it provides, even for a single day, hour, or minute, depending on its current use. It may happen while streaming your favorite TV show or playing an online game. It may occur before handing in an assignment or while you are trying to read the news. Losing connection to the Internet can happen any day and at any time, some may experience it more frequently than others depending on the Internet service provider (ISP). In addition to featuring multiple interfaces, we would like our system to provide continuous Internet access. The system will be given a mobile broadband interface which acts as a secondary connection to the Internet and will automatically turn itself on at suitable occasions. The system will have multiple purposes, depending on the user that takes the device into use. It may serve as a node in an already established local area network (LAN) or be used in other scenarios.

1.1 Background

There are some existing products on the market which provides continuous Internet access by using mobile broadband as a second connection to the Internet. Companies such as Asus [1], Cisco [2], TP-Link [3] and Netgear [4] are a few examples of manufacturers that have developed such devices. The amount of devices which are specialized in providing continuous Internet access are few. Of these there are few or none multi-purpose devices able to communicate on other interfaces. The gateway provided by the dissertation will become an addition to the group of devices which uses mobile broadband as failover while providing the user the possibility to use it in scenarios suited for their needs. It is important to note that this embedded system may be modified by users by expanding its capabilities.

1.2 Problem Statement

Although mentioned companies has developed existing products, there are still some features that may be added. For instance, multiple interfaces for multi-purpose use in other types of networks, as well as a watchdog to improve its availability. Keeping these improvements in mind, the objective of this dissertation is to create a similar product as well as improving its current state by adding new features. The newly developed product's performance will be evaluated through experiments. More specifically, the following goals are identified:

- **Goal 1:** To develop a multi-purpose embedded gateway by installing the following interfaces: LAN, Wireless Local Area Network (WLAN), and Bluetooth, ZigBee, and Third/Fourth Generation Telecom technology (3G/4G).
- **Goal 2:** To develop automatic loss detection to the Internet that may be used together with mobile broadband to provide continuous Internet access.
- **Goal 3:** To improve the system's availability by adding a watchdog.
- **Goal 4:** To improve the system's reliability through script optimization and loading specific folder's into the memory. The kernel of the operating system will be recompiled to bring stability to the system as well as removing unnecessary load.
- **Goal 5:** To propose a solution for data synchronization to a remote server.

1.3 Problem Solution

The embedded system will be developed using a Raspberry Pi with several types of interface adapters. Continuous Internet access will be achieved using shell scripts with mobile broadband acting as a backup connection to the Internet. In order to provide availability, an Arduino will function as a watchdog. The reliability of the system will be achieved through code optimization in

addition to loading specific folders into the memory to protect the Secure Digital (SD) card. The kernel of the operating system will be recompiled to suit the system specification and to provide stability. The dissertation will also propose a solution for implementing data synchronization between the embedded system and a server.

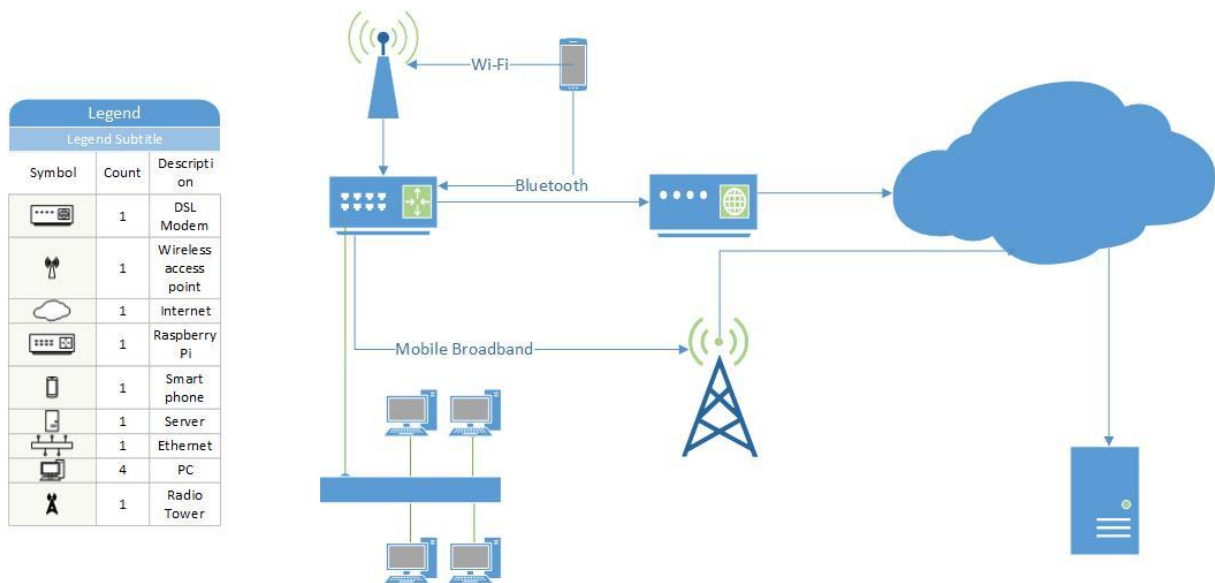


Figure 1 - Server synchronization scenario

By including multiple interfaces, the embedded system will have additional usages. Figure 1 shows that it may be used to synchronize local nodes on different interfaces with an external server. Here a smart phone is connected to the system via both Bluetooth and Wi-Fi, while a computer network is connected through Ethernet. The system is connected to the internet via Digital Subscriber Line (DSL) and mobile broadband for redundancy.

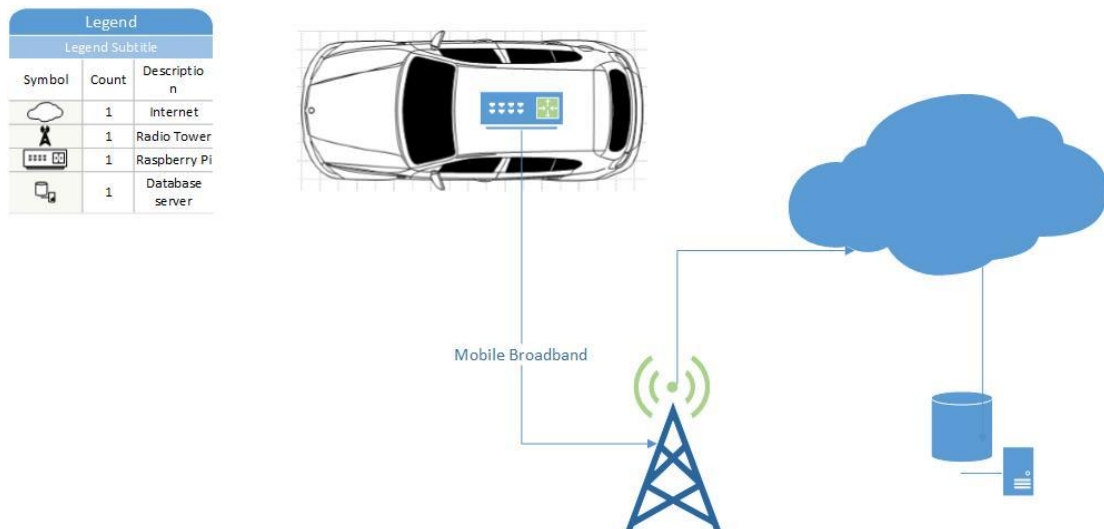


Figure 2 - Car scenario

Figure 2 shows that another use for the system can be inside a car. The system could provide routing for sensors in the car, and transmit log files to an online location if a emergency should occur.

1.4 Organization of the Dissertation

The main body of the report is organized into several different chapters ranging from chapter 2-7. Chapter 2 is titled Existing Work and provides information about already existing products and technology related to our communication system. Chapter 3 is called System Design and contains an overview of the different hardware and software that have been looked at during the semester. The chapter will also present the system overview in terms of chosen hardware and software and give the reasons behind the specific selections. The dissertation will then move on to Chapter 4 Implementation. This chapter will provide information on how to install and develop the different features that are included in the multi-purpose embedded communication system. The chapter will in addition include how to compile the Linux kernel to suit the system's specifications and make it more reliable. Testing and validation will be provided in Chapter 5 Testing. The chapter will inform about the several different experiments that were done to validate our solution. The end of the report, Chapter 6 and 7, will provide a discussion and conclusion of our work.

2 Existing Work

2.1 Commercial Products

It has already been mentioned in the previous sections that there are existing products on the market which provides continuous Internet access by using mobile broadband as a second connection to the Internet. One of them is NetGear.

Netgear has three routers which were developed with purpose of providing continuous Internet access. The only router which supports a DSL connection is the DGN2200M [4]. It was released in 2010 and is still in production by the company [5]. The router was originally intended to be used with 3G modems, but is currently supporting a wide arrangement of 4G modems as well [6]. DSL is connected to the DGN2200M using the RJ11 connector. Its internal modem will act as an access point and provide Internet access to the other nodes in the network which eliminates the need of an independent modem.

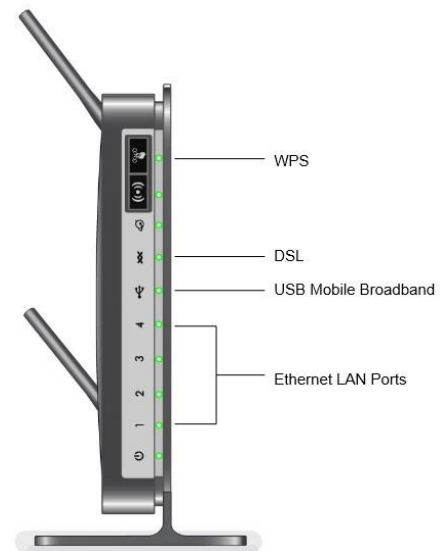


Figure 3 - Netgear DGN2200M [7]

The DGN2200M, in addition to the DSL modem, supports mobile broadband through its Universal Serial Bus (USB) port. The mobile broadband can be used to provide continuous Internet access when the DSL is unavailable or as a dedicated mobile broadband router. Another useful feature which is included in the router is the traffic meter [7]. The traffic meter is used to control the amount of data used while the mobile broadband is active. A limit may be set to a specific value which will suit the user's specification and needs. The mobile broadband will be disconnected when the limit of maximum data transfer is reached.

The routers provide continuous Internet access in the same manner as our embedded communication gateway. Our embedded communication gateway does however provide multiple interface which makes it more universal than the products of the mentioned companies. It is also worth mentioning that our system is open source and may be modified to suit other purposes compared to the DGN2200M which is a closed system and protected by companies' patents. In addition, the embedded communication gateway has an integrated watchdog which removes the need for manual rebooting of the system. The result is that it may be used in situation where manual reboots is not an option, for instance in a data sensor scenario where there are not any active administrators.

2.2 Research Studies

2.2.1 Network Failover

The first paper we're going to address is a patent paper[8]. The paper itself describes a network which use Remote Access Points (RAP) that associates and employees connect to. The RAP is given the same configuration parameters as the local company network. By using the same Service Set identifiers (SSIDs), encryption and authentication settings, the users can automatically connect to the office network after an initial setup. The paper describes a strategy which lets the RAP stay connected to the company network at all times. This is done through several parameters such as connection priority, signal strength, connection capacity, bit error rate (BER), round-trip time(RTT), and traffic amount limitations.

If the amount of data transfer is exceeded in one connection between the user and RAP, the system will switch its user over to another RAP with the right specifications. If the user does not wish to do so, it should disconnect from the Internet. The process of choosing a connection is decided through different attributes such as BER and RTT. They are however subject to change due to weather conditions, amount of users on the specific access point as well as physical disturbances. The system will then measure these parameters for all available alternative connections before choosing one to connect through. The patent paper also address other requirements for the system such as security.

The strategies and parameters mentioned in this article are relevant for the network selection in the embedded system. It is however uncertain if the requirements set for the network described in this article applies to the system developed during the thesis. If the solution is supposed to handle secure communication, it would be a good idea to implement some quality requirements. On the other hand, in applications with small amounts of data transferred, the quality of the connection does not need to meet the same requirements. There it would probably be a good alternative to implement connection quality as a user available setting. [9]

2.2.2 Data Synchronization Technologies

This article provides four different tests of the file transfer technologies cfengine and rsync [10]. The tests are with native cfengine, native cfengine through a Secure Shell (SSH) tunnel, native cfengine with Triple Data Encryption Standard (3DES) encryption, and rsync over SSH called externally from cfengine.

For each of these tests the data transfers were divided in four different sizes: From the smallest 128kb in 22 files and 1 directory to the largest at 258Mb in 5646 files and 528 directories. The change in file sizes during the test may help to detect different performances during diverse workloads. To produce enough data points, the tests were run ten times – five with debugging and five without debugging. And to eliminate eventual bottlenecks in the client, the same transfers were done in a secondary client as well.

The results of the tests are illustrated in order to show the time taken for each transfer size and technology. To summarize the results, rsync on Linux performs better than cfengine only when there are a large amount of files to be synchronized and Solaris rsync is equal or faster than cfengine with all techniques. Non-encrypted cfengine over SSH performed poorly overall as cfengine and SSH was connected through Transmission Control Protocol (TCP) socket connections and therefore created a heavier load on the operating system's TCP stack.

These results are interesting as rsync is popular file synchronization technology. However, it is important to keep in mind that these tests were performed in 2001 and a lot has changed since then. Network transmission speeds, operating systems and hardware capacities as well technologies themselves are subject to change. Even if the results in this article were conclusive for the concurrent time, the same tests performed with the latest versions on the current hardware and operating systems may show different results. [9]

2.2.3 Watchdog as a Reliability Improvement

The purpose of this article is to describe the different types of watchdogs that may be used in a system [11]. It describes in details where they may be placed and what types of trigger mechanisms that are used based on their location. There are three possible locations for a watchdog in a system:

- 1 In software.
- 2 On-chip of the device.
- 3 External

The article describes the advantages and disadvantage of each location. For instance, if the watchdog is placed inside the software and is to be triggered by a software malfunction it will result in being unusable if the hardware freezes. However, an on-chip watchdog in the mentioned scenario may provide a system reboot. This way the system “knows” if the system is required a normal or an extraordinary boot. The third alternative is an external watchdog. An external watchdog is installed on an external board which is connected to the main system.

Other aspects which are described in the article are methods which can be used in order to detect a malfunction. One method is to implement a timer which is connected to the watchdog. The system will reset the timer at a given time, would the system fail to do so, the watchdog will act according to its settings. The article is relevant for our work as it provides information about the different types of watchdog which exists, and its strengths and weaknesses.

As it has been mentioned in the previous subchapters, the goal of this dissertation is to develop a multi-purpose embedded communication gateway. The motivation itself lies within the mutual interest of development and networking. Personal motivation also plays a part of the project as it seems to be a practical device to have. It also lets the user to add more features to the device if necessary.

Based on the above observations, we realize that some of the features our communication gateway brings to the market already exists. As it was mentioned in the previous section, there are already existing commercial products that provides continuous Internet access through LTE and the discussion of bringing network failover, data backup, and availability to to users is a fairly discussed topic. The LTE router mentioned in the previous section does however not provide any of the features that the articles discusses. It is important to notice that even though the features discussed in the articles are common in the corporate side, it is not so in a regular user's home. We would like to provide our users with a communication device that features LTE as a mean to provide continuous Internet access, as well as integrating the corporate advantages that has been discussed in the recent articles. Our device will also integrate different communication interfaces to ensure that the device can be used in different types of scenarios and not only in home-environment. [9]

3 System Design

3.1 System Overview

The embedded system is developed on the Raspberry Pi using a recompiled kernel of the operating system: Raspbian. Multiple communication interfaces such as Ethernet, Wi-Fi, Bluetooth, ZigBee, and LTE has been added through compatible chipsets to make it suitable for different types of scenarios. The communication dongles are connected to the Raspberry Pi through a self-powered USB-hub, and an external board is connected to the Raspberry Pi which functions as a watchdog. The current board which has been used in the system is an Arduino, the reason for choosing the specific hardware will be explained in Section 3.2.2.

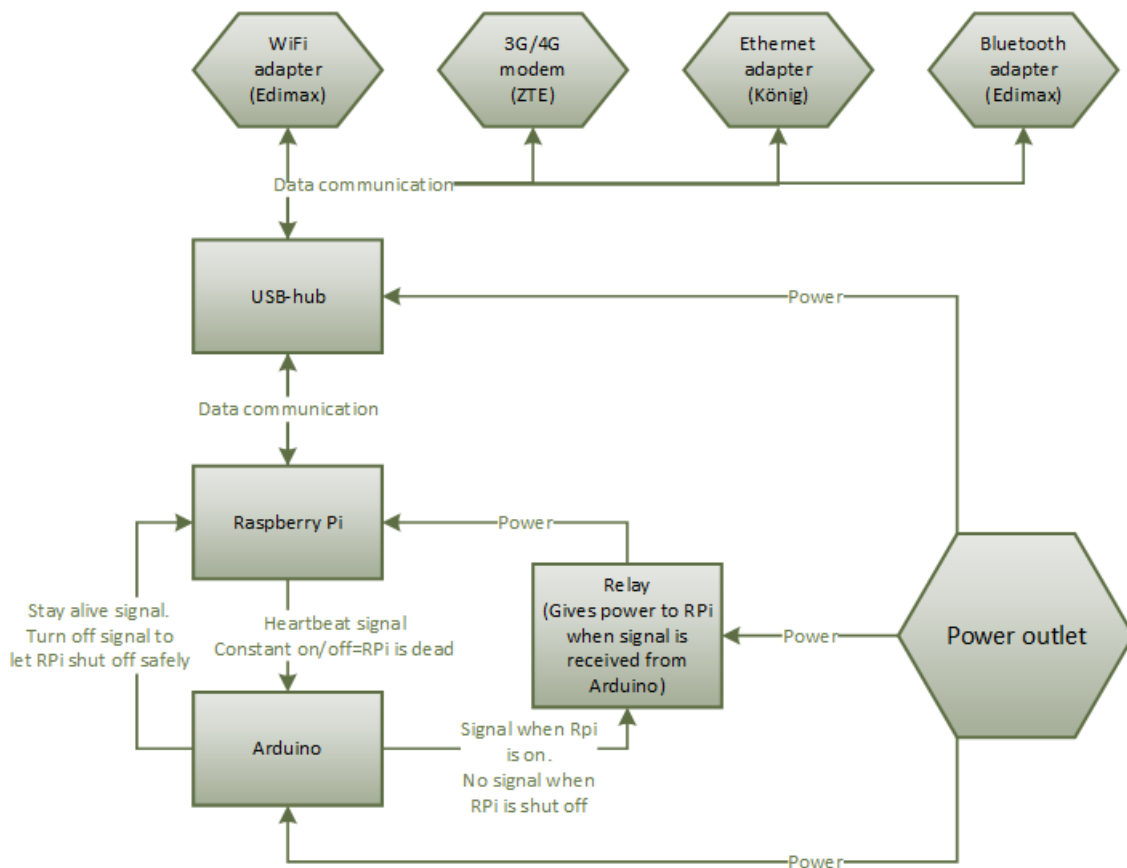


Figure 4 - Overview of the system

The system has been developed to monitor its Ethernet ports in real-time to detect abrupt loss of connection and automatically switch to mobile broadband to provide continuous Internet access. It is designed to act as a wireless access point for wireless devices. To extend the life of the SD card the system has been designed to write only to the Raspberry Pi's memory.

3.2 Overview of Hardware

This part of the report will provide some information about the different types of hardware which has been looked at during the past year, and the reasons for choosing the specific computer boards to support our communication gateway.

Table 1 - Hardware comparison 1

Computer Board	Central Processing Unit (CPU)	Random Access Memory (RAM)	Storage	USB-connections
A13-OLinuXino-WIFI [12][13][14][15]	1 GHz	512 MB	Micro SD-card	3
Arduino Mega 2560 [16][17][18]	16 MHz	8196 B	Internal 4092 B	1
BeagleBoard [19][20]	720 MHz	2048 MB	Memory card	1
BeagleBone Black [14][20][21]	1 GHz	512 MB	Micro SD card	1
Cubietruck [14][22][23][24]	Dual-core 1GHz	2048 MB	Internal 2 GB+S-ATA	2
Elektor Linux Board [25][26]	180 MHz	32 MB	4 SD card	1
Gooseberry [27][28][29]	1 GHz	512 MB	4 GB internal + micro SD	1 Mini-USB
Hackberry [30][31]	1 GHz	512 MB	4 GB internal + SD card up to 32 GB	2
Mars Board [32][33]	1.2 GHz	1024 MB	4 GB internal + SD card up to 32 GB	2 + OTG
Netduino plus 2 [34][35][36]	168 MHz	100+ KB	Micro SD	0
Parallella [37][38][39]	Dual-core 1 GHz	1024 MB	16 GB micro SD	1+OTG
Raspberry Pi mod. B [40][41]	700 MHz	512 MB	SD card	2
Utilite Value [42][43]	1 GHz	512 MB	Micro SD	4 + OTG

Table 2 - Hardware comparison 2

Computer Board	Ethernet	Wi-Fi	Bluetooth	Power consumption	Price
A13-OLinuxXino-WIFI [12][13][14][15]	No	Yes	No	6 W	55€
Arduino Mega 2560 [16][17][18]	No	No	No	N/A	39€
BeagleBoard [19][20]	No	No	No	6 W	91€
BeagleBone Black [14][20][21]	Yes	No	No	6 W+	35€
Cubietruck [14][22][23][24]	Yes	Yes	Yes	2 A	70€
Elektor Linux Board [25][26]	No	No	No	50-70 mA	71€
Gooseberry [44][28][29]	No	Yes	No	4 W	50€
Hackberry [30][31]	Yes	Yes	No	0.7 A @ 5.5 V	48€
Mars Board [32][33]	Yes	No	No	N/A	37€
Netduino plus 2 [34][35][36]	Yes	No	No	N/A	55€
Parallella [37][38][39]	Yes	No	No	5 W	72€
Raspberry Pi mod. B [40][41]	Yes	No	No	700-1000 mA	25€
Utilite Value [42][43]	Yes	No	No	4-6 W	72€

3.2.1 Main Computer Board – Raspberry Pi

Table 1 and Table 2 illustrate the different types of computer boards which have been studied in the past year. There is variation in power and price as well as utilities such as Ethernet, Wi-Fi, Bluetooth and power consumption. The weakest boards, the Arduino, The Elektor Linux board, and the Netduino Plus 2, all operate with processor frequencies below 200MHz. The mentioned boards' specifications are too low to support our embedded system and are removed from the list of candidates. The other boards possess similar aspects when it comes to the CPU frequency as they all range from 700 MHz to 1.2 GHz dual core and are all able to run advanced operating systems such as Linux. Another important aspect is the amount of RAM each of the devices possess. They all range from 512MB to 2GB. The BeagleBoard and Cubietruck ships with the highest amount of RAM of the listed devices, but are also the most expensive ones to order, the BeagleBoard at 91€ and Cubietruck at 71€. Other expensive boards are the A13-OLinuxXino-WIFI, the Hackberry, the Parallella and the Utilite Value, which all have a price above 47€ and basically have the same capabilities as the cheaper models. It is important to note that the BeagleBoard and Cubietruck, with highest amount of RAM, are not suitable for the tasks that the embedded system will perform as 2048MB of RAM is too much.

This leaves us with BeagleBone Black, The Gooseberry, the Mars Board, and the Raspberry Pi. The Gooseberry was removed from the list of candidates since it does not possess an Ethernet port or a standard USB port. The next step is to look at the power adapters that are powering the different computer boards. The Raspberry Pi and BeagleBone Black are both powered by a mini-USB, while the Mars Board is only able to power up by use of a 5 volt direct current (5VDC) adapter. In terms of availability we decided to remove the Mars Board from the list of candidates as being able to power the Raspberry Pi (RPI) and BeagleBone Black (BBB) through a mini-USB is simpler.

The two remaining boards are quite similar to each other. They both have the memory of 512 MB, as well as an High-Definition Multimedia Interface (HDMI) and Ethernet output. The RPi has the advantage of shipping with two USB ports, while the BBB has a stronger CPU at 1 GHz compared to Raspberry Pi's 700 MHz. The RPi is however our chosen as our platform of development as it is more than powerful enough to support the systems function, and comes at a cheaper price than BBB. The RPi is well suited for running a light Linux distribution, and the CPU can easily be overclocked to gain the necessary performance level. [45]

3.2.2 Watchdog – Arduino

The Arduino Mega 2560, Elektor Linux board and Netduino Plus 2 were all removed from the list of candidates in Section 3.2.1 for being too weak. As the watchdog uses a fairly simple piece of software, the processing power in the hardware does not have to be as great as in one who will run an operating system. The watchdog needs to be reliable and as inexpensive as possible. With the price ranging from 37€ for the Arduino to 71€ for the Elektor Linux Board, it was the most economical choice to choose the Arduino. Some other deciding factors is that the Elektor Linux Board, as well as the Netduino plus 2, require a memory card to function, in addition to the Arduino's large community which can be helpful if needing assistance. It is however important to mention that there are other types of hardware that might be more suitable for a service like watchdog. The Arduino is in reality too powerful for a simple task such as a watchdog; the decision to use it was made due to the availability of this hardware in the lab. We can easily transplant our program to another simpler device in the future if necessary. [45]

3.3 Overview of Software

This subchapter will present the operating system (OS) and other software which has been chosen for our embedded system as well as the reason for choosing the specific types.

3.3.1 Operating System for Raspberry Pi

Table 3 contains an overview of free operating systems that supports the Advanced RISC Machine (ARM) CPU architecture. Popular operating systems such as Windows and OSX is not listed as they are not free, and do not support the type of architecture which is needed. Table 3 also contains a combination of suggestions from Distrowatch.com [46] and elinux.com [47]. It is important to note that not all of the distributions from the latter source list are taken into consideration as they do not support ARM Hard Float (armhf).

Table 3 - Operating systems for ARM processors

General Linux Distributions	Lightweight OS	Unix-like OS	Entertainment	Special Features
Alt Linux	Arch Linux	FreeBSD	GeeXbox	IPFire
Debian	Bodhi Linux	NetBSD	OpenELEC	Kali Linux
Fedora	Moebius	openSUSE	PiBox	Plan9
Gentoo Linux	PiBang Linux	Slackware Linux	PiMAME	
Lubuntu	Pidora		Raspbmc	
RISC OS	Raspbian		RaspyFi	
Ubuntu	SliTaz		Xbian	
	Tiny Core Linux			

The main board which was chosen for the embedded system runs ARM version 6 (ARMv6) CPU architecture. The natural step was to eliminate the distributions which did not support the current architecture, which is why the following operating systems were removed from the list of candidates: Alt Linux, Debian, Fedora, IPFire, Lubuntu, NetBSD, Slackware Linux and Ubuntu. Ubuntu and Lubuntu supports the ARM version 7 (ARMv7) architecture, but has limited ARMv6 support [48]. IPFire does not support ARMv6 architecture as the developers felt that it would be too slow for their operating system [49]. Slackware Linux does not exist for the Raspberry Pi [50] while NetBSD is under development [51]. Debian and Fedora does not support the Raspberry Pi, but there is however rebuilds of the two operating systems; Raspbian and Pidora.

GeeXbox [52], OpenELEC [53], PiBox [54], Raspbmc [55], RaspyFi [56] and Xbian [57] are distributions that were developed to act as media centers on the Raspberry Pi which is not suited for our systems' specifications. Another OS that was excluded is Kali Linux. Kali Linux is an operating system for network testing, and is made by the same developers who made the live compact disc (CD) tool BackTrack [58]. The PiMame is an OS with a collection of emulated game consoles while the Plan9 OS

is designed to be used as a part of a distributed system [59][60]. None of the mentioned operating systems suited our preferences and was removed.

Limited information could be gathered from reading system specifications. The next step of the process was to install the different operating systems on the Raspberry Pi and check if the installed operating systems supported the basic features which were needed for the embedded system. The installation of RISC OS had several flaws. The operating system performed poorly with frequent crashes. Arch Linux was not able to detect nearby devices using Bluetooth and Bodhi Linux did not have the support for Bluetooth on the Raspberry Pi. Moebius was not able to transmit files even though the pairing had been successfully accomplished. We experienced Wi-Fi errors with several operating systems. Arch Linux, Bodhi Linux, Moebius, PiBang Linux and Pidora all had Wi-Fi connection uses and was able to detect nodes over Wi-Fi, but unable to connect to them.

The Raspbian OS was the only operating system that met the system specifications of our embedded system and passed the basic installation experiments. It experienced some compatibility issues with the current USB-hub which made it unable to boot, but was easily resolved by using another hub. [45]

3.4 Overview of Interfaces

3.4.1 Technologies

The table gives a description of the different values of the properties shared by the different communication systems. The table was used to compare and evaluate the different types of interfaces which were selected for our embedded communication gateway.

Table 4 - Interface comparison

Feature(s)	Ethernet [61]	IEEE 802.11g [62]	Bluetooth [63][64]	ZigBee [65]	3G/4G [66][67]
Power Profile	N/A	Hours	Days	Years	Days
Complexity	Very Complex	Very Complex	Complex	Simple	Very Complex
Nodes/Master			7	64000	
Latency	0.3 ms	Up to 3 sec.	Up to 10 sec.	30 ms	
Range	N/A	100 m	10 m	70-300m	Radio Tower
Data Rate		11Mbps	1Mbps	250Kbps	40Mbps

Most of the interfaces from Table 4 is implemented in our embedded system. One interface which did not get implemented was WiMax. WiMax technology provides fast data rate up to 30-40 Mbps and has a large broadcast area compared to Wi-Fi [68]. There is a high amount of devices which supports the technology, and it may even rival Wi-Fi in some areas, but it was decided to not implement it as it is not a widespread technology in Norway. The Internet service providers which offer WiMax services are NextGentel, NextNet, and Telenor, and this is only in certain areas of the country [69]. The WiMax wireless communication standard is supported only by 4G era technology, which also only exists within the largest city in Norway and is costly. We believe it that it would be better to wait until it becomes a more widespread technology, and more people takes interest in it.

The chosen interfaces for our system are Ethernet, Wi-Fi, Bluetooth, Zigbee and 3G/4G. We would like our embedded system to support multiple types of interfaces to make it a universal device which may be used in several situations. The user is then able to choose between the interfaces that may suit his or her needs. Ethernet is the most common and widespread technology used in LAN. It provides the network with high data transfer rate up to 100Gbps as well as security, as people would need to be physically connected to do any damage.

The Ethernet technology is constrained by cables and it is the reason why Wi-Fi is included in the embedded system. Wi-Fi has the advantage of providing mobility, and allows LANs to be easily deployed while being cost efficient. The use of Wi-Fi will remove hurdles such as cabling, drilling and setting up multiple switches as access points.

Bluetooth offers the same advantages as Wi-Fi, but at a much shorter range. Bluetooth was developed to be used by portable equipment and its application. The main difference between the two technologies is that Wi-Fi is dependent on access points. In addition Bluetooth demands low processing power, has low battery usage and, and gives little signal interference.

The next interface which is implemented on the communication gateway is ZigBee. ZigBee is developed for the purpose of communication on a sensor network, and is a low power intensive interface. Compared to Bluetooth, ZigBee has the ability to transmit data over longer distance by use of multi-hop communication. Multi-hop communication gives the node the capability of forwarding information.

The last interface which is going to be implemented in our system is 3G and 4G mobile broadband technology. The technology is usually used by mobile phones and other mobile devices. The technology comes with universal global roaming, and supports different types of multimedia

services. Mobile broadband is one of the most important features in our system, as it provides a second connection to the Internet. The interface provides better security than Wi-Fi, and allows a data transfer rate up to 40 Mbps while the device is stationary, which makes it an exceptional candidate for replacing DSL during downtime [67].

During the process of selecting the different types of candidates we thought about how the mentioned interfaces could cover their strength and weaknesses. We would like users to choose between the different types of interface suited for their situation. By implementing multiple interfaces our embedded system may be used in a wide area of different tasks and makes our gateway universal. [45]

3.4.2 Communication Dongles

The embedded communication gateway provides several different communication interfaces. It requires several dongles which can serve as antennas for the respective interface. Functional dongles was acquired by recommendations from the Linux community. The system is currently using ZTE MF821D Telenor as an antenna to provide mobile broadband, and reason for choosing the specific dongle was due to our connection with Telenor. They allowed us to borrow and test the dongle and offered us a special discount. The communication gateway serves as a wireless access point and is connected to a Edimax EW-7811Un. It was however not our first choice when choosing dongles. The D-Link DWL-123 Wi-Fi dongle was installed and worked as a client, but did not support the feature of serving as an access point for other devices.

The Bluetooth dongle which was chosen for our gateway was the ASUS USBBT211. The Bluetooth was accessible at the current time and had been used by several users in the Raspberry Pi community. The ZigBee interface was provided by a ZigBee module named XBee, which required an additional circuit board to function. We had access to the Xbee module at the lab and the circuit board was ordered online as it was manufactured for the Raspberry Pi.

4 Implementation

4.1 Software Installation and Interface Initialization

4.1.1 Operation System Installation

When having a brand new Raspberry Pi at hand, one need to install an operating system. There are different methods on how to do this, depending on which OS is used. As we chose to use Raspbian, the method is quite simple. By downloading an image file and extracting this onto the SD card using another computer, the installation process is almost done.

The Raspberry Pi will boot immediately after inserting the SD card and being connected to the power supply. First the operating system will load, and then the script “raspi-config” will be shown. Since this is the first boot of the Raspberry Pi, it would be recommended to do some basic configuration before getting started using the OS.

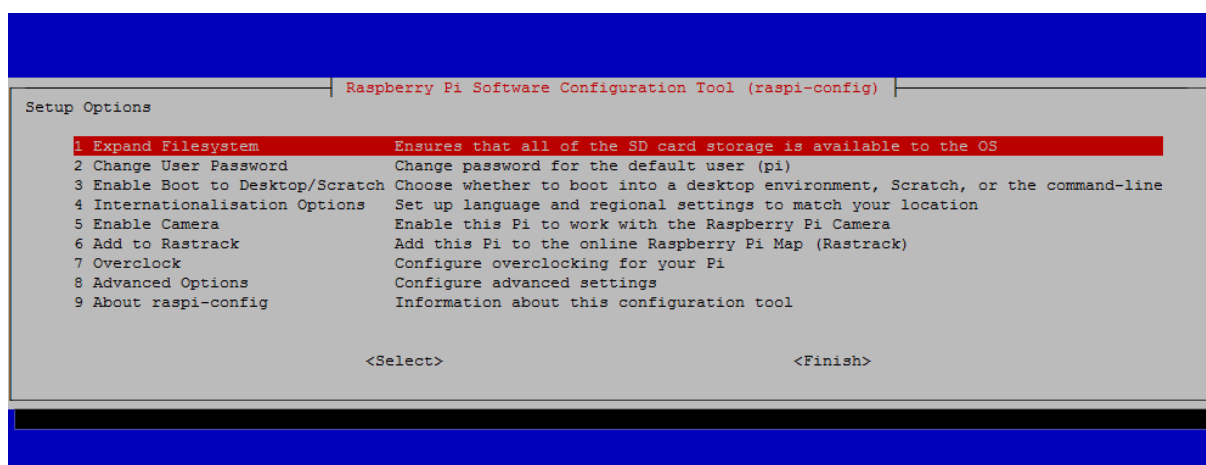


Figure 5 - Raspi-config main menu

Most of the configurations in this script may be changed later, but we recommend to do some changes as soon as possible. Following is a list of configurations we recommend changing, where the naming is based the menu shown in Figure 5. Also note that some menu choices are unmentioned, as they are irrelevant for our setup.

- 1 Expand Filesystem: It is advisable to expand the file system, to make sure no space is unallocated and then unusable by the OS.
- 2 Change User Password: The default password for the user “pi” is “password”, but to improve the security, this should be changed.

- 3 Enable Boot to Desktop/Scratch: If one wish to use the graphical user interface provided by Raspbian, it is necessary to enable that in the menu. By default the RPi will boot to the text console.
- 4 Internationalisation Options

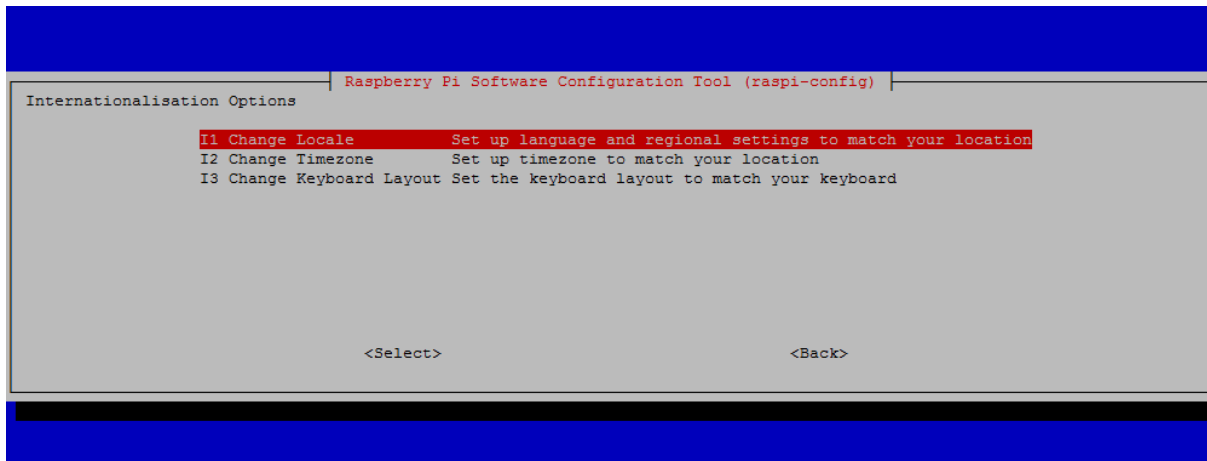


Figure 6 - Raspi-config internationalisation options

- I2 Change Timezone: To get the clock correct one has to change the time zone setting.
- I3 Change Keyboard Layout: By default the keyboard layout is set to “English”, and should be changed if the keyboard has another character set.
- 7 Overclock: In order to speed up the process of upgrading existing and installing new software, it is possible to overclock the CPU, core, SDRAM and voltage. However, it might weaken the stability and reduce the lifetime of the Raspberry Pi.

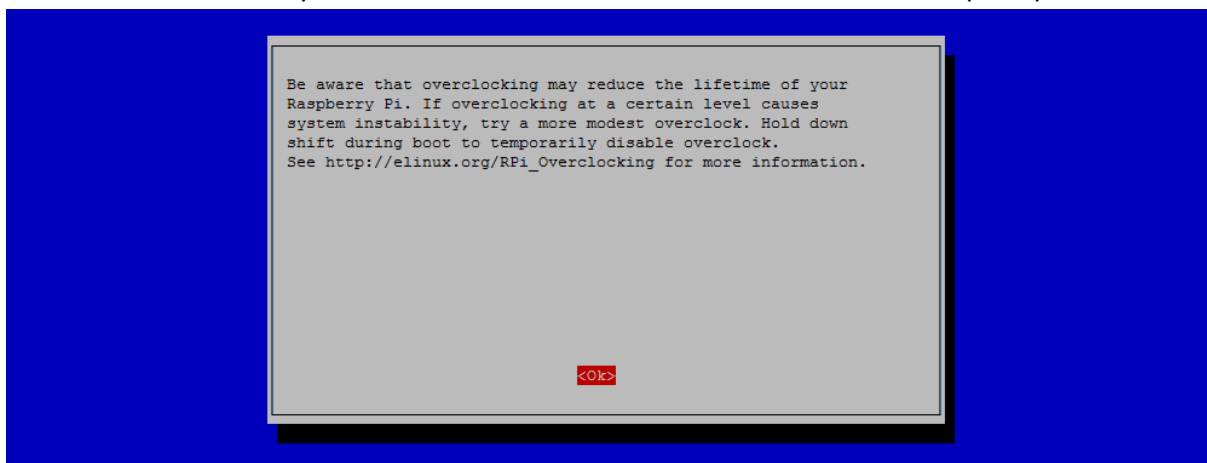


Figure 7 - Raspi-config overclock warning

- The Raspberry Pi Foundation has made some improvements, and enabling the highest level of overclocking will not void the warranty anymore [70]. As stability is a great factor in this implementation, we recommend to be careful when overclocking, but if it is only for a short

period of time, it can be a way to save some time.

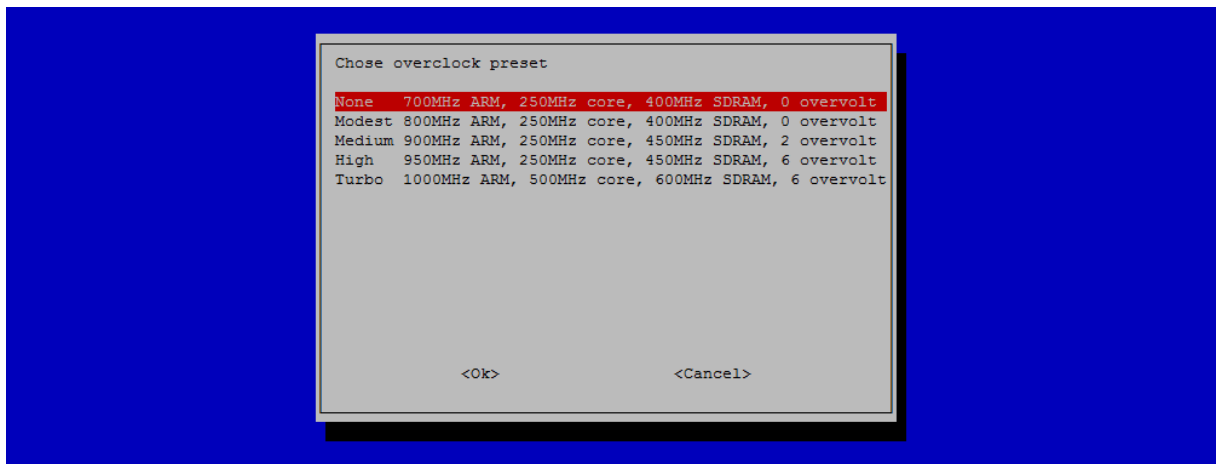


Figure 8 - Rasp-config overclock menu

- When we did the installation of software, we used the preset called “High”, as the likelihood of stability issues increases with heavier overclocking. In earlier firmware versions it was a common problem with SD card corruption when using the “Turbo” preset, but this should have been fixed in newer firmware versions [71].
- 8 Advanced Options

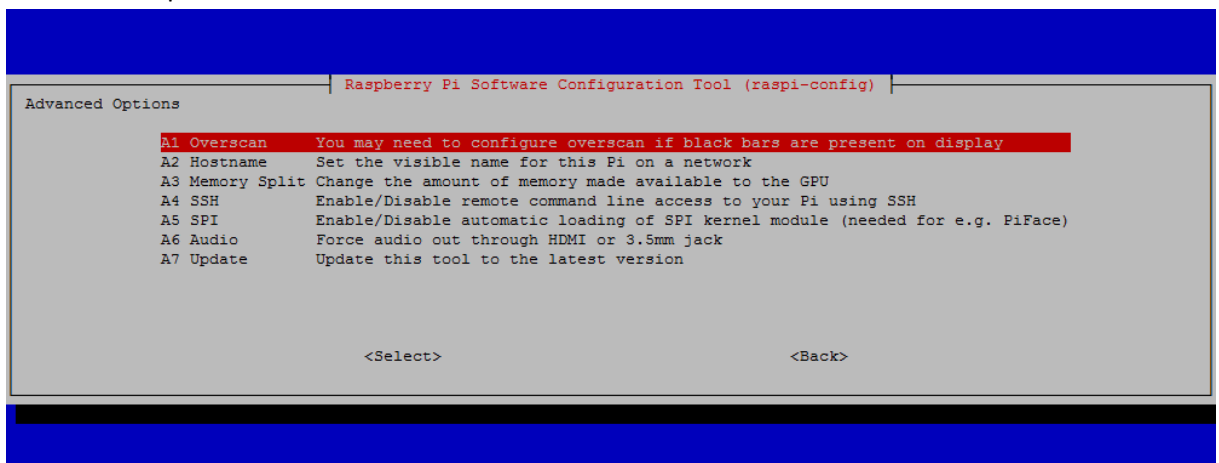


Figure 9 - Rasp-config advanced options

- A4 SSH: To control the Raspberry Pi remotely in the future, we suggest to enable the SSH server. Until this is done, a keyboard and a monitor is required to control the system.

To exit the “**raspi-config**” script, choose “Finish” and answer yes to reboot the Raspberry Pi.

During the development of this system, there has been a lot of fresh installs. Therefore we made a script to automate the installation. This script is somewhat interactive in the way that it asks the user which features to install, and saves the configuration details in between reboots. Then it is able to

continue where it left off and without the need of the same input twice[72]. The complete script can be read in Appendix 4.

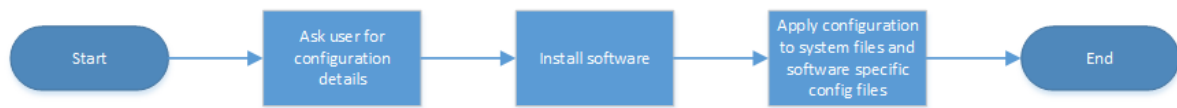


Figure 10 - Basic flow diagram for the installation script

In general the installation script will ask the user for input, followed by installation of software and finally changing configuration files according to the collected input.

We will not explain the script in detail, as the interested reader can study it in the appendices. However, an installation guide of different functions will be provided. The installation script contains some functionality that can be useful, such as selection of system functionality and e-mail notification when user input is needed.

Before installing anything, one should get an updated version of the package lists from the repositories. This can be done in console by running “**sudo apt-get update**”.

After the update, the application manager will know if there are any new versions of the installed packages on the computer. To install the newest versions of everything available from the package repositories, run the command “**sudo apt-get upgrade -y**”. This will take a while, but afterwards the system will be ready for the software installation.

4.1.2 Mobile Broadband Software

The mobile broadband can be difficult because the hardware often run in different modes. First it will start in storage mode which allows the user to install the supported driver software, but then one will have to enable the modem mode. In Windows this usually is no problem, but in Linux one will have to reboot or reconnect the device. Even then there might be an issue. Reconnecting the device will not always work, as it stays in storage mode after connecting it to the system. Also, if the modem is connected to a self-powered USB-hub (as we recommend), one will have to disconnect the external power supply, or the modem will not detect the reboot.

A third alternative is to use a program named “usb-modeswitch”. It needs some configuration, but it is much more effective to use than any of the previously mentioned methods.

As a step in the configuration you will have to run “**lsusb**” to get a list of USB-devices connected to the Raspberry Pi [73].


```
pi@raspberrypi ~ $ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 002: ID 0424:9512 Standard Microsystems Corp.
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 004: ID 05e3:0608 Genesys Logic, Inc. USB-2.0 4-Port HUB
Bus 001 Device 005: ID 0cf3:3005 Atheros Communications, Inc. AR3011 Bluetooth
Bus 001 Device 006: ID 19d2:0166 ZTE WCDMA Technologies MSM
```

Figure 11 - Output from "lsusb"

Find the modem in the list and take a note of the vendor ID and product ID. In Figure 11 these values are given as 19d2:0166 (the vendor ID and the product ID, respectively, separated with a colon), but these values will be different for other devices[73].

If an externally powered USB-hub is used, disconnect the power supply and then reboot. After rebooting the system, the power can be reconnected to the hub. Then, run "lsusb" again.

```
pi@raspberrypi ~ $ lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 001 Device 002: ID 0424:9512 Standard Microsystems Corp.
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp.
Bus 001 Device 004: ID 05e3:0608 Genesys Logic, Inc. USB-2.0 4-Port HUB
Bus 001 Device 005: ID 7392:7811 Edimax Technology Co., Ltd EW-7811Un 802.11n Wireless Adapter [Realtek RTL8188CUS]
Bus 001 Device 006: ID 0cf3:3005 Atheros Communications, Inc. AR3011 Bluetooth
Bus 001 Device 008: ID 19d2:0167 ZTE WCDMA Technologies MSM
```

Figure 12 - Output from "lsusb" after reboot

This time, the vendor and product IDs should have changed. In Figure 12 they are 19d2:0167. Note that the vendor ID does not have to change. Again, take a note of these values as they will be used at a later stage[73].

To install the software, run the command "sudo apt-get install ppp wvdial usb-modeswitch -y". "wvdial" is creating the connection to the mobile broadband provider and "ppp" is a program responsible for creating a tunnel using the ppp protocol.



Now that usb-modeswitch is installed, it can be configured. The command "tar -xzf /usr/share/usb_modeswitch/configPack.tar.gz 19d2\ :0166 && cat 19d2\ :0166 | grep MessageContent" will output one or more lines to be used in the configuration [73]. Note that "19d2" and "0166" has to be replaced with the correct values for vendor ID and product ID from before the reboot (when the modem was in storage mode). This output is used when editing the configuration file /etc/usb_modeswitch.conf [73]. If using text console and not the GUI, the file can be changed using "sudo nano /etc/usb_modeswitch.conf".

```

GNU nano 2.2.6 File: /etc/usb_modeswitch.conf
DefaultVendor=0x19d2
DefaultProduct=0x0166

TargetVendor=0x19d2
TargetProduct=0x0167

MessageContent="55534243123456782400000080000685000002400000000000000000000"

[ Read 7 lines ]
^G Get Help      ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text    ^T To Spell

```

Figure 13 - Content of usb_modeswitch.conf

Create the file as given in Figure 13, but use the vendor IDs, product IDs and the message content values from earlier. As the configuration file is complete, usb-modeswitch can be run with “**sudo usb_modeswitch -c /etc/usb_modeswitch.conf**” [73].

The only thing left before being able to use the mobile broadband, is the configuration of wvdial. The command “**wvdialconf**” creates a file (**/etc/wvdial.conf**) which can be edited using “nano”. There are three variables in this configuration file that depends on the mobile broadband provider. As we are using Telenor, the APN, username and password is all “**telenor**”. When using another provider, these settings will have to be changed accordingly.

```

GNU nano 2.2.6 File: /etc/wvdial.conf
[Dialer telenor]
Init1 = ATZ
Init2 = ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0
Init3 = AT+CGDCONT=1,"IP","telenor"
Stupid mode = 1
Modem Type = Analog Modem
ISDN = 0
Phone = *99#
Modem = /dev/ttyUSB2
Username = {telenor}
Password = {telenor}
Baud = 9600

[ Read 12 lines ]
^G Get Help      ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify      ^W Where Is     ^V Next Page    ^U UnCut Text    ^T To Spell

```

Figure 14 - Content of wvdial.conf

As soon as the file is similar to Figure 14, connection can be tried using “**wvdial telenor**”, where “**telenor**” in this command is the apn name for the mobile broadband provider [73]. Press Ctrl+C to disconnect the mobile broadband connection.

4.1.3 Wi-Fi Access Point

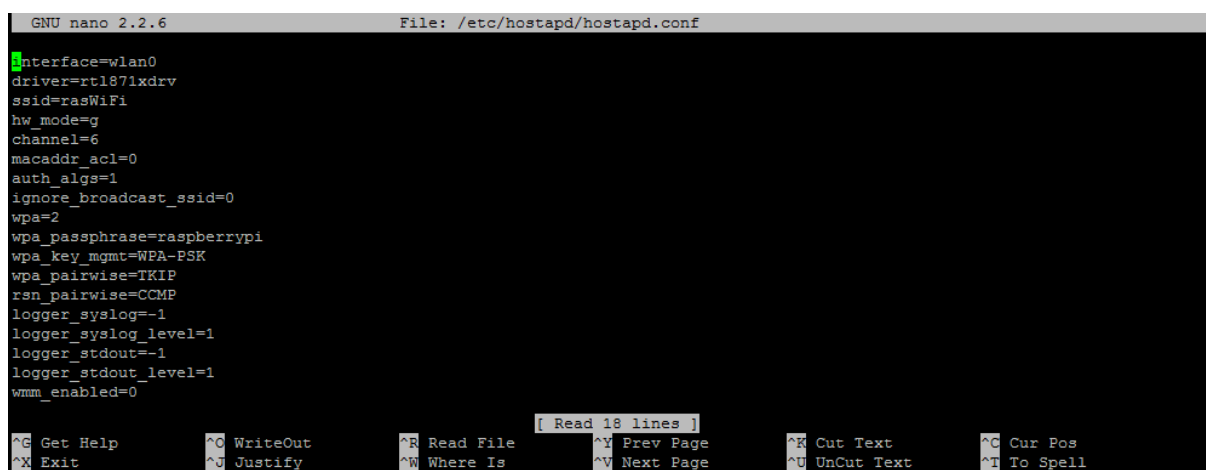
If the objective with the system is to be a router, it is probably more useful to implement a wireless access point instead of client functionality. It is only possible to use one of these features, as the programs **wpa_supplicant** and **hostapd** is having counter-effects on each other. The latter is used to

handle the authentication and connection functions in the access point. The setup of Wi-Fi client is described in Appendix 7.

To install hostapd, run “**sudo apt-get install hostapd -y**” [74]. This will install the necessary files to use a Linux computer as a Wi-Fi access point. Unfortunately, this is not completely supported by the Raspberry Pi. Therefore a modified version is needed. At the time of writing this thesis, the modified hostapd could be downloaded and installed using the following commands:

```
wget http://www.adafruit.com/downloads/adafruit_hostapd.zip
unzip adafruit_hostapd.zip
sudo mv /usr/sbin/hostapd /usr/sbin/hostapd.ORIG
sudo mv hostapd /usr/sbin/hostapd
sudo chmod 755 /usr/sbin/hostapd
```

The configuration process of hostapd is done by creating a configuration file. To edit or create the configuration file, run “**sudo nano /etc/hostapd/hostapd.conf**”. If the file does not exist yet, it will be created.



```
GNU nano 2.2.6 File: /etc/hostapd/hostapd.conf
interface=wlan0
driver=rtl871xdrv
ssid=rasWiFi
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=rasberrypi
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
logger_syslog=-1
logger_syslog_level=1
logger_stdout=-1
logger_stdout_level=1
wmm_enabled=0
```

Figure 15 - The contents of the configuration file for hostapd

Inside the configuration file, write exactly the same as shown in Figure 15, except for the lines with “**ssid**” and “**wpa_passphrase**” [74]. This is where the name of the network and the password needed to connect is inserted. In Figure 15 the name of the network is “rasWiFi” and the WPA password is “rasberrypi”. If the connection is slow, changing the value for “channel” can solve the problem, as the frequency may be less disturbed by other channels. The best value is depending on the given environment.

The completion of the access point setup is described in Section 4.1.5 as some of the steps are depending on whether the extra Ethernet port is going to be installed or not.

4.1.4 Ethernet

The main reason for installing the extra Ethernet port would be for routing purposes in a larger network. As the Wi-Fi interface has a physical limitation on the amount of concurrent users, the users on cabled network will not be affected by each other. On Wi-Fi, traffic from all other users will be received as noise, which will cause collisions and degraded signal.

Other reasons for a limited Wi-Fi signal are noise from home appliances and limited signal range. In addition to these factors, the signal is vulnerable to building structures. Ethernet is much more resistant for external noise and therefore more stable. It also supports communications over a longer range.

In order to install the extra Ethernet interface, connect the adapter and run the following commands:

```
sudo ifconfig eth1 up  
sudo ifconfig eth1 10.0.1.1
```

These commands will turn on the Ethernet port (if it is not on already) and set its IP address to 10.0.1.1. For Ethernet clients this will be the address to their gateway.

As the next steps are different if the Wi-Fi access point is installed, the final part of the setup is described in Section 4.1.5.

4.1.5 Routing

This subsection is only relevant when installing the Wi-Fi access point, the second Ethernet port or both.

4.1.5.1 Network settings

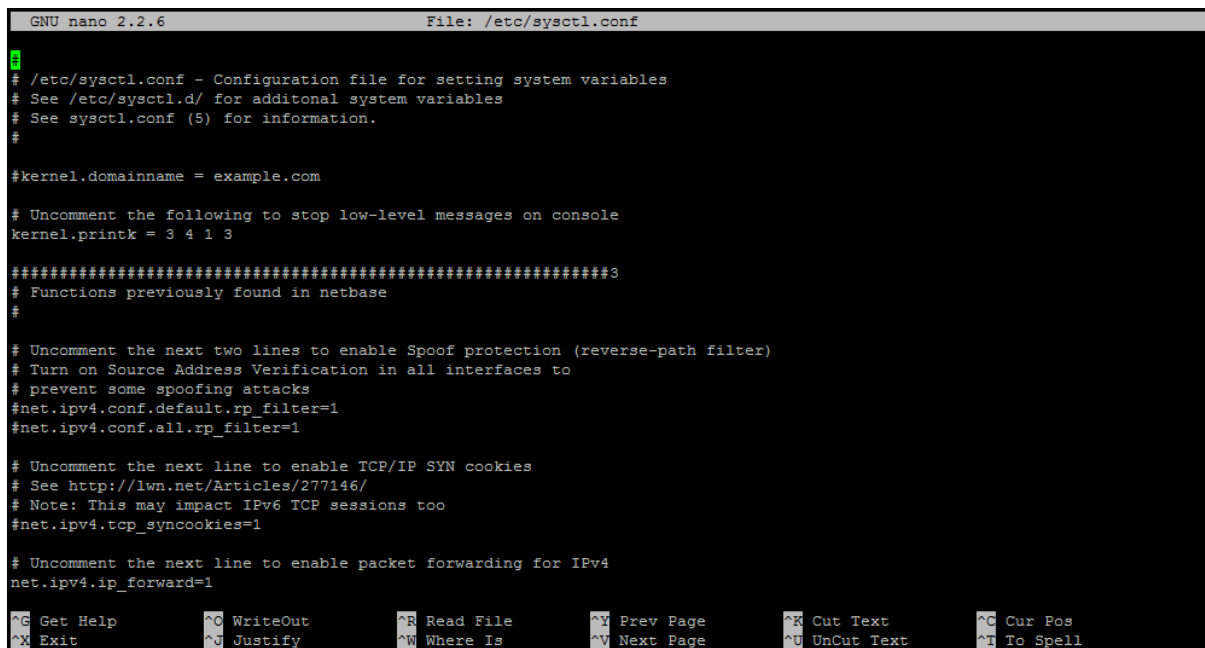
In order to be able to route network packets between the Internet and the local network, Linux need to enable forwarding. This is done by editing a file called “`/etc/sysctl.conf`” and running some commands afterwards.

In the configuration file, uncomment the line containing “`net.ipv4.ip_forward=1`”, so it becomes identical to the bottom line in Figure 16 [74].

To finalize setting up the forwarding, run the following lines of code [74]. The first line will activate forwarding in Linux, and the rest is setting up iptables.

```
sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"  
sudo iptables --flush  
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

```
sudo iptables -A FORWARD -i eth0 -o eth1 -m state --state RELATED,ESTABLISHED
sudo iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
sudo iptables-save > /etc/iptables.rules
```



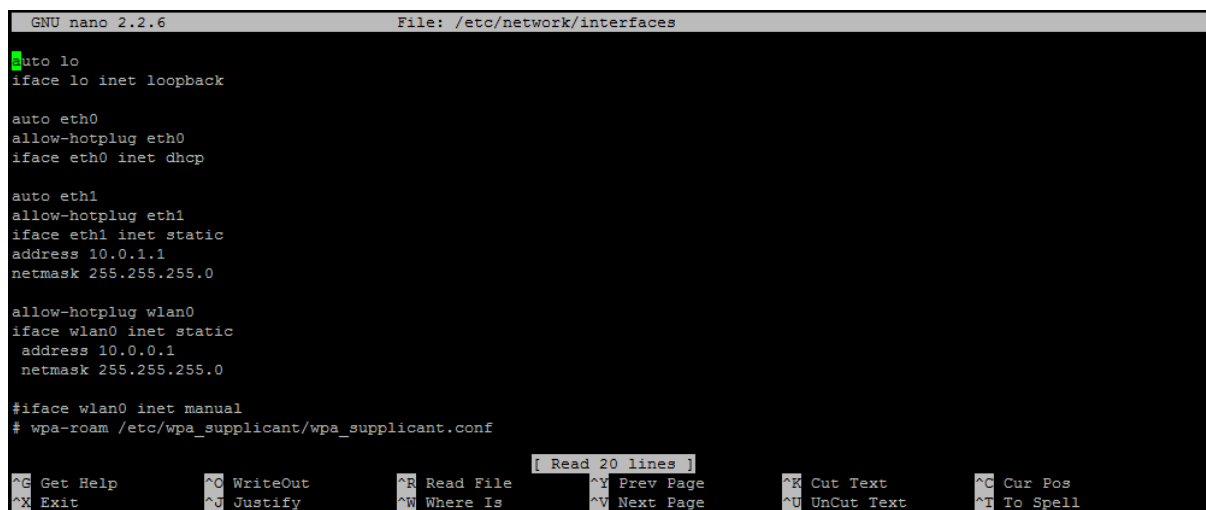
```
GNU nano 2.2.6 File: /etc/sysctl.conf
# /etc/sysctl.conf - Configuration file for setting system variables
# See /etc/sysctl.d/ for additional system variables
# See sysctl.conf (5) for information.
#
#kernel.domainname = example.com
# Uncomment the following to stop low-level messages on console
kernel.printk = 3 4 1 3
#####
# Functions previously found in netbase
#
# Uncomment the next two lines to enable Spoof protection (reverse-path filter)
# Turn on Source Address Verification in all interfaces to
# prevent some spoofing attacks
#net.ipv4.conf.default.rp_filter=1
#net.ipv4.conf.all.rp_filter=1
# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1
# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1
^G Get Help      ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit         ^J Justify      ^W Where Is    ^V Next Page    ^U UnCut Text  ^T To Spell
```

Figure 16 - The first part of the file "/etc/sysctl.conf"

Iptables is a firewall in Linux, which is also used to help forward traffic. “`iptables -t nat`” sets up NAT, so the local network addresses is disguised for an external viewer. The parameters “`-i eth0 -o eth1 -m state --state RELATED,ESTABLISHED`” is blocking traffic from the outside which is not from an already existing connection, while “`-i eth1 -o eth0 -j ACCEPT`” lets through all traffic from the local network to the Internet. If only the Wi-Fi access point is in use, and not the extra Ethernet port, please replace `eth1` in this example with `wlan0`. If both are in use, it is unnecessary to add `wlan0` to the iptables rules.

As the forwarding part is now complete, it is necessary to set some interface specific settings. These are set in the file “`/etc/network/interfaces`” [75].

If both the access point and the Ethernet is set up, the file from Figure 17 can be directly copied. If only one is being used, just skip the part of the file regarding the interface not in use.



```
GNU nano 2.2.6 File: /etc/network/interfaces
auto lo
iface lo inet loopback

auto eth0
allow-hotplug eth0
iface eth0 inet dhcp

auto eth1
allow-hotplug eth1
iface eth1 inet static
address 10.0.1.1
netmask 255.255.255.0

allow-hotplug wlan0
iface wlan0 inet static
address 10.0.0.1
netmask 255.255.255.0

#iface wlan0 inet manual
# wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
```

Figure 17 - The content of the file `"/etc/network/interfaces"`

4.1.5.2 DHCP Server

In modern IPv4 networks, there is almost always a DHCP server present. Since devices need an IP address which corresponds to the current network in order to be able to communicate, it is quite impractical to use static IP addresses. To add DHCP functionality, start with installing the DHCP server:

```
sudo apt-get install isc-dhcp-server -y
```

This command will probably give an error message as the DHCP server will not be able to start before the configuration is complete. First open the file `"/etc/default/isc-dhcp-server"` and find the line starting with `"INTERFACES="`. Add the names of the interfaces being used inside the quotation marks. Use a space to separate the names if using more than one. For instance, the line will be `"INTERFACES="wlan0 eth1"` if both interfaces is going to be used.

The rest of the configuration is done when editing the file `"/etc/dhcp/dhcpd.conf"`.

At the start of the configuration file, comment the lines with `"option domain-name"` and `"option domain-name-servers"`, and uncomment the line containing `"#authoritative;"`. The result should be similar to Figure 18.

Before quitting the text editor, the subnet declarations for the interfaces you want to use must be added. These are added to the end of the same file. If both the Wi-Fi access point and extra Ethernet interface is to be used, the end of `"dhcpd.conf"` should look like Figure 19.

```
GNU nano 2.2.6 File: /etc/dhcp/dhcpd.conf
# Sample configuration file for ISC dhcpd for Debian
#
#
# The ddns-update-style parameter controls whether or not the server will
# attempt to do a DNS update when a lease is confirmed. We default to the
# behavior of the version 2 packages ('none', since DHCP v2 didn't
# have support for DDNS.)
ddns-update-style none;

# option definitions common to all supported networks...
#option domain-name "example.org";
#option domain-name-servers ns1.example.org, ns2.example.org;

default-lease-time 600;
max-lease-time 7200;

# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.
authoritative;
```

Figure 18 - The start of the file "/etc/dhcp/dhcpd.conf"

On the other hand, if only the Wi-Fi access point is installed, then only add the first half with “**subnet 10.0.0.0**” (including everything inside the first set of curly braces). Alternatively, if the access point is skipped, but the extra Ethernet interface is installed, add the last part of Figure 19, starting with “**subnet 10.0.1.0**”.

```
GNU nano 2.2.6 File: /etc/dhcp/dhcpd.conf
subnet 10.0.0.0 netmask 255.255.255.0 {
range 10.0.0.10 10.0.0.59;
option broadcast-address 10.0.0.255;
option routers 10.0.0.1;
default-lease-time 600;
max-lease-time 7200;
option domain-name "local";
option domain-name-servers 8.8.8.8, 8.8.4.4;
}

subnet 10.0.1.0 netmask 255.255.255.0 {
range 10.0.1.10 10.0.1.59;
option broadcast-address 10.0.1.255;
option routers 10.0.1.1;
default-lease-time 600;
max-lease-time 7200;
option domain-name "local";
option domain-name-servers 8.8.8.8, 8.8.4.4;
}
```

Figure 19 - The end of the file "/etc/dhcp/dhcpd.conf"

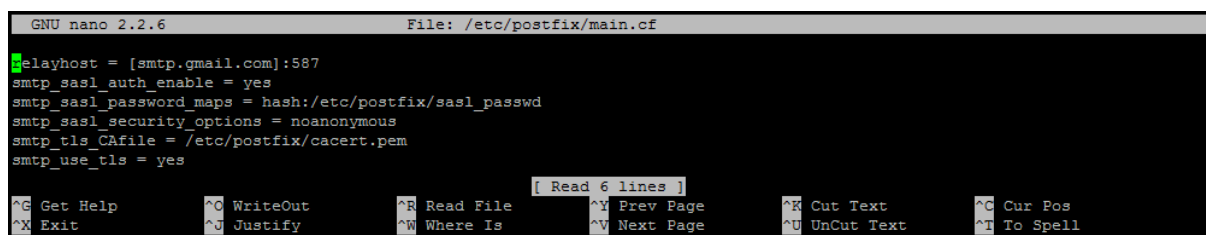
The interfaces and DHCP server are now ready. To start the DHCP server and make it start during boot, run the two following commands:

```
sudo service isc-dhcp-server start
sudo update-rc.d isc-dhcp-server enable
```

The first line is only starting the DHCP server for this session, and the second line makes it start at every boot.

4.1.6 E-mail

If the user wants to be notified on special events, such as on errors or when the main Internet connection is lost, the gateway can send an e-mail to inform the user automatically. To do so, an e-mail server can be installed. This is purely for sending mail, and not for receiving. To install the required packages, run “`sudo apt-get install postfix mailutils libsasl2-2 ca-certificates libsasl2-modules -y`” [76]. Postfix is the program used to send e-mail, and has to be configured using the file `/etc/postfix/main.cf`. In this example we are using a Gmail address, but it is possible to use most other e-mail accounts as well.



```
GNU nano 2.2.6 File: /etc/postfix/main.cf
relayhost = [smtp.gmail.com]:587
smtp_sasl_auth_enable = yes
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
smtp_sasl_security_options = noanonymous
smtp_tls_CAfile = /etc/postfix/cacert.pem
smtp_use_tls = yes
[ Read 6 lines ]
^G Get Help      ^O WriteOut     ^R Read File    ^Y Prev Page    ^K Cut Text     ^C Cur Pos
^X Exit          ^U Justify      ^W Where Is    ^V Next Page    ^U UnCut Text   ^T To Spell
```

Figure 20 - The main configuration of Postfix

If using Gmail, the configuration file should be identical to Figure 20 [77][76]. To finish the setup of postfix, run the following commands. Make sure the “`USERNAME`” and “`PASSWORD`” fields are replaced with the correct login credentials.

```
sudo echo "[smtp.gmail.com]:587 USERNAME@gmail.com:PASSWORD" >
/etc/postfix/sasl_passwd
sudo chmod 400 /etc/postfix/sasl_passwd
sudo postmap /etc/postfix/sasl_passwd
sudo cat /etc/ssl/certs/Thawte_Premium_Server_CA.pem | sudo tee -a
/etc/postfix/cacert.pem
sudo /etc/init.d/postfix reload
```

Now that postfix is completely installed, e-mails can be sent using “`echo “E-mail text” | mail -s “Subject” example@example.com`”, where `E-mail text` is the body of the mail, `Subject` is the subject and `example@example.com` is the receiver of the e-mail [78].

4.1.7 Bluetooth

In this setup, the Bluetooth interface is listening for incoming files and will automatically save them to a directory. By default the Bluetooth agent will need to pair with the sending device, in addition to accepting the file transfer, with help of a user input. These issues are solved in the following installation description.

For this section, only three modules will be installed. This is done by running “`sudo apt-get install bluetooth bluez-utils obexpushd -y`” [79][80]. The third of these packages

(**obexpushd**) is the daemon responsible for accepting and receiving files over Bluetooth. The configuration is done by creating a file with `mkdir /home/pi/bluetooth && nano /etc/init/obexpushd.conf` and adding the following lines to this file [80].

```
chdir /home/pi/bluetooth
exec obexpushd -n
start on startup
```

Save the file and run `sudo initctl reload-configuration` and `sudo start obexpushd`, and the system will be automatically receive all files sent from paired devices [80]. As previously mentioned, the system needs some user input to pair with a Bluetooth device. This is circumvented by running the following commands [81].

```
sudo hciconfig hci0 piscan
sudo hciconfig hci0 sspmode 1
```

The first command will enable Bluetooth visibility and the second will turn off the PIN code when pairing. When these commands are run, they do not set the visibility and security modes permanently. If the Bluetooth should always stay in these modes, the commands may be added to a startup file using `sudo nano /etc/rc.local`. Please note that the last line should be `exit 0`, and any extra code should be inserted before this line.

When files are received they will be saved in the previously given folder. This dissertation is not covering an automatic backup or file transfer in detail, but we have made an example where the user can have the incoming files sent by mail (given that the mail feature is installed in Section **Feil! Fant ikke referansekinden**). This example is easy to modify by changing the `echo` and `mail` commands inside the `while read file` loop. When executing the following code, the folder for incoming files will constantly be monitored and files automatically sent to an e-mail address.

```
#!/bin/bash
while true
do
    cd /home/pi/bluetooth
    ls > /tmp/bluetoothfiles.tmp
    while read file
    do
        echo "File received from Bluetooth client" | mail -s "New Bluetooth
file" -a $file test@example.com
        rm $file
    done </tmp/bluetoothfiles.tmp
    sleep 10
done
```

In our code example above the `ls` command is run within the folder `/home/pi/bluetooth` and the output is printed to a temporary file. In each line in this temporary file, a name of a file is listed, and for each file an e-mail will be sent to the e-mail address `test@example.com`. Then the file will be

deleted to make sure it is not sent again and to prevent wasting disk space. This whole process is repeated every 10 seconds, by the use of the `sleep` command.

4.1.8 ZigBee

The hardware part of the ZigBee was installed in Section 4.5.3. This part of the dissertation will explain how to configure the XBee to join a PAN network with other nodes. The Xbee may be configured in different ways, it may be done through a software on a remote computer or through the Raspberry Pi itself. This part of the report will cover how to do it on a remote computer which does not run a Linux operating system.

The first step to configure one's XBee is to download and install the open source application for XBee named XCTU on the remote computer. The application is a free multi-platform application that is both compatible with Windows and MacOS and offers simple wireless network configuration tools for the user [82].

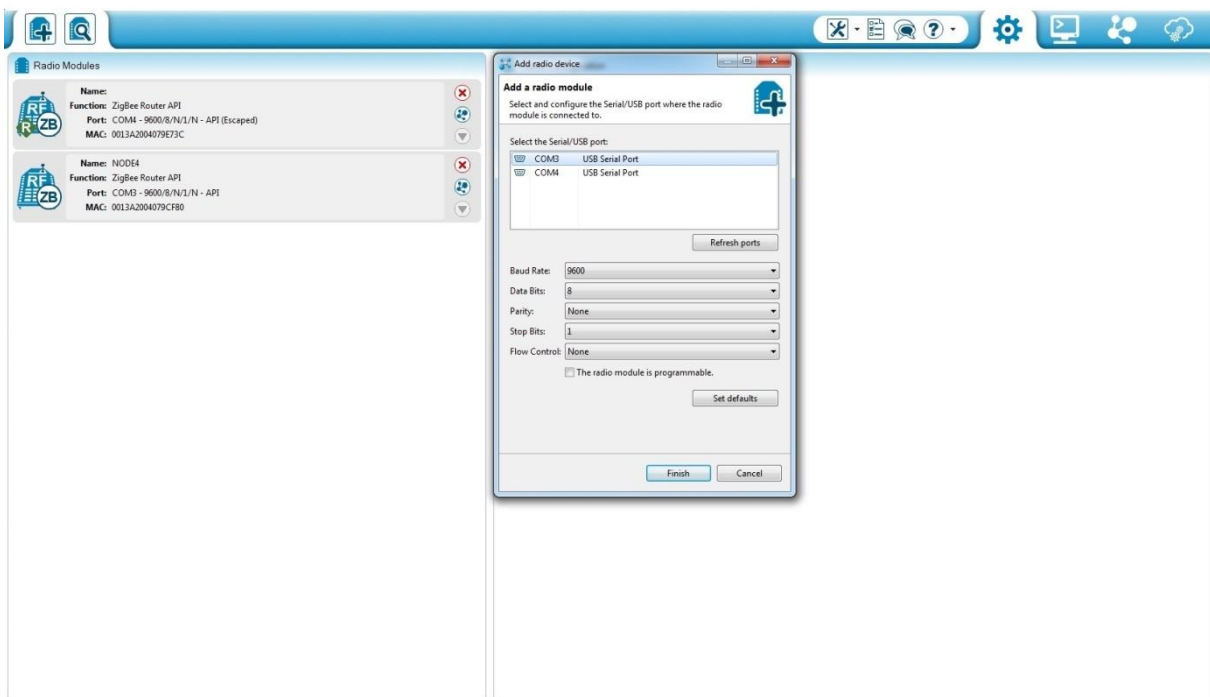


Figure 21 - XCTU initial launch

During the installation the XBee module can be prepared and connect them to a remote computer by use of a serial/USB port. The application will not automatically search for the modules and the search must be started manually by pressing the button in the top left corner with a plus sign. A window is shown in Figure 21. This window allows the user to add a radio device to the application. Choose the

respective serial port and press finish. It will most likely be able to find the device if it is new and in its default settings. However if the baud rate has been changed by another user, it might be necessary to try and go through all the different baud rate options, unless the specific number is known.

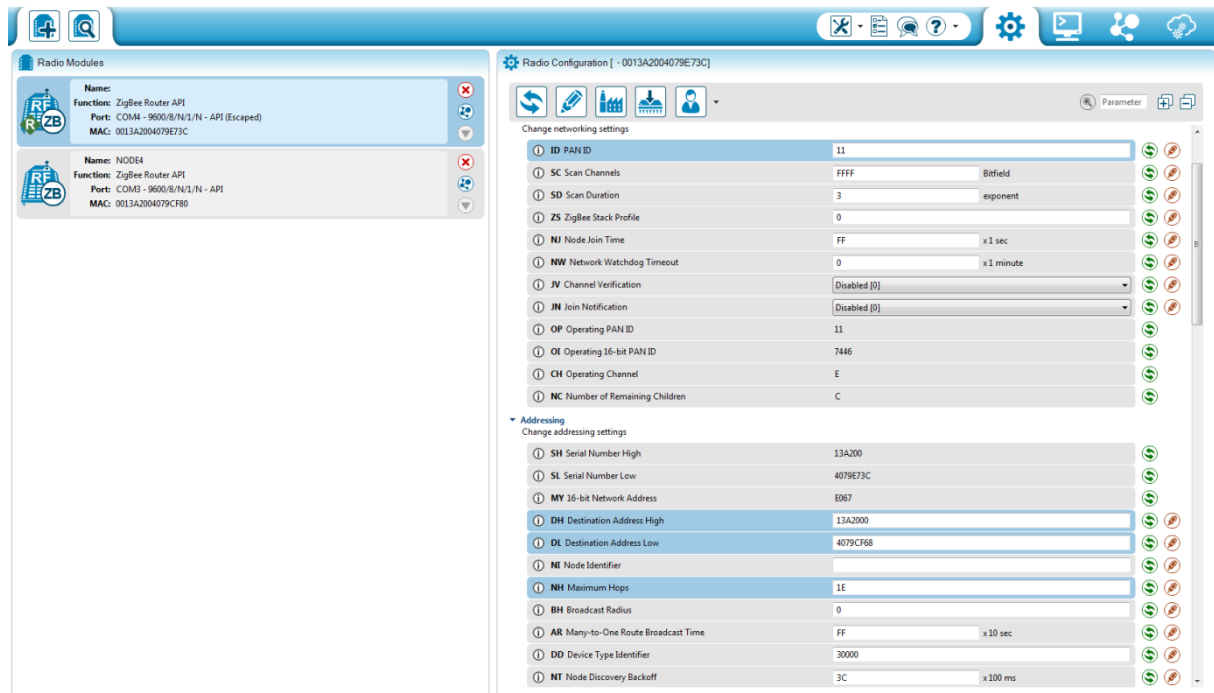


Figure 22 - XCTU XBee Configuration

After discovering the XBee module it can be configured. As seen in Figure 22 there are a lot of configuration parameters that may be set. The most important ones, however, are ID, CH, SH and SL, MY, DH and DL, and BD. The ID (PAN ID) allows the user to specify which networks the module is supposed to join. In other words, all the modules which are going to be in the same network needs to have the same ID. The CH (Operating channel) lets one specify the channel that the ZigBee modules will use while SH and SL specifies the unique address of the modules. The SH and SL can however not be changed. DH and DL allows setting destination address high, and destination address low. The addresses are found on the back of the XBee and will be given to the adjacent XBee in the PAN. The baud rate (BD) is set to 9600 which is the default rate to communicate. As we have not determined what to use the sensors for we use the default rate. An example were you could change it, would be in relation to a Nintendo Wii remote were a fast baud rate is necessary.

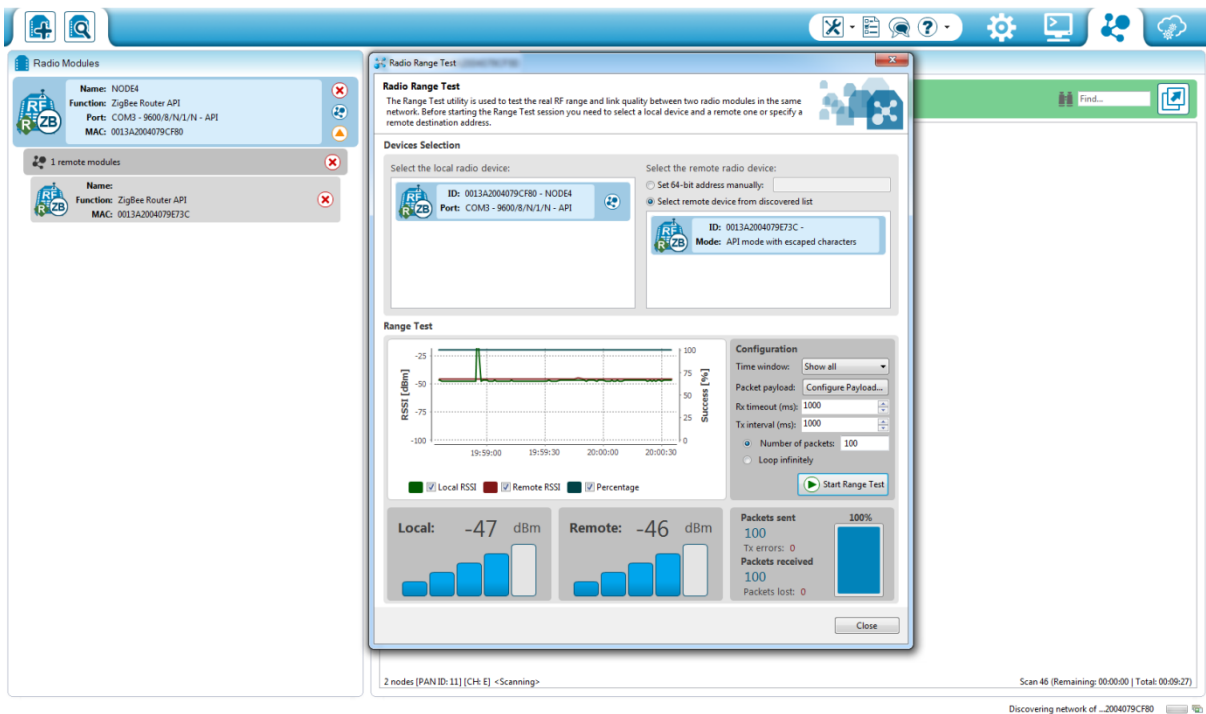


Figure 23 - XCTU Range Test

By using XCTU we check if the XBee has been properly configured and may join a PAN. This requires another XBee which it can be join up with. After configuring the second XBee unit we switch to networking mode in XCTU. Figure 23 shows that the ZigBee modules are able to detect each other. To properly test the XBee modules and observe if they are able to transmit packets between each other we proceed to use the range test tool found in XCTU.

By running the XCTU range test we now know that the two modules are properly configured and may now be integrated onto the Raspberry Pi to serve as one of the many communication interfaces the system has to offer.

4.1.9 Kernel Compilation

There are basically two ways of compiling the Linux kernel. It can be done locally on the computer that is going to be upgraded, or it can be done on a secondary computer. As the Raspberry Pi is only equipped with a 700 MHz CPU, we figured it would take too much time. Especially since each build had to be tested and if it turned out to be erroneous, it had to be recompiled. Therefore we installed Ubuntu on a virtual PC using VirtualBox. It was given access to one CPU core running at 3.2 GHz and 4096 MB of RAM.

The compilation process is started when downloading the source code to the Linux kernel, and a compiler designed for the target computer – in our case the RPi. Then the source code is primed with the current config, which is copied directly from the Raspberry Pi. When this is done, the kernel can be modified by adding or removing features. In order to make the kernel more stable, we removed a lot of unnecessary modules. After repeating this process until we had a reduced version that worked without flaws, we transferred the kernel and the modules to RPi.

Below is a set of commands to download the necessary files and prepare for selecting modules. We have not automated the selection, as this was done in an interactive menu [83].

```
mkdir ~/rpi
cd ~/rpi
wget https://github.com/raspberrypi/linux/archive/rpi-3.2.27.tar.gz
wget https://github.com/raspberrypi/tools/archive/master.tar.gz
tar xzf rpi-3.2.27.tar.gz
tar xzf master.tar.gz
export CCPREFIX=/home/picompile/rpi/tools-master/arm-bcm2708/arm-bcm2708-linux-
gnueabi/bin/arm-bcm2708-linux-gnueabi-
export KERNEL_SRC=/home/picompile/rpi/linux-rpi-3.2.27
cd $KERNEL_SRC
make mrproper
scp pi@192.168.1.18:/proc/config.gz ./
zcat config.gz > .config
make ARCH=arm CROSS_COMPILE=${CCPREFIX} oldconfig
```

Please note that the username on the Ubuntu install is “**picompile**” and the IP address of the Raspberry Pi is “**192.168.1.18**” in the code example. The code above is first making a folder in the home directory, then downloading two .tar.gz-files. These files are then extracted. The “**export**” commands are creating variables to ease the rest of the process. Instead of using the long folder paths, the variables can be used.

The “**cd**” command changes directory to perform the following commands from inside the kernel source folder. When inside this directory “**make mrproper**” cleans up old files before the compilation is started [83]. As this command also will remove configuration files, we wait until after this to copy the configuration from the RPi.

“**scp**” is the command used to transfer the current configuration from RPi. This starts a SSH session and will copy the file to the Ubuntu machine. Then the configuration file is extracted and used to select the modules in the source that is already installed on the Raspberry Pi. In order to make changes in the kernel, run the following command and select or unselect modules [84]

```
make ARCH=arm CROSS_COMPILE=${CCPREFIX} menuconfig
```

When this is done, the kernel will be compiled using the command “**make ARCH=arm CROSS_COMPILE=\${CCPREFIX}**”, and the modules will be built with “**make ARCH=arm**

`CROSS_COMPILE=${CCPREFIX} modules` [84]. Be prepared to wait, as this is the part that may potentially take the most time, along with the transfer of the modules later.

As all of the modules will now be built, they should be installed to a folder. To do this, run the following commands.

```
mkdir ~/rpi/modules
export MODULES_TEMP=~/.rpi/modules
make ARCH=arm CROSS_COMPILE=${CCPREFIX} INSTALL_MOD_PATH=${MODULES_TEMP}
modules_install
```

On the Raspberry Pi, we designed the following commands to install the newly built kernel along with the modules.

```
scp -q picompile@192.168.1.76:/home/picompile/rpi/linux-rpi-
3.2.27/arch/arm/boot/Image ./
scp -r -q picompile@192.168.1.76:/home/picompile/rpi/modules/lib ./
cp Image /boot/kernel_new.img
cp -r lib/* /lib
rm -r Image lib/
cat /boot/config.txt | grep -v "kernel=" > config.txt
echo "kernel=kernel_new.img" >> config.txt
cp config.txt /boot/config.txt
rm config.txt
```

This time “`scp`” is used to copy the kernel image and the modules to the Raspberry Pi from the virtual PC with IP 192.168.1.76. Use “`scp -r`” to copy an entire directory [85]. Then reboot and check if everything works the way it should. If there are errors, the process has to be repeated from “`make mrproper`” and afterwards. In case of error it can be helpful to only remove a small number of modules at a time, so that it is easier to troubleshoot. The solution is probably to add the module that turned out to be needed after all.

4.1.10 SD-Card Protection

The multi-purpose embedded communication gateway has to be as reliable as possible. We looked at several different techniques to introduce reliability to the system, one of them revolved around the SD-card. It is known that writing multiple times to a SD-card might damage it over time. To reduce the amount of writing to the SD-card we found a couple of solutions to handle the problem.

The Linux operating system has an in-memory file system that lets you write files to it. The files which are written to the memory will only exist in the hardware's memory and will be erased if the system is shut off. The Linux file system includes two different mount types; ramfs and tmpfs. Ramfs is not used in our solution as it does not give the option to set a memory limit to prevent overflow of RAM usage. The Raspberry Pi frequently writes to `/var/log` and `/var/run`. One could say that they are the worst offenders. The `var/log` file is mostly used for troubleshooting, but has been moved to the RAM

disk for the moment as it can be mounted back on the SD-card if needed. It would also benefit the system to write the temporary files from **tmp** to the RAM-disk.

The **/var/log**, **/var/run** and **tmp** folders are mounted to the RAM disk by opening and editing the **/etc/fstab** configuration file. This is done by “**sudo nano /etc/fstab**”.

The following lines should be added to the configfile [86][87].

#<file system>	<dir>	<type>	<options>	<dump><pass>
proc	/proc	proc	defaults	0 0
/dev/mmcblk0p1	/boot	vfat	ro,noatime	0 2
/dev/mmcblk0p2	/ext4	defaults,noatime		0 1
none	/var/run	tmpfs	defaults,noatime	0 0
none	/var/log	tmpfs	defaults,noatime	0 0
none	/tmp	tmpfs	defaults,noatime	0 0

The **<filesystem>** column determines the partition or storage that is going to be mounted. This column will not be used in relation to our solution. The **<dir>** column is used to specify the directory which is going to be written to the RAM instead of the SD-card. It is also possible to specify the type of file system that will be mounted. There are a lot of different types of file system besides tmpfs. **<dump>** can be set as **0** or **1** which specifies if to create a backup of the specific directory while **<pass>** is set to **0** or **1**, depending on if it is desirable to have the file system checked by fsck utility. In the end we have the **<options>** column which lets a user specify the setting for the specific folder. The fstab config files comes with a lot of different types of options. The options can be found at [kildenummer <https://wiki.archlinux.org/index.php/fstab>]. Currently the options for tmpfs has been set to default which means that tmpfs is allowed to use up to half of the total RAM that the system offers. The **noatime** option which has been set is the most important setting as it will disable the feature were Linux writes to the SD card every time something is read.

By using tmpfs we will prevent some of the most used directories to be written to the SD card. The tmpfs does however come with a weakness. If a situation arise were the system is need of higher amounts of memory, the file system swap back to using the SD card when writing to a folder. To prevent swapping we can run the following commands to disable it [86].

```
sudo dphys-swapfile swapoff  
sudo dphys-swapfile uninstall  
sudo update-rc.d dphys-swapfile remove
```

The changes will take effect on system reboot.

4.1.11 MySQL Installation Proposal

This section of the dissertation will provide information about how to setup data synchronization between two servers using MySQL. The information provided here has not been tested and is only a proposal on how to install and configure MySQL to act as a master slave replication on the multi-purpose communication gateway. The MySQL server is installed by using the following command:

```
sudo apt-get install mysql-server mysql-client
```

The next step of the process is to open up the MySQL configuration file on the Raspberry Pi. This is found in `/etc/mysql/my.cnf` and is accessed by use of the "nano" tool in Raspbian. Scroll down to `bind-address` in the configuration file and replace its IP address with the IP address of the server. In our case this will be the IP address of the Raspberry Pi.

```
bind-address = 109.189.105.151
```

The next line to change in the configuration file is the server id. The server id is used to identify the SQL server and must be unique. For the sake of simplicity we give the Raspberry Pi the id: 1.

```
server-id = 1
```

The last line to configure in the file is the `log_bin` line. `log_bin` is where all the changes that are made to the SQL server are registered. The slave server is going to copy these changes. The line will be commented and one only has to uncomment the line to activate it.

```
log_bin = /var/log/mysql/-bin.log
```

Once the changes has been made save and exist the configuration file.

The next step of the configuration will take place in the MySQL shell. To access MySQL use the command `mysql -u root -p`. Access has to be given to the slave server and set up their password. To do so, give the following command.

```
GRANT REPLICATION SLAVE ON *.* TO 'slave_user'@'%' IDENTIFIED BY 'password';  
FLUSH PRIVILEGES;
```

For the next step open a new window or tab in addition to the one already in use. In the new tab run the following commands.


```
USE newdatabase;  
FLUSH TABLES WITH READ LOCK;  
SHOW MASTER STATUS;
```

The command "SHOW MASTER STATUS" will provide the user with a table which should be remembered for later use. The user will then go back to the previous shell tab and export the database using the following command.

```
mysqldump -u root -p --opt newdatabase > newdatabase.sql.
```

It is important to note that the command is run in bash shell and not in MySQL. Return to the MySQL shell and unlock the tables and quit.

```
UNLOCK TABLES;  
QUIT;
```

Now that we are done configuring the master database (Raspberry Pi) we have to configure the slave database. Open up the slave's MySQL shell with `mysql -u root -p`. In the SQL's shell use the following command.

```
CREATE DATABASE newdatabase;  
EXIT;
```

Import the database we previously exported from the master database by use of `mysql -u root -p newdatabase < /path/to/newdatabase.sql`. The next step is to configure the server in the same way that the master server was configured. The configuration file is accessed through the same means by `sudo nano /etc/mysql/my.cnf`. In the configuration file set the following parameters:

```
server-id      = 2  
relay-log     = /var/log/mysql/mysql-relay-bin.log  
log_bin       = /var/log/mysql/mysql-bin.log  
binlog_do_db = newdatabase
```

Exit the SQL shell and restart the service through `sudo service mysql restart`. Open the shell again after its reboot and type the following details:

```
CHANGE MASTER TO MASTER_HOST='109.189.105.151', MASTER_USER='slave_user',  
MASTER_PASSWORD='password', MASTER_LOG_FILE='mysql-bin.000001', MASTER_LOG_POS=  
107;
```

The information used was given when running `SHOW MASTER STATUS` on the master server. The command used designates the current server as the slave of our master server and provides the server the correct login credentials. And lastly it tells the slave server were to replicate from by using the log file from the master server.

The configuration is now complete and one can start the slave server by typing “**START SLAVE;**” [88].

4.2 Mobile Broadband Failover

In order to manage automatic failover of the Internet connection, we design the following script that will run continuously. An overview of this script is shown in Figure 24.

When the script is started, it will set some settings before reaching the first loop. This loop runs while the Internet connection is available. Inside the loop, the script will check if the Internet connection is lost, and will also distinguish if there is local network access or not. If the connection is lost, it will connect the mobile broadband and set up forwarding from this, but where it goes from there depends on which level of network access is still available. When there is access to the local network, it will perform some tests towards the Internet. This is done within the B-loop in Figure 25. Another outcome is if there is no network connection at all. The script will then run loop C in Figure 25. Then the script will only perform tests on the local network. When the network connection is available, the script will exit loop C and enter loop B. If it then detects normal Internet connection, it will enter loop A. The mobile broadband connection will stay connected for a predefined number of seconds. This number is set in the start of the script.

When taking a look at Figure 25, note that there is no direct link going from loop C to loop A. That is to make sure the network is back before switching off the mobile broadband.

In this brief overview, we have divided the script into four parts. The following subsections will go into detail on each of these.

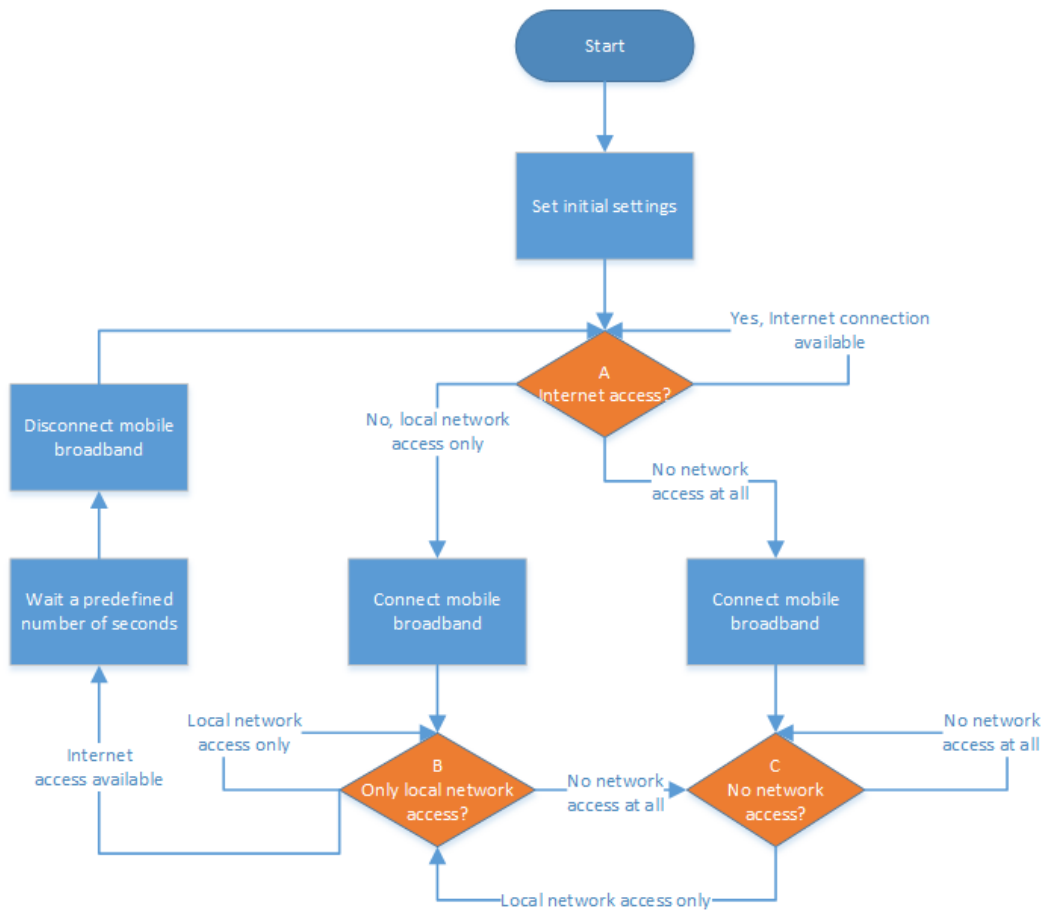


Figure 24 - Overview of the network failover script

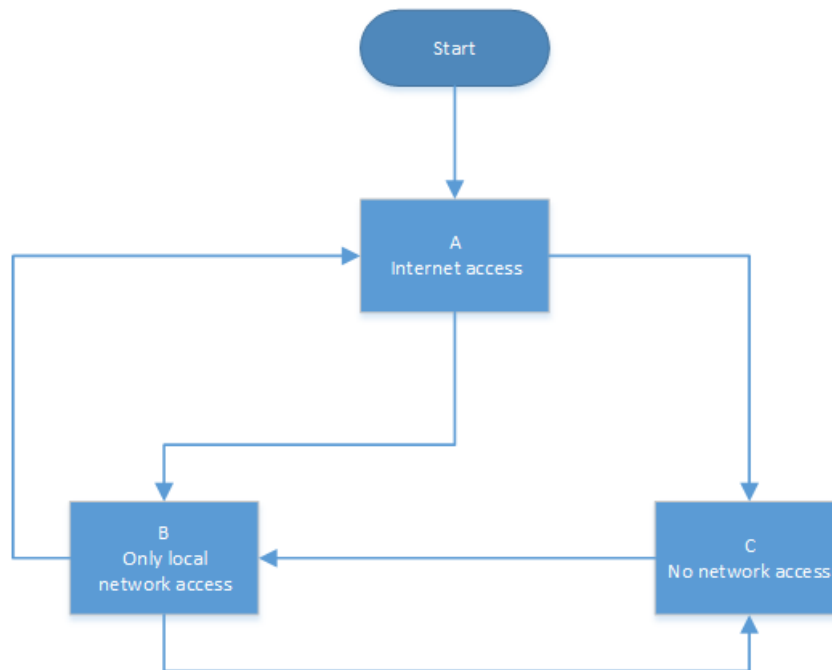


Figure 25 - Basic overview of the states in the network failover script

4.2.1 The Initialization of the Script

The first part of the script is only used to set some parameters to be used later in the process. Some of the variables are set automatically, but some is also available for the user to change. Following is a complete list of the user definable variables and an explanation of their meaning:

- **IF** – The interface which has internet access and is monitored by the script. By default it is eth0.
- **pausetime** – The number of seconds which the script is going to wait between each check. Setting this to a low value increases the system load, but a high value may result in a longer waiting period before the failover script reacts on connection loss. Default: 2.
- **staymin** – When mobile broadband is connected due to loss of Internet connection, and the main connection becomes available, this value says how long the mobile broadband shall stay connected. As it takes some seconds to connect the mobile broadband, this can help if there is a high risk of a new connection loss in near future. If the main connection is usually stable, this value can be set to 0. Default: 2.
- **networkstate** – The initial network state. Can be 0, 1 or 2. 0 means that there is internet connectivity, 1 means only local access and 2 is no network access. This value is also deciding which of the while loops the script should stay in at all times. Default: 0.
- **rxthreshold** – One of the connectivity tests are measuring the amount of packets that can be received and still have no internet access.
- **pingcount** – The number of ping requests to be sent when performing ping tests. There is a delay on one second between each ping. Default value: 3.
- **pingexternal** – An IP address in the Internet that is always available to ping. Default: 8.8.8.8 (Google DNS server)
- **losslimit** – This is the amount of ping requests that is acceptable to lose, given in percentage. The value must be between 0-100. Default: 0.
- **pridns** – The primary DNS server that is to be used. It is possible to use the ISPs own DNS, but some servers are only for the ISPs IP subnet. As the mobile broadband is on some other subnet, and maybe behind a NAT server as well, it is possible that the ISP DNS will be unreachable when using the mobile broadband connection. Default value: 8.8.8.8 (hosted by Google).
- **secdns** – The secondary DNS server that is to be used. Default: 8.8.4.4 (hosted by Google).

In addition to the variables available to the user, the script sets some values automatically. These values include setting the default gateway on both the main interface and the 3G/4G interface. Also, the **staymin** value is converted to seconds, to match the timers in the script.

By running **ifconfig**, the values for the amount of bytes sent and received can be read. These values are stored to the variables **txold** and **rxold**.

Before entering the main part of the script, `usb_modeswitch` will run to make sure the modem is ready, and not locked in storage mode. Then it will wait the number of seconds as the variable `pausetime`.

4.2.2 While Loop When There is Internet Connectivity

Right before the most common while loop, `iptables` will set its rules back to normal. If the variable `networkstate` is 0, the script will enter the loop. It will again wait the amount of seconds as defined in the variable `pausetime`. This is the best place of the script to add a delay. The delay has to be inside the loop, but the middle and the end of the loop is more time critical in case of failure.

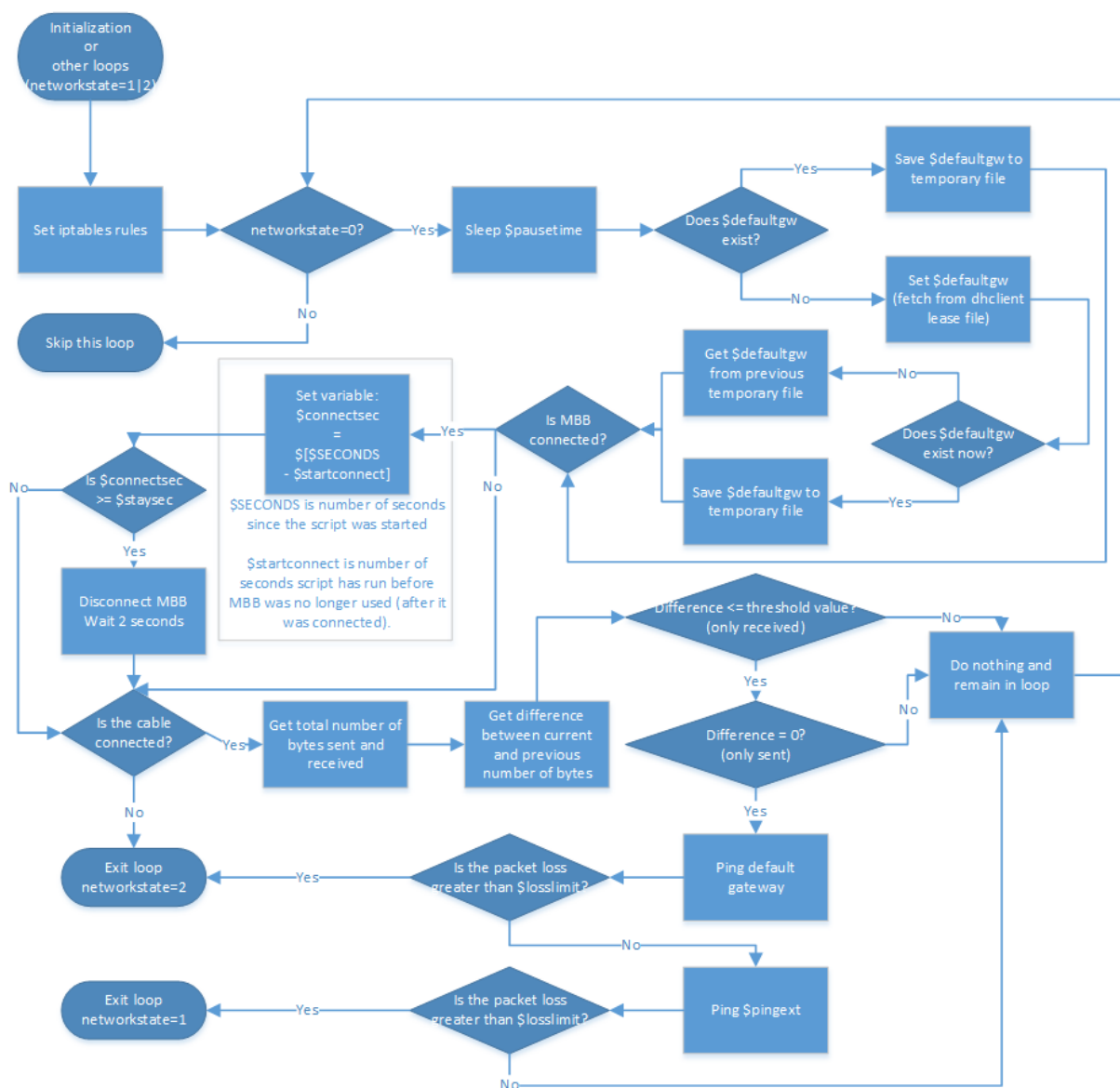


Figure 26 - Loop in the failover script where there is Internet access

After the delay, the script checks if there is anything stored in the variable **defaultgw**. Preferably the IP address of the nearest router on the current interface (given in the variable **IF**), should be stored in the variable. If the IP address is already stored in the variable, it will be saved to a file to remember until next time the script runs. On the other hand, if the variable is not set, the script will try to read it from the DHCP client lease file. If the IP address was found now, it will be saved in a file. It may happen that the address cannot be found, and it will then try reading the value from a file generated last time the IP address was known.

In the next step there is a check if the variable **mbbconnected** equals **1**. In that case, mobile broadband is most likely connected as well. If it is connected, it will first figure out how long it has been connected. This is done with comparing the variable **SECONDS** to the variable **startconnect**. The first of which is an internal Linux variable, that counts seconds from the start of the script, and the latter is set to the same value as **SECONDS** at the time the main connection is yet again available. The difference of these values will at all times show the number of seconds the mobile broadband has been connected after the main connection is back. If this difference, **connectsec**, is equal to or larger than **staysec** (which is the amount of seconds the mobile broadband shall stay connected after it is no longer in use), the mobile broadband will disconnect. This calculation makes sure that if the main connection disconnects again, it will be faster to switch over to mobile broadband. Depending on the contract and operator, it may be unwise to let the mobile broadband stay connected for longer periods of time, mostly because of the potential costs.

In the next test, the script checks the content of a file called **/sys/class/net/eth0/carrier** (in this example **eth0** is the value of the variable **IF**). If the content of this file is **1**, the cable is detected in the port. That means that there must be a device in the other end of the cable. If the value here is **0**, the loop will exit immediately and **networkstate** will be set to **2**.

When the cable is detected, the values for **rxbytes** and **txbytes** will be fetched from **ifconfig**. These are the numbers of bytes that has been received and transmitted (respectively) in total since boot. As the values **rxold** and **txold** contains the corresponding values from earlier, it is possible to measure the amount of traffic since last iteration of the script, or from the start of the script if this is its first round. The differences are stored to the values **rxdiff** and **txdiff**.

If **rxdiff** is less than **rxthreshold**, the script will continue to check **txdiff**. Otherwise nothing will be done and the script will continue from the start of the loop. The next test checks if **txdiff** is equal to **0**. If this is the case, nothing is probably expected to be received, and the script will continue. If **txdiff** is more than **0**, the script will return to the start of the loop.

As it is now defined that there has been no measurable traffic since the last run, it is necessary to generate some traffic in order to find out if there is Internet access. This is done using ping tests. First the IP in **defaultgw** is pinged. If this fails (ergo; the fail percentage is higher than the value in **LossLimit**), the script exits the loop, and sets **networkstate** to **2**, as there is probably no network connection available. The number of attempts is depending on the value of **pingcount**. If the test is successful, some new attempts are done to the IP given in **pingext**. The result of this test will decide if the loop should be restarted or not. There is probably no working Internet connection if this test turns out negative, but it is already determined that the local network is available, so the variable **networkstate** will be set to **1** and the loop will be exited. If the test is positive, it seems to be no problem with the Internet connection.

In total, there are three network tests in this loop. They are deliberately sorted by complexity, duration and resource intensity. As the simple **cat** command to determine if the cable is connected is so fast, it is best to try this first, as a network error will then be found earlier. A ping test demands more of the local system, and all nodes in between the sender and the receiver, so it is best to wait until this is the last option.

4.2.3 While Loop When There is Only Local Network Access

Before this while-loop is reached, there are some lines of code that enables the mobile broadband connection if it is not already connected. Similar to the in Subsection 4.2.2 that checks for content in the variable **defaultgw**, in this part the script checks **mbbgw** for content. If it is empty, it will attempt to find it using the **route** command. On the other hand, if the IP address to the gateway for the mobile broadband connection is already stored in the variable, it will create a file to save it for later.

If this fails, and the variable was not found, it will attempt to read the IP address from a previously saved file. Then the default route will be updated to go via the mobile broadband gateway, the DNS servers are replaced and the iptables rules is changed to include **ppp0** (rather than the interface stored in the **IF** variable). When this is done, the script will have arrived at the start of the loop.

The first action within the while-loop, is to wait the number of seconds that is defined in **pausetime**. Then it will temporarily enable the main interface as the default route and reset the DNS server addresses. This is done mainly because there is a risk of getting false replies on the ping requests when the mobile broadband is connected. For instance, if the ping replies were received through the mobile broadband interface, the script could reset the connection back to the main interface, while in fact the connection was still down.

The first ping attempt is to the IP in `pingext`. If this succeeds, the script will exit the loop and set `networkstate` to 0, as there was now detected an Internet connection. However, if the attempt fails, the script will ping its nearest router, the IP from `defaultgw`. If this succeeds, the loop will restart after setting `mbbgw` as the default route and adding the new DNS servers, but if it fails, `networkstate` will become 2 and the loop will exit.

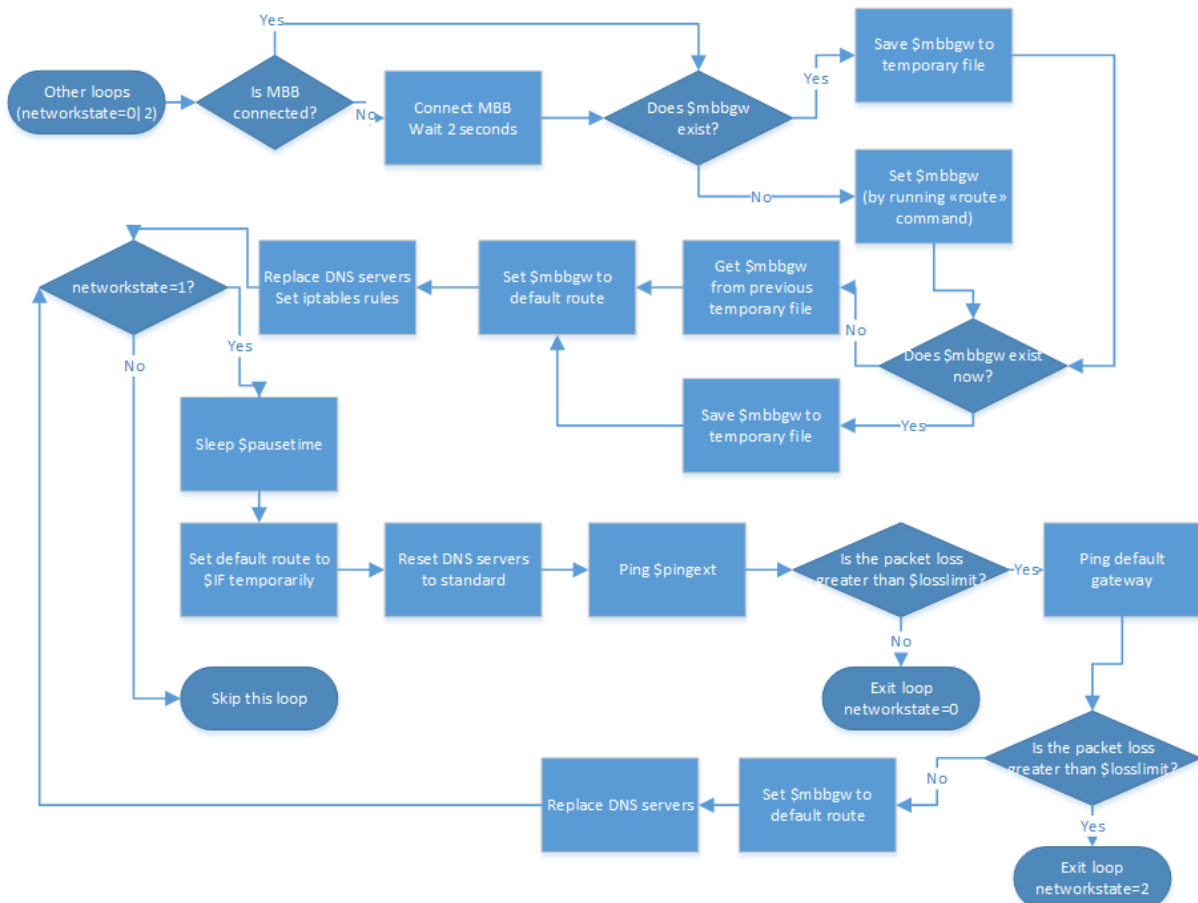


Figure 27 - Loop in the failover script where there is local network access

4.2.4 While Loop When There is No Network Access

This while-loop is slightly different from the other two, in the way that the script can come from either the first or second loop (`networkstate=0` or `networkstate=1`). If the process runs from `networkstate=0`, then the processes in front of the loop is identical to the part in front of the loop described in Subsection 4.2.3. In fact, this is the exact same piece of code that is run. However, if the script runs from `networkstate=1`, the given code has been run earlier. Therefore it does not matter for the outcome of the script, except for the flow described in Figure 28. Compared to the other while-loops, it is different as there are two possible starting points, instead of just one.

If the script comes from the loop where **networkstate=0**, the steps in the process before this while-loop is described in Subsection 4.2.3. There are no code in between the two last loops in the script, so when exiting the loop where **networkstate=1**, this loop is entered at once.

As soon as the loop is entered, the script stops as long as it should according to the value in **pausetime**. Then it will run the cat command to check **/sys/class/net/eth0/carrier** if the cable is detected. If so, the loop quits immediately and returns to the loop where **networkstate=1**.

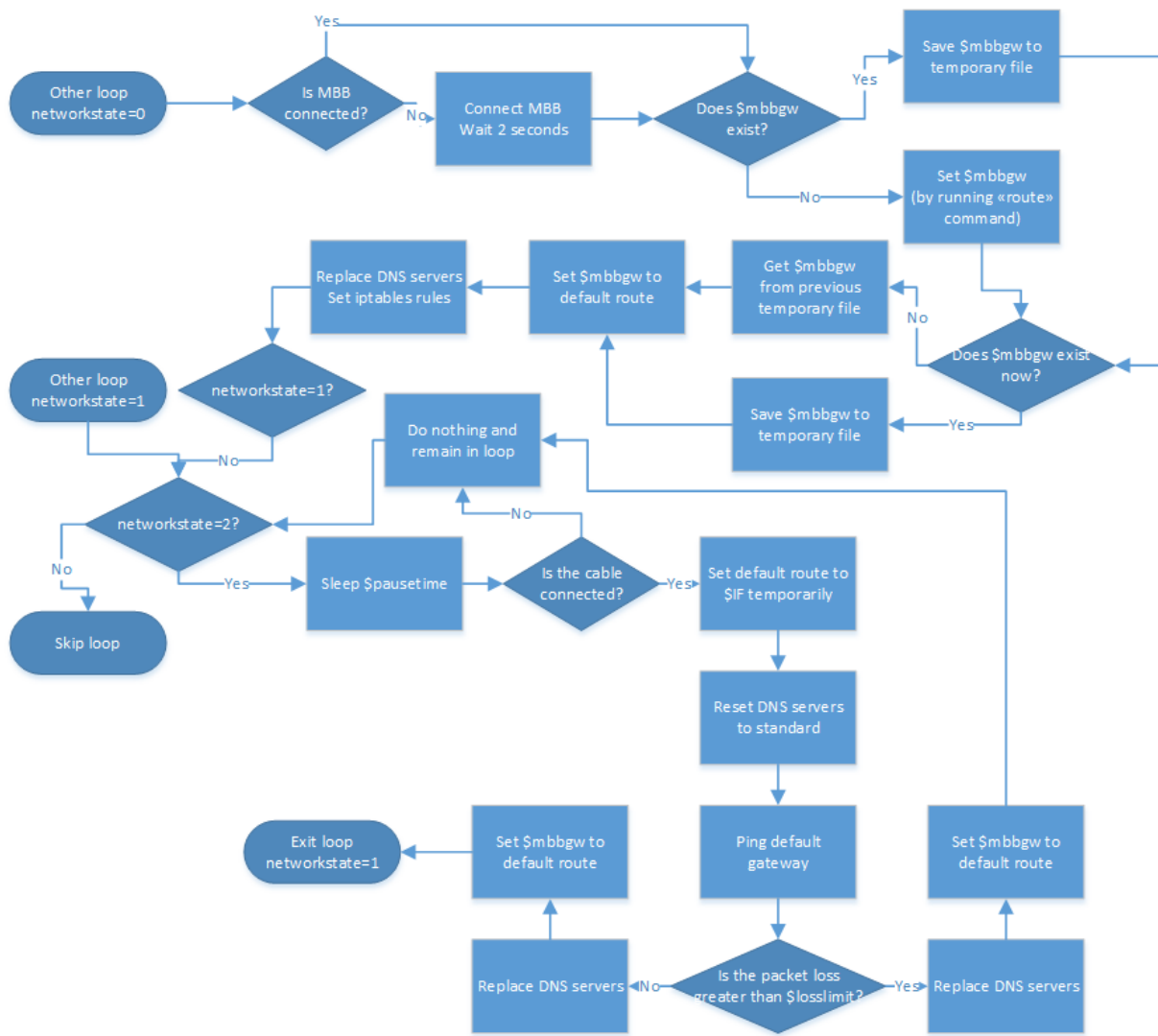


Figure 28 - Loop in the failover script with no network access

On the other hand, if the cable is undetected, the script sets default route temporarily back to the standard value. The DNS servers are also updated. After this, the DNS servers are set and the default gateway is pinged and **mbbgw** is again set at default route. Depending on the verdict of the ping test, the script will either return to **networkstate=1** or repeat the current loop (**networkstate=2**). None

of the loops will ever be exited unless the **networkstate** changes, so if the ping test gives a positive result, the loop will be exited.

4.3 The Watchdog

As stated in Section 3.2.2, we chose to use the Arduino Mega 2560 as our watchdog. What we might not have mentioned is that there is a watchdog on the Raspberry Pi as well. This internal watchdog is able to react sooner when something is wrong, but if the system has stopped working completely, chances are the watchdog itself has stopped running too. As this is a risk, we chose to use both the internal and external watchdogs.

4.3.1 Internal Watchdog

In order to use the onboard watchdog in the Raspberry Pi, the watchdog driver has to be added to the kernel. This can be done during the kernel compilation, or via a simple command and by editing a file. Also, the software for the watchdog has to be installed.

If the kernel is not going to be recompiled, the watchdog driver can be added with the command `“sudo modprobe bcm2708_wdog”` [89].

Edit the file called `/etc/modules` and add a new line with the text `“bcm2708_wdog”` [89]. When this is done, the watchdog will be recognized by the operating system.

To install the necessary software, run `“sudo apt-get install watchdog chkconfig -y”`. The installation will create a file `“/etc/watchdog.conf”` [89]. Edit this file by uncommenting the lines containing the following:

- `max-load`
- `min-memory`
- `watchdog-device`
- `admin`
- `interval`
- `logtick`
- `log-dir`
- `pidfile`

Make sure the file is identical to the file in Figure 30. When the file is saved, run the two following commands, and the setup of the internal watchdog will be complete [89].

```
sudo chkconfig watchdog on
sudo /etc/init.d/watchdog start
```

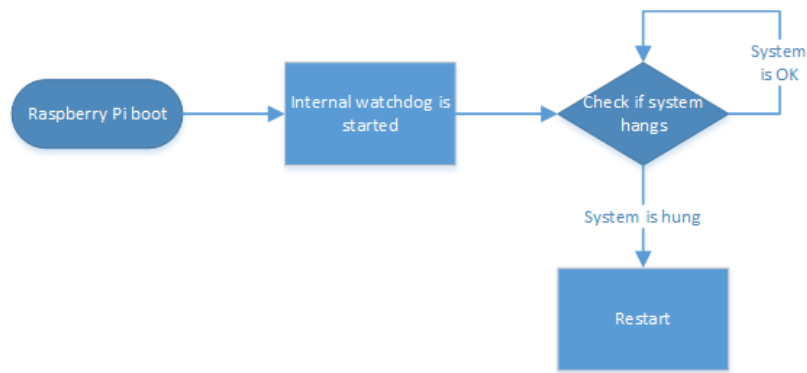


Figure 29 - Simple flow diagram of the internal watchdog

```

GNU nano 2.2.6 File: /etc/watchdog.conf
#ping = 172.31.14.1
#ping = 172.26.1.255
#interface = eth0
#file = /var/log/messages
#change = 1407

# Uncomment to enable test. Setting one of these values to '0' disables it.
# These values will hopefully never reboot your machine during normal use
# (if your machine is really hung, the loadavg will go much higher than 25)
max-load-1 = 24
max-load-5 = 18
max-load-15 = 12

# Note that this is the number of pages!
# To get the real size, check how large the pagesize is on your machine.
min-memory = 1

#repair-binary = /usr/sbin/repair
#repair-timeout =
#test-binary =
#test-timeout =

watchdog-device = /dev/watchdog

# Defaults compiled into the binary
#temperature-device =
#max-temperature = 120

# Defaults compiled into the binary
admin = root
interval = 1
logtick = 1
log-dir = /var/log/watchdog

# This greatly decreases the chance that watchdog won't be scheduled before
# your machine is really loaded
realtime = yes
priority = 1

# Check if syslogd is still running by enabling the following line
pidfile = /var/run/syslogd.pid

[ Read 41 lines ]
^G Get Help      ^O WriteOut      ^R Read File     ^Y Prev Page     ^X Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where Is     ^V Next Page     ^U UnCut Text    ^T To Spell
  
```

Figure 30 - The Watchdog configuration file

As long as the system is still running, the watchdog will continue to monitor the system state. If an exception should occur, the watchdog will reboot the system. The watchdog will however not be able to reboot the system if the process has stopped working. The reliability will then rely on the external watchdog.

4.3.2 External Watchdog

Unlike the internal watchdog, the function of the Arduino is clearly not depending on the system load on the Raspberry Pi. The internal watchdog is unusable if the process controlling it stops working.

In order to create and run a script on the Arduino, it needs to be connected to a computer running the Arduino IDE. This software is used to write the code, compile it and transfer to the Arduino. As the computer is quite basic, it will only run one script at a time, and any previous scripts will be deleted when transferring a new. As soon as the transfer is complete, the Arduino will restart and run the newly transferred script. Basically, an Arduino script can be divided into three parts: The initialization, the setup and the loop.

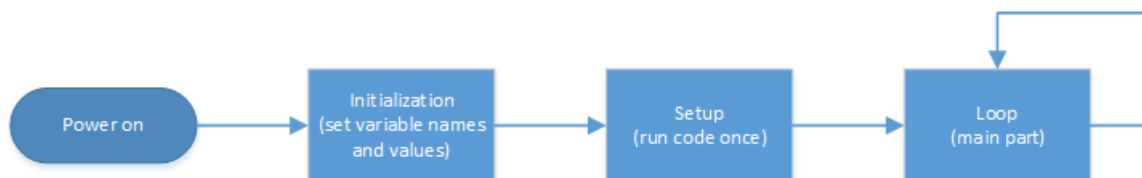


Figure 31 - A general flow diagram of an Arduino script

In the initialization, variables are defined and assigned start values. The setup contains code that is only supposed to run once. When this is done, the loop is started. This is the main part of the code, and will run continuously until the Arduino loses power, or is manually reset using its reset button.

Our complete script can be read in Appendix X, where there also is a complete model. As the script is fairly complex, it will be described in detail below.

4.3.2.1 Initialization Part

This is the first part of the code and contains only declarations of variables. Following is a list of the variables, their initial values and a description:

- **buttonPin** – The number of the pin where the button is connected. Default: 3.
- **beaconPin** – The number of the pin where the Raspberry Pi sends its heartbeat signal. Default: 8.
- **onSignalPin** – The number of the pin where the Arduino sends a signal to Raspberry Pi when it is supposed to be turned on. Default: 30.
- **relaysignalPin** – The number of the pin where the Arduino sends a signal to the relay which is controlling the power supply to Raspberry Pi. Default: 52
- **startupTime** – A measured number of milliseconds it takes to power up the Raspberry Pi. Default: 75000.

- **shutdownTime** – A measured number of milliseconds it takes to shut off Raspberry Pi. Default: 15000.
- **time** – A value used to measure time between each alternation of the signal from Raspberry Pi. Default: 0.
- **newtime** – A value used to measure time between each alternation of the signal from Raspberry Pi. Default: 0.
- **runtime** – A value used to measure time between each alternation of the signal from Raspberry Pi. Default: 0.
- **buttonval** – The measured value on buttonPin on a scale 0-1023. Default: 0.
- **beaconval** – The measured value on beaconPin on a scale 0-1023. Default: 0.
- **initialState** – A variable that is either 0 or 1. It is used to keep track of which section of the main loop that should be run. Default: 0.
- **startupState** – A variable that is either 0 or 1. It is used to keep track of which section of the main loop that should be run. Default: 0.
- **normalState** – A variable that is either 0 or 1. It is used to keep track of which section of the main loop that should be run. Default: 0.
- **buttonshutdownState** – A variable that is either 0 or 1. It is used to keep track of which section of the main loop that should be run. Default: 0.
- **hardshutdownState** – A variable that is either 0 or 1. It is used to keep track of which section of the main loop that should be run. Default: 0.

4.3.2.2 *The Setup*

The setup part only contains six lines of code. This includes setting the direction on the pins (input or output), starting a serial session for debugging and setting the variable **initialState** to 1. Pin direction is important as this tells the Arduino if the pin is used to send or receive signals.

4.3.2.3 *Main Code*

The main part of the script is one large “**while**”-loop that will run continuously [90]. Inside this loop, there are five smaller while loops. Three of them are only run once, and behave like “**if**”-tests. The other two will not stop until they receive a proper input, in form of a button press or a signal loss from the Raspberry Pi.

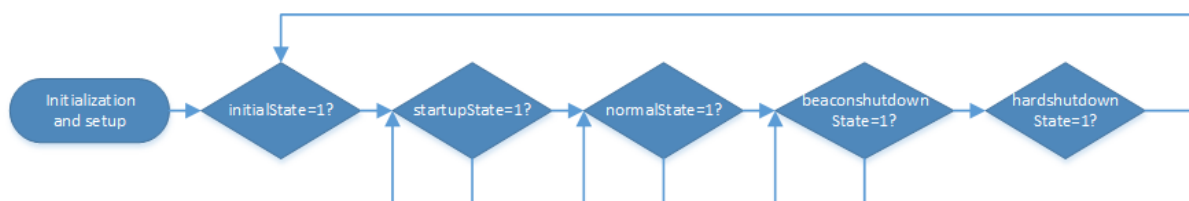


Figure 32 - Arduino main loop overview

In the first while loop, which is only entered when **initialState=1**, the Arduino starts sending signal to the relay and the Raspberry Pi. It also sets **initialState** to **0** and **startupState** to **1**. This means that it will always exit after its first run.

When **startupState=1**, the second while loop is started. This is a state where the Arduino waits for the Raspberry Pi to boot. As soon it detects the signal from the Raspberry Pi, it will exit the loop and continue with its normal operation.

Inside the main loop, there is code designed to detect the alternating signal from the Raspberry Pi, while also detecting if the button is being pressed. In order to do this, we have designed two smaller loops, one for when the signal is received and one for when it is not received.

The first action in the main loop is to set the variable **time** to the output from the function **millis()** [91]. This is a number of milliseconds since the script was started and will be used at a later time.

Afterwards, it will check if the signal from RPi is on or off [92]. If it is on it will check if the button is pressed [92]. If it is, the variables **normalState** and **buttonshutdownState** will be changed (to **0** and **1** respectively). Then the script will not restart the main loop when it reaches the end, but instead continue to the next step.

However, if the button was not pressed, the script will continue and check if the RPi is still sending a signal, and getting a new output from **millis()** [92][91]. This time the value is stored to the variable **newtime**. The difference between **time** and **newtime** is stored to **runtime**, which is compared to the value **startupTime** [93][94]. If **runtime** is longer than **startupTime**, the Raspberry Pi is declared “dead” and the script will enter the last loop [94]. This is done with setting the variables **normalState**, **beaconval** and **hardshutdownState** to **0**, **1024**, and **1**, respectively.

On the other hand, if **runtime** is small enough, the loop will return to the start. Just before **runtime** was calculated, there was also a set a new value for **beaconval**. If it has changed, the current loop will be exited, and the right loop in Figure 33 will run. This loop is designed exactly the same, except that it will run until the signal is detected again. As the signal is designed to alternate every 500 ms, the script will switch between the loops pretty fast.

When the main loop is exited, it is always because the Raspberry Pi needs to shut down. It is either the user’s decision to or because the signal is constant on or off, so it’s suspected it has stopped working. Either way the Arduino stops sending the signal to Raspberry Pi. This is done so the Raspberry Pi has a chance to turn itself off in a controlled manner and to prevent damaging the SD

card when cutting the power. The Arduino will wait for the predefined number of milliseconds it usually takes to shutdown the RPi before switching off the relay.

If the system was shut down by the user, the Arduino will wait for a new button press from the user before turning on the Raspberry Pi. On the other hand, if the Raspberry Pi was shutdown to recover from an error, it will be restarted instantaneously. In both of these scenarios, the new startup of the Raspberry Pi is done by returning to the start of the script, in the first while loop[90].

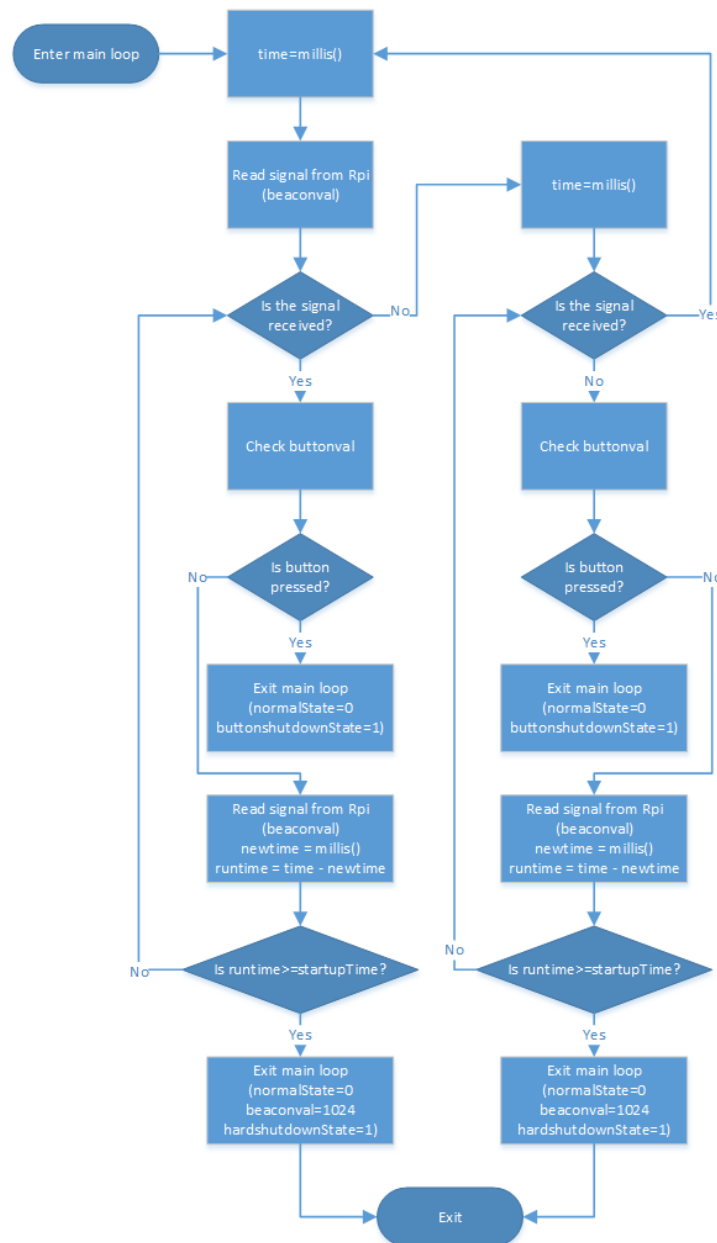


Figure 33 - A detailed view of the main loop in the Arduino script

4.4 Other Reliability Measures

As the watchdog only reacts when the system has stopped working, there is a need for another level of reliability. This would be when clients fail to connect to the router, when the network interfaces need to restart or if processes have stopped. A reboot is too time consuming and should be the absolute last resort.

For instance, if the process which is signaling the watchdog stops, there will only be a constant signal, or loss of signal. Then the watchdog will react and restart the whole system. A restart may take 90 seconds or more, so it would be beneficial for the user if the system just could restart the given process.

As a complement to the watchdog, we have created a script that will check different important parameters and fix them if necessary. This script is started during boot and continues to run as long as the system is on. Through extensive testing we have found multiple parameters which can be used to automate a troubleshooting:

- Check if secondary Ethernet and Wi-Fi has IP addresses. If not, the script will set IP addresses.
- Check if DHCP process is running. If not, start it.
- Check if hostapd process is running. If not, start it.
- Capture network traffic on local interfaces and check if DHCP requests are received without being answered. Restart DHCP server if there are any problems with this.
- Check that the processes that is capturing network traffic is running. If not, start them.
- Check the system logfile if there has been any sudden deauthentications of Wi-Fi clients that has previously been connected. In that case, restart hostapd.

The script is available for reading in Appendix 4.

4.5 Physical Implementation

The previous subchapters have shown how we installed the software and designed the various scripts. In this subchapter we will show how the system was built, and how the final version is connected physically.

4.5.1 The Casing

In the early stages of the project, the main focus was to get the features to work independently. It was first of all important that the Wi-Fi, 3G/4G and external Ethernet worked on its own before

connecting the parts. Figure 34 shows the Raspberry Pi (to the right) connected to power and the USB-hub (the device to the bottom left). The primary Ethernet is not connected in the figure. On the lower part of the USB-hub, there are two cables: The external power and the USB-cable to the RPi. We experienced that the interfaces would become unstable if the external power was disconnected, so we will recommend to always keep this connected, except during setup of the mobile broadband software.



Figure 34 - The Raspberry Pi with the hub and some adapters

On the top of the hub, the Edimax Wi-Fi adapter is connected (directly to the right of the Logik-logo in Figure 34). The secondary Ethernet and the 3G/4G modem is located on the top side of the hub, in the picture. Note that in Figure 34 the Bluetooth and ZigBee adapters are not present, and the devices are unpowered. The light emitting diodes (LEDs) are turned off.

After the picture in Figure 34 was taken, the system would also get support for Bluetooth and ZigBee, and the watchdog was added. In order to make it transportable, we decided to make a box for it. The box was made primarily by plexiglass and plastic glue, but the devices were mounted using Sugru [95].



Figure 35 - The front of the box

A button is placed in the front side of the box. This button is used to turn the system on and off. The relay is also placed on this wall, but on the inside.



Figure 36 - The rear side of the box

The rear wall has a hole with a narrow opening above going to the top of the box, as in Figure 36. The intention is that the cables will be pulled through the hole. The opening is located there for guiding the cables when installing, if the connectors are too large to be pulled through.



Figure 37 - The top of the box

On the inside of the top side, the USB-hub is mounted. In order to make the box compact, we found that this was the best solution. The top is glued to one of the side walls, and these two parts are separate from the rest of the box. This is practical as one might have to perform changes in the connections inside the box. The one side wall is also working as a stabilizer for the roof when it is stuck between the two adjacent walls.

When the roof is lifted off, it is easier to see the placement of the devices. See Figure 38 for an overview. The two embedded computers are placed on the bottom, in parallel to each other. It seems to be a lot of cables, but the USB cable from the hub was quite long. We could have shortened it, but wanted to keep it intact for the future.

4.5.2 Cables and Connections

We did however modify the power cables for the Arduino and the RPi. As there was limited extra space for the connectors in the box, we dismantled the USB type B connector for the Arduino, the USB connector to the USB hub, and the micro USB to the Raspberry Pi. Then we were able to angle them by resoldering and encapsulating the cables using Sugru. As Sugru is formable and sets like rubber, in addition to having a very high level of electrical resistance [95], it was ideal to use instead of the original plastic coating on the connectors.

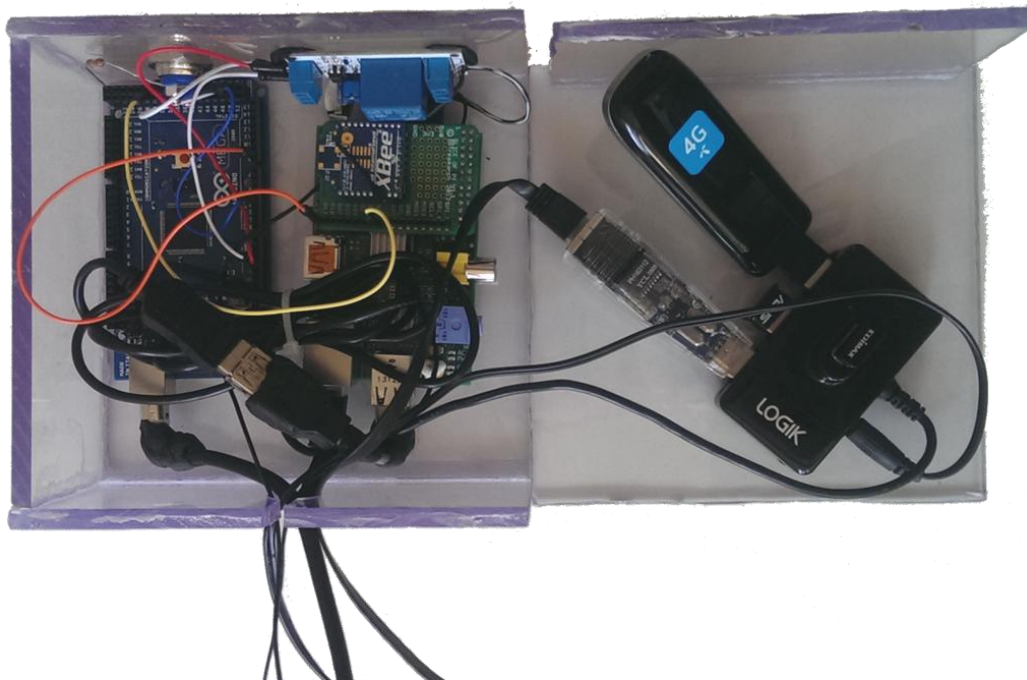


Figure 38 - The inside of the box, seen from above

When the power cable to RPi was modified, it was also cut in half. This was done in order to connect the relay switch in between, to make the Arduino able to control the power status of the Raspberry Pi. The 5VDC lead was connected to the normally-open (NO) contacts on the relay, while the ground was practically unaltered, going directly from the RPi to the power source. The relations between the Arduino, Raspberry Pi, Slice of Pi, on/off button and relay is illustrated in Figure 39, and a short explanation to each connection is given in Table 5.

The Arduino is connected to the other side of the relay. The VCC on the relay is always powered from a 5VDC output on the Arduino, and the ground is connected to ground on the relay. In between VCC and ground on the relay, is the IN1 pin [96]. This is supposed to get signal when the relay should be switched on. In our setup we connected this to digital pin 52, but another pin can be chosen as long as the correct pin number is defined in the watchdog script.

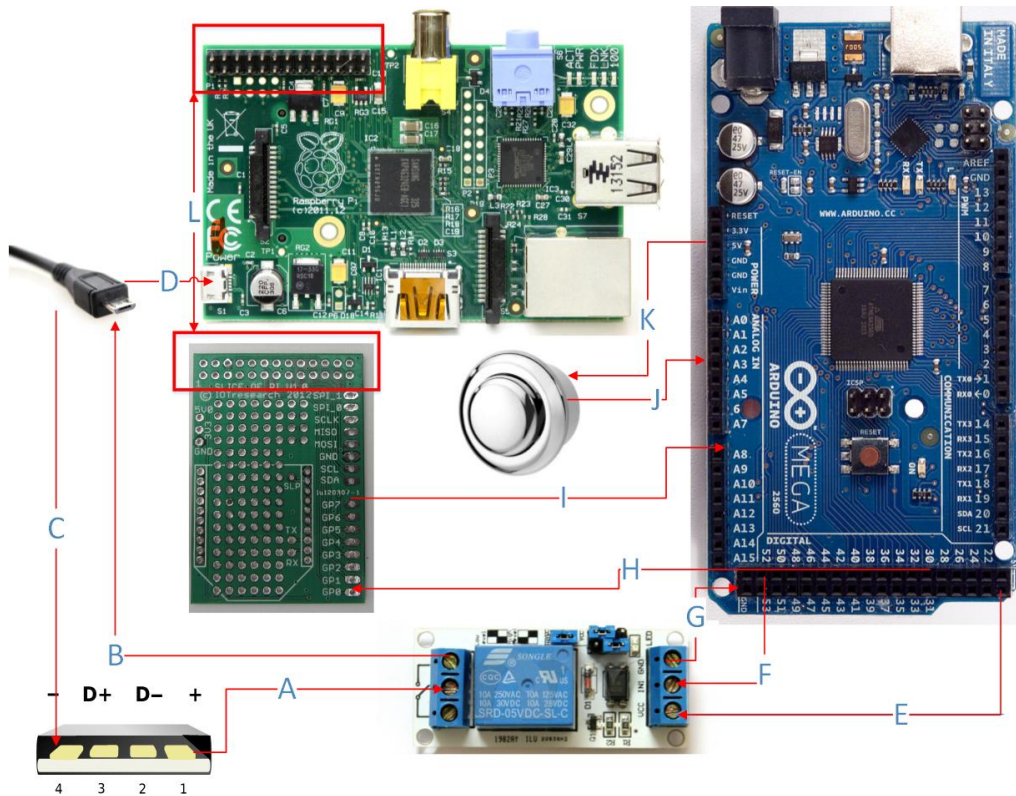


Figure 39 - Illustration of connections [97][98][99][44][100][101][102]

Table 5 - Explanation to Figure 39

Connection	From	To	Description
A	Power source	Relay	5VDC to relay
B	Relay	Micro USB connector	Relay to 5VDC (micro USB connector)
C	Micro USB connector	Power source	Ground (connector) to power source
D	Micro USB connector	Raspberry Pi	Micro USB connector connected to RPi
E	Arduino	Relay	5VDC (Arduino) to VCC (relay)
F	Arduino	Relay	Digital pin 52 (Arduino) to IN1 (relay)
G	Relay	Arduino	Ground (relay) to ground (Arduino)
H	Arduino	Slice of Pi	Digital pin 30 to GPIO 17
I	Slice of Pi	Arduino	GPIO 4 to analog pin 8
J	Button	Arduino	Button to analog pin 3
K	Arduino	Button	5VDC (Arduino) to button
L	Raspberry Pi	Slice of Pi	All GPIO pins (RPi) to Slice of Pi

The next connection on the list in Table 5 is going from the Arduino to the Slice of Pi. As the Slice of Pi is connected to all of the GPIO pins on the Raspberry Pi, and not all of them are used by the Xbee,

there are some GPIO extensions available. When connecting the Arduino to GPIO 4 and GPIO 17 on the Slice of Pi, it is in reality connected to the same GPIO pins on the Raspberry Pi. In our setup we used GPIO 17 to receive the signal from Arduino that only stops when RPi needs to shut down. This signal was sent from digital pin 30, but as with the relay signal, this can be changed as long as the pin is defined in the script on the Arduino. From GPIO 4 on RPi is the heartbeat signal sent to analog pin 8 on Arduino. This is the alternating signal that is designed to let the watchdog know if the system has stopped working.

The second input for the Arduino comes from the on/off button. The button is constantly connected to 5VDC on Arduino and when pressed it connects the circuit so that the Arduino detects input on analog pin 3.

4.5.3 ZigBee Module

We mentioned that we used the printed circuit board (PCB) Slice of Pi in between the Xbee and the GPIO connections. This came as a kit which needed to be assembled. Figure 40 shows the parts separated as they were delivered.

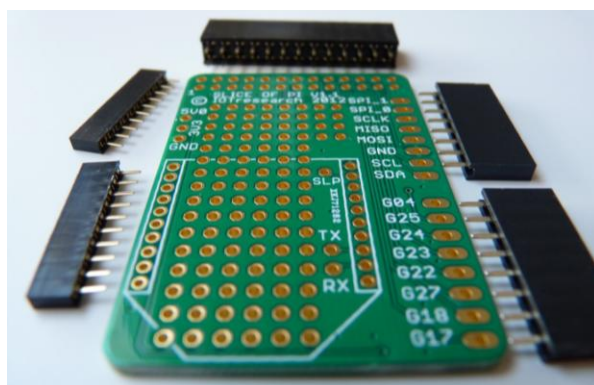


Figure 40 - Slice of Pi before assembly [103]

Please note that the PCB in Figure 40 is version 1.1, while the Slice of Pi in Figure 39 is version 1.0. The differences seems minimal, but the naming scheme is different. Version 1.1, which we use, uses the naming scheme of the second revision of Raspberry Pi, while version 1.0 uses a numbering scheme ranging from 0 to 7 [104]. This difference does not have any practical impact on the system, except that it is important to connect to the correct pins according to the settings in the scripts.

The pins are easier to connect to after assembling the Slice of Pi kit. Figure 41 shows the PCB after soldering the connectors. For connecting the Xbee, the two nearest, parallel connection sets are used.

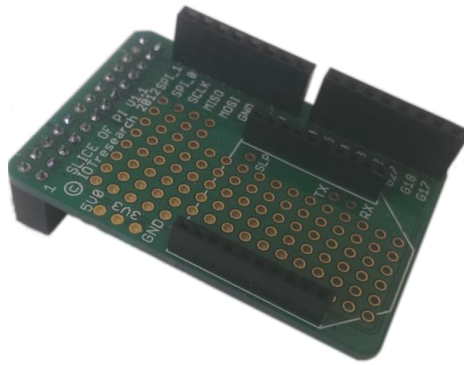


Figure 41 - The Slice of Pi after soldering the connectors

There is even an outline to show which direction the Xbee should be placed in. How the Xbee is connected, is shown in Figure 42.



Figure 42 - Our Slice of Pi with Xbee

This is the Slice of Pi before it is connected to the Raspberry Pi. In Figure 43 and Figure 44, one can see how it looks when mounted on the RPi.



Figure 43 - RPi with Slice of Pi



Figure 44 - RPi with Slice of Pi

5 Testing and Validation

5.1 Reliability

The multi-purpose embedded communication gateway provides several types of features. The features themselves are independent of each other will be validated independently of each other. The data of the experiments will be collected through sheer observation.

5.1.1 Wi-Fi Access Point

In addition to acting as a gateway, it may also act as an access point for wireless devices. The setup of the wireless access point has already been explained in the previous section and this test will validate the feature of the device. The process of the test will simply be to:

1. Install the necessary software and hardware and setup the system as an access point as described in Section 4.1.3.
2. Connect several wireless devices (smart-phones and laptops) to the access point and let them stay connected for several hours.
3. Access the Internet through Raspberry Pi using the wireless devices.

The result of the experiment was successful. The different wireless devices were connected to the access point for several hours. The wireless devices received updates from social media sites and applications such as Facebook and Steamworks through the communication gateway, and users who used it for playing online games did not get sudden connection drops.

5.1.2 Ethernet Routing

The embedded system will provide various communication interfaces and may act as a gateway in a local area network. One of the interfaces which are provided by the gateway is Ethernet. Ethernet is one of the most common data transfer protocols, and the procedure for verifying Ethernet will simply be:

1. Install an additional Ethernet port on to the Raspberry Pi as described in Section 4.1.4.
2. Let Raspberry Pi a gateway between a computer and a modem.
3. Access the Internet through the Raspberry Pi using the computer.

The result of the test was a success. The computer in use was able to access the Internet, as well as remotely control the Raspberry Pi through SSH. The test also showed us that the Raspberry Pi did not

act as a bottleneck between the computer, and the modem and kept its throughput throughout the session.

5.1.3 Mobile Broadband (3G/4G)

The mobile broadband provided by the mobile communication gateway, is the interface which provides continuous Internet access to the users of the system. The mobile broadband is one of the most important features of the system, and will be validated through the given process.

1. Install the hardware and software, and make the proper configurations of the mobile broadband and the routing as shown in Chapter 0.
2. Disconnect the Ethernet cable and connect the mobile broadband modem.
3. Access the Internet.

The result of the experiment was a success. The devices connected to the mobile broadband were able to connect to the Internet and browse web, as well as receiving files through application services. The mobile broadband offered a sufficient throughput.

5.1.4 Network Failover

The experiment of this feature includes several different sub-features which are implemented into the communication gateway. The features which will be validated through the experiment are the device's ability to detect connection losses to the Internet, turn on mobile broadband, detect Internet connection through Ethernet, and switch off mobile broadband. The experiment is given as:

1. Create or install the script for automatic failover before running it, as described in Subchapter 4.2.
2. Disconnect the Ethernet cable or turn off the DSL modem.
3. Test the Internet connection by use of a wireless or physical connected device.
4. Connect the Ethernet cable or turn on the DSL modem.
5. Observe if the mobile broadband process is killed.
6. Test the Internet connection by use of a wireless or physical connected device.

The experiment was successfully completed. The mobile broadband was connected once the Ethernet cable was pulled out, and the connected devices were able to access the Internet. The time it took for the system to switch from Ethernet to mobile broadband was approximately 10 seconds. The process of disconnecting the modem was instantaneous as the script is designed to kill the connection process when detecting another source to the Internet.

5.1.5 Bluetooth

The gateway provides Bluetooth support for situations where users decide to transfer files in a simple manner from a device without Wi-Fi. It is also important to notice that the mobile communication gateway needs to accept the pairing of any device by default.

1. Install the software and hardware as described in Section 4.1.7
2. Use a device to pair with the gateway and observe if it is accepted automatically by default.
3. Choose a file to transfer from the device to the gateway and vice versa.

The experiment was successfully completed, and both devices were able to retrieve and store a PDF file of 6000 KB. The pairing was accepted by the Raspberry Pi and remembered by the Bluetooth device used in the process.

5.1.6 Internal Watchdog

In order to make the system run reliably a watchdog is used. The internal watchdog is having more control over the system status, and can reboot the system fast if there are any problems. The following steps were done to test if the onboard watchdog is working properly.

1. Install the watchdog as described in Section 4.3.1.
2. Try crashing the system.
3. Wait to see if the system crashes and reboots on its own.

To crash the system we ran a piece of code (“: C){ :|:& };:”) called a fork bomb [89]. This is a process which creates subprocesses that creates more processes. After a short while there has been created so many processes that the system is drained of resources and is unable to perform any actions. The code was successful and simply by waiting we noticed that the system rebooted. The result was expected and we think it was highly satisfactory.

5.1.7 External Watchdog

As the internal watchdog, the external watchdog will also monitor the system. The external watchdog will have the advantage that it is separate from the rest of the system. In case the hardware crashes on the Raspberry Pi, it is a possibility that the internal watchdog has crashed as well. The probability for needing the external watchdog is fairly low, but it provides a great deal of reliability, as the chance of both the internal watchdog and the external watchdog crashing at the same time is very small. In order to test the external watchdog, we will do the following.

1. Install the external watchdog as described in 4.3.2.
2. Press the button and check that the RPi turns off.
3. Check if the RPi turns on when button is pressed.
4. Stop the signal from the Raspberry Pi.
5. Wait and see if the watchdog restarts the RPi.
6. Check that the watchdog does not turn on or off the system unexpectedly.

When first connecting power to the Arduino, it started sending a signal to the relay, which is supplying power for the Raspberry Pi. According to the design of the watchdog script, nothing will be done before the Raspberry Pi has started completely. When the Raspberry Pi was done with the startup, we tried pushing the button. The Raspberry Pi shut down almost immediately and its power was cut by the relay. Then we pushed the button again the relay was switched on and the RPi booted up again.

The secondary task of the watchdog is to listen for changes in the signal from Raspberry Pi. This feature was tested by killing the process in charge of alternating the signal. As the script is designed to wait until the RPi have had enough time to boot on its own if the signal was stopped by a controlled reboot, nothing happened until around 75 seconds had passed. Then the relay was shut off, then on again and the system booted once again.

Until now, the system has been acting as expected. The last step in the list was tested over two hours. During this time the system shut down once. After some troubleshooting we found that the watchdog had gotten some false signals on the button input. As the input signal is read on an analog GPIO pin, it will always detect some input values, but usually not more than a fraction of what is needed to trigger the watchdog. The input values on analog pins are read in a range of 0-1023, where 1023 is the value when the current is sent deliberately. Unless a step-down converter is used, the input value will never be 0. By debugging this value, we found that the value is highly dependent on the environment. Just by holding the hand over the Arduino, without touching, we were able to trigger a shutdown of the system.

At the current state of the system, the watchdog acts fairly reliable, but as there is a risk for getting wrong values on the analog pin, it is advisable to keep the chassis closed and place the system in a distance from other electrical devices in order to prevent erroneous signals.

5.2 Scenarios

5.2.1 Wi-Fi Access Point Range Experiment

The Wi-Fi access point configured for the embedded communication gateway may vary in connection quality when one take into the account of objects that may disrupt the signal. This experiment's purpose is to estimate the signal strength of the communication gateway's AP inside a building. The experiment was performed inside a house of 144 square meters, and the wireless access point was placed at the far end of the house. We observed that the users were able to connect to the access point at all locations in the house, maintaining an excellent connection quality throughout the 10 hour session. The quality of the signal did however drop significantly outside the house, unable to detect its signal 5 meters past the outer walls. The reason for such sudden drop is that the outer wall of the house is thicker and made of high density materials such as concrete and cement. A comparison was made with the home owner's router and the same results were observed.

5.2.2 Mobile Broadband Gateway in a Car

In order to find out if the system is suited for use in a car, we performed an experiment where the intention was to connect to the Internet through the Wi-Fi access point. The watchdog was powered by the USB-port on a computer and the Raspberry Pi was connected to a 12V cigarette lighter adapter with 1 ampere (A) power capacity. As there was no 230V outlet in the car, the external power supply for the USB-hub was not used.

The result of the experiment was that the Wi-Fi access point turned on, and provided Internet access, but only for a few seconds before rebooting. This was most likely caused by lack of power, as the Ethernet adapter did not even turn on like it usually does. As the RPi is designed to use almost 700 milliamper (mA), there will be roughly 300 mA left of the capacity of the power adapter to serve the USB-hub including Wi-Fi, Ethernet, 3G/4G modem and Bluetooth. Therefore we can conclude that with the current setup, the system is not suited for use in a car. With an increase of power, this test should be retried.

5.3 Validation

Most of the goals for this project are met, but the watchdog is not quite as reliable as we wanted. This is because of the sensitivity of the analog pins. As they can give false results by just holding objects near them, we feel it is an issue that should be fixed in the future. Also, in the car scenario, the system requires more power in order to function properly.

Because of these issues, we have to say that the solution as of now is not completely satisfactory. The solution is however well working in an environment with little electrical disturbances. As the Ethernet, Wi-Fi, mobile broadband, and Bluetooth interfaces have all been working with no errors during the testing period, there is not much left until the system is comparable to existing products.

6 Discussion

This part of the dissertation will provide some of the additional options which have been looked at during the semester. A lot of software and hardware has been looked at. It has been mentioned in the previous chapters that we've had a lot of different hardware and software candidates to choose between, but they were all eliminated during the planning stage of our embedded system. It is however somewhat hard to eliminate all the options during the phase of planning, such as solutions for a given feature. A solution to a given feature comes in many different forms, and this part will present some of the other solutions that has been worked on during our thesis, and was not used due to particular reasons. In addition to presenting former, we will also recommend the next step for the project.

The first chosen operating system for our system base our system on an operating system named PiBang Linux. The operating system proved to be unstable and had several errors during the installation process. As our time were limited, starting a project on an unstable platform could have given us a hard time in the future and it was removed as our system's operating system.

In addition to the operating system, we encounter different solutions to install mobile broadband to our device. It was decided on the earlier stage that the system would use a software named sakis3G. Sakis3G is a tool in Linux which is used to configure and connect a device to the Internet through LTE. The software required usb-modeswitch to successfully connect to the Internet and ran in a stable manner during our experiments. It was however not used in the final stage of the prototype as it ran too slow. In our experiments the modem used an average 40 seconds to connect to the Internet, compared to the 10 seconds used by wvdial.

We had different hardware solution as well as software solutions. As one can recall, several types of communication interface dongles were tested during our master thesis as mentioned in Section 3.4.2. The most noticeable one was a Wi-Fi dongle named D-Link DWA-123. The D-Link DWA-123 dongle worked perfectly fine as a standard Wi-Fi connector, but lacked the feature to act as an access point which was crucial for our solution. When discovered it got replaced by an Edimax dongle which supported the feature. Another problem we encountered during our communication interface hardware selection, was during our time implementing a second Ethernet port to the Raspberry Pi. The Ethernet port created a serious issue that froze the Raspberry Pi on random occasions. We suspect that it was driver related problem, but were not able to determine the exact cause since the Raspberry Pi kept crashing. To solve the ongoing issue we bought a new Ethernet port KÖNIG CMP-NWUSB2.0 which was connected to the Raspberry Pi without further problems. In addition to the

solutions that was not implemented due to various problems there were features that we would like to add to our system, but did not have the appropriate time to do.

For future work we would recommend to go onto the direction of making the device a commercial product. Then the issue with the watchdog should be resolved. The analog pins are likely to give false values when read. This can be fixed either by using the digital pins instead, or by adding a resistor as a step-down converter.

To improve the product further, we suggest to make it easier to configure for the regular user by developing a graphical user interface. The graphical user interface would give the user the option to configure the device without prior knowledge to Linux and scripting.

We would also recommend to design and create a proper casing for the embedded system that makes it more durable as well as giving it a slicker design. At this point the embedded communication system is powered through three power adapters. In the future we we would also like to combine all three of the power adapters into an single unit, as well as implementing our suggesting to synchronizing the Raspberry Pi with a server through use of MySQL.

7 Conclusion

The purpose of the dissertation was to develop a multi-purpose embedded communication gateway for providing continuous Internet access. A gateway was developed for multiple interfaces such as Ethernet, Wi-Fi, Bluetooth, ZigBee, and 3G/4G. The device would use the mobile broadband as a failover to provide continuous Internet action, and provide reliability and availability through script optimization and a watchdog.

The result of our work, is a stable gateway that may be used in different types of scenarios, as long as the amount of power needed for the system is appropriately delivered. The gateway may act as an access point for different types of communication interfaces, and forward information to the Internet by use of Ethernet or mobile broadband. Compared with other commercial devices, it includes different communication interfaces, and features a watchdog which strengthens the device's availability and reliability in an environment without electrical disturbances. It is also important to mention that the gateway is open source, and new modules may be added to suit the user's specifications.

References

- [1] ASUSTEK Computer Inc. "SDHC 3-in-1 Wireless-N150 Router with 3G/4G backup". Internet: http://www.asus.com/Networking/RTN10U_B/, [Nov., 2013].
- [2] Cisco Systems, Inc. "Cisco 880G Series 3G Wireless Integrated Services Router". Internet: http://www.cisco.com/en/US/prod/collateral/routers/ps380/ps10082/data_sheet_c78_498096.html, [Nov., 2013].
- [3] TP-Link Technologies CO., Ltd. "3G/4G Wireless N Router". Internet: <http://www.tp-link.com/no/products/details/?model=TL-MR3420>, [Nov., 2013].
- [4] NETGEAR. "NETGEAR® Mobile Broadband Wireless Routers". Internet: <http://www.netgear.com/3G>, [Nov., 2013].
- [5] M. Webb. "Netgear unveils Wireless-N 300 router featuring automatic 3G/4G/WiMax failover". Internet: <http://www.gizmag.com/netgear-wireless-router-failover-dgn2200m/13759/>, Jan., 2010 [Nov., 2013].
- [6] NETGEAR. "DGN2200M Compatible 3G USB Modems in Denmark". Internet: http://kb.netgear.com/app/answers/detail/a_id/24181, Nov., 2013 [Nov., 2013].
- [7] NETGEAR. "N300 Wireless ADSL2+ Modem Router DGN2200M Mobile Edition User Manual". Internet: http://www.downloads.netgear.com/files/GDC/DGN2200M/DGN2200M_UM_04Nov11.pdf, Nov., 2011 [Jan., 2014].
- [8] J. Vadivelu. "Method and system for network failover and network selection with multi-mode modem in remote access points". Internet: <http://patentimages.storage.googleapis.com/pdfs/US8537715.pdf>, Sep., 2013 [Nov., 2013].
- [9] C. W. Ea and M. Sørnbø, "Multipurpose Embedded Communication Gateway," University of Agder, Grimstad, Project report for IKT416 Scientific Methods 2013.
- [10] M. J. Pont and R. H.L. Ong. "Using watchdog timers to improve the reliability of single-processor embedded systems: Seven new patterns and a case study". Internet: <http://www.le.ac.uk/eg/mjp9/mjpv02.pdf>, Sep., 2002 [Mar., 2014].
- [11] A. Mayhew. "File Distribution Efficiencies: cfengine vs. rsync". Internet: https://www.usenix.org/legacy/events/lisa2001/tech/full_papers/mayhew/mayhew.pdf, Dec., 2001 [Nov., 2013].
- [12] Wikipedia. "A13-OLinuDino-WIFI". Internet: <http://en.wikipedia.org/wiki/OLinuDino#A13->

- [OLinuXino-WIFI](#), [Nov., 2013].
- [13] Olimex Ltd. "Universal EXTension connector". Internet:
<https://www.olimex.com/Products/Modules/UEXT/>, [Nov., 2013].
- [14] J. Vilaça. "Cubieboard vs OlinuXino A13 WiFi vs Beaglebone". Internet:
<http://vilaca.eu/cubieboard-olinuxinoa13-beaglebone/>, [Nov., 2013].
- [15] Olimex Ltd. "A13-OLinuXino-WIFI". Internet:
<https://www.olimex.com/Products/OLinuXino/A13/A13-OLinuXino-WIFI/open-source-hardware>, [Nov., 2013].
- [16] Atmel Corporation. "ATmega2560". Internet:
<http://www.atmel.com/devices/atmega2560.aspx>, [Nov., 2013].
- [17] Arduino SA. "Arduino Mega 2560". Internet:
<http://arduino.cc/en/Main/ArduinoBoardMega2560>, [Nov., 2013].
- [18] Arduino SA. "Arduino Mega2560 Rev3". Internet:
http://store.arduino.cc/ww/index.php?main_page=product_info&cPath=11_12&products_id=196, [Nov., 2013].
- [19] The BeagleBoard.org Foundation. "BeagleBoard". Internet:
<http://beagleboard.org/Products/BeagleBoard>, [Nov., 2013].
- [20] G. Coley. "BeagleBone Black vs BeagleBone". Internet:
<https://groups.google.com/d/msg/beagleboard/9aLzt68mq9I/hfTVH9mAqhMJ>, May., 2013
[Nov., 2013].
- [21] The BeagleBoard.org Foundation. "BeagleBone Black". Internet:
<http://beagleboard.org/Products/BeagleBone%20Black>, [Nov., 2013].
- [22] Cubieboard. "Cubieboard Products". Internet: <http://docs.cubieboard.org/products/start>,
[Nov., 2013].
- [23] Miniand Pty Ltd. "Cubietruck A20 Dev Board". Internet:
<https://www.miniand.com/products/Cubietruck%20A20%20Dev%20Board#specifications>,
[Nov., 2013].
- [24] Seeed Studio. "Cubietruck Kit - Dual Core Single-board Computer". Internet:
<http://www.seeedstudio.com/depot/cubietruck-kit-dual-core-singleboard-computer-p-1628.html>, [Nov., 2013].
- [25] Elektor International Media BV. "Elektor Academy - Embedded Linux Made Easy – Q&A

- session". Internet:
http://www.element14.com/community/servlet/JiveServlet/previewBody/50560-102-1-261603/QA_Sep_final.docx, [Nov., 2013].
- [26] Elektor International Media BV. "Embedded Linux Made Easy (120026-91)". Internet:
<http://www.elektor.com/products/kits-modules/modules/120026-91-elektor-linux-board.2135310.lynkx>, [Nov., 2013].
- [27] B. Linder. "Gooseberry developer boards: £40 alternative to the Raspberry Pi". Internet:
<http://liliputing.com/2012/06/gooseberry-developer-boards-40-alternative-raspberry-pi.html>, Jun., 2012 [Nov., 2013].
- [28] P. Georgiadis. "Gooseberry – An alternative to Raspberry Pi". Internet:
<http://www.unixmen.com/gooseberry-an-alternative-to-raspberry-pi/>, Jul., 2012 [Nov., 2013].
- [29] Gooseberry. "Gooseberry - Store". Internet: http://gooseberry.atspace.co.uk/?page_id=31, [Nov., 2013].
- [30] Miniand Pty Ltd. "Hackberry A10 Dev Board". Internet:
<https://www.miniand.com/products/Hackberry%20A10%20Developer%20Board>, [Nov., 2013].
- [31] Miniand Pty Ltd. "Hackberry-Power". Internet: <https://www.miniand.com/wiki/Hackberry-Power>, Oct., 2012 [Nov., 2013].
- [32] HAOYU Electronics. "MarsBoard A10 Dev Board". Internet:
<http://www.hotmcu.com/marsboard-a10-dev-board-p-59.html>, [Nov., 2013].
- [33] HAOYU Electronics. "MarsBoard A10 Dev Board - An ARM GUN/Linux box". Internet:
<http://www.marsboard.com/>, [Nov., 2013].
- [34] Secret Labs LLC. "netduino - hardware". Internet: <http://netduino.com/hardware/>, [Nov., 2013].
- [35] Secret Labs LLC. "Netduino Board Comparison". Internet:
<http://wiki.netduino.com/Comparison.ashx>, [Nov., 2013].
- [36] Cool Components Ltd. "Netduino Plus 2.NET / C# Development Board". Internet:
<http://www.coolcomponents.co.uk/dev-boards/net/netduino/netduino-plus-2-net-c-development-board.html>, [Nov., 2013].
- [37] Parallella. "Parallella Computer Specifications". Internet: <http://www.parallella.org/board/>,

- [Nov., 2013].
- [38] Adapteva Inc. "Parallella Reference Manual 4.13.2.13". Internet: http://www.adapteva.com/wp-content/uploads/2013/02/parallella_reference_4.13.2.13.pdf, [Nov., 2013].
- [39] Adapteva Inc. "Parallella-16". Internet: <http://shop.adapteva.com/collections/featured-products/products/parallella-16>, [Nov., 2013].
- [40] Wikipedia. "Raspberry Pi". Internet: http://en.wikipedia.org/wiki/Raspberry_Pi, [Nov., 2013].
- [41] Raspberry Pi Foundation. "FAQs". Internet: <http://www.raspberrypi.org/faqs>, [Nov., 2013].
- [42] CompuLab. "Utilite Value Specifications". Internet: <http://utilite-computer.com/web/utilite-value-specifications>, [Nov., 2013].
- [43] CompuLab. "Order Utilite Direct". Internet: <http://utilite-computer.com/web/order-utilite-direct>, [Nov., 2013].
- [44] A. D. Lindsay. "Picture of Slice of Pi". Internet: http://blog.thiseldo.co.uk/wp-photos/pi_001.jpg, [May., 2014].
- [45] C. W. Ea and M. Sørbrø, "Multipurpose Embedded Communication Gateway," University of Agder, Grimstad, Project report for IKT508 Specialization Project 2013.
- [46] Unsigned Integer Limited. "DistroWatch.com - Search Distributions". Internet: <http://distrowatch.com/search.php?ostype=All&category=All&origin=All&basedon=All¬basedon=None&desktop=All&architecture=arm&status=Active>, [Nov., 2013].
- [47] eLinux.org. "RPi Distributions". Internet: http://elinux.org/RPi_Distributions, [Nov., 2013].
- [48] Canonical Ltd. "Ubuntu.com - ARM". Internet: <https://wiki.ubuntu.com/ARM>, [Nov., 2013].
- [49] M. Tremer. "The Raspberry Pi dilemma". Internet: <http://planet.ipfire.org/post/the-raspberry-pi-dilemma>, Apr., 2012 [Nov., 2013].
- [50] Slackware Linux, Inc. "Slackware ARM on the Raspberry Pi". Internet: <http://docs.slackware.com/howtos:hardware:arm:raspberrypi>, [Nov., 2013].
- [51] The NetBSD Foundation, Inc. "NetBSD/evbarm on Raspberry Pi". Internet: http://wiki.netbsd.org/ports/evbarm/raspberry_pi/, [Nov., 2013].
- [52] GeeXboX. "GeeXboX". Internet: <http://www.geebox.org/>, [Nov., 2013].
- [53] OpenELEC. "OpenELEC MediaCenter". Internet: <http://openelec.tv/>, [Nov., 2013].

- [54] M. J. Hammel. "PiBox: An Embedded Distribution for the Raspberry Pi. And more!" Internet: <http://www.graphics-muse.org/wiki/pmwiki.php/RaspberryPi/RaspberryPi>, [Nov., 2013].
- [55] S. Nazarko. "Raspbmc". Internet: <http://www.raspbmc.com/>, [Nov., 2013].
- [56] RaspyFi. "RaspyFi Pi never sounded so good!" Internet: <http://www.raspyfi.com/>, [Nov., 2013].
- [57] Xbian. "XBian | XBMC on the Raspberry Pi, The bleeding edge". Internet: <http://www.xbian.org/>, [Nov., 2013].
- [58] Offensive Security Ltd. "Kali Linux | Rebirth of BackTrack, the Penetration Testing Distribution." Internet: <http://www.kali.org/>, [Nov., 2013].
- [59] S. Silverman. "PIMAME | Gaming and Emulators for the Raspberry Pi | Shea Silverman". Internet: <http://pimame.org/>, [Nov., 2013].
- [60] Alcatel-Lucent. "Expanding your Grid". Internet: http://plan9.bell-labs.com/wiki/plan9/expanding_your_grid/index.html, [Nov., 2013].
- [61] K. J. Sitjar. "Pro's And Cons Of Ethernet". Internet: <http://www.techwench.com/pros-and-cons-of-ethernet/>, Sep., 2012 [Nov., 2013].
- [62] WyzGuys Computer Tutors. "Pro and Cons of WiFi". Internet: <http://www.wifi.wyzguys.net/04Pros&Cons.htm>, [Nov., 2013].
- [63] P. S. Kwak. "Pros & Cons of Bluetooth". Internet: http://www.ehow.com/info_8258446_pros-cons-bluetooth.html, [Nov., 2013].
- [64] R. Thadani. "Advantages and Disadvantages of Bluetooth Technology". Internet: <http://www.buzzle.com/articles/advantages-and-disadvantages-of-bluetooth-technology.html>, Jun., 2011 [Nov., 2013].
- [65] Wikipedia. "ZigBee". Internet: <http://en.wikipedia.org/wiki/Zigbee>, [Nov., 2013].
- [66] F. Y. Li. "IKT429 Lecture 7: Wireless Sensor Networks". Internet: [https://fronter.com/uia/links/files.phtml/478215379\\$150667710\\$/Fagstoff/Lecture+slides/IKT429_L7_WSN.pdf](https://fronter.com/uia/links/files.phtml/478215379$150667710$/Fagstoff/Lecture+slides/IKT429_L7_WSN.pdf), Oct., 2013 [Oct., 2013].
- [67] Telenor. "Dekning - 4G og 3G". Internet: http://www.telenor.no/privat/dekning/?cid=p-00741-bet-gog-mbb-generiskdekning&gclid=CjgKEAjw2KCCbRCQ_6mztcunhEgSJABPxOF13gjWdmbv6cmh1punsQPKeH_01AHndEpgIXZ8bn2vKPD_BwE, [May., 2014].
- [68] Wikipedia. "WiMax". Internet: <http://en.wikipedia.org/wiki/WiMAX>, [Nov., 2013].

- [69] Diskusjon.no-user: Zailenser. "WiMAX dekkning i Norge". Internet: <http://www.diskusjon.no/index.php?showtopic=1052585&p=12720383>, Nov., 2008 [Feb., 2014].
- [70] E. Upton. "Introducing Turbo Mode: Up to 50% More Performance For Free". Internet: <http://www.raspberrypi.org/introducing-turbo-mode-up-to-50-more-performance-for-free/>, [May., 2014].
- [71] eLinux.org. "RPiconfig". Internet: <http://elinux.org/RPiconfig>, [May., 2014].
- [72] A. Gutierrez. "read a file and save it in variable using shell script". Internet: <http://stackoverflow.com/questions/7427262/read-a-file-and-save-it-in-variable-using-shell-script>, May., 2012 [Jan., 2014].
- [73] The Fan Club. "How to setup a USB 3G Modem on Raspberry PI using usb_modeswitch and wvdial". Internet: <http://www.thefanclub.co.za/how-to/how-setup-usb-3g-modem-raspberry-pi-using-usbmodeswitch-and-wvdial>, Nov., 2013 [Feb., 2014].
- [74] eLinux.org. "RPI-Wireless-Hotspot". Internet: <http://elinux.org/RPI-Wireless-Hotspot>, [Jan., 2014].
- [75] E. Carvalho. "How to set up Wi-Fi Hotspot for sharing using 2 wired and 1 Wi-Fi network cards?" Internet: <http://askubuntu.com/questions/186516/how-to-set-up-wi-fi-hotspot-for-sharing-using-2-wired-and-1-wi-fi-network-cards>, Sep., 2012 [Apr., 2014].
- [76] rtCamp. "Configure Postfix to Use Gmail SMTP on Ubuntu". Internet: <https://rtcamp.com/tutorials/linux/ubuntu-postfix-gmail-smtp/>, [Apr., 2014].
- [77] postfix.org. "Postfix main.cf file format". Internet: <http://www.postfix.org/postconf.5.html>, [Apr., 2014].
- [78] rtCamp. "Debugging Postfix Config, Mail Logs & more". Internet: <https://rtcamp.com/tutorials/mail/postfix-debugging/>, [Apr., 2014].
- [79] Dream Green House Ltd. "Raspberry Pi & Bluetooth". Internet: <http://www.dreamgreenhouse.com/projects/2012/rpiibt/index.php>, [May., 2014].
- [80] askubuntu.com-user: James. "How do you make Ubuntu Server accept files sent over bluetooth?" Internet: <http://askubuntu.com/questions/408429/how-do-you-make-ubuntu-server-accept-files-sent-over-bluetooth>, Jan., 2014 [Apr., 2014].
- [81] archlinux.org. "Bluetooth". Internet: <https://wiki.archlinux.org/index.php/Bluetooth>, [Apr., 2014].

- [82] Digi International Inc. "XCTU - Next generation configuration platform for XBee". Internet: <http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/xctu>, [May., 2014].
- [83] B. Chavez. "Compiling Kernel Modules for Raspberry Pi". Internet: <http://bchavez.bitarmory.com/archive/2013/01/16/compiling-kernel-modules-for-raspberry-pi.aspx>, Jan., 2013 [Apr., 2014].
- [84] eLinux.org. "RPi Kernel Compilation - Cross compiling from Linux". Internet: http://elinux.org/RPi_Kernel_Compilation#2._Cross_compiling_from_Linux, [Mar., 2014].
- [85] stackoverflow.com-user: Gryphius. "How to scp a folder from remote to local?" Internet: <http://stackoverflow.com/questions/11304895/how-to-scp-a-folder-from-remote-to-local>, Jul., 2012 [Apr., 2014].
- [86] N. Wertzberger. "Stopping SD Card Corruption on Raspberry Pi's Raspbian". Internet: <http://www.ideaheap.com/2013/07/stopping-sd-card-corruption-on-a-raspberry-pi/>, Mar., 2014 [May., 2014].
- [87] A-netz.de. "Ramdisks for the Raspberry". Internet: <http://www.a-netz.de/2013/02/ramdisks-for-the-raspberry/>, Feb., 2013 [May., 2014].
- [88] E. Sverdlov. "How To Set Up Master Slave Replication in MySQL". Internet: <https://www.digitalocean.com/community/articles/how-to-set-up-master-slave-replication-in-mysql>, Jul., 2012 [May., 2014].
- [89] binerry.de. "Raspberry Pi Watchdog Timer". Internet: <http://binerry.de/post/28263824530/raspberry-pi-watchdog-timer>, [May., 2014].
- [90] Arduino Team. "Arduino Tutorial - While Loop". Internet: <http://arduino.cc/en/Tutorial/WhileLoop>, [Apr., 2014].
- [91] Arduino Team. "Arduino Reference - millis()". Internet: <http://arduino.cc/en/Reference/millis>, [May., 2014].
- [92] Arduino Team. "Arduino Reference - analogRead()". Internet: <http://arduino.cc/en/Reference/analogRead>, [Apr., 2014].
- [93] Arduino Team. "Arduino Reference - Addition, Subtraction, Multiplication, & Division". Internet: <http://arduino.cc/en/Reference/Arithmetic>, [May., 2014].
- [94] Arduino Team. "Arduino Reference - if (conditional) and ==, !=, (comparison operators)". Internet: <http://arduino.cc/en/Reference/if>, [May., 2014].

- [95] FormFormForm Ltd. "About sugru". Internet: <http://sugru.com/about>, [May., 2014].
- [96] hobbyist.co.nz. "Step 2: Interfacing the relay modules to the Arduino". Internet: <http://www.hobbyist.co.nz/?q=interfacing-relay-modules-to-arduino>, [Apr., 2014].
- [97] eDampf-Shop.com. "Picture of Micro USB connector". Internet: http://www.edampf-shop.com/WebRoot/Store/Shops/47213485610/5211/E481/5A43/612F/69D9/596E/87E8/B5A/USB-Kabel_auf_micro_USB2.png, [May., 2014].
- [98] Wikipedia. "USB". Internet: <http://en.wikipedia.org/wiki/USB>, [Apr., 2014].
- [99] R. J. Kinch. "Picture of Raspberry Pi". Internet: http://www.truetex.com/raspberry_pi_2-0.jpg, [May., 2014].
- [100] ebaths.co.uk. "Picture of silver button". Internet: [http://www.ebaths.co.uk/ekmps/shops/ebaths1/images/twyford-air-button-single-flush-plastic-small-button-chrome-plated-b97940-900-p\[ekm\]288x288\[ekm\].jpg](http://www.ebaths.co.uk/ekmps/shops/ebaths1/images/twyford-air-button-single-flush-plastic-small-button-chrome-plated-b97940-900-p[ekm]288x288[ekm].jpg), [May., 2014].
- [101] NY PLATFORM. "Picture of relay". Internet: <http://www.ebay.com/itm/321341659057>, [May., 2014].
- [102] hifiduino.wordpress.com. "Picture of Arduino Mega 2560". Internet: http://hifiduino.files.wordpress.com/2012/03/arduinomega2560_r2_front.jpg, [May., 2014].
- [103] pihomeserver.fr. "Picture of Slice of Pi v1.1". Internet: <http://www.pihomeserver.fr/wp-content/uploads/2013/07/P1060953.jpg>, [May., 2014].
- [104] S. Breuning. "Controlling GPIO with wiringPi". Internet: <http://raspberryywebserver.com/gpio/wiringPi.html>, [Apr., 2014].
- [105] debian.org. "Ralink RT2070, RT2770, RT2870, RT3070, RT3071, RT3072, RT3370, RT3572, RT5370, RT5372 devices (rt2800usb)". Internet: <https://wiki.debian.org/rt2800usb>, [Jan., 2014].
- [106] T. Sounthala. "How to Setup Asus USB N10 Wifi Adapter On Raspberry Pi". Internet: http://databoyz.wordpress.com/tag/how-to-setup-network-and-wpa_supplicant-conf-file-on-raspberry-pi/, Oct., 2012 [Feb., 2014].
- [107] eLinux.org. "Rpi Low-level peripherals - GPIO hardware hacking". Internet: http://elinux.org/Rpi_Low-level_peripherals#GPIO_hardware_hacking, [Apr., 2014].
- [108] wikia.com. "How to set up a NAT router on a Linux-based computer". Internet: http://how-to.wikia.com/wiki/How_to_set_up_a_NAT_router_on_a_Linux-based_computer, [Feb.,

- 2014].
- [109] adafruit.com. "Setting up a Raspberry Pi as a WiFi access point". Internet: <http://learn.adafruit.com/setting-up-a-raspberry-pi-as-a-wifi-access-point>, [Apr., 2014].
- [110] Wireshark Foundation. "Filtering while capturing". Internet: http://www.wireshark.org/docs/wsug_html_chunked/ChCapCaptureFilterSection.html, [Apr., 2014].
- [111] Me in IT Consultancy. "A shell script to measure network throughput on Linux machines". Internet: <http://meinit.nl/shell-script-measure-network-throughput-linux-machines>, [Jan., 2014].
- [112] kernel.org. "hostapd Linux documentation page". Internet: <http://wireless.kernel.org/en/users/Documentation/hostapd>, [Apr., 2014].
- [113] debian.org. "NetworkConfiguration". Internet: <https://wiki.debian.org/NetworkConfiguration>, [Mar., 2014].
- [114] H.W. Tseng, L. M. Grupp, and S. Swanson. "Understanding the Impact of Power Loss on Flash Memory". Internet: <http://cseweb.ucsd.edu/users/swanson/papers/DAC2011PowerCut.pdf>, [Apr., 2014].
- [115] S. Moorthy. "Unix Nohup: Run a Command or Shell-Script Even after You Logout". Internet: <http://linux.101hacks.com/unix/nohup-command/>, [Apr., 2014].
- [116] E. Groothuis. "General Commands Manual - dhcping". Internet: <http://www.mavetju.org/unix/dhcping-man.php>, [May., 2014].
- [117] M. Griffa. "BASH Programming - Introduction HOW-TO - Misc." Internet: <http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-10.html>, Jul., 2000 [Jan., 2014].
- [118] linuxquestions.org-user: Matir. "how to specify default network interface?" Internet: <http://www.linuxquestions.org/questions/debian-26/how-to-specify-default-network-interface-373181/>, Oct., 2005 [Apr., 2014].
- [119] linuxquestions.org-user: jschiwal. "bash command for removing special characters from string". Internet: <http://www.linuxquestions.org/questions/linux-newbie-8/bash-command-for-removing-special-characters-from-string-644828/>, May., 2008 [Jan., 2014].
- [120] nixCraft. "Bash For Loop Examples". Internet: <http://www.cyberciti.biz/faq/bash-for-loop/>, Oct., 2008 [Apr., 2014].
- [121] M. Garrels. "Introduction to if". Internet: <http://tldp.org/LDP/Bash-Beginners->

- [Guide/html/sect_07_01.html](#), Dec., 2008 [Mar., 2014].
- [122] T. Amoyal. "How can I read pcap files in a friendly format?" Internet: <http://serverfault.com/questions/38626/how-can-i-read-pcap-files-in-a-friendly-format>, Aug., 2009 [Apr., 2014].
- [123] nixCraft. "How To Use awk In Bash Scripting". Internet: <http://www.cyberciti.biz/faq/bash-scripting-using-awk/>, Aug., 2009 [Feb., 2014].
- [124] thegeekstuff.com-user: Sasikala. "4 Bash If Statement Examples (If then fi, If then else fi, If elif else fi, Nested if)". Internet: <http://www.thegeekstuff.com/2010/06/bash-if-statement-examples/>, Jun., 2010 [Mar., 2014].
- [125] nixCraft. "Bash String Comparison: Find Out IF a Variable Contains a Substring". Internet: <http://www.cyberciti.biz/faq/bash-find-out-if-variable-contains-substring/>, Mar., 2010 [Feb., 2014].
- [126] stackoverflow.com-user: wsware. "How to get a password from a shell script without echoing". Internet: <http://stackoverflow.com/questions/3980668/how-to-get-a-password-from-a-shell-script-without-echoing>, Oct., 2010 [Feb., 2014].
- [127] R. Natarajan. "7 Linux Grep OR, Grep AND, Grep NOT Operator Examples". Internet: <http://www.thegeekstuff.com/2011/10/grep-or-and-not-operators/>, Oct., 2011 [Mar., 2014].
- [128] nixCraft. "Bash Infinite Loop Examples". Internet: <http://www.cyberciti.biz/faq/bash-infinite-loop/>, Jan., 2011 [Jan., 2014].
- [129] J. Bennett. "Deauthenticated due to local deauth request on a tp-link wa901nd v2 running wpa2-psk". Internet: <https://dev.openwrt.org/ticket/10025>, Aug., 2011 [May., 2014].
- [130] K. Horvath. "How can I add numbers in a bash script". Internet: <http://stackoverflow.com/questions/6348902/how-can-i-add-numbers-in-a-bash-script>, Jun., 2011 [Apr., 2014].
- [131] T. Anderson. "How can I measure a duration in seconds in a Linux shell script?" Internet: <http://stackoverflow.com/questions/5152858/how-can-i-measure-a-duration-in-seconds-in-a-linux-shell-script>, Mar., 2011 [Apr., 2014].
- [132] M. Jdidi. "Uncoment lines using bash script". Internet: <http://stackoverflow.com/questions/7751065/uncoment-lines-using-bash-script>, Oct., 2011 [Feb., 2014].

- [133] nixCraft. "Bash Shell: Find Out If a Variable Is Empty Or Not". Internet: <http://www.cyberciti.biz/faq/unix-linux-bash-script-check-if-variable-is-empty/>, Dec., 2012 [Apr., 2014].
- [134] K. Jones. "Bash: test mutual equality of multiple variables?" Internet: <http://stackoverflow.com/questions/8812089/bash-test-mutual-equality-of-multiple-variables>, Jan., 2012 [Apr., 2014].
- [135] O. Warner. "List all MAC addresses and their associated IPs in my local network(LAN)". Internet: <http://askubuntu.com/questions/406792/list-all-mac-addresses-and-their-associated-ips-in-my-local-networklan>, Jan., 2014 [Apr., 2014].
- [136] J. Tokar. "Hotspot – WiFi Access Point". Internet: <http://raspberry-at-home.com/hotspot-wifi-access-point/>, Jun., 2013 [Mar., 2014].
- [137] J. Malinen. "hostapd.conf". Internet: <http://hostap.epitest.fi/cgit/hostap/plain/hostapd/hostapd.conf>, Jan., 2013 [Mar., 2014].
- [138] linux.org-user: rob.1. "File Permissions - chmod". Internet: <http://www.linux.org/threads/file-permissions-chmod.4094/>, Jul., 2013 [Apr., 2014].

Appendices

Appendix 1 – Installation script..... 92

Appendix 2 – Failover script 105

Appendix 3 – Arduino script 111

Appendix 4 – Selftest script..... 115

Appendix 5 – Heartbeat signal to watchdog (script)..... 118

Appendix 6 – Stay alive when signal from watchdog is received (script) 119

Appendix 7 - Wi-Fi Client..... 120

Appendix 1 – Installation script

```
#!/bin/bash
if [ -e progress.txt ]
then
    progress=$(<progress.txt)
else
    progress=0
    echo $progress > progress.txt
fi

while [ $progress -eq 0 ]
do
    mkdir temp
    apt-get update
    echo "Enabling SSH server"
    update-rc.d ssh enable
    invoke-rc.d ssh start
    echo "Overclocking... Valid after next reboot"
    sed -e "s/^\`cat /boot/config.txt | grep arm_freq`/arm_freq=950/g" -e "s/^\`cat
/boot/config.txt | grep core_freq`//g" -e "s/^\`cat /boot/config.txt | grep
sdram_freq`//g" -e "s/^\`cat /boot/config.txt | grep over_voltage`//g"
/boot/config.txt > /boot/config.txt
    echo "core_freq=250" >> /boot/config.txt
    echo "sdram_freq=450" >> /boot/config.txt
    echo "over_voltage=6" >> /boot/config.txt
    echo "Do you want to install e-mail? You need an Gmail address to do so!"
    read installemail

    y="y"
    yes="yes"
    client="c"
    AP="a"

    if [ "$installemail" == "$y" ]
    then
        instmail=1
        echo "Will install email!"
        apt-get install postfix -y
        echo "What is your Gmail username (before @gmail.com)?"
        read gmailuser
        echo "What is your Gmail password?"
        read -s gmailpass
        echo "Do you want to be notified on mail when this installation needs
input?"
        read installnotification
        if [ "$installnotification" == "$y" ]
        then
            notify=1
            echo "Will set up notification. On which mail address do you
wish to be notified?"
            read mailaddress
            elif [ "$installnotification" == "$yes" ]
```

```
        then
            notify=1
            echo "Will set up notification. On which mail address do you
wish to be notified?"
            read mailaddress
        else
            notify=0
            echo "Will not set up notification"
        fi
    elif [ "$installemail" == "yes" ]
    then
        instmail=1
        echo "Will install email!"
        apt-get install postfix -y
        echo "What is your Gmail username (before @gmail.com)?"
        read gmailuser
        echo "What is your Gmail password?"
        read -s gmailpass
        echo "Do you want to be notified on mail when this installation needs
input?"
        read installnotification
        if [ "$installnotification" == "y" ]
        then
            notify=1
            echo "Will set up notification. On which mail address do you
wish to be notified?"
            read mailaddress
        elif [ "$installnotification" == "yes" ]
        then
            notify=1
            echo "Will set up notification. On which mail address do you
wish to be notified?"
            read mailaddress
        else
            notify=0
            echo "Will not set up notification"
        fi
    else
        instmail=0
        echo "Skipping install of email"
    fi
    echo "Disconnect all wireless network devices now! Do not connect until
asked! Press ENTER to continue."
    read
    echo "Do you want to install 3G/4G?"
    read installmbb
    echo 'Do you want to install Wi-Fi client or access point? Type "c" for
client, "a" for accesspoint or "n" to skip Wi-Fi.'
    read installwifi
    echo "Do you want to install secondary ethernet for routing?"
    read installeth1

    if [ "$installmbb" == "y" ]
    then
```

```
        instmbb=1
        echo "will install mobile broadband"
    elif [ "$installmbb" == "yes" ]
    then
        instmbb=1
        echo "will install mobile broadband"
    else
        instmbb=0
        echo "Skipping install mobile broadband"
    fi

    if [ "$installwifi" == "$client" ]
    then
        instwificlient=1
        instwifiap=0
        echo "will install wi-Fi client"
    elif [ "$installwifi" == "$AP" ]
    then
        instwifiap=1
        instwificlient=0
        echo "will install wi-Fi access point"
    else
        instwificlient=0
        instwifiap=0
        echo "Skipping install of Wi-Fi"
    fi

    if [ "$installeth1" == "$y" ]
    then
        insteth1=1
        echo "will install ethernet router"
    elif [ "$installeth1" == "yes" ]
    then
        insteth1=1
        echo "will install ethernet router"
    else
        insteth1=0
        echo "Skipping install of secondary interface"
    fi

    if [ $instmbb -eq 1 ]
    then
        echo "Please connect 3G/4G modem now! Will now ask for mobile
broadband details."
        echo "Enter apn"
        read apnname
        echo "Enter apn username"
        read apusername
        echo "Enter apn password"
        read apnpassword
        echo "Running lusb... Will then ask for ID of mobile broadband
device! It's given in the format ID xxxx:yyyy."
        sleep 3
    fi
```

```
    lsusb
    echo "Enter vendor ID (xxxx in the xxxx:yyyy format)"
    read vendorid
    echo "Enter product ID (yyyy in the xxxx:yyyy format)"
    read productid
fi
if [ $instwificlient -eq 1 ]
then
    echo "Will now ask for Wi-Fi details. Do NOT connect Wi-Fi adapter
yet!"
    echo "Enter SSID"
    read ssidname
    echo "Enter WPA2 key"
    read wpa2password
fi
if [ $instwifiap -eq 1 ]
then
    echo "Will now ask for Wi-Fi details. Do NOT connect Wi-Fi adapter
yet!"
    echo "Enter SSID"
    read ssidname
    echo "Enter WPA2 key"
    read wpa2password
fi
echo "Upgrading current software..."
if [ $notify -eq 1 ]
then
    echo "This might take a few minutes. You will be notified by mail when
you are needed."
fi
apt-get upgrade -y
apt-get install raspi-config -y

if [ $instmail -eq 1 ]
then
    apt-get install mailutils libsasl2-2 ca-certificates libsasl2-modules
-y
    sed -e "s/^\`cat /etc/postfix/main.cf | grep relayhost`//g" -e "s/^\`cat
/etc/postfix/main.cf | grep smtp_sasl_auth_enable`//g" -e "s/^\`cat
/etc/postfix/main.cf | grep smtp_sasl_password_maps`//g" -e "s/^\`cat
/etc/postfix/main.cf | grep smtp_sasl_security_options`//g" -e "s/^\`cat
/etc/postfix/main.cf | grep smtp_tls_CAfile`//g" -e "s/^\`cat /etc/postfix/main.cf |
grep smtp_use_tls`//g" /etc/postfix/main.cf > /etc/postfix/main.cf
    echo "relayhost = [smtp.gmail.com]:587" >> /etc/postfix/main.cf
    echo "smtp_sasl_auth_enable = yes" >> /etc/postfix/main.cf
    echo "smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd" >>
/etc/postfix/main.cf
    echo "smtp_sasl_security_options = noanonymous" >>
/etc/postfix/main.cf
    echo "smtp_tls_CAfile = /etc/postfix/cacert.pem" >>
/etc/postfix/main.cf
    echo "smtp_use_tls = yes" >> /etc/postfix/main.cf
    echo "[smtp.gmail.com]:587 $gmailuser@gmail.com:$gmailpass" >
/etc/postfix/sasl_passwd
    chmod 400 /etc/postfix/sasl_passwd
    postmap /etc/postfix/sasl_passwd
```



```
cat /etc/ssl/certs/Thawte_Premium_Server_CA.pem | sudo tee -a
/etc/postfix/cacert.pem
/etc/init.d/postfix reload
if [ $notify -eq 1 ]
then
    echo $mailaddress > temp/mailaddress.txt
    echo "Sending a test mail"
    echo "Test mail from postfix" | mail -s "Test Postfix"
$mailaddress
fi
fi

if [ $instmbb -eq 1 ]
then
    echo "Installing mobile broadband software..."
    apt-get install ppp wvdial usb-modeswitch -y
fi

echo "Storing progress before reboot"
echo $instmail > temp/installmail.txt
echo $notify > temp/notify.txt
echo $instmbb > temp/installmbb.txt
echo $instwificlient > temp/installwificlient.txt
echo $instwifiap > temp/installwifiap.txt
echo $insteth1 > temp/installeth1.txt
if [ $instmbb -eq 1 ]
then
    echo $apnname > temp/apnname.txt
    echo $apnusername > temp/apnuser.txt
    echo $apnpassword > temp/apnpassword.txt
    echo $vendorid > temp/defaultvendor.txt
    echo $productid > temp/defaultproduct.txt
fi
if [ $instwificlient -eq 1 ]
then
    echo $ssidname > temp/ssid.txt
    echo $wpa2password > temp/wpa2.txt
fi
if [ $instwifiap -eq 1 ]
then
    echo $ssidname > temp/ssid.txt
    echo $wpa2password > temp/wpa2.txt
fi
progress=1
echo 1 > progress.txt
if [ $instmbb -eq 1 ]
then
    echo "If mobile broadband modem is connected to self-powered USB-hub,
disconnect the power supply of the hub while rebooting! Press ENTER to reboot!"
    if [ $notify -eq 1 ]
    then
        echo "If mobile broadband modem is connected to self-powered
USB-hub, disconnect the power supply of the hub while rebooting! Confirm reboot!" |
mail -s "Installation progress" $mailaddress
```

```
        fi
        read
        reboot
        sleep 10
    fi
done

while [ $progress -eq 1 ]
do
    instmail=$(cat temp/installmail.txt)
    notify=$(cat temp/notify.txt)
    if [ $notify -eq 1 ]
    then
        mailaddress=$(cat temp/mailaddress.txt)
    fi
    instmbb=$(cat temp/installmbb.txt)
    instwificlient=$(cat temp/installwificlient.txt)
    instwifiap=$(cat temp/installwifiap.txt)
    insteth1=$(cat temp/installeth1.txt)
    if [ $instmbb -eq 1 ]
    then
        echo "Continuing from last time. Reconnect power supply to the USB-
hub"
        echo "Running lsusb... Will then ask for ID of mobile broadband
device! It's given in the format ID xxxx:yyyy."
        if [ $notify -eq 1 ]
        then
            echo "Reconnect power supply to the USB-hub! waiting for
input..." | mail -s "Installation progress" $mailaddress
        fi
        lsusb
        echo "Enter vendor ID (xxxx in the xxxx:yyyy format)"
        read targetvendorid
        echo "Enter product ID (yyyy in the xxxx:yyyy format)"
        read targetproductid
        if [ $notify -eq 1 ]
        then
            echo "This might take a few minutes. You will be notified by
mail when you are needed."
        fi
        echo "Customizing /etc/usb_modeswitch.conf"
        defaultvendor=$(cat temp/defaultvendor.txt)
        defaultproduct=$(cat temp/defaultproduct.txt)
        cwd=$(pwd)
        echo "DefaultVendor=0x$defaultvendor" > /etc/usb_modeswitch.conf
        echo "DefaultProduct=0x$defaultproduct" >> /etc/usb_modeswitch.conf
        echo "" >> /etc/usb_modeswitch.conf
        echo "TargetVendor=0x$targetvendorid" >> /etc/usb_modeswitch.conf
        echo "TargetProduct=0x$targetproductid" >> /etc/usb_modeswitch.conf
        echo "" >> /etc/usb_modeswitch.conf
        cd /tmp
        tar -xzf /usr/share/usb_modeswitch/configPack.tar.gz
        $defaultvendor\:$defaultproduct
        for mc in $(cat $defaultvendor\:$defaultproduct | grep MessageContent)
```

```
do
    echo $mc >> /etc/usb_modeswitch.conf
done
cd $cwd

echo "Customizing /etc/wvdial.conf"
wvdialconf > temp/wvdialconf.txt
wvdialmodem=`cat temp/wvdialconf.txt | grep "Found a modem on " | awk
'{ print $5 }'`
modem=${wvdialmodem//[.]/}
apnname=$(<temp/apnname.txt)
apnuser=$(<temp/apnuser.txt)
apnpassword=$(<temp/apnpassword.txt)
echo "[Dialer $apnname]" > /etc/wvdial.conf
echo "Init1 = ATZ" >> /etc/wvdial.conf
echo "Init2 = ATQ0 V1 E1 S0=0 &C1 &D2 +FCLASS=0" >> /etc/wvdial.conf
echo "Init3 = AT+CGDCONT=1,\"IP\", \"$apnname\"" >> /etc/wvdial.conf
echo "Stupid mode = 1" >> /etc/wvdial.conf
echo "Modem Type = Analog Modem" >> /etc/wvdial.conf
echo "ISDN = 0" >> /etc/wvdial.conf
echo "Phone = *99#" >> /etc/wvdial.conf
echo "Modem = $modem" >> /etc/wvdial.conf
echo "Username = {$apnuser}" >> /etc/wvdial.conf
echo "Password = {$apnpassword}" >> /etc/wvdial.conf
echo "Baud = 9600" >> /etc/wvdial.conf
fi
progress=2
echo 2 > progress.txt
done

while [ $progress -eq 2 ]
do
    instmail=$(<temp/installmail.txt)
    notify=$(<temp/notify.txt)
    if [ $notify -eq 1 ]
    then
        mailaddress=$(<temp/mailaddress.txt)
    fi
    instmbb=$(<temp/installmbb.txt)
    instwificlient=$(<temp/installwificlient.txt)
    instwifiap=$(<temp/installwifiap.txt)
    insteth1=$(<temp/installeth1.txt)
    if [ $instwificlient -eq 1 ]
    then
        echo "Installing wi-Fi client..."
        echo "Removing old Wi-Fi drivers"
        apt-get remove firmware-ralink -y

        ###https://wiki.debian.org/rt2800usb:
        echo "Temporarily changing apt-get sources"
        cp /etc/apt/sources.list /tmp/sources.list.tmp
        echo "deb http://http.debian.net/debian/ wheezy main contrib non-free"
    > /etc/apt/sources.list
```

```
apt-get update
echo "Install Wi-Fi drivers"
apt-get install firmware-ralink -y --force-yes
echo "Changing back to old apt-get source"
cp /tmp/sources.list.tmp /etc/apt/sources.list
apt-get update

ssidname=$(cat temp/ssid.txt)
wpa2password=$(cat temp/wpa2.txt)

echo "Connect Wi-Fi-adapter now! Press ENTER to continue."
if [ $notify -eq 1 ]
then
    echo "Connect Wi-Fi-adapter now! Press ENTER to continue." |
mail -s "Installation progress" $mailaddress
fi
read
sleep 5
###http://databoyz.wordpress.com/tag/how-to-setup-network-and-
wpa_supplicant-conf-file-on-raspberry-pi/:
echo "Updating /etc/wpa_supplicant/wpa_supplicant.conf"
echo "ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev" >
/etc/wpa_supplicant/wpa_supplicant.conf
echo "update_config=1" >> /etc/wpa_supplicant/wpa_supplicant.conf
echo "network={ " >> /etc/wpa_supplicant/wpa_supplicant.conf
echo "ssid=\"$ssidname\" " >> /etc/wpa_supplicant/wpa_supplicant.conf
echo "scan_ssid=1" >> /etc/wpa_supplicant/wpa_supplicant.conf
echo "psk=\"$wpa2password\" " >>
/etc/wpa_supplicant/wpa_supplicant.conf
echo "proto=RSN" >> /etc/wpa_supplicant/wpa_supplicant.conf
echo "key_mgmt=WPA-PSK" >> /etc/wpa_supplicant/wpa_supplicant.conf
echo "pairwise=CCMP" >> /etc/wpa_supplicant/wpa_supplicant.conf
echo "auth_alg=OPEN" >> /etc/wpa_supplicant/wpa_supplicant.conf
echo "}" >> /etc/wpa_supplicant/wpa_supplicant.conf

fi

if [ $insteth1 -eq 1 ]
then
    defaultgw=`cat /var/lib/dhcp/dhclient.eth0.leases | grep router | tail
-1 | awk '{ print $3 }'`
    gw=${defaultgw//[;]/}
    if [ $instwifiap -eq 1 ]
    then
        echo "Connect Ethernet and Wi-Fi adapters now! Press ENTER to
continue."

        if [ $notify -eq 1 ]
        then
            echo "Connect Ethernet and Wi-Fi adapters now! Press
ENTER to continue." | mail -s "Installation progress" $mailaddress
        fi
        read
        sleep 5
        echo "Setting up eth1"
        ifconfig eth1 up
    fi
fi
```

```

        ifconfig eth1 10.0.0.1
        echo "Setting up wlan0"
        ifconfig wlan0 up
        ifconfig wlan0 10.0.0.1
        apt-get install hostapd isc-dhcp-server -y
    else
        echo "Connect Ethernet adapter now! Press ENTER to continue."
        if [ $notify -eq 1 ]
        then
            echo "Connect Ethernet adapter now! Press ENTER to
continue." | mail -s "Installation progress" $mailaddress
            fi
            read
            sleep 5
            echo "Setting up eth1"
            ifconfig eth1 up
            ifconfig eth1 10.0.0.1
            apt-get install isc-dhcp-server -y
        fi
        sed -e "s/^#authoritative;/authoritative;/g" /etc/dhcp/dhcpd.conf >
temp/dhcpd.conf
        cp temp/dhcpd.conf /etc/dhcp/dhcpd.conf
        sed -e "s/^option domain-name/#option domain-name/g"
/etc/dhcp/dhcpd.conf > temp/dhcpd2.conf
        cp temp/dhcpd2.conf /etc/dhcp/dhcpd.conf
        echo "" >> /etc/dhcp/dhcpd.conf
        echo "subnet 10.0.0.0 netmask 255.255.255.0 {" >> /etc/dhcp/dhcpd.conf
        echo "range 10.0.0.10 10.0.0.59;" >> /etc/dhcp/dhcpd.conf
        echo "option broadcast-address 10.0.0.255;" >> /etc/dhcp/dhcpd.conf
        echo "option routers 10.0.0.1;" >> /etc/dhcp/dhcpd.conf
        echo "default-lease-time 600;" >> /etc/dhcp/dhcpd.conf
        echo "max-lease-time 7200;" >> /etc/dhcp/dhcpd.conf
        echo 'option domain-name "local";' >> /etc/dhcp/dhcpd.conf
        echo "option domain-name-servers 8.8.8.8, 8.8.4.4;" >>
/etc/dhcp/dhcpd.conf
        echo "}" >> /etc/dhcp/dhcpd.conf
        sed -e 's/^#net.ipv4.ip_forward=1/net.ipv4.ip_forward=1/g'
/etc/sysctl.conf > temp/sysctl.conf
        cp temp/sysctl.conf /etc/sysctl.conf
        sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
        if [ $instwifiap -eq 1 ]
        then
            sed -e 's/^INTERFACES=""/INTERFACES="eth1 wlan0"/g'
/etc/default/isc-dhcp-server > temp/isc-dhcp-server
        else
            sed -e 's/^INTERFACES=""/INTERFACES="eth1"/g' /etc/default/isc-
dhcp-server > temp/isc-dhcp-server
        fi
        cp temp/isc-dhcp-server /etc/default/isc-dhcp-server
        iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
        iptables -A FORWARD -i eth0 -o eth1 -m state --state
RELATED,ESTABLISHED -j ACCEPT
        iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
        if [ $instwifiap -eq 1 ]
        then

```

```
iptables -A FORWARD -i eth0 -o wlan0 -m state --state
RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
fi
iptables-save > /etc/iptables.rules
route del default
route add default gw $gw eth0
elif [ $instwifiap -eq 1 ]
then
    defaultgw=`cat /var/lib/dhcp/dhclient.eth0.leases | grep router | tail
-1 | awk '{ print $3 }'`
    gw=${defaultgw//[;]/}
    echo "Connect Wi-Fi adapter now! Press ENTER to continue."
    if [ $notify -eq 1 ]
    then
        echo "Connect Wi-Fi adapter now! Press ENTER to continue." |
mail -s "Installation progress" $mailaddress
        fi
        read
        sleep 5
        echo "Setting up wlan0"
        ifconfig wlan0 up
        ifconfig wlan0 10.0.0.1
        apt-get install hostapd isc-dhcp-server -y
        sed -e "s/^#authoritative;/authoritative;/g" /etc/dhcp/dhcpd.conf >
temp/dhcpd.conf
        cp temp/dhcpd.conf /etc/dhcp/dhcpd.conf
        sed -e "s/^option domain-name/#option domain-name/g"
/etc/dhcp/dhcpd.conf > temp/dhcpd2.conf
        cp temp/dhcpd2.conf /etc/dhcp/dhcpd.conf
        echo "" >> /etc/dhcp/dhcpd.conf
        echo "subnet 10.0.0.0 netmask 255.255.255.0 {" >> /etc/dhcp/dhcpd.conf
        echo "range 10.0.0.10 10.0.0.59;" >> /etc/dhcp/dhcpd.conf
        echo "option broadcast-address 10.0.0.255;" >> /etc/dhcp/dhcpd.conf
        echo "option routers 10.0.0.1;" >> /etc/dhcp/dhcpd.conf
        echo "default-lease-time 600;" >> /etc/dhcp/dhcpd.conf
        echo "max-lease-time 7200;" >> /etc/dhcp/dhcpd.conf
        echo 'option domain-name "local";' >> /etc/dhcp/dhcpd.conf
        echo "option domain-name-servers 8.8.8.8, 8.8.4.4;" >>
/etc/dhcp/dhcpd.conf
        echo "}" >> /etc/dhcp/dhcpd.conf
        sed -e 's/^#net.ipv4.ip_forward=1/net.ipv4.ip_forward=1/g'
/etc/sysctl.conf > temp/sysctl.conf
        cp temp/sysctl.conf /etc/sysctl.conf
        sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
        sed -e 's/^INTERFACES=""/INTERFACES="wlan0"/g' /etc/default/isc-dhcp-
server > temp/isc-dhcp-server
        cp temp/isc-dhcp-server /etc/default/isc-dhcp-server
        iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
        iptables -A FORWARD -i eth0 -o wlan0 -m state --state
RELATED,ESTABLISHED -j ACCEPT
        iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
        iptables-save > /etc/iptables.rules
        route del default
        route add default gw $gw eth0
```

```
fi

echo "Updating /etc/network/interfaces"
echo "auto lo" > /etc/network/interfaces
echo "iface lo inet loopback" >> /etc/network/interfaces
echo " " >> /etc/network/interfaces
echo "auto eth0" >> /etc/network/interfaces
echo "allow-hotplug eth0" >> /etc/network/interfaces
echo "iface eth0 inet dhcp" >> /etc/network/interfaces
echo " " >> /etc/network/interfaces
if [ $insteth1 -eq 1 ]
then
    echo "auto eth1" >> /etc/network/interfaces
    echo "allow-hotplug eth1" >> /etc/network/interfaces
    echo "iface eth1 inet static" >> /etc/network/interfaces
    echo "address 10.0.0.1" >> /etc/network/interfaces
    echo "netmask 255.255.255.0" >> /etc/network/interfaces
    echo "gateway 10.0.0.1" >> /etc/network/interfaces
    echo " " >> /etc/network/interfaces
fi
if [ $instwificlient -eq 1 ]
then
    echo "allow-hotplug wlan0" >> /etc/network/interfaces
    echo "iface wlan0 inet manual" >> /etc/network/interfaces
    echo " wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf" >>
/etc/network/interfaces
fi
if [ $instwifiap -eq 1 ]
then
    echo "allow-hotplug wlan0" >> /etc/network/interfaces
    echo "iface wlan0 inet static" >> /etc/network/interfaces
    echo " address 10.0.0.1" >> /etc/network/interfaces
    echo " netmask 255.255.255.0" >> /etc/network/interfaces
    echo "" >> /etc/network/interfaces
    echo "#iface wlan0 inet manual" >> /etc/network/interfaces
    echo "# wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf" >>
/etc/network/interfaces

    echo "Setting up hostapd"
    ssidname=$(cat temp/ssid.txt)
    wpa2password=$(cat temp/wpa2.txt)
    echo "interface=wlan0" > /etc/hostapd/hostapd.conf
    echo "driver=rtl871xdrv" >> /etc/hostapd/hostapd.conf
    echo "ssid=$ssidname" >> /etc/hostapd/hostapd.conf
    echo "hw_mode=g" >> /etc/hostapd/hostapd.conf
    echo "channel=6" >> /etc/hostapd/hostapd.conf
    echo "macaddr_acl=0" >> /etc/hostapd/hostapd.conf
    echo "auth_algs=1" >> /etc/hostapd/hostapd.conf
    echo "ignore_broadcast_ssid=0" >> /etc/hostapd/hostapd.conf
    echo "wpa=2" >> /etc/hostapd/hostapd.conf
    echo "wpa_passphrase=$wpa2password" >> /etc/hostapd/hostapd.conf
    echo "wpa_key_mgmt=WPA-PSK" >> /etc/hostapd/hostapd.conf
    echo "wpa_pairwise=TKIP" >> /etc/hostapd/hostapd.conf
```

```
echo "rsn_pairwise=CCMP" >> /etc/hostapd/hostapd.conf
echo "logger_syslog=-1" >> /etc/hostapd/hostapd.conf
echo "logger_syslog_level=2" >> /etc/hostapd/hostapd.conf
echo "logger_stdout=-1" >> /etc/hostapd/hostapd.conf
echo "logger_stdout_level=2" >> /etc/hostapd/hostapd.conf
sed -e
's/\^#DAEMON_CONF=\\"/DAEMON_CONF=\\"/etc/hostapd/hostapd.conf"/g'
/etc/default/hostapd > temp/hostapd
cp temp/hostapd /etc/default/hostapd

echo "Updating hostapd"
route
echo "Press ENTER if eth0 is default route"
if [ $notify -eq 1 ]
then
    echo "Confirm if eth0 is default route." | mail -s "Installation
progress" $mailaddress
fi
read
wget http://www.adafruit.com/downloads/adafruit_hostapd.zip
unzip adafruit_hostapd.zip
mv /usr/sbin/hostapd /usr/sbin/hostapd.ORIG
mv hostapd /usr/sbin
chmod 755 /usr/sbin/hostapd
else
echo "iface default inet dhcp" >> /etc/network/interfaces
fi

if [ $insteth1 -eq 1 ]
then
    sed -e 's/\^#\!\/bin\/sh\ \-e\/\^#\!\/bin\/bash/g' /etc/rc.local >
temp/rc.local2
    sed -e 's/\^exit 0//g' temp/rc.local2 > temp/rc.local
    echo "" >> temp/rc.local
    echo "iptables-restore < /etc/iptables.rules" >> temp/rc.local
    echo "" >> temp/rc.local
    echo "gw=\`cat /var/lib/dhcp/dhclient.eth0.leases | grep router | tail
-1 | awk '{ print \$3 }'\`" >> temp/rc.local
    echo "defgw=\${gw//[;]/}" >> temp/rc.local
    echo "route del default" >> temp/rc.local
    echo "route add default gw \$defgw eth0" >> temp/rc.local
    echo "" >> temp/rc.local
    echo "exit 0" >> temp/rc.local
    cp temp/rc.local /etc/rc.local
    if [ $instwifiap -eq 1 ]
    then
        service hostapd start
        update-rc.d hostapd enable
    fi
    service isc-dhcp-server start
    update-rc.d isc-dhcp-server enable
elif [ $instwifiap -eq 1 ]
then
```



```
sed -e 's/^\#\!\|\bin\sh\ \-e/^\#\!\|\bin\bash/g' /etc/rc.local >
temp/rc.local2
sed -e 's/^\exit 0//g' temp/rc.local2 > temp/rc.local
echo "" >> temp/rc.local
echo "iptables-restore < /etc/iptables.rules" >> temp/rc.local
echo "" >> temp/rc.local
echo "gw=\`cat /var/lib/dhcp/dhclient.eth0.leases | grep router | tail
-1 | awk '{ print \$3 }'\`" >> temp/rc.local
echo "defgw=\${gw//[;]}/" >> temp/rc.local
echo "route del default" >> temp/rc.local
echo "route add default gw \$defgw eth0" >> temp/rc.local
echo "" >> temp/rc.local
echo "hostapd /etc/hostapd/hostapd.conf" >> temp/rc.local
echo "" >> temp/rc.local
echo "exit 0" >> temp/rc.local
cp temp/rc.local /etc/rc.local
service hostapd start
update-rc.d hostapd enable
service isc-dhcp-server start
update-rc.d isc-dhcp-server enable

fi
echo "Resetting overclock to default. Valid after reboot."
sed -e "s/^\`cat /boot/config.txt | grep arm_freq`/arm_freq=700/g" -e "s/^\`cat
/boot/config.txt | grep core_freq`/core_freq=250/g" -e "s/^\`cat /boot/config.txt |
grep sdram_freq`/sdram_freq=400/g" -e "s/^\`cat /boot/config.txt | grep
over_voltage`/over_voltage=0/g" /boot/config.txt > /boot/config.txt
echo "Press ENTER to reboot"
if [ $notify -eq 1 ]
then
echo "Press ENTER to reboot and finalize setup!" | mail -s
"Installation progress" $mailaddress
fi
read
progress=3
echo 3 > progress.txt
rm -r temp
reboot
sleep 10
done

while [ $progress -eq 3 ]
do
echo "Completely installed!"
progress=4
done
exit
```

Appendix 2 – Failover script

```
#!/bin/bash

#Initial settings:
IF=eth0
pausetime=2 #Number of seconds between each run
staymin=1 #Number of minutes to stay connected to mobile broadband after primary
interface is back online
networkstate=0 #0=Internet access; 1=Local network access; 2=No network access
rxthreshold=150

#Initialization:
touch rx.txt
touch tx.txt
touch defaultgw.txt
touch mbbgw.txt
gw=`cat /var/lib/dhcp/dhclient.eth0.leases | grep router | tail -1 | awk '{ print
$3 }'`
defaultgw=${gw//[;]/}
mbbgw=$(
```

```
if [[ -z "$defaultgw" ]]
then
    defaultgw=$(  
defaultgw.txt)
else
    echo $defaultgw > defaultgw.txt
fi

if [[ -z "$mbbgw" ]]
then
    mbbgw=$(  
mbbgw.txt)
else
    echo $mbbgw > mbbgw.txt
fi
#echo "Mobile broadband gateway $mbbgw"

if [ $mbbconnected -eq 1 ]
then
    connectsec=$((SECONDS - $startconnect)
    echo "Still connected to mbb - will stay connected for  

[$$staysec - $connectsec] seconds."
    while [ $connectsec -ge $staysec ]
    do
        killall wvdial &
        mbbconnected=0
        connectsec=0
        sleep 2
    done
else
    echo "Not connected to mbb"
fi

#Check if cable connected
for interface in $(ls /sys/class/net/ | grep -e $IF);
do
    if [[ $(cat /sys/class/net/$interface/carrier) = 1 ]]
    then
        echo "Cable connected."
        #Calculation of sent and received bytes since boot
        ##Get current values
        rxbytes=$(ifconfig $IF | grep "RX bytes" | cut -d: -f2 |
awk '{ print $1 }')
        txbytes=$(ifconfig $IF | grep "TX bytes" | cut -d: -f3 |
awk '{ print $1 }')
        ##Get values from last run
        oldrx=$(  
rx.txt)
        oldtx=$(  
tx.txt)
        ##Write current values to files
        echo $rxbytes > rx.txt
        echo $txbytes > tx.txt
        ##Get difference between old and new values
        diffrx=$(expr $rxbytes - $oldrx)
        difftx=$(expr $txbytes - $oldtx)
```

```

diffsum=`expr $diffrx + $difftx`

#Print visible output before connectivity test
#echo "Default next hop gateway $defaultgw on interface
$IF"

#echo "Mobile broadband gateway $mbbgw"
#echo "Received bytes since last run: $diffrx"
#echo "Sent bytes since last run: $difftx"
echo "Total packages transmitted: $diffsum"
#echo $oldrx
#echo $oldtx

if [ $diffrx -le $rxthreshold ]
then
    echo "Packets received since last run! No more
tests needed."
    #Get gateway addresses on primary interface and
mobile broadband
elif [ $difftx -le 0 ]
then
    defaultgw=$(

```

```

                                echo "Packets sent since last run! No more tests
needed."
                                fi
                                else
                                echo "Cable disconnected! Setting mobile broadband as
default..."
                                networkstate=2
                                fi
                                done
                                done

                                if [ $mbbconnected -eq 0 ]
                                then
                                    echo "Connecting mobile broadband..."
                                    wvdial telenor &
                                    sleep 2
                                    mbbconnected=1
                                fi
                                #
                                route del default
                                mbbgw=`route | grep "ppp0" | awk '{ print $1 }'`
                                if [[ -z "$mbbgw" ]]
                                then
                                    mbbgw=$(<mbbgw.txt)
                                else
                                    echo $mbbgw > mbbgw.txt
                                fi
                                route add default gw $mbbgw
                                echo "nameserver $pridns" > /etc/resolv.conf
                                echo "nameserver $secdns" >> /etc/resolv.conf
                                currentgw=`route | grep 0.0.0.0 | awk '{ print $2 }'`
                                currentif=`route | grep 0.0.0.0 | awk '{ print $8 }'`
                                echo "Default gateway $currentgw on $currentif."
                                #
                                iptables --flush
                                #
                                iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
                                #
                                iptables -A FORWARD -i ppp0 -o eth1 -m state --state RELATED,ESTABLISHED
                                #
                                iptables -A FORWARD -i eth1 -o ppp0 -j ACCEPT
                                while [ $networkstate -eq 1 ]
                                do
                                    sleep $pausetime
                                    defaultgw=$(<defaultgw.txt)
                                #
                                route del default
                                route add default gw $defaultgw $IF
                                cp resolv.conf.backup /etc/resolv.conf
                                pingext=`ping -c $pingcount -l $pingcount -I $IF $pingexternal | grep
"$pingcount packets transmitted" | cut -d, -f3 | awk '{ print $1 }'`
                                pktlossext=${pingext//[%/]}
                                echo "$pktlossext percent packet loss on ping request."
                                if [ $pktlossext -le $losslimit ]
                                then
                                    echo "Internet connection, setting default settings"
                                    networkstate=0
                                    startconnect=$SECONDS
                                else

```

```

#           route del default
#           route add default gw $mbbgw
           echo "nameserver $pridns" > /etc/resolv.conf
           echo "nameserver $secdns" >> /etc/resolv.conf
grep "$pingcount packets transmitted" | cut -d, -f3 | awk '{ print $1 }'`
           pingtestnh=`ping -c $pingcount -l $pingcount -I $IF $defaultgw |
           pktlossnh=${pingtestnh//[%/]}
           if [ $pktlossnh -le $losslimit ]
           then
               echo "Still only local network access"
           else
               echo "No network connection on $IF. Check cables and
equipment!"
               networkstate=2
           fi
       fi
       currentgw=`route | grep 0.0.0.0 | awk '{ print $2 }'`
       currentif=`route | grep 0.0.0.0 | awk '{ print $8 }'`
       echo "Default gateway $currentgw on $currentif."
done

while [ $networkstate -eq 2 ]
do
    firstrun=1
    sleep $pausetime
    for interface in $(ls /sys/class/net/ | grep -e $IF);
    do
        if [[ $(cat /sys/class/net/$interface/carrier) = 1 ]]
        #pingtestnh=`ping -c $pingcount -l $pingcount -I $IF $defaultgw
| grep "$pingcount packets transmitted" | cut -d, -f3 | awk '{ print $1 }'`
        #pktlossnh=${pingtestnh//[%/]}
        #if [ $pktlossnh -le $losslimit ]
        then
            echo "Local network access"
            networkstate=1
        else
            echo "Still no network connection on $IF. Check cables
and equipment!"
        fi
    done
    currentif=`route | grep 0.0.0.0 | awk '{ print $8 }'`
    currentgw=`route | grep 0.0.0.0 | awk '{ print $2 }'`
    echo "Default gateway $currentgw on $currentif."

    while [ $firstrun -eq 1 ]
    do
        sleep $pausetime
        for interface in $(ls /sys/class/net/ | grep -e $IF);
        do
            if [[ $(cat /sys/class/net/$interface/carrier) = 1 ]]
            #pingtestnh=`ping -c $pingcount -l $pingcount -I $IF
$defaultgw | grep "$pingcount packets transmitted" | cut -d, -f3 | awk '{ print $1
}'`
            #pktlossnh=${pingtestnh//[%/]}

```

```

        #if [ $pcktlossnh -le $losslimit ]
        then
            echo "Local network access"
            networkstate=1
            firstrun=0
        #else
        #echo "Still no network connection on $IF. Check
cables and equipment!"
        fi
    done
done
done
done
exit
```

Appendix 3 – Arduino script

```
int buttonPin = 3;    // on/off button connected to analog pin 3
int beaconPin = 8;    // beacon signal from Raspberry Pi connected to analog pin 8
int onsignalPin = 30; // keep alive signal to Raspberry Pi on digital pin 30
int relaysignalPin = 52; // signal to relay switch on digital pin 52
int startupTime = 75000; // maximum time spent to start up Raspberry Pi
int shutdownTime = 15000; // standard time in milliseconds spent to shutdown
Raspberry Pi
int time = 0;
int newtime = 0;
int runtime = 0;

int buttonval = 0;    // variable to store the value read on button pin
int beaconval = 0;    // variable to store the value read on beacon pin

int initialState = 0; // variable to store state. This is the first phase of
the process
int startupState = 0; // variable to store state. If in startup phase, value
will be 1
int normalState = 0; // variable to store state. If in normal phase, value will
be 1
int buttonshutdownState = 0; // variable to store state. If button is pressed
to shutdown, value will be 1
int hardshutdownState = 0; // variable to store state. If beacon signal is not
received, value will be 1
int val = 0;

int run = 0; // variable to check for alternating beacon signal. if value is
too high, beacon from the Raspberry Pi is probably dead

void setup()

{
  pinMode(buttonPin,INPUT); // set signal direction
  pinMode(beaconPin,INPUT); // set signal direction
  pinMode(onsignalPin,OUTPUT); // set signal direction
  pinMode(relaysignalPin,OUTPUT); // set signal direction
  Serial.begin(9600); // setup serial
  initialState = 1; // enter initial state
}

void loop()

{
  while (initialState == 1) {
    digitalWrite(relaysignalPin,HIGH); // send signal to power the Pi
    digitalWrite(onsignalPin,HIGH); // send keep-alive signal to the Pi
    initialState = 0; // exit initial state
    startupState = 1; // enter startup state
  }

  while (startupState == 1) {
    beaconval = analogRead(beaconPin); // read the input pin
```



```
    if (beaconval == 1023) {
        startupState = 0;    // exit startup state
        normalState = 1;    // enter normal state
    }
}

while (normalState == 1) {
    time = millis();
    beaconval = analogRead(beaconPin);    // read the input pin
    while (beaconval == 1023) {
        buttonval = analogRead(buttonPin);    // read the input pin
        if (val == 1023) {
            normalState = 0;    // Exit the state
            buttonsShutdownState = 1;    // Restart the Raspberry Pi
        }
        beaconval = analogRead(beaconPin);    // read the input pin
        newtime = millis();
        runtime = newtime - time;
        if (runtime >= startupTime) {    // If the beacon does not change after this
amount of runs, the RPi is probably dead. (MUST CHECK THIS VALUE)
//        delay(startupTime);    // wait for beacon in case onboard watchdog has
restarted the raspberry pi
            normalState = 0;    // Exit the state
            beaconval = 1024;    // exit the while loop by returning a value that
could never be read on the analog pin (0-1023)
            hardShutdownState = 1;    // Restart the Raspberry Pi
        }
    }
}

time = millis();
while (beaconval == 0) {
    buttonval = analogRead(buttonPin);    // read the input pin
    if (val == 1023) {
        normalState = 0;    // Exit the state
        buttonsShutdownState = 1;    // Restart the Raspberry Pi
    }
    beaconval = analogRead(beaconPin);    // read the input pin
    newtime = millis();
    runtime = newtime - time;
    if (runtime >= startupTime) {    // If the beacon does not change after this
amount of runs, the RPi is probably dead. (MUST CHECK THIS VALUE)
//        delay(startupTime);    // wait for beacon in case onboard watchdog has
restarted the raspberry pi
        normalState = 0;    // Exit the state
        beaconval = 1024;    // exit the while loop by returning a value that
could never be read on the analog pin (0-1023)
        hardShutdownState = 1;    // Restart the Raspberry Pi
    }
}
}

// When exiting normal state, the Raspberry Pi is shutting down, either forced og
because the button was pressed.
// The following three lines are mutual for both states, and is kept in front of
the while loops as they only should be run once.
```

```
digitalWrite(onsignalPin,LOW);
delay(shutdownTime);
digitalWrite(relaysignalPin,LOW);

while (buttonshutdownState == 1) {
  buttonval = analogRead(buttonPin);    // read the input pin
  if (val == 1023) {
    buttonshutdownState = 0;    // Exit the state
    initialState = 1;    // Restart the Raspberry Pi
  }
}

while (hardshutdownState == 1) {
  hardshutdownState = 0;    // Exit the state
  initialState = 1;    // Restart the Raspberry Pi
}
}
```


Appendix 4 – Selftest script

```
#!/bin/bash
cat /var/log/syslog | grep hostapd > /tmp/syslog.tmp
lastSyslog=`cat /tmp/syslog.tmp | tail -1`
while true
do
iswlanIPset=`ifconfig wlan0 | grep -c 'inet addr'`
isEth1IPset=`ifconfig eth1 | grep -c 'inet addr'`
dhcpStatus=`service isc-dhcp-server status`
hostapdProcessRunning=`ps aux | grep hostapd | grep -v grep | grep -c hostapd`
wlan0tsharkProcessRunning=`ps aux | grep tshark | grep wlan0 | grep -v grep | grep -c tshark`
eth1tsharkProcessRunning=`ps aux | grep tshark | grep eth1 | grep -v grep | grep -c tshark`
currentwlan0LogEdit=`ls /tmp | grep outfile_wlan0 | cut -d'_' -f4 | tail -1`
lastwlan0LogEdit=`ls /tmp | grep outfile_wlan0 | grep -v $currentwlan0LogEdit`
tcpdump -ttttnr /tmp/$lastwlan0LogEdit | grep -v IP6 > /tmp/wlan0.tmp
cat /tmp/wlan0.tmp | cut -d' ' -f4 | cut -d'.' -f1-4 | grep . > /tmp/wlan0src.tmp
cat /tmp/wlan0.tmp | cut -d' ' -f6 | cut -d'.' -f1-4 | grep . > /tmp/wlan0dst.tmp
cat /var/log/syslog | grep hostapd > /tmp/syslog.tmp
chmod 777 /tmp/syslog.tmp
currentSyslog=`cat /tmp/syslog.tmp | tail -1`
lnOfLastSyslog=`cat /tmp/syslog.tmp | grep -n '$lastSyslog' | cut -d':' -f1`
nextSyslog=`expr $lnOfLastSyslog + 1`
recentDeauth=`cat /tmp/syslog.tmp | tail -n +$nextSyslog | grep -c 'IEEE 802.11: deauthenticated due to local'`
lastSyslog=$currentSyslog
wlan0clients=`arp -an | grep -c wlan0`
if [ $wlan0clients -ne 0 ]
then
i=1
offlineclients=0
while [ $i -le $wlan0clients ]
do
clientIP=`arp -an | grep wlan0 | cut -d'(' -f2 | cut -d')' -f1`
pingresult=`ping -c 3 -I wlan0 $clientIP | grep 'packet loss' | cut -d' ' -f6 | cut -d'%' -f1`
if [ $pingresult -eq 100 ]
then
offlineclients=`expr $offlineclients + 1`
fi
i=`expr $i + 1`
done
onlineclients=`expr $wlan0clients - $offlineclients`
else
onlineclients=0
fi
echo $onlineclients
currentEth1LogEdit=`ls /tmp | grep outfile_eth1 | cut -d'_' -f4 | tail -1`
lastEth1LogEdit=`ls /tmp | grep outfile_eth1 | grep -v $currentEth1LogEdit`
```

```
tcpdump -ttttnr /tmp/$lastEth1LogEdit | grep -v IP6 > /tmp/eth1.tmp
cat /tmp/eth1.tmp | cut -d' ' -f4 | cut -d'.' -f1-4 | grep '*.*.*.*' >
/tmp/eth1src.tmp
cat /tmp/eth1.tmp | cut -d' ' -f6 | cut -d'.' -f1-4 | grep '*.*.*.*' >
/tmp/eth1dst.tmp
lnnr=0
while read line
do
    lnnr=`expr $lnnr + 1`
    destination=`cat /tmp/eth1dst.tmp | tail -$lnnr | head -1`
    echo '$lnnr:$line:$destination'
done </tmp/eth1src.tmp

if [ $iswlanIPset -eq 1 ]
then
    echo 'wlan0 IP:          OK'
else
    echo 'wlan0 IP:          Not set'
    ifconfig wlan0 10.0.0.1
fi
if [ $isEth1IPset -eq 1 ]
then
    echo 'eth1 IP:           OK'
else
    echo 'eth1 IP:           Not set'
    ifconfig eth1 10.0.1.1
fi
if [[ $dhcpStatus == 'Status of ISC DHCP server: dhcpd is running.' ]]
then
    echo 'DHCP-server:       OK'
else
    echo 'DHCP-server:       Not running'
    sudo isc-dhcp-server restart
fi
if [ $hostapdProcessRunning -ne 0 ]
then
    if [[ ($recentDeauth -ne 0) && ($onlineClients -eq 0) ]]
    then
        echo 'Hostapd process:    Restarting...'
        killall hostapd
        ifdown wlan0
        ifup wlan0
        nohup hostapd /etc/hostapd/hostapd.conf &
        service isc-dhcp-server restart
        rm /tmp/outfile_wlan0*
        nohup tshark -b duration:20 -b files:2 -w /tmp/outfile_wlan0log -t d -
    fi
    echo 'Hostapd process:    OK'
else
    echo 'Hostapd process:    Not running'
```

```
        nohup hostapd /etc/hostapd/hostapd.conf &
        service isc-dhcp-server restart
    fi
    if [ $wlan0tsharkProcessRunning -ne 0 ]
    then
        echo 'wlan0 logging:          OK'
    else
        echo 'wlan0 logging:          Starting...'
        rm /tmp/outfile_wlan0*
        nohup tshark -b duration:20 -b files:2 -w /tmp/outfile_wlan0log -t d -i wlan0
    &
    fi
    if [ $eth1tsharkProcessRunning -ne 0 ]
    then
        echo 'Eth1 logging:          OK'
    else
        echo 'Eth1 logging:          Starting...'
        rm /tmp/outfile_eth1*
        nohup tshark -b duration:20 -b files:2 -w /tmp/outfile_eth1log -t d -i eth1 &
    fi

    if [[ ($wlan0ok -eq 1) && ($wlan0ok == $eth1ok) && ($eth1ok == $dhcpok) && ($dhcpok
    == $hostapdok) ]]
    then
        check=1
    fi
    sleep 10
done
exit
```

Appendix 5 – Heartbeat signal to watchdog (script)

```
#!/bin/bash
while true
do
    echo "1" > /sys/class/gpio/gpio4/value
    echo "on"
    sleep 0.5
    echo "0" > /sys/class/gpio/gpio4/value
    echo "off"
    sleep 0.5
done
exit
```

Appendix 6 – Stay alive when signal from watchdog is received (script)

```
#!/bin/bash
i=0
while true
do
    keepalive=`cat /sys/class/gpio/gpio17/value`
    if [[ "$keepalive" -ne "1" ]]
    then
#         echo "recieving shutdown signal"
        i=`expr $i + 1`
        if [[ "$i" -eq 10 ]]
        then
            shutdown now
            sleep 15
        fi
    else
#         echo "stay alive"
        i=0
    fi
#     echo $i
    sleep 1
done
exit
```


Appendix 7 - Wi-Fi Client

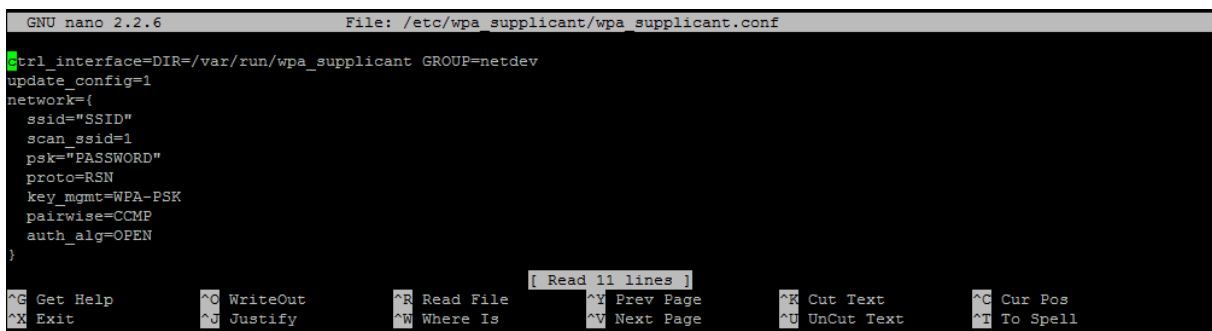
As the Wi-Fi client software excludes the possibility for access point, and vice versa, one cannot choose both. If the Wi-Fi access point is going to be installed, please read Section 4.1.3.

As we were using D-Link DWA-123 when the system featured a Wi-Fi client, it was necessary to download a driver that was not available in the Raspbian repositories [105]. If using another adapter, chances are that the installed drivers are fully compatible. In that case it is not required to install new drivers.

This can be done using the following command set. Each command is here written on a new line to improve readability, but to run multiple commands at once, write them on a single line and put “&&” in between.

```
sudo apt-get remove firmware-ralink -y
sudo cp /etc/apt/sources.list ./sources.list.tmp
sudo echo "deb http://http.debian.net/debian/ wheezy main contrib non-free" > cp
/etc/apt/sources.list
sudo apt-get update
sudo apt-get install firmware-ralink -y --force-yes
sudo mv ./sources.list.tmp /etc/apt/sources.list
sudo apt-get update
```

With the drivers installed, the Wi-Fi software is ready to be configured. This is done by editing “/etc/wpa_supplicant/wpa_supplicant.conf”.



```
GNU nano 2.2.6 File: /etc/wpa_supplicant/wpa_supplicant.conf
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
  ssid="SSID"
  scan_ssid=1
  psk="PASSWORD"
  proto=RSN
  key_mgmt=WPA-PSK
  pairwise=CCMP
  auth_alg=OPEN
}
```

Content of wpa_supplicant.conf

Edit the file so that it becomes identical to the Figure above, except for the **ssid=""** and **psk=""**. Insert the name of the network you wish to connect to, in the quotation marks following **ssid**, and insert the network password in the quotation marks after **psk**.

Finally, add the following three lines to the end of /etc/network/interfaces [106].

```
allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
```

When this is done reboot the system and it should connect automatically to the configured wireless network.