



NTNU – Trondheim
Norwegian University of
Science and Technology

Internet Control for Residential Users

Torgeir Pedersen Cook

Master of Science in Communication Technology

Submission date: June 2013

Supervisor: Poul Einar Heegaard, ITEM

Co-supervisor: Bjørn J. Villa, ITEM

Norwegian University of Science and Technology
Department of Telematics

Title: Internet Control for Residential Users

Student: Torgeir Pedersen Cook

Problem description:

Some security issues exist in Wifi hot-spot systems found in airports and cafés. These systems often employ unencrypted Wifi access, which gives that opportunity to mount a variety of different attacks. Also, it is easy to obtain unauthorized access to these systems by spoofing the MAC- and IP address of authorized clients. The first part of this project will provide an analysis of the security issues found in Wifi hot-spots and present some proposed solutions to these issues.

In the second part, a prototype Wifi hot-spot system for residential environments will be implemented. The prototype will use the appropriate hardware and should be managed from a central server. Similar residential Wifi hot-spot systems have shortcomings when it comes to how end-users manage the system. This increases users reluctance to learn and use the system. A detailed specification of the system will be provided, including functional- and quality requirements. High emphasis will be placed on usability, responsiveness and interoperability. The identified requirements will impact the technology and design decisions made in the implementation. The system will include:

- Appropriate hardware providing Wifi access and basic IP traffic filtering functionality
- Central server for management and storage of end-user information
- Mobile-application intended for end-users to manage and control clients' Internet access
- Web-application intended for end-users to manage and control clients' Internet access. The web-application will also be used by requesting clients to obtain Internet access

Testing of the most important functional requirements will be conducted to ensure that these requirements are fulfilled.

Responsible professor: Poul Heegaard, ITEM

Supervisor: Bjørn J. Villa, ITEM

Abstract

The Internet has become instrumental in modern society. Many Internet based services are a necessary part of people's lives. Other services are intended solely for entertainment purposes. For certain groups there exists a need to control access to these services. This project will implement a prototype Internet control system for the residential environment. The implementation aims to amend existing Internet control solutions by improving user interaction. The implemented system bears resemblance to Wifi hot-spots found in coffee shops and book stores. This project will explore security vulnerabilities present in these Wifi hot-spot systems.

Contents

List of Figures	vii
List of Tables	ix
List of Code Snippets	xi
List of Acronyms	xv
1 Introduction	1
1.1 Motivation	1
1.2 Related Work	1
1.3 Scope	2
1.4 Rapport Structure	3
1.5 Method	3
2 Wifi Hot-spot Security	5
2.1 Wifi Access Domains	5
2.1.1 Enterprise Access	5
2.1.2 Public Access	6
2.1.3 Residential Access	6
2.2 Security Mechanisms	6
2.2.1 Authentication	6
2.2.2 Authorization	7
2.2.3 Accounting	8
2.3 Common Security Threats	8
2.3.1 Eavesdropping	8
2.3.2 MAC Address Spoofing	9
2.3.3 Freeloading	9
2.3.4 Session Hi-jacking	10
2.3.5 Rouge Access Point	11
2.4 Discussion	11
3 System Requirements	13

3.1	Overview	13
3.2	Functional Requirements	14
3.3	Quality Requirements	16
3.3.1	Usability	16
3.3.2	Responsiveness	16
3.3.3	Interoperability	17
3.3.4	Modifiability	17
4	Development Platform	19
4.1	Residential Access Point	19
4.1.1	Operating System	20
4.1.2	Hostapd	20
4.1.3	Dnsmasq	20
4.1.4	Iptables	20
4.1.5	Application Language	23
4.2	Management Server	23
4.2.1	Server Language	23
4.2.2	Client Permission Format	24
4.2.3	Front-end Framework	24
4.3	Mobile Management Application	24
4.3.1	Platform	24
4.3.2	Push Notification	24
4.4	Integrated Development Environment	25
4.5	Revision Control	25
5	Implementation	27
5.1	Residential Access Point	27
5.1.1	Wifi Access	27
5.1.2	DHCP Server	28
5.1.3	Authorization	28
5.1.4	Obtaining Client Identifier	29
5.1.5	Polling Server	31
5.2	Management Server	33
5.2.1	Storage	33
5.2.2	Structure	35
5.2.3	Requesting Access	36
5.2.4	Remote Management	36
5.3	Mobile Management Application	37
5.3.1	Push Notification	37
5.3.2	Manager Login	41
5.3.3	Manage Internet Permissions	41
5.4	Web Management Application	44

5.5	Functionality Overview	46
6	System Testing	47
6.1	Allow and Block Internet Access	47
6.2	Multiple Customers	48
7	Discussion and Further Work	51
7.1	Authentication	51
7.2	Authorization	51
7.3	Accounting	51
7.4	System Feedback	52
	7.4.1 Requesting Access	52
	7.4.2 Visible Permissions	53
7.5	Uniform Management Interface	53
7.6	Validating Permissions	53
7.7	Double Network Address Translation	54
7.8	Modes of Notification	54
7.9	Customer Management	55
8	Conclusion	57
	References	59
	Appendices	
A	Raspberry Pi	63
A.1	Configuration files	63
	A.1.1 hostapd.conf	63
	A.1.2 dnsmasq.conf	63
	A.1.3 interfaces	64
A.2	Bash Scripts	64
	A.2.1 add_static_lease	64
	A.2.2 reload_dhcp_leases	65
	A.2.3 iptables_setup	66
A.3	Python Modules	67
	A.3.1 Polling Server	67
	A.3.2 Proxy Server	70
B	Management Server	75
C	Mobile Application	77
D	Development Aid	79
D.1	Raspberry Pi	79

D.1.1	Commands	79
D.1.2	Directories	80
D.2	Management Server	80
D.2.1	Apache Tomcat	81
D.2.2	MySQL	81
D.3	Mobile Application	81
D.4	Software Tools	82
D.4.1	Eclipse	82
D.4.2	Git	82

List of Figures

2.1	OSI Reference Model	5
2.2	Captive Portal	7
2.3	Freeloading Attack	9
2.4	Session Hi-jacking Attack	10
3.1	System Architecture	14
4.1	Raspberry Pi Type B Specifications	19
4.2	Iptables Table-Chain-Rule Structure	21
4.3	Iptables Overview	21
4.4	Custom Client Chain	23
5.1	Authorized and Unauthorized Sub-net	29
5.2	User and Client Table	34
5.3	Request Access Web Page	36
5.4	Get Clients	37
5.5	GCM Registration Procedure	38
5.6	Access Request Notification	39
5.7	Push Notification	40
5.8	Select Client Internet Permission	40
5.9	Manager Login	41
5.10	Get Clients	42
5.11	Client List	44
5.12	Manager Web Page	45
5.13	Functionality Overview	46
7.1	Access Request Feedback	52
7.2	Client Internet Permissions	53
D.1	Install Android Application	82

List of Tables

3.1	Functional Requirements	15
6.1	Allow and Block Access Test Results	48
6.2	Multiple Customers Test Results	49

List of Code Snippets

4.1	Iptables Command: Accept Rule	22
4.2	Iptables Command: FORWARD Chain Policy Rule	22
5.1	Iptables Command: NAT Rule	27
5.2	Dnsmasq Initial Configuration	28
5.3	DHCP Lease	28
5.4	Linux Command: /etc/network/server/proxy.py	30
5.5	Iptables Command: HTTP Port Redirect Rule	30
5.6	Python Method: /etc/network/server/util.py	30
5.7	Python Method: /etc/network/server/util.py	31
5.8	JavaScript Object Notation: Client	32
5.9	MySQL Command: /etc/init.d/mysql	33
5.10	Management Server Structure	35
5.11	JavaScript Object Notation: Client Array	43
7.1	Pseudocode: Access Request Notification Procedure	55
A.1	Configuration File: /etc/hostapd/hostapd.conf	63
A.2	Configuration File: /etc/dnsmasq.conf	63
A.3	Configuration File: /etc/network/interfaces	64
A.4	Bash Script: /etc/add_static_lease.sh	64
A.5	Bash Script: /etc/reload_dhcp_leases.sh	65
A.6	Bash Script: /etc/network/if-up.d/iptables_setup.sh	66
A.7	Python Module: /etc/network/server/configserver.py	67
A.8	Python Module: /etc/network/server/transproxy.py	70

List of Acronyms

- ADT** Android Development Tools
- AJAX** Asynchronous JavaScript and XML
- API** Application Programming Interface
- CA** Certificate Authority
- CSS** Cascading Style Sheets
- DHCP** Dynamic Host Configuration Protocol
- DNS** Domain Name System
- DTD** Data Retention Directive
- EAP** Extensible Authentication Protocol
- EU** European Union
- GB** Gigabyte
- GCM** Google Cloud Messaging
- GUI** Graphical User Interface
- HTML** HyperText Markup Language
- HTTP** Hypertext Transfer Protocol
- IDE** Integrated Development Environment
- IEEE** Institute of Electrical and Electronics Engineers
- iOS** iPhone OS
- IP** Internet Protocol

ISO International Organization for Standardization

ISP Internet Service Provider

JRE Java Runtime Environment

JSON Java Script Object Notation

LTS Long Term Support

MAC Media Access Control

Mbps Megabit per second

NAT Network Address Translation

NIC Network Interface Controller

NTNU Norwegian University of Science and Technology

OS Operating System

OSI Open Systems Interconnection

PEAP Protected Extensible Authentication Protocol

PHP Hypertext Preprocessor

RPI Raspberry Pi

SDHC Secure Digital High Capacity

SIM Subscriber Identity Module

SMS Short Message Service

SQL Structured Query Language

SSH Secure Shell

SSID Service Set Identifier

SSL Secure Sockets Layer

STUN Session Traversal Utilities for NAT

TCP Transmission Control Protocol

TTLS Tunnelled Transport Layer Security

TURN Traversal Using Relays around NAT

URL Uniform Resource Locator

USB Universal Serial Bus

VoIP Voice over Internet Protocol

WEP Wired Equivalent Privacy

WPA WiFi Protected Access

XML Extensible Markup Language

Chapter 1

Introduction

1.1 Motivation

Internet based services have become an important part of people's lives. In the residential environment there exists a need to control Internet access for certain groups. Reports exist of children being addicted to Internet services such as *World of Warcraft* and *Facebook*.

Many different Internet control systems for the residential environment are available on the market today. A common weakness of existing systems is how users interact with the system. Extensive configuration is often required by the user to enable the offered Internet control features. Also, existing Internet control systems give little consideration to how clients request Internet access.

1.2 Related Work

Existing Internet Control solutions for the residential environment fall into three categories.

- Software centric
- Router centric
- Dedicated hardware

Software centric solutions is installed on client devices to provide Internet control. Solutions in this category offer a wide range of Internet control features. The software needs to be installed on each device requiring Internet control. This is impractical in environments containing a variety of devices such as laptops and smartphones. Different software needs to be configured on each device to provide complete Internet control.

The second category is router centric solutions. Some major router manufacturers have Internet control functionality implemented in their routers. This centralizes the Internet control on the residential network since all Internet traffic passes through the router. Internet control is one of many services offered by routers. Extensive configuration is often required by the user to enable the offered Internet control features. Also, no consideration is given to how Internet access is requested and granted.

The third category is Internet control realized on dedicated hardware. A prototype system in this category was implemented in a prior project at Norwegian University of Science and Technology (NTNU) [1]. Systems in this category are connected to home-routers. More consideration is given to how users interact with these systems than in the two other categories. The Internet control feature provided by these systems usually demand a substantial annual fee.

1.3 Scope

The first part of this project will analyze the security provided in different Wifi access domains. The analysis will uncover security threats found in the public- and residential access domain. Some countermeasures to the identified threats are presented. The second part of the project will implement an Internet control prototype system for the residential environment. The implementation will not provide countermeasures to the threats uncovered in the first part of the project.

The Internet control system implemented aims to prototype a managed service where customers can either purchase or subscribe to the service. To support multiple customers, a central server will be utilized for storage of customer information. The Internet control features of the system will be implemented on dedicated hardware that is connected to the home-network of the customer. The dedicated hardware can be managed from the central server. Clients can request Internet access through the Wifi network offered by the dedicated hardware. The system customer can manage clients' Internet permissions through a mobile- and web application. This project aims to improve how Internet access is requested and granted on existing Internet control systems.

Having customer management is necessary for the service provided by the planned system. No customer management functionality will, however, be implemented. The main focus of this project is to create a system suited for demonstration purposes. The project also aims to lie the groundwork for further development. Initially, the project intended to implement a mobile- and web application that managed clients' Internet permissions, but due to time limitations only a mock-up implementation of the web application was created. The mobile application was given priority as it is

better suited to demonstrate the functionality of the system.

1.4 Rapport Structure

In the Wifi Hot-spot Security Chapter the main Wifi access domains will be presented. Some common security threats found in the public- and residential domain are analyzed. The analysis includes some defenses against the identified threats. The System Requirements Chapter gives an overview of the planned prototype system and specifies the functional- and quality requirements. The Development Platform Chapter provides a description of the hardware- and software components needed to meet the requirements specified in the System Requirements Chapter. The Implementation Chapter gives a description of the implementation work performed in the project. The most important functionality of the systems is tested in the System Testing Chapter. The Discussion and Further Work Chapter presents some challenges and limitations of the system. This chapter also suggests issues and features that should be considered in further development.

1.5 Method

An overview of the system domain was established through the work performed in a prior project [1]. This project aimed to improve on poor design and implementation choices in the prior project. The functional- and quality requirements were specified subsequent to the system implementation. The requirements were used to plan and direct development efforts.

A deep understanding of *Python*, *Hypertext Preprocessor (PHP)* and *Java Script Object Notation (JSON)* was obtained to meet the specified requirements. Working with these software components yielded valuable insight into challenges and limitations faced in a distributed system. Learning to work with these components in a distributed environment was challenging. Much time was spent on debugging and tracing the origin of system bugs.

Chapter 2

Wifi Hot-spot Security

2.1 Wifi Access Domains

2.1.1 Enterprise Access

Enterprise access is used in access networks that require high security, such as networks found in companies and government offices. Wifi access is encrypted at the data link layer in the Open Systems Interconnection (OSI) reference model [2].

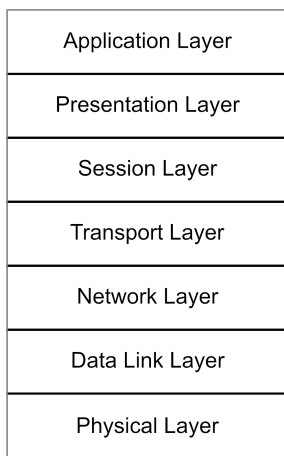


Figure 2.1: OSI Reference Model

These systems often utilize the WiFi Protected Access (WPA)-Enterprise standard [3] to provide encrypted access. Since companies can have thousands of employees, enterprise systems require a reliable and scalable scheme to distribute the authentication information. Enterprise systems offer strong security, but configuration is often required of client equipment to obtain access. The institutions utilizing enterprise

access often have technical support departments that aid clients in configuration of their equipment.

2.1.2 Public Access

Public hot-spots are used to grant clients access in public places such as coffee shops and book stores. The primary focus of these systems is to provide access free of configuration. For this reason most public hot-spots employ open networks, i.e the access is not encrypted. Some public hot-spots do however provide encrypted access. The encryption key can often be obtained at the counter of the coffee shop or book store. This does not provide stronger security than the open network approach since anyone can obtain the encryption key.

2.1.3 Residential Access

Resident access is used in the home environment. The access offered often uses encryption standards such as Wired Equivalent Privacy (WEP) [3] and WPA-Personal [3]. Clients can obtain access by providing the encryption key. For the purpose of this project a residential hot-spot is a Wifi access point that provides Internet control features.

2.2 Security Mechanisms

2.2.1 Authentication

Authentication is the process of identifying an individual or a systems component. In enterprise systems authentication is provided through the Extensible Authentication Protocol (EAP) [4]. EAP provides multiple modes of authentication. The modes most commonly used are Protected Extensible Authentication Protocol (PEAP), EAP-Subscriber Identity Module (SIM) and EAP-Tunneled Transport Layer Security (TTLS). These modes provide mutual authentication, i.e the clients are authenticated to the network and the network is authenticated to clients. Clients authenticate themselves to the network by providing the authentication information provided by their organization or company.

In public hot-spots *captive portals* are widely used to provide client authentication. Captive portals utilize a special gateway between the access point and the rest of the network.

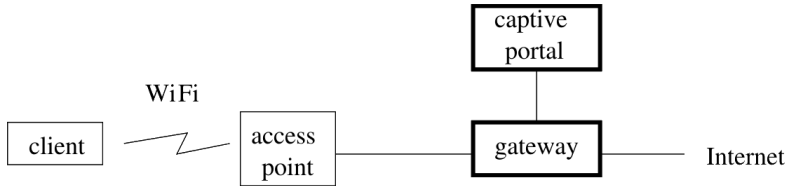


Figure 2.2: Captive Portal

Source [5]

The gateway redirects all web requests originating from unauthenticated clients to a Secure Sockets Layer (SSL) web page located on the *captive portal*. Clients usually authenticate themselves by providing payment details.

In the residential environment a client is authenticated by providing the encryption key to the hot-spot. This provides weaker authentication than in enterprise- and public hot-spots, since any client with knowledge of the encryption key can authenticate them self.

2.2.2 Authorization

Authorization is granting an individual or a system component permission to perform a particular activity. In Wifi hot-spots authorization is provided by employing traffic filtering and bandwidth management. Some authorization services are listed below.

- Allow and Block Internet access
- Time based Internet access
- Allow and Block Internet services
- Transferred content limit
- Allocate bandwidth

These services are enforced by filtering traffic based on Media Access Control (MAC)- and Internet Protocol (IP) address. In enterprise systems, traffic is encrypted using client specific encryption keys. Bypassing the authorization services of these systems would require breaking the WPA-Enterprise encryption, which is considered infeasible [6].

In public hot-spots, the authentication procedure implemented is considered secure since the procedure is encrypted using application layer protocols such as SSL. However, the authorization services provided by these hot-spots can often be bypassed with minimal effort. This will be discussed in Section Common Security Threats.

The access in residential systems is often encrypted at the data link layer. Clients can act in collusion to bypass the systems authorization services. This will also be discussed in Section Common Security Threats.

2.2.3 Accounting

Accounting is to track an individual's or system component's consumption of resources. In Wifi hot-spots accounting can be provided by recording:

- Which web domains are accessed
- Which web services are used
- How much data is transferred

Accounting services are provided by coupling the recorded data with the MAC- and IP address of clients. The system can utilize the recorded data to track which and how much system resources clients use.

In accordance with the establishment of the European Union (EU) Data Retention Directive (DTD) in 2006, hot-spot owners may be required to provide their national government with client Internet traffic data. Implementing reliable and secure accounting services thus becomes important in a juridical sense, since hot-spot owners may be held responsible for illegal activities on their network.

2.3 Common Security Threats

Public and residential hot-spots aim to provide clients Wifi access free of configuration. Less consideration is given to security than in enterprise systems. Some security threats found in public and residential hot-spots are presented in the preceding sections.

2.3.1 Eavesdropping

The Wifi access offered by public and residential hot-spots is often left unencrypted or the encryption key is available to the other clients on the network. An attacker

may eavesdrop on Wifi traffic by using software such as *Wireshark* [7]. The attacker can recover passwords and other content of web services the victim is using.

2.3.2 MAC Address Spoofing

Public and residential hot-spots provide authorization services based on clients' MAC address. By spoofing the MAC address, an attacker can gain the same permissions in the system as an authorized client. The MAC address of authorized clients can easily be obtained by eavesdropping on the Wifi traffic. Several Wifi drivers in Windows support spoofing of the MAC address. Other tools for MAC address spoofing are available on the Internet, e.g *smac*¹.

2.3.3 Freeloading

A problem with the MAC spoofing is that public and residential hot-spots often provide authentication services based on the IP address, in addition to the MAC address. An attacker can easily circumvent this problem by assigning an unauthorized device, the same IP address as an authorized device. This will result in that the attacker also receiving traffic intended for the victim (since the attacker and victim have the same IP address). However, if both the attacker and victim run personal firewalls, a freeloading attack will perform reliably. Personal firewalls typically only allow packets that are part of a session. This results in no traffic interference between the attacker and victim. This attack can be performed by a single attacker or in collusion. In a collusion attack clients cooperate to obtain unauthorized access.

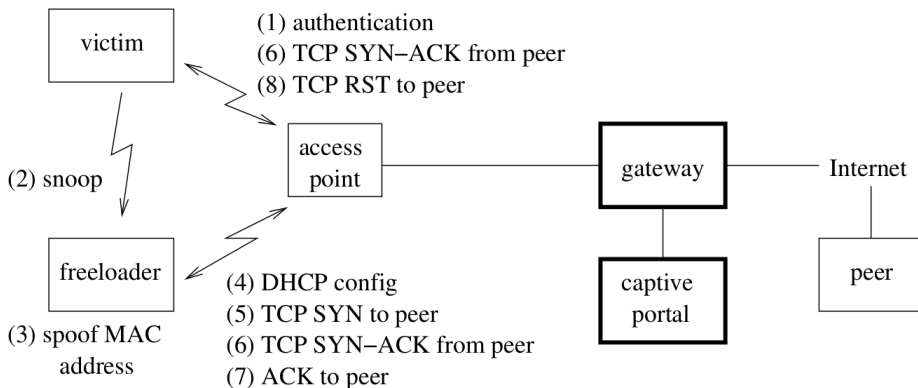


Figure 2.3: Freeloading Attack

Source [5]

¹<http://www.klccconsulting.net/smac>

A freeloading attack can be prevented by detecting when a MAC address is spoofed. The appropriate actions can be initiated to deny the attacker access. A novel MAC spoofing detection scheme is presented in [5]. The proposed solution traces the sequence number of data link layer frames. Data link layer frames contain a 12 bit sequence number that is incremented for each network layer datagram sent. This number is set and verified by the Network Interface Controller (NIC) hardware, and can typically not be altered by software. By recording trend-lines for each client, the solution can determine if abnormal changes in the MAC sequence number is the result of a freeloading attack. This MAC spoofing detection solution is performed in the hot-spot and is completely transparent to clients.

2.3.4 Session Hi-jacking

A session high-jacking attack can be mounted by eavesdropping on the traffic of a victim authenticating with the hot-spot. The attacker can factor de-authentication frames so that they are seemingly coming from the hot-spot. These frames can be periodically sent to prevent the victim from gaining access. By spoofing the victim's MAC- and IP address the attacker can obtain unauthorized access.

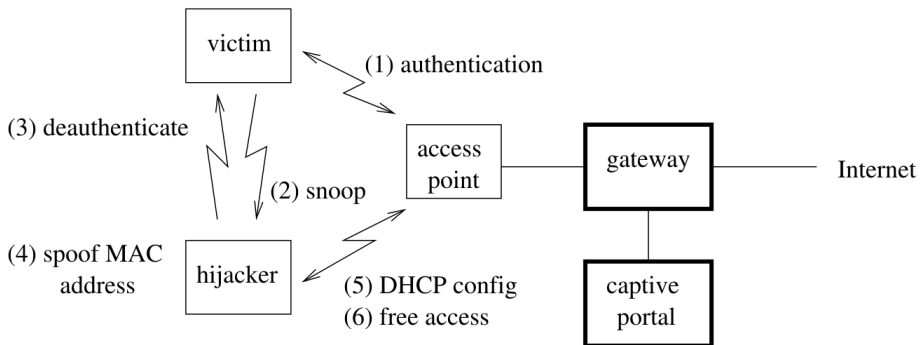


Figure 2.4: Session Hi-jacking Attack

Source [5]

This attack cannot be detected by the MAC spoofing detection solution described in Section Freeloading, since there has not been enough traffic to create reliable trend-lines for the attacker and victim. A defense is presented in [5]. This solution presents clients with a SSL session management web page. The web page has an associated cookie containing a session id, i.e a cryptographic random number, and is configured to refresh automatically. A session high-jacking attack is detected when

no refresh requests containing the session id are made within a certain time interval. This solution requires that the session management web page remains open.

2.3.5 Rouge Access Point

In public and residential hot-spots clients authenticate themselves to the hot-spot, but the hot-spot does not authenticate itself to clients. An attacker can configure a rouge hot-spot with the same Service Set Identifier (SSID) as a legitimate hot-spot with a stronger signal. Most client devices will automatically change to the hot-spot providing the stronger signal. This attack is difficult for clients to detect since most operating systems do not provide any information regarding activity at the data link layer. Once associated with the rouge hot-spot, the attacker has complete control over the victim's traffic.

2.4 Discussion

Enterprise hot-spot systems provide strong authentication, authorization and accounting services. Configuration is often required of the client devices to gain access. Enterprise systems also require a scheme to distribute authentication information. In public- and residential hot-spots clients can gain access with minimal effort. These systems are however vulnerable to the attacks presented in Section Common Security Threats. Bypassing the authorization services will result in a revenue loss for the public hot-spot owner. In the residential environment a likely attack scenario is that clients act in collusion to obtain unauthorized access. A successful attack will defeat the purpose of the system, as residential hot-spots aim to control clients' Internet access permissions. A defense against the identified attacks should be considered in development of public and residential hot-spot systems. The defense mechanisms should be transparent to clients since ease of configuration and use is key to successful hot-spot deployment.

Chapter 3

System Requirements

3.1 Overview

This project will implement a prototype Internet control system for the residential environment. The implemented system aims to provide a managed service. A central *management server* is thus required to store customer information such as name and email address. A *management server* is also required to provide remote management of the *residential access point* located on the home network. the Internet control features of the system are realized on the *residential access point*. The *residential access point* will offer clients Wifi access and provide IP packet filtering functionality. Customers of the system can grant clients Internet permissions through a *mobile-and web management application*. Clients can request Internet access through a web page located on the management server.

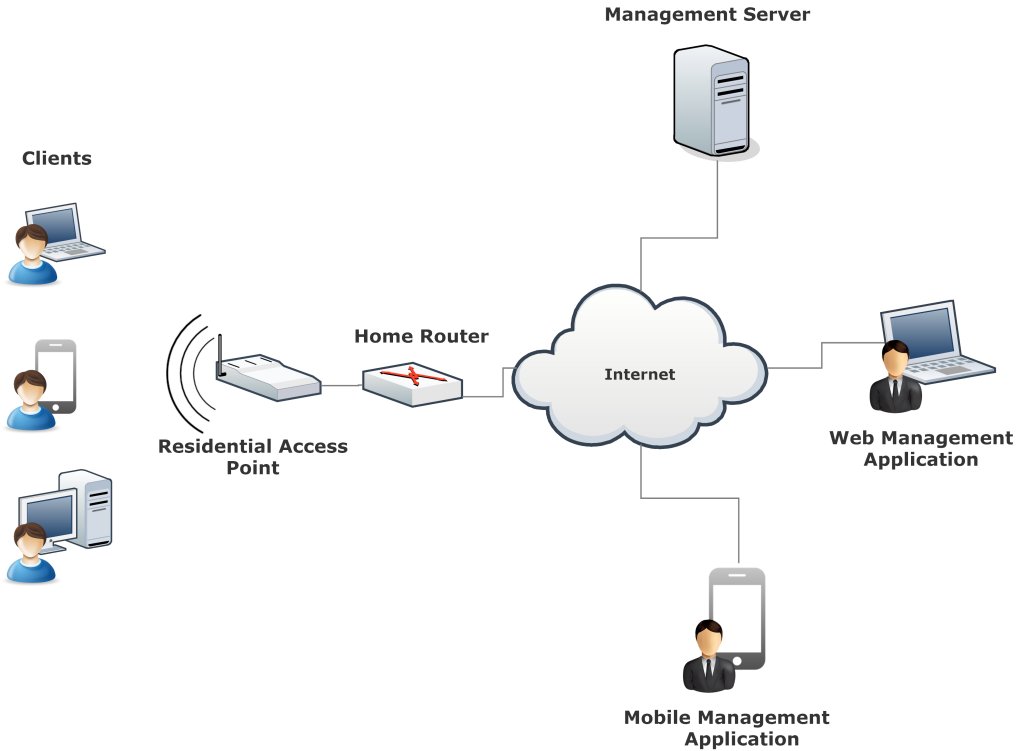


Figure 3.1: System Architecture

In the following sections the functional- and quality requirements of the system will be presented. The identified requirements will impact the choices of software- and hardware components made in Chapter Development Platform.

3.2 Functional Requirements

The focus of this project was to implement a system suited for demonstration purposes. The functional requirements will thus prioritize the core aspects of the system, i.e to allow and block clients' Internet access. As the system aims to prototype a managed service, priority will be given to enable support for multiple customers. Less emphasis will be placed on providing a rich set of Internet control features. The Functional requirements for the *residential access point*, *management server*, *mobile management application* and *web management application* are listed below. The requirements are listed in order of importance.

#	Title	Functional Requirement	Priority
1	Wifi access	The residential access point can provide Wifi access to clients	High
2	Unauthorized Permissions	The residential access point can block all traffic from unauthorized clients. Only traffic required to request Internet access is allowed	High
3	Allow Internet Access	The residential access point can allow clients Internet access based on MAC- and IP address	High
4	Block Internet Access	The residential access point can block clients Internet access based on MAC- and IP address	High
5	Time Based Internet Access	The residential access point can allow or block clients Internet access based on time of day	Low
7	Domain Black List	The residential access point can allow or block clients Internet traffic to custom web domains	Low
8	Accounting	The residential access point can gather IP traffic statistics from clients	Low
9	Customer Information	The management server can store customer information	High
10	Remote Management	The residential access point can be managed remotely from the management sever	High
11	Multiple customers	The management server can support managing of residential access points belonging to different customers	High
12	Allow Internet Access	The mobile application can allow clients Internet access	High
13	Block Internet Access	The mobile application can block clients Internet access	High
14	Access Request Notification	The mobile application can receive push notifications when clients request Internet access	High
15	Time Based Internet Access	The mobile application can allow or block Internet access based on time of day	Low
16	Domain Black List	The mobile application can allow or block Internet traffic to custom web domains	Low
17	Allow Internet Access	The web application can allow clients Internet access	High
18	Block Internet Access	The web application can block clients Internet access	High
19	Request Access	Internet access can be requested by through the web application	High
20	Time Based Internet Access	The web application can allow or block Internet access based on time of day	Low
21	Domain Black List	The web application can allow or block Internet traffic to custom web domains	Low
22	Accounting	The web application can display clients IP traffic statistics	Low

Table 3.1: Functional Requirements

3.3 Quality Requirements

As the system implemented in this project is a prototype, emphasis will be placed on user interaction with the system. The quality attributes addressed will be usability, responsiveness and interoperability. Since the system is likely to be the subject of further work, emphasis will also be placed on the modifiability of the system.

The following sections will present the importance of each quality requirement and provide quality scenarios for each requirement.

3.3.1 Usability

International Organization for Standardization (ISO) 9241-11, Guidance on Usability [8] states that:

Usability is the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

Usability is key to encourage users to learn and use this system.

No Access Point Configuration When the residential access point is initially connected to the home network of the customer, no configuration should be required to provide Wifi access.

Managing Clients Internet Access Managing clients' Internet access permissions through the mobile- and web application should not require a tutorial.

Requesting Internet Access Requesting Internet access should not require a tutorial.

3.3.2 Responsiveness

Research indicates that responsiveness is a fundamental issue for positive human-computer interaction. Long response times induces reactions such as annoyance, stress and decreased productivity.

Internet Permission Update Time When a client's Internet permissions are updated by the manager, through the mobile- or web application, the affect of the changes should take less than eight seconds.

Responsive User Interface The user interface of the mobile- and web application should have response times of less than 200 milliseconds.

3.3.3 Interoperability

Interoperability in the context of this system is supporting a wide range of client devices. Utilizing web technologies not supported by client devices will prohibit the clients from requesting Internet access. Not supporting the most common Wifi access standards would remove the opportunity for some devices to obtain Wifi access. The system should support client devices ranging from desktop computers to smartphones. The system aims to support only devices containing a web browser. An increasing number of devices are gaining Wifi access, e.g smart-grid appliances such as fridges. Arguably, these devices do not require Internet control features.

Web Application Support The web application should support all client equipment containing a web browser.

Wifi Support The local hardware should support client hardware implementing the following IEEE 802.11 Wifi access standards: *a*, *b*, *g* [3][9][10].

3.3.4 Modifiability

To simplify further development of the system, consideration should be given to code quality and structure of software components. A detailed guide of the software components used should also be provided to aid further development.

Code Quality This project will involve multiple programming languages. Code written in high-level languages should have self-explanatory variable-, method- and class names. Code written in low-level languages should contain comments where the functionality of the code is unclear.

Software Component Structure As the systems will contain multiple software components, each component should have a consistent and logical structure.

Adding Client Permissions The functional requirements only specify a limited set of Internet access permissions. The system should have the capability to add new client Internet permissions without undergoing architectural changes.

Chapter 4

Development Platform

4.1 Residential Access Point

Raspberry Pi (RPI) is a single board computer the size of a credit card. The RPI was originally intended to promote computer science education. Two different models are currently available, type A and B. The main difference is that type A is not equipped with an Ethernet port. Because network access was required in this project, type B was chosen. A price of 25\$ to 35\$ has resulted in a rapidly growing development community for the RPI. Development projects range from using the RPI as a home media server to home automation and robotics¹.

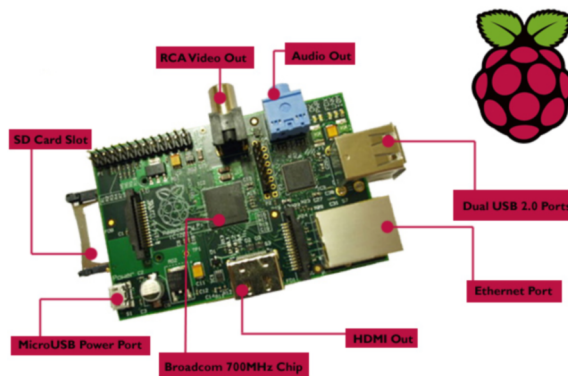


Figure 4.1: Raspberry Pi Type B Specifications

Source²

Since the RPI is not equipped with on-board Wifi, a Wifi Universal Serial Bus (USB) dongle was required. Several different dongles were tested. The RPI was

¹<http://www.raspberrypi.org/phpBB3/>

²<http://www.trustedreviews.com/opinions/raspberry-pi> page - 2

found compatible with *D-link DWL-G122*. For storage a *Samsung 16 Gigabyte (GB) Secure Digital High Capacity (SDHC) Class 10 Card* was used.

The RPI provides a good prototype platform since it resembles the resource-constrained hardware of similar residential Internet control systems.

4.1.1 Operating System

The RPI is capable of running a variety of OSs. The recommended beginner Operating System (OS) is *Rasbian wheezy* [11]. This is a Linux based OS with a large number of pre-installed packages.

4.1.2 Hostapd

To provide Wifi access the *hostapd* package [12] was used. Hostapd supports the Institute of Electrical and Electronics Engineers (IEEE) 802.11 access point management [3] and offers a wide range of configuration options for Wifi access control and authentication.

4.1.3 Dnsmasq

The Dynamic Host Configuration Protocol (DHCP) [13] is used to dynamically allocate IP addresses to clients on a network. In the system a DHCP server was required to grant IP addresses. *Dnsmasq* [14] provides Domain Name System (DNS) forwarder and DHCP server. Dnsmasq was easy to configure and offers rich functionality.

4.1.4 Iptables

To provide Internet control functionality *iptables* [15] was utilized. Iptables is the most used traffic filtering package available for Linux. Iptables can set up and maintain tables of IP packet filter rules in the Linux kernel. When a IP packet arrives at the RPI, it is sequentially checked against the rules in iptables. The main tables in iptables are *filter*, *nat* and *mangle*. Each table contains a number of chains and each chain has a set of rules.

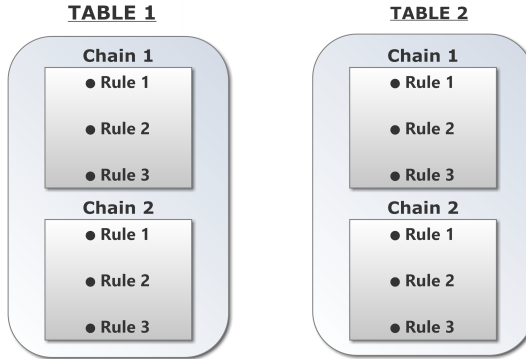


Figure 4.2: Iptables Table-Chain-Rule Structure

Adopted from³

The *filter* table contains three chains, i.e *INPUT*, *OUTPUT* and *FORWARD*. Packets destined for a local process visit the *INPUT* chain in the *filter* table. Locally generated traffic visits the *OUTPUT* chain. Packets not destined or generated by a local process visit the *FORWARD* chain. The *nat* table is consulted when new connections are encountered. The *mangle* table is utilized for specialized packet alteration.

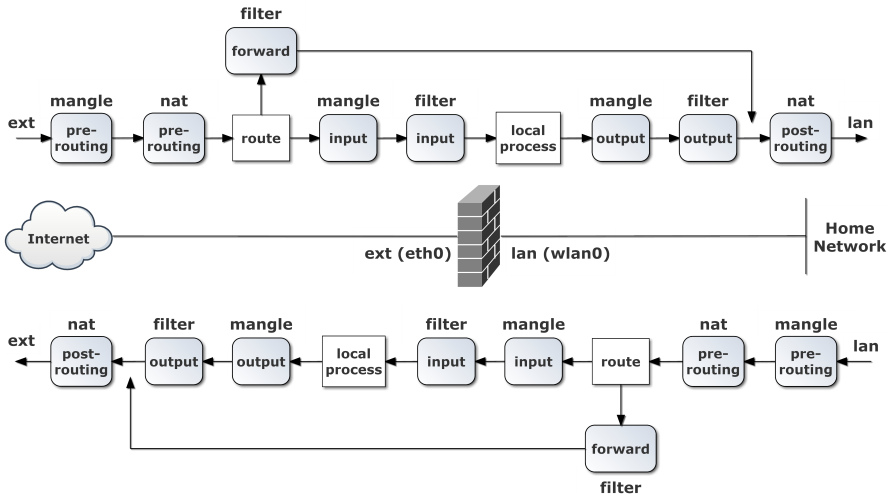


Figure 4.3: Iptables Overview

Adopted from⁴

³<http://www.thegeekstuff.com/2011/01/iptables-fundamentals/>

Iptables allows the creation of a wide range of filtering rules. A rule can contain a set of *matching criteria* and a *target*. The *matching criteria* specifies which packets the rule applies to. Examples of matching criteria is source- and destination IP address. The *target* specifies the destination of a packets once a *matching criteria* is fulfilled. Iptables has several pre-defined *targets* such as *ACCEPT* and *DROP*.

Code Snippet 4.1 Iptables Command: Accept Rule

```
iptables -t filter -A INPUT -s 10.0.0.6 -j ACCEPT
```

Rules are added to iptables by running commands in the command line. This rule is added to the *INPUT* chain in the *filter* table. The *matching criteria* is the *source* IP-address. The *target* is *ACCEPT*, i.e packets with source address *10.0.0.6* are allowed through iptables.

If none of the rules in a chain apply to a packet, the policy of the chain is followed.

Code Snippet 4.2 Iptables Command: FORWARD Chain Policy Rule

```
iptables -t filter -P FORWARD DROP
```

This rule sets the *policy* of the *FORWARD* in the *filter* table to *DROP*. A packet visiting the *FORWARD* chain is then dropped if none of the rules in the chain apply to the packet.

An additional feature provided in iptables is the creation of custom chains. This feature can be utilized to create client specific chains. Each client allowed Internet access can have a custom chain in iptables where the client's Internet access permissions are added. Traffic can be directed towards the custom chains based on a client specific criteria such as MAC- or IP address.

⁴<http://www.linuxnetmag.com/en/issue9/m9iptables1.html>

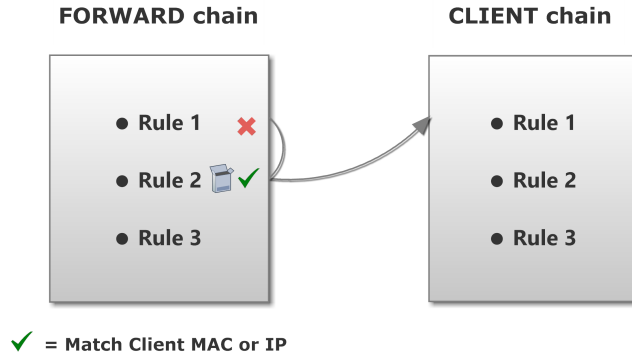


Figure 4.4: Custom Client Chain

Utilizing custom chains will yield better performance, as less rules will have to be consulted than in a purely sequential approach.

4.1.5 Application Language

A programming language was required to enable communication between the RPI and management server. Python is a free, general purpose programming language that offers a comprehensive standard library. Python's lightweight and elegant syntax makes it an ideal language for scripting and prototype development. Python is also the native application language for the RPI, which makes it a good candidate for implementing the specified functionality.

4.2 Management Server

Due to previous experience *MySQL 5.1.66* [16] database server was used for storage. An *Ubuntu 10.04.4 Long Term Support (LTS)* [17] virtual server was provided at the start of the project. To provide an interface for clients requesting access, an *Apache Tomcat 6* [18] web server was installed on the management server. Apache Tomcat is an open source web server that supports multiple server languages.

4.2.1 Server Language

A language was required on the management server to provide the communication with the RPI. The language should also support communication with the mobile- and web application. PHP is a free and widely used server language designed for web development. PHP contains native extensions for handling Hypertext Transfer

Protocol (HTTP) traffic. Database interaction is made simple through the native extensions included in PHP. *PHP 5.3.2* [19] was installed on the management server.

4.2.2 Client Permission Format

A format for representing client Internet permissions was required. The format should be complex enough to represent a wide range of Internet permissions. JSON is a format used for storing and exchanging text information. JSON is derived from the JavaScript scripting language, where it is used for representing data structures. Despite the name, JSON is language independent, i.e parsers exist in numerous programming languages. Extensible Markup Language (XML) was initially considered as it provides more structure than JSON. XML also provides the capability of validation through XML schemas. XML schemas describe the valid structure of XML documents. A XML document can be validated towards its respective XML schema. This would provide the capability to validate clients Internet permissions. The complexity of XML was however a major obstacle. The XML parsers available in the relevant programming proved to be complex and difficult to work with. The capabilities of JSON was considered sufficient to represent clients Internet permissions.

4.2.3 Front-end Framework

Twitter Bootstrap [20] was utilized to simplify the front-end work of the web application. Twitter Bootstrap is a front-end web application framework built around HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript. Twitter Bootstrap provides a clean and uniform solution for the development of most web applications, while remaining more flexible than other front-end frameworks such as *Drupal*⁵.

4.3 Mobile Management Application

4.3.1 Platform

Android is a Linux-based operating system designed primarily for smartphones and tablets [21]. Android application code is written in Java. Previous experience with both Java and Android made Android an appealing platform to realize the mobile application.

4.3.2 Push Notification

Mobile network providers aim to protect their customers by blocking Internet sessions not initiated by their customers. Without this restriction, customer devices could

⁵<http://www.drupal.org/>

be flooded with unwanted Internet traffic. A requirement for this system was to notify the customer when a client requests Internet access. To bypass the restriction described above, Google Cloud Messaging (GCM) [22] was utilized. GCM is a free cloud service that allows push notifications from a server to Android devices.

4.4 Integrated Development Environment

Due to previous experience, *Eclipse Juno 4.2.2* was chosen as the Integrated Development Environment (IDE) for software development in this project. *Eclipse* supports plug-ins for numerous software components and programming languages. The *PyDev* plug-in was used for Python development. This plug-in provides Python auto-complete and debugging features. The *Web Tools 3 Platform* was used for web development on the management server. This platform provides an *Apache Tomcat* plug-in that creates a local Apache Tomcat Server. This simplified the development on the management server since the web application code could be deployed to the local Apache Tomcat server. Small changes to the web application code could thus be tested without deploying the web application to the remote Apache Tomcat server on the management server. *Android Development Tools (ADT)* was installed to provide and environment for Android development. A guide on how to set up and use Eclipse is provided in Appendix D.4.1.

4.5 Revision Control

Git [23] is a commonly used revision control technology. Application code written in this project was published on the public repositories hosted by *GitHub* [24]. Utilizing revision control eases the distribution of application code in further development.

Chapter 5 Implementation

5.1 Residential Access Point

5.1.1 Wifi Access

Hostapd was configured to provide Wifi access to clients via the Wifi interface (*wlan0*) on the RPI. The configuration file of *hostapd* is available in Appendix A.1.1. Network Address Translation (NAT) [25] was implemented on the RPI to grant clients an internal IP address on the network. NAT provides a mapping from the externally visible IP address to internal IP addresses. This provides some security as external hosts cannot initiate connections with the internal clients. The main purpose of NAT is to remedy the lack of available IPv4 addresses.

Code Snippet 5.1 Iptables Command: NAT Rule

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

This rule is added to the *POSTROUTING* chain in the *nat* table. The rule applies to packets leaving the *eth0* interface on the RPI. The *target* is *MASQUERADE*. The *MASQUERADE target* provides a mapping between the externally visible *eth0* interface and the internal IP addresses of clients

In the case of connecting the RPI to the router of a customer, providing NAT will simplify the set-up of the RPI as IP addresses are granted dynamically by *dnsmasq* to clients on the network. This setup will however result in double NAT, i.e NAT is performed on the RPI and home-router. This can cause complications for commonly used NAT traversal schemes such as Session Traversal Utilities for NAT (STUN) and Traversal Using Relays around NAT (TURN). This will be discussed in Section Double Network Address Translation.

5.1.2 DHCP Server

Minimal configuration was required to enable dnsmasq as a DHCP server.

Code Snippet 5.2 Dnsmasq Initial Configuration

```
interface=wlan0
dhcp-range=10.0.0.1,10.0.0.100,2h
```

The following lines were added to the the configuration file of dnsmasq to bind dnsmasq to the *wlan0* interface and specify the range of IP addresses offered. The configuration file of dnsmasq is available in Appendix A.1.2

When a client request Wifi access a DHCP lease is granted by dnsmasq.

Code Snippet 5.3 DHCP Lease

```
74:2f:68:37:d5:a2 10.0.0.83 TorgeirLaptop
```

DHCP lease line from the lease file of dnsmasq. The DHCP lease contains the *MAC*, *IP* and *name* of the client device

With hostapd and dnsmasq configured, the RPI functioned as a router. Hostapd provided Wifi access and dnsmasq granted clients IP addresses in the range specified in Code Snippet 5.2.

5.1.3 Authorization

To provide client authorization, the IP address range offered by dnsmasq was split into two ranges. The upper part of the range was dedicated to clients who have not been granted Internet access permissions (unauthorized clients). The lower part of the range is for clients who have been granted Internet access permissions (authorized clients).

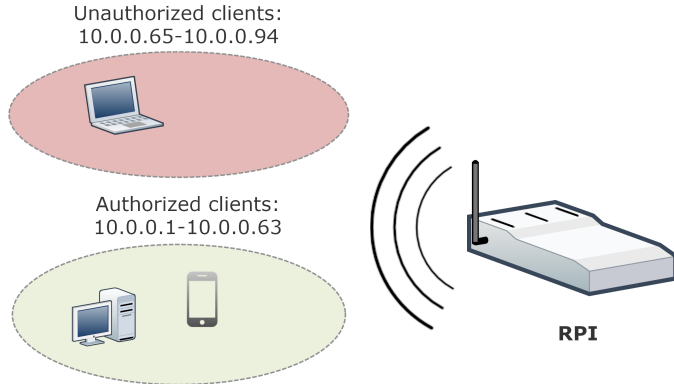


Figure 5.1: Authorized and Unauthorized Sub-net

A client requesting Wifi access is granted a lease in the unauthorized range. A client is granted Internet access by adding a static lease within the authorized range. A static lease is added by the bash script available in Appendix A.2.1.

5.1.4 Obtaining Client Identifier

For clients requesting Internet access, an identifier must be obtained by the management server to identify the client. The client's MAC address is a good identifier as it should be globally unique. In an implementation made prior to this project the MAC address was obtained by running a Java Applet in the web page where clients requested access. This proved to be a poor solution as Java Applets assume that the client device has a Java Runtime Environment (JRE) installed. Java Applet is also an outdated web technology.

The approach adopted in this project was to intercept unauthorized clients' HTTP traffic at the RPI. The RPI already has knowledge of clients' MAC address through the DHCP lease file of dnsmasq, as illustrated in Code Snippet 5.3. The MAC address was appended to the HTTP payload and propagated to the management server.

To intercept HTTP traffic, a Python HTTP proxy server was utilized. Multiple Python proxies were available at the public repositories hosted by GitHub. The proxy available at¹ was chosen due to its simplicity. This proxy was modified to realize the required functionality. The implemented Python proxy server is available in Appendix A.3.2.

¹<https://github.com/erijo/transparent-proxy>

Code Snippet 5.4 Linux Command: `/etc/network/server/proxy.py`

```
sudo python proxy.py
```

This command starts the Python proxy server. The proxy was run on port 8089

All client Internet traffic visited the *FORWARD* chain in iptables. HTTP traffic originating from clients in the unauthorized sub-net was directed towards the proxy server by adding the following rule to iptables.

Code Snippet 5.5 Iptables Command: HTTP Port Redirect Rule

```
iptables -t nat -A PREROUTING -s $IPunauth -p tcp --dport 80
-j REDIRECT --to-ports 8089
```

This rule is appended to the *PREROUTING* chain in the *nat* table. The rule specifies that *Transmission Control Protocol (TCP)* traffic with *destination port 80* (the default HTTP port number) is directed *to port 8089*, i.e the port of the proxy server

When HTTP requests arrive at the proxy server, the source IP address is read from the request packet, and the MAC address of the client is obtained from the DHCP lease file.

Code Snippet 5.6 Python Method: `/etc/network/server/util.py`

```
def getMacAddressFromIP(ip):
    dhcpleases = open("/var/lib/misc/dnsmasq.leases", "r")

    for line in dhcpleases:
        if (ip in line):
            s = line.split()
            mac = s[1]
            return mac
```

This method locates the line in the DHCP lease file of dnsmasq that contains the input parameter *ip*. The MAC address in the located line is returned

The RPI MAC address was also required at the management server to determine which RPI the access request originated from.

Code Snippet 5.7 Python Method: `/etc/network/server/util.py`

```
def getMacAddress(iframe):
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    info = fcntl.ioctl(s.fileno(), 0x8927, struct.pack('256s',
    iframe[:15]))

    return ''.join(['%02x:' % ord(char) for char in
    info[18:24]][:-1])
```

This method returns the MAC address of a network interface on the RPI. The MAC address of the `eth0` interface on the RPI was used as the RPI identifier.

The client- and RPI MAC address were appended to the payload of the HTTP traffic of unauthorized clients. The proxy server modified the HTTP request URL to point to the management server. This ensured that all HTTP requests made by unauthorized clients were directed to the *Request Access Web Page* on the management server. This web page is displayed in Figure 5.3.

5.1.5 Polling Server

In an implementation made prior to this project remote management was realized through establishment of a Secure Shell (SSH) tunnel from the RPI to the management server. When a client's Internet access permissions were changed, the management server utilized the SSH tunnel to apply the changes to the RPI. This would not scale well in a real implementation. Significant load would be placed on the management server as SSH tunnels would have to be maintained with potentially thousands of RPIs.

In this project remote management was provided by running a Python process on the RPI. The Python process polls the management server every five seconds for changes in client permissions. The Python process, `configserver.py`, is available in Appendix A.3.1. If changes have been made to any client permissions, the management server will respond to a polling request with a JSON string containing `client_info` and `permissions`.

Code Snippet 5.8 JavaScript Object Notation: Client

```
{ "client" : { "client_info" : { "mac" : "23:F3:45:34:44",
    "name" : "Torgeir"
  },
  "permissions" : { "access" : { "allow" : "true" } }
}
```

The management server responds polling requests from the Python process. The payload of the HTTP response is a JSON string. The JSON string contains the client's Internet *permissions* and the *client_info* required to convert the permissions to iptables rules

Upon the reception of a JSON string, the Python process initiates the following sequence:

1. Read the JSON string of the HTTP response
2. Convert the JSON string to a Python JSON object
3. Convert the JSON object to a client object. The client object is defined in the Python module available in Appendix A.3.1
4. Assign the client object an IP address in the authorized sub-net. As described in Section Authorization
5. Add a lease to the DHCP lease file containing the authorized IP address. A new DHCP is added using the bash script available in Appendix A.2.1.
6. Create a custom chain in iptables based on the client's MAC- and IP address
7. Add rules to the FORWARD chain in iptables to direct traffic from the authorized client to the custom chain just created. As illustrated in Figure 4.4
8. Add the permissions from the received JSON string to the custom client chain created in step 6
9. Finally, reload dnsmasq configurations by using the bash script available in Appendix A.2.2. The next time the client requests a DHCP lease, the leases added in step 5 is granted.

It was discovered late in the implementation that the dnsmasq does not provide a way to force DHCP lease renewal. The minimal lease time allowed in dnsmasq is 2 minutes. It may take up to 2 minutes before the DHCP lease added in *step 5* is granted to the client and the permissions added in *step 8* are enforced. This will be discussed in Section Authorization.

5.2 Management Server

The management server lies at the center of the system and requires functionality for communication with the RPI as well as the mobile- and web application.

In implementation made prior to this project, communication between the management server and RPI was provided utilizing *Java Servlets*. Using a compiled language like Java proved to be cumbersome as any changes to the management server required recompilation of the code. PHP is an interpreted language, i.e it avoids explicit compilation. PHP is also better attuned to web development than Java Servlets. PHP was thus chosen as the server language in this project. The functionality already implemented with Java Servlets was rewritten to PHP.

5.2.1 Storage

The MySQL server on the management server contains the database *WifiAccess*. The database has two tables, *user* and *client*. The *user* table contains information related to the system customers. Customers were added to the system by manually adding the customer information to the user table in the WifiAccess database.

Code Snippet 5.9 MySQL Command: `/etc/init.d/mysql`

```
INSERT INTO user (name, email, rpi_mac) VALUES
('Torgeir', 'torgeirpc@gmail.com', 'b8:27:eb:5e:c5:53')
```

This command is used to insert a customer's *name*, *email* and *rpi_mac* into the *user* table.

It is assumed the customer is the same individual that manages clients' Internet permissions. *Customer* and *manager* will be used interchangeably for the remainder of this report. The client table contains information about clients either requesting or granted Internet permissions. An overview of the two tables is displayed below.

User						
Field	Type	Null	Key	Default	Extra	
rpi_mac	char(40)	NO	PRI	NULL		
rpi_ip	char(40)	NO		NULL		
name	varchar(50)	NO		NULL		
email	varchar(50)	NO		NULL		
app_id	varchar(200)	YES		NULL		

Client						
Field	Type	Null	Key	Default	Extra	
mac	varchar(40)	NO	PRI			
name	varchar(50)	YES		NULL		
rpi_mac	varchar(40)	YES		NULL		
json_permission	longtext	YES		NULL		
update_flag	tinyint(1)	YES		0		

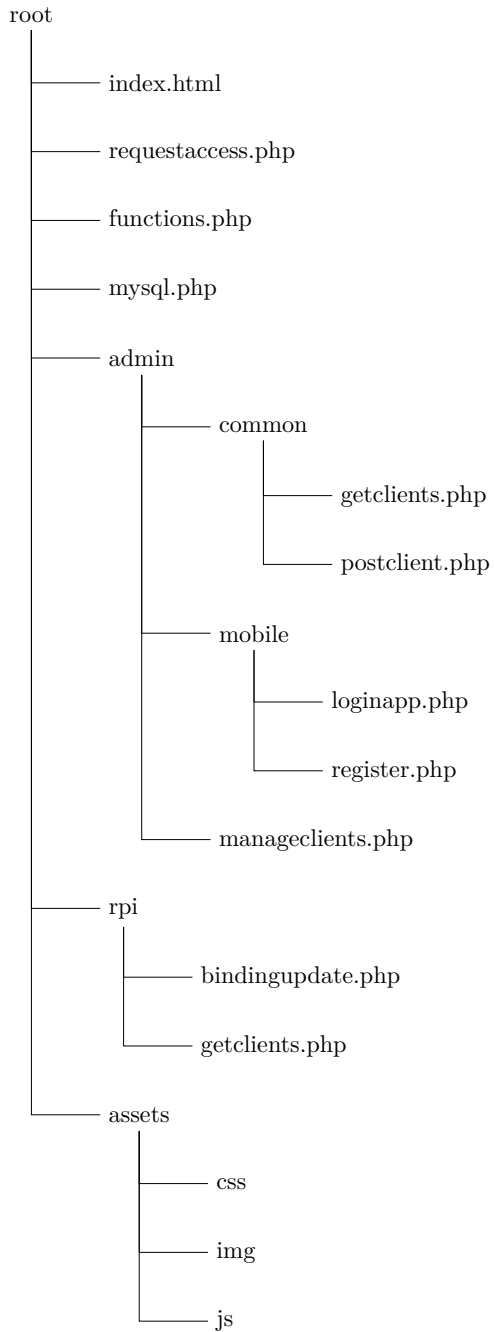
Figure 5.2: User and Client Table

The *rpi_mac* and *rpi_ip* fields in the *user* table, contain the MAC- and IP address of the RPI. The *name* and *email* fields are the name and email of the a customer. The *app_id* is granted by the GCM service and is necessary to allow the transmission of push notification to customer mobile devices.

The *mac* and *name* fields in the *client* table are the MAC address and name of clients requesting Internet access. The *rpi_mac* is the MAC address of the RPI from which an access request originated. This field is required to support the management of RPIs belonging to different customers. The *json_permission* field contains a JSON formatted string describing a clients Internet permissions. The *update_flag* field is used to minimize the traffic sent to the RPI. When the manager changes a client's permissions, this flag is set. The management server receives polling requests from the Python process described in Section Polling Server. Clients having the update flag are included in the response to the polling request. This ensures that client permissions already present on the RPI are not sent from the management server.

5.2.2 Structure

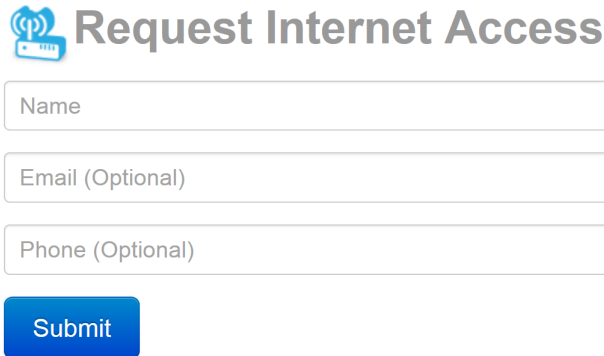
Code Snippet 5.10 Management Server Structure



The tree structure in Code Snippet 5.2.2 illustrates the file structure on the management server. *Index.html* and *requestaccess.php* are accessed by clients who request Internet access. *Fuctions.php* and *mysql.php* contain methods used by the other PHP scripts in the structure. The *admin* directory contains PHP scripts related to the manager functionality of the system. The PHP scripts in the *common* directory are utilized by both the mobile- and web application to *get clients* permissions and *post clients*. The *loginapp.php* and *register.php* in the *mobile* directory are used to provide login and register functionality for the mobile application. The *rpi* directory contains PHP scripts that respond to the polling requests sent from the Python process described in Section Polling Server. The *css*, *img* and *js* contain resources provided by Twitter Bootstrap. The management server is available in Appendix B.

5.2.3 Requesting Access

Clients requesting Internet access are directed to the Uniform Resource Locator (URL) of the management server, i.e `apc.item.ntnu.no`. Client traffic is directed to the management server using the Python process described in Section Obtaining Client Identifier.



The image shows a web form titled "Request Internet Access". At the top left of the form area is a blue icon of a Wi-Fi signal. Below the title are three text input fields stacked vertically. The first field is labeled "Name". The second field is labeled "Email (Optional)". The third field is labeled "Phone (Optional)". Below these fields is a blue rectangular button with the word "Submit" written in white text.

Figure 5.3: Request Access Web Page

Index.html contains a web form for clients requesting Internet access. No functionality is currently implemented behind the *email* and *phone* fields, these fields may be used for client authentication in further development. This will be discussed in Section Authentication. The only input required by the client is his/her name. When pressing the *submit* an access request is sent to the management server

5.2.4 Remote Management

The management server implements remote management of the RPI through the *rpi/getclients.php* script.

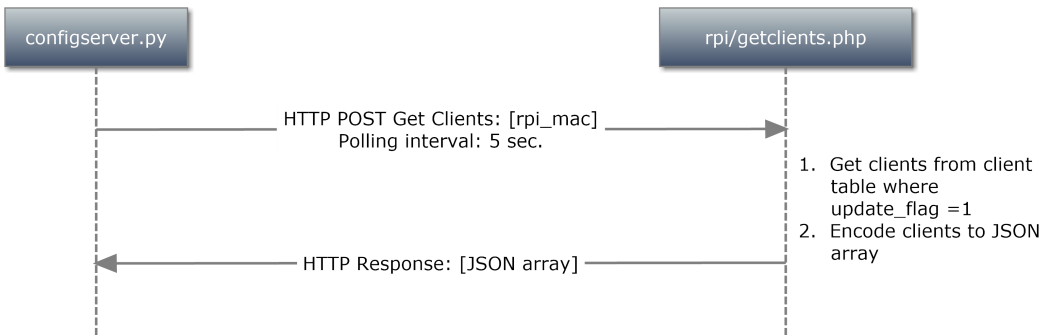


Figure 5.4: Get Clients

This *rpi/getclients.php* script responds to the polling requests from the python process described in Section Polling Server. The response holds a *JSON array* containing clients' Internet permissions. A sample JSON array is displayed in Code Snippet 5.11

5.3 Mobile Management Application

The manager should be able to grant clients Internet access from any location. The mobile application implements functionality to receive push notifications when clients request Internet access. The mobile application is available in Appendix C.

The main functionality of the mobile application is implemented in: *ClientListActivity*, *ClientPermissionsActivity* and *LoginActivity*. In the Android framework an *activity* is usually associated with a single screen presented to the user. The *LoginActivity* is used to login to the mobile application. The *ClientListActivity* and *ClientPermissionsActivity* are used to manage clients' Internet permissions.

5.3.1 Push Notification

To allow the mobile application to receive notifications when a client requests access, GCM was used. To receive push notifications, the mobile application must be registered with the management server and GCM service. The tutorials available at² were utilized to implement the GCM registration procedure.

²<http://developer.android.com/google/gcm/index.html>

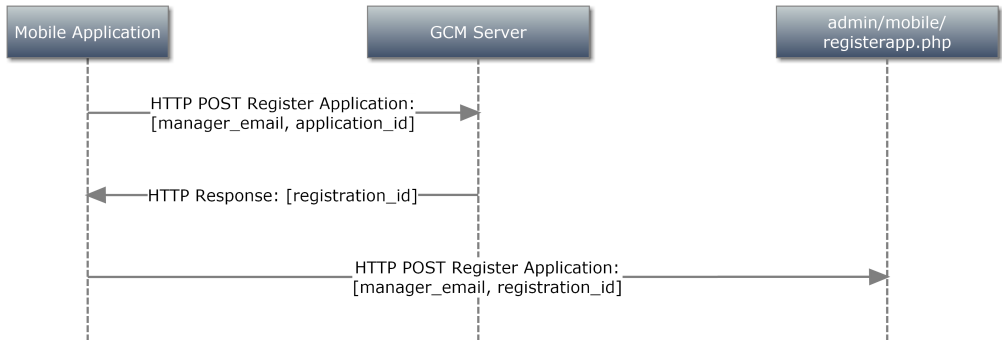


Figure 5.5: GCM Registration Procedure

The mobile application provides the *application_id* and *manager_email* to the GCM server. The *application_id* is an identifier generated by the mobile application, the *manager_email* is obtained through the *LoginActivity* described in Section Manager Login.

The GCM server response with a unique *registration_id*. To register the mobile application on the management server, the *manager_email* and *registration_id* is sent to the *admin/mobile/registerapp.php* script

With the mobile application registered with the management server and GCM service, the management server can send notifications to the mobile application by utilizing the *registration_id* provided in the registration procedure in Figure 5.5.

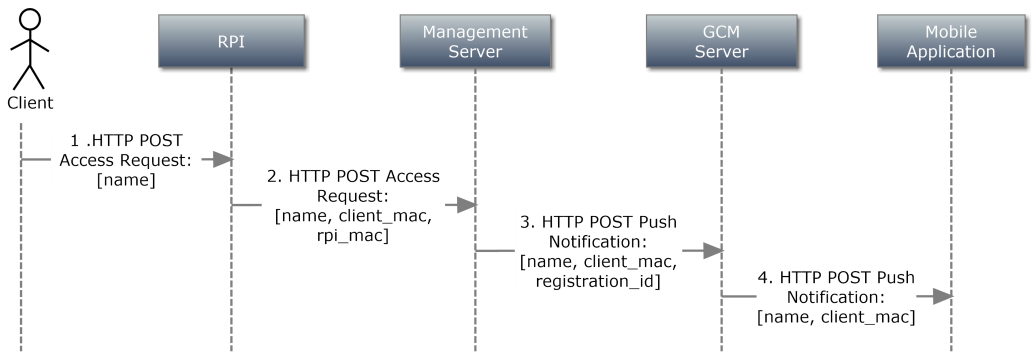


Figure 5.6: Access Request Notification

1. The client requests access through the web page in Figure Request Access Web Page.
2. The *client_mac* and *rpi_mac* are appended to the request at the RPI as described in Section Obtaining Client Identifier. The management server locates the appropriate *app_id* based on the received *rpi_mac*. In the creation of the WifiAccess database fields, a mismatch was made in the naming of the *app_id* field. This field holds the *registration_id* provided by the GCM registration procedure.
3. The management server sends the *name*, *client_mac* and *registration_id* to the GCM server. The *registration_id* is used by the GCM service to locate the destination of the mobile application.
4. GCM sends a push notification to the mobile application containing the *name* and *client_mac*

Upon receiving the push notification, the notification below is displayed in the status bar on the Android device.

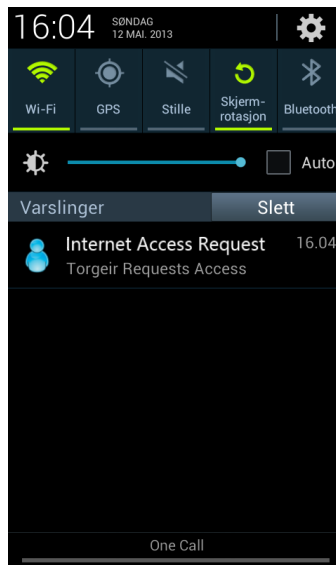


Figure 5.7: Push Notification

When pressing the notification displayed in Figure 5.7, the `ClientPermissionActivity` is started.

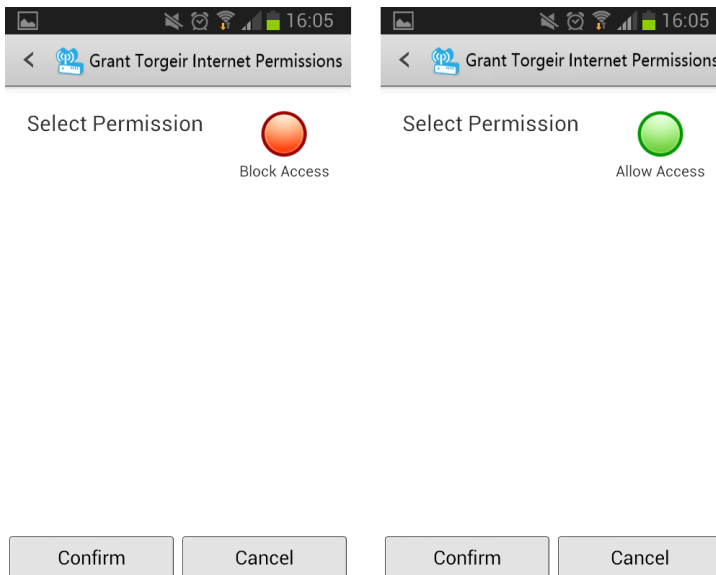


Figure 5.8: Select Client Internet Permission

5.3.2 Manager Login

The LoginActivity is displayed to the manager when the mobile application is initially started. The manager can login by providing an email address that was added to the *WifiAccess* database, as shown in Code Snippet 5.9. The password field is not used in the current implementation.

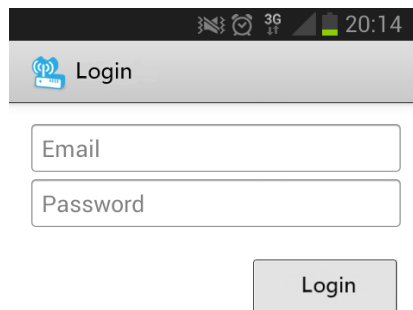


Figure 5.9: Manager Login

5.3.3 Manage Internet Permissions

After logging into the mobile application, clients' Internet permissions are requested from the management server.

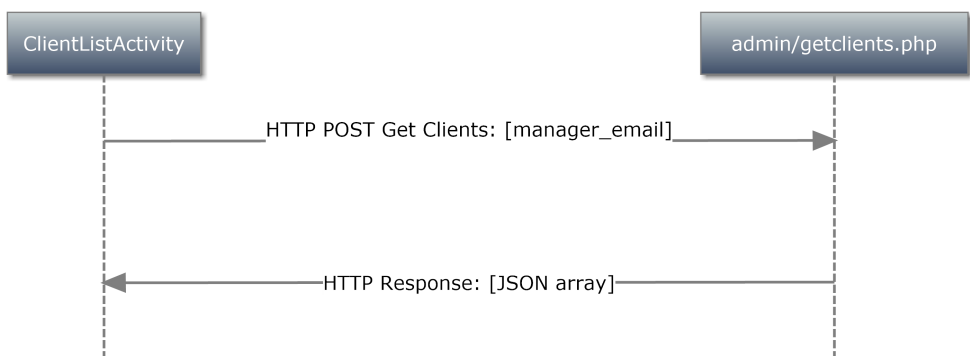


Figure 5.10: Get Clients

The *manager_email* obtained through the LoginActivity is included in the request. The request is sent to `admin/common/getclients.php` on the management server. This PHP script uses the *manager_email* to locate the appropriate clients in the *WifiAccess* database.

The script responds to the request with a *JSON array* containing the clients' Internet permissions. A sample JSON array is displayed in Code Snippet 5.11.

Code Snippet 5.11 JavaScript Object Notation: Client Array

```
[
  {
    "client": {
      "client_info": {
        "mac": "5c:ff:35:22:bd:4e",
        "name": "Alf"
      }
    }
  },
  {
    "client": {
      "permissions": {
        "access": {
          "allow": false
        }
      },
      "client_info": {
        "mac": "f0:7d:68:14:4c:78",
        "name": "Ingrid"
      }
    }
  },
  {
    "client": {
      "permissions": {
        "access": {
          "allow": true
        }
      },
      "client_info": {
        "mac": "74:2f:68:37:d5:a2",
        "name": "Torgeir"
      }
    }
  }
]
```

The `ClientListActivity` uses the JSON array displayed in Code Snippet 5.11 to populate the client list.

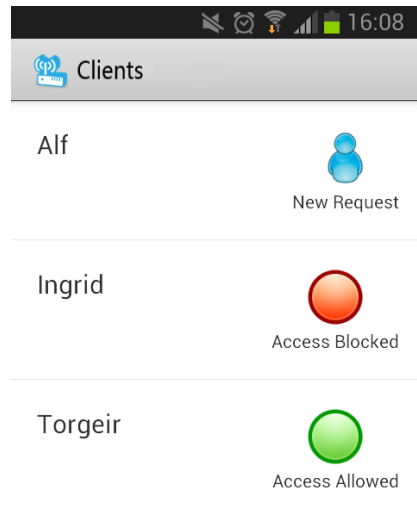


Figure 5.11: Client List

The client *Alf* is identified as a new access request, as no *permissions* are present in the received JSON array. *Ingrid* and *Torgeir* have been granted permissions by the `ClientPermissionActivity`

5.4 Web Management Application

Due to time limitations the web application does not meet the functional requirements specified in Section Functional Requirements. The web application is a mock-up implementation of the specified functionality. The mobile application was given priority since it is better suited for demonstration purposes.

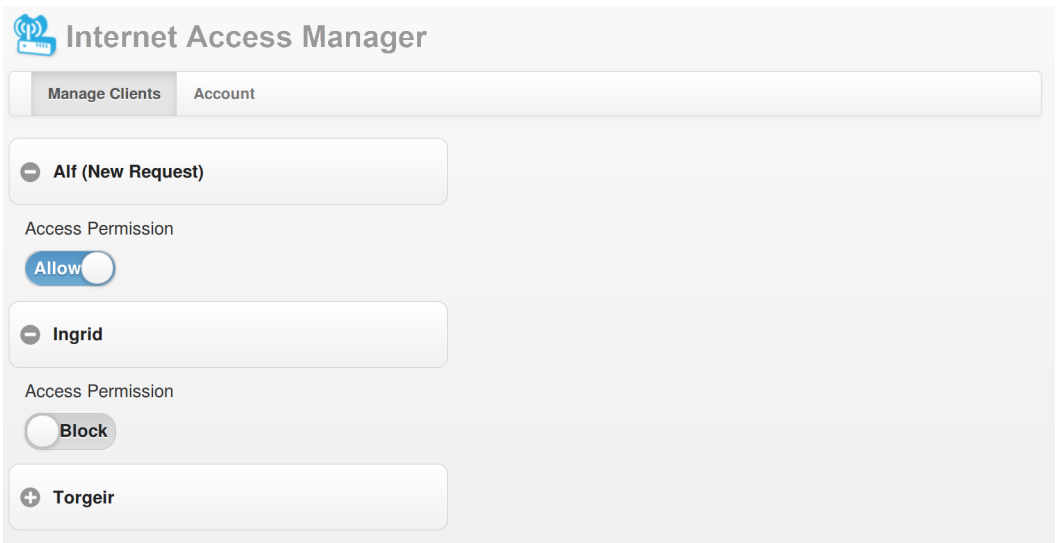


Figure 5.12: Manager Web Page

5.5 Functionality Overview

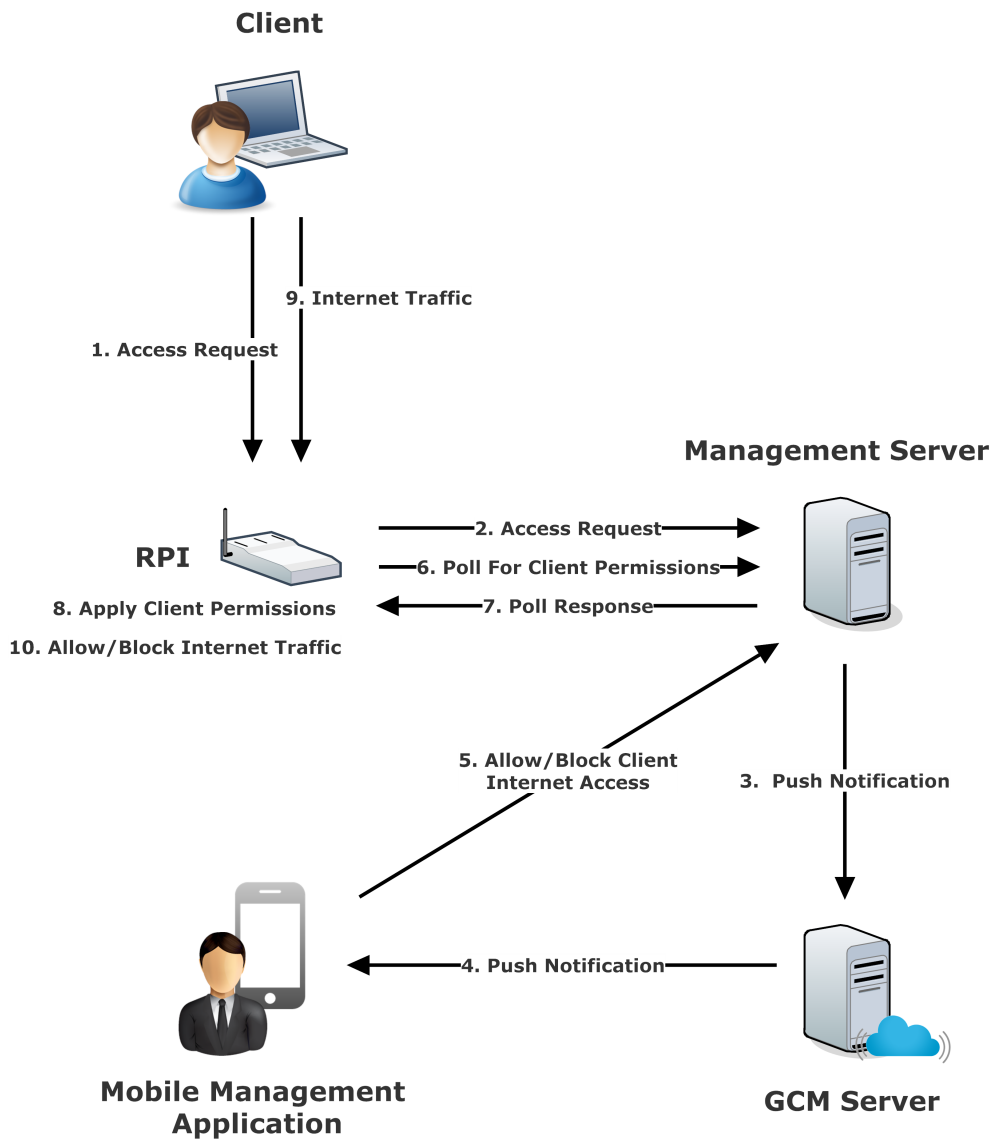


Figure 5.13: Functionality Overview

Chapter 6

System Testing

To ensure that the highest ranked functional requirements were fulfilled, system testing was conducted. The first system aimed to test the core functionality of the system, namely allowing and blocking clients Internet access. The second system test performs the same steps as the first test, but also tests the support for multiple customers in the system.

The system test-setup included the following equipment: *2 RPI type B, 1 Samsung Galaxy 2 Android 4.1.3 Smartphone, 1 Asus Transformer Android 4.0.1 Tablet, 1 Asus U31SD Windows 8 Laptop computer, 1 iPhone 5 iPhone OS (iOS) Smartphone.* The Android devices were used by managers of each RPI. The laptop and iPhone were used to request Internet access. Prior to the system tests the following system configuration was performed.

1. Configure each RPI to offer Wifi access with different SSID. This was performed by editing the configuration file of hostapd.
2. Add 2 entries to the user table in the WifiAccess database. Each entry included the MAC address of its respective RPI as well as the name and email of the manager. This was performed using the MySQL command in Code Snippet 5.9.
3. Install the mobile application on each Android device and provide the appropriate login credentials.

6.1 Allow and Block Internet Access

In this system test the following steps were performed:

1. Clients on the laptop and iPhone gain access to the Wifi offered by the RPI
2. The clients request Internet access through the web browser on the device

3. The manager allows Internet access through the mobile application
4. After 2 minutes have elapsed, the manager blocks Internet access through the mobile application

Step	Title	Functional Requirement	Result
1	Wifi access	The residential access point can provide Wifi access to clients	Verified
1	Unauthorized Permissions	The residential access point can block all traffic from unauthorized clients. Only traffic required to request Internet access is allowed	Verified
2	Request Access	Internet access can be requested by through the web application	Verified
3	Allow Internet Access	The residential access point can allow clients Internet access based on MAC- and IP address	Verified
3	Remote Management	The residential access point can be managed remotely from the management sever	Verified
3	Access Request Notification	The mobile application can receive push notifications when clients request Internet access	Verified
3	Allow Internet Access	The mobile application can allow clients Internet access	Verified
4	Block Internet Access	The residential access point can block clients Internet access based on MAC- and IP address	Verified
4	Block Internet Access	The mobile application can block clients Internet access	Verified

Table 6.1: Allow and Block Access Test Results

6.2 Multiple Customers

The systems should support the management of RPIs associated with multiple customers. For this test the second RPI in the test set-up was employed. This test included the steps in the previous section. The steps were however performed by requesting access to the Wifi network offered by each of the RPIs. Internet access was allowed and blocked by the Android device associated with its receptive RPI.

Step	Title	Functional Requirement	Result
1	Wifi access	The residential access point can provide Wifi access to clients	Verified
1	Unauthorized Permissions	The residential access point can block all traffic from unauthorized clients. Only traffic required to request Internet access is allowed	Verified
2	Request Access	Internet access can be requested by through the web application	Verified
3	Allow Internet Access	The residential access point can allow clients Internet access based on MAC- and IP address	Verified
3	Remote Management	The residential access point can be managed remotely from the management sever	Verified
3	Multiple Customers	The management server can support managing of residential access point belonging to different customers	Verified
3	Access Request Notification	The mobile application can receive push notifications when clients request Internet access	Verified
3	Allow Internet Access	The mobile application can allow clients Internet access	Verified
4	Block Internet Access	The residential access point can block clients Internet access based on MAC- and IP address	Verified
4	Block Internet Access	The mobile application can block clients Internet access	Verified

Table 6.2: Multiple Customers Test Results

Chapter 7

Discussion and Further Work

7.1 Authentication

The client authenticating procedure implemented in the current system requires clients to provide their name into the *Request Access Web Page* displayed in Figure 5.3. A client requesting access could potentially masquerade as a different client by providing a false name. No restriction is placed on the amount of times a client can request access. A malicious client could flood the manager with bogus access requests. These attacks can be prevented by implementing a stronger authentication procedure. Clients could authenticate themselves by providing their email or phone number. An authentication code could be provided by the management server either by email or Short Message Service (SMS).

7.2 Authorization

The current implementation provides client authorization by splitting the IP address range offered by the DHCP server into two ranges. When a client is given Internet permissions a lease is granted in the authorized range, as depicted in Figure 5.1. The change from unauthorized range to the authorized range may take up to 2 minutes due to the minimum DHCP lease time allowed by *dnsmasq*. This delay is significantly higher than any other response-time and thus becomes the responsiveness bottle-neck.

This problem can be eliminated by having *iptables* maintain a white-list of MAC address. The white-list should contain the MAC-addresses of clients who have been granted Internet permissions. Clients who are not on the white-list should be directed towards the *Request Access Web Page* displayed in Figure 5.3.

7.3 Accounting

Gathering IP traffic statistics from clients on the RPI was a low-priority requirement specified in Section Functional Requirements. Clients' Internet traffic statistics could

be gathered on the RPI and sent to the management server. The statistics could be presented to the manager through the mobile- and web application. Some Internet traffic statistics that might interest the manager are listed below.

- Most visited web sites
- Internet services used, e.g online games or video-streaming
- Peak Internet usage period
- Average bandwidth used

7.4 System Feedback

In the current implementation no feedback is given to clients regarding the operation of the system. Providing feedback is key in promoting a satisfactory user experience.

7.4.1 Requesting Access

When clients request Internet access a considerable amount of time may elapse before the manager reacts to the access request. To prevent the client from growing impatient, the system should give feedback to the client about the state of the access request.

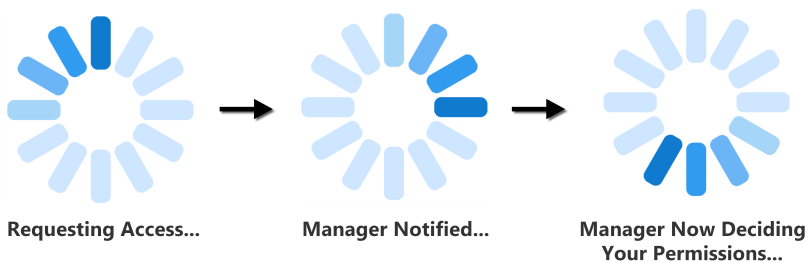


Figure 7.1: Access Request Feedback

This functionality can be realized by using Asynchronous JavaScript and XML (AJAX). AJAX provides the functionality to update elements of a web page without reloading the entire page. After a client has initially requested access, AJAX-requests could be sent querying about the state of the request.

7.4.2 Visible Permissions

In the current implementation no functionality is realized to display clients Internet permissions. If traffic is blocked by iptables, no feedback is given to the client explaining why the traffic was blocked. Feedback could be provided by directing clients to web page explaining why their traffic was blocked.



Figure 7.2: Client Internet Permissions

7.5 Uniform Management Interface

The interface provided by the mobile- and web application offers the same management functionality. Developing a separate mobile- and web application proved to be a time consuming task. *PhoneGap* [26] is free and open source development platform based around HTML, CSS and Javascript. PhoneGap supports the major mobile platforms such as Android, iOS and Windows Phone. Developing a PhoneGap application would provide many benefits to the approach used in this project. Open standards such as HTML, CSS and JavaScript would allow reuse of code between the mobile- and web application. Reusing code would provide a more uniform manager interface and also greatly reduce development time. PhoneGap would also provide support for other mobile platforms in addition to Android.

7.6 Validating Permissions

In the current system, client Internet permissions are represented using JSON. Creating the JSON objects representing clients' permissions is a trivial task in the current implementation, since the only client permissions supported is to allow and block Internet access. The light-weight nature of JSON makes it unsuitable to represent more complex permissions. The JSON permissions are read and written by multiple components in the system, providing some form of permission validation becomes very important to identify invalid permissions. Consider the case where

the mobile- or web application generate overlapping client Internet permissions, e.g *block all Internet traffic* and *allow Internet traffic between 18:00 and 21:00*. These permissions are mutually exclusive and cannot be enforced simultaneously. In the current implementation this overlap would not be detected, and the permissions would be converted to iptables rules on the RPI. Which permissions are enforced depends on the order they are added to iptables. The practical work of this project revealed that it was very difficult do detect overlapping rules in iptable.

One might argue that extensive care could be given to the mobile- and web application code generating the JSON permissions. Specifically, Graphical User Interface (GUI) elements could be enabled or disabled to prevent the creation of overlapping JSON permissions. This approach would leave great margin for error. The application programmer would require a complete overview of overlapping Internet permissions, and enable or disable GUI elements accordingly.

XML provides document validation through the use of XML schemas, as described in Section Client Permission Format. A XML schema could be created defining the valid set of client Internet permissions. XML schemas have the capability to define mutually exclusive elements. XML also provides more structural features than JSON. In spite of this, XML is harder to work with. An approach utilizing the strengths of both XML and JSON should be considered in further development.

7.7 Double Network Address Translation

The system implemented is likely to impose double NAT on the network of the customer, i.e NAT is performed on the RPI and home-router. Double NAT causes complications for applications using peer-to-peer and Voice over Internet Protocol (VoIP) traffic. A solution to this problem not involving configuration of the home router of customer was not found. A way to solve this issue would be to remove the home router and let the RPI function as the router on the home network. This would, however, give rise to new challenges of how to configure the RPI to function with the Internet Service Provider (ISP) of the customer.

7.8 Modes of Notification

The current implementation allows push notifications to the mobile application when clients request Internet access. The mobile application may not be able to receive the push notification for a variety of reasons. If the mobile application is unable to receive the push notification, a requesting client must wait for an indefinite time before the manager reacts to the access request. Other modes of notification should be considered in further development. The manager could be notified of clients' Internet access requests through email and SMS. The functionality described below

could be implemented on the management server to decrease the manager response time.

Code Snippet 7.1 Pseudocode: Access Request Notification Procedure

```

if (is mobile application activty) do:

    # Send push notification to managers mobile application

    if not (is push notification response received within 60 sec) do:

        # Send SMS to the managers mobile phone

        if not (is sms response received within 60 sec) do:

            # Send e-mail to manager

```

7.9 Customer Management

The main focus of this project was to implement a prototype system suited for demonstration purposes. Consideration was not given to how customers are managed in the system. Further development of the system should consider the implementation of an administration interface that can be utilized by the service provider of the system. Through this interface a link could be created between the customer and the RPI. The administration interface could support other services such as billing and customer support.

Chapter 8

Conclusion

This project has implemented a working prototype Internet control system for the residential environment. The prototype offers a simple and intuitive interface to users, and thus provides a good platform to demonstrate how effortless residential Internet control can be. The practical work performed yielded valuable insight into software development in distributed systems. Strengths and weaknesses of the relevant platforms and technologies were discovered during the course of implementation.

The current system has shortcomings both in regard to security and offered Internet control features. However, this report lays the groundwork to make amends in further development.

References

- [1] Torgeir Pedersen Cook. Wifi parental control. Technical report, May 2013.
- [2] Hubert Zimmermann. Osi reference model-the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4):425–432, 1980.
- [3] Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1–2793, 2012.
- [4] Bernard Aboba, Larry Blunk, John Vollbrecht, James Carlson, Henrik Levkowetz, et al. Extensible authentication protocol (eap). Technical report, RFC 3748, June, 2004.
- [5] Haidong Xia and José Brustoloni. Detecting and blocking unauthorized access in wi-fi networks. In *NETWORKING 2004. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications*, pages 795–806. Springer, 2004.
- [6] Wi-Fi Alliance. Wi-fi protected access: Strong, standards-based, interoperable security for today’s wi-fi networks. *Retrieved March*, 1:2004, 2003.
- [7] Gerald Combs. Wirehark, network protocol analyzer tool. Software. <http://www.wireshark.org/>, 1998.
- [8] Ergonomic requirements for office work with visual display terminals. (ISO 9241-11), 1998.
- [9] Supplement to ieee standard for information technology- telecommunications and information exchange between systems- local and metropolitan area networks-specific requirements- part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications: Higher-speed physical layer extension in the 2.4 ghz band. *IEEE Std 802.11b-1999*, pages i–90, 2000.

- [10] Iso/iec 8802-11:2005/amd4 [ieee std 802.11g-2003] information technology– local and metropolitan area networks– part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications–amendment 4: Further higher data rate extension in the 2.4 ghz band. *ISO/IEC 8802-11:2005/Amd.4:2006(E) IEEE Std 802.11g-2003 (Amendment to IEEE Std 802.11-1999)*, pages 1–83, 2006.
- [11] Raspberry Pi Foundation. Raspbian wheezy, debian based operating system. Software. <http://www.raspberrypi.org/downloads>, 2012.
- [12] Jouni Malinen. hostapd, ieee 802.11 ap, ieee 802.1x/wpa/wpa2/eap/radius authenticator. software. <http://hostap.epitest.fi/hostapd/>, 2009.
- [13] R. Droms. Dynamic host configuration protocol. Technical report, 1997. IETF, RFC2131.
- [14] Simon Kelley. dnsmasq, linux dns forwarder and dhcp server. Software. <http://www.thekelleys.org.uk/dnsmasq/doc.html>, 2001.
- [15] Rusty Russell. iptables, linux kernel ip packet filtering. Software. <http://www.netfilter.org/projects/iptables/index.html>, 1998.
- [16] Oracle. Mysql, open source database community server. Software. www.mysql.com, 1995.
- [17] Canonical Ltd. / Ubuntu community. Ubuntu lts server, linux based operating system. Software. <http://releases.ubuntu.com/lucid/>, April 2010.
- [18] Apache Software Foundation. Tomcat 6 web server. Software, May.
- [19] The PHP Group. Php, server scripting language. Software. <http://www.php.net/>, 1995.
- [20] Twitter. Twitter bootstrap, sleek, intuitive, and powerful front-end framework for faster and easier web development. Software. <http://twitter.github.io/bootstrap/>, August 2011.
- [21] Google. Android, linux based smartphone operating system. Software. <http://www.android.com/>, 2007.
- [22] Google. Google cloud messaging, push notification service for android. Software. <http://developer.android.com/google/gcm/index.html>, June 2012.
- [23] Linus Torvalds. git, technology for distributed version control and source code managemnet. Software. <http://git-scm.com/>, 2005.
- [24] PJ Hyett Chris Wanstrath and Tom Preston-Werner. Github, web-based hosting service for software development projects that use the git revision control system. Software. <http://www.github.com>, 2008.
- [25] Pyda Srisuresh and Kjeld Egevang. Traditional ip network address translator (traditional nat). Technical report, 2001.

- [26] Adobe Systems. Phonegap, front-end web application framework. Software. <http://phonegap.com/>, 2009.

Appendix

Raspberry Pi



A.1 Configuration files

A.1.1 hostapd.conf

Code Snippet A.1 Configuration File: /etc/hostapd/hostapd.conf

```
interface=wlan0
driver=nl80211
ctrl_interface=/var/run/hostapd
ctrl_interface_group=0
ssid=raspberry
hw_mode=g
channel=8
wpa=0
beacon_int=100
auth_algs=1
wmm_enabled=1
```

A.1.2 dnsmasq.conf

Code Snippet A.2 Configuration File: /etc/dnsmasq.conf

```
interface=wlan0
dhcp-range=unauth,10.0.0.65,10.0.0.94,2m
dhcp-option-force=unauth,1,255.255.255.224
dhcp-option-force=unauth,6,129.241.200.170,129.241.200.170
dhcp-range=auth,10.0.0.1,static,2m
dhcp-option-force=auth,1,255.255.255.192
dhcp-authoritative
dhcp-hostsfile=/etc/dnsmasq.hosts
```

A.1.3 interfaces

Code Snippet A.3 Configuration File: /etc/network/interfaces

```

auto lo
iface lo inet loopback
iface eth0 inet dhcp

iface wlan0 inet static
address 10.0.0.1
netmask 255.255.255.0

post-up /etc/network/if-up.d/router.sh

```

A.2 Bash Scripts

A.2.1 add_static_lease

Code Snippet A.4 Bash Script: /etc/add_static_lease.sh

```

#Add lease to the DHCP lease file of dnsmasq
if ! grep -q $1 /etc/dnsmasq.hosts; then
    line=$(grep $1 /var/lib/misc/dnsmasq.leases)
    myarr=($line)
    device=${myarr[3]}
    dhcphost="dhcp-host=$1,$2,$device,$3"
    echo $dhcphost >> /etc/dnsmasq.hosts
    echo "lease added to dnsmasq.hosts: $dhcphost"
else
    echo "dnsmasq.hosts already contains a lease with mac: $1"
fi

```

A.2.2 reload_dhcp_leases

Code Snippet A.5 Bash Script: /etc/reload_dhcp_leases.sh

```
#Reload dnsmasq configurations
pid=$(pgrep dnsmasq) # store the return value
echo "dnsmasq pid: $pid"

sudo /bin/kill -s SIGHUP $pid
echo "dnsmasq config reloaded"
```

A.2.3 iptables_setup

Code Snippet A.6 Bash Script: /etc/network/if-up.d/iptables_setup.sh

```
#DNS server of unathorized subnet
DNS1="129.241.200.170"
#IP range of unthorized subnet
IPunauth="10.0.0.64/27"
#IP range authorized subnet
IPauth="10.0.0.1/26"

#Flush all tables and delete all custom chains
iptables -F
iptables -X
iptables -t nat -F
iptables -t nat -X
iptables -t filter -F
iptables -t filter -X
iptables -t mangle -F
iptables -t mangle -X
iptables -t raw -F
iptables -t raw -X

#Default policies for chains in filter table
iptables -t filter -P INPUT ACCEPT
iptables -t filter -P FORWARD DROP
iptables -t filter -P OUTPUT ACCEPT

#NAT
iptables -A POSTROUTING -t nat -o eth0 -j MASQUERADE

#Allow traffic to and from the management sever
iptables -A FORWARD -s $IPunauth -d $DNS1 -j ACCEPT
iptables -A FORWARD -s $DNS1 -d $IPunauth -j ACCEPT

#Direct http traffic to local proxy to obtain client MAC address
iptables -t nat -A PREROUTING -s $IPunauth -p tcp --dport 80
-j REDIRECT --to-ports 8089

#Redirect locally generated DNS traffic
iptables -t nat -A OUTPUT -p udp --dport 53 -j DNAT --to $DNS1
iptables -t nat -A OUTPUT -p tcp --dport 53 -j DNAT --to $DNS1
```

A.3 Python Modules

A.3.1 Polling Server

Code Snippet A.7 Python Module: /etc/network/server/configserver.py

```
#!/usr/bin/python
import httplib
import socket
import fcntl
import struct
import urllib
import json
import clienthandler
import util
import iptablesapi
import time, threading

class Server:

    def __init__(self, client_list=None):

        self.clientlist = clienthandler.ClientList()
        iptablesapi.initialSetup()
        self.isInitialRequestSent = False
        self.bootFlag = 0;

    def getClientPermissions(self):
        #Start polling thread
        threading.Timer(5.0, self.getClientPermissions).start()

        print 'Boot Flag: %s' % self.bootFlag

        httpServ = httplib.HTTPConnection("129.241.200.170", 80)
        httpServ.connect()
        payload = urllib.urlencode({'macaddress':
            util.getMacAddress('eth0'), 'boot_flag' : self.bootFlag})
        headers = {"Content-type":
            "application/x-www-form-urlencoded", "Accept":
            "text/plain"}
```

```
request = httpServ.request('POST', '/rpi/getclients.php',
payload , headers)
response = httpServ.getresponse()
print 'Permissions Polling Request Sent'

if response.status == httplib.OK:
    array = response.read()
    data = json.loads(array)
    self.bootFlag = 1;

if len(data) == 0:
    print 'No client updates recieved from server'

else:
    for str in data:
        client_string = json.dumps(str)
        print 'Client from server : %s'
        % client_string
    client_json = json.loads(client_string)
    key = client_json ['client']['client_info']
    ['mac']

    self.clientlist.removeClient(key)
    self.clientlist.addClient(client_json)

    reload_leases_cmd =
    'bash /etc/reload_dhcp_leases.sh'
    iptablesapi.executeCommand(reload_leases_cmd)

httpServ.close()

def sendBindingUpdate(self):
    #Start posting thread
    threading.Timer(60.0, self.sendBindingUpdate).start()

httpServ = httplib.HTTPConnection("129.241.200.170", 80)
httpServ.connect()
payload = urllib.urlencode(
{'macaddress': util.getMacAddress('eth0'), 'ipaddress':
headers = {"Content-type":
"application/x-www-form-urlencoded","Accept": "text/plain"}
request = httpServ.request('POST', '/rpi/bindingupdate.php',
payload, headers)
```

```
        response = httpServ.getresponse()
        print 'Binding Update Sent'
        if response.status == httplib.OK:
            print "Binding Update Success"

    httpServ.close()

serv = Server()
serv.getClientPermissions()
serv.sendBindingUpdate()
```

Repository The Python code is published to the the public repositories hosted by GitHub. The code can be viewed either through a web browser or retrieved to your computer by following the procedure described in Appendix D.4.2. Repository URL: <https://github.com/TorgeirCook/RPIServerConfigurationServer.git>.

```
        else:
            self.sendHeader(key, value)

    self.endHeaders()

def sendPostData(self):
    log.msg("Sending POST data")
    self.transport.write(self.postData)
    log.msg('The POST data sent: %s' % self.postData)

def connectionMade(self):
    log.msg("HTTP connection made")
    self.sendRequest()
    self.sendHeaders()
    if self.method == 'POST':
        self.sendPostData()

def handleStatus(self, version, code, message):
    log.msg("Got server response: %s %s %s"
           % (version, code, message))
    self.originalRequest.setResponseCode(int(code), message)

def handleHeader(self, key, value):
    if key.lower() == 'content-length':
        self.contentLength = value
    else:
        self.originalRequest.responseHeaders
            .addRawHeader(key, value)

def handleResponse(self, data):
    data = self.originalRequest.processResponse(data)

    if self.contentLength != None:
        self.originalRequest.setHeader('Content-Length',
                                       len(data))

    self.originalRequest.write(data)

    self.originalRequest.finish()
    self.transportloseConnection()
```

```
class ProxyClientFactory(protocol.ClientFactory):

    def __init__(self, method, uri, postData, headers,
originalRequest):
        self.protocol = ProxyClient
        self.method = method
        self.uri = uri
        self.postData = postData
        self.headers = headers
        self.originalRequest = originalRequest

    def buildProtocol(self, addr):
        return self.protocol(self.method, self.uri, self.postData,
                             self.headers, self.originalRequest)

    def clientConnectionFailed(self, connector, reason):
        log.err("Server connection failed: %s" % reason)
        self.originalRequest.setResponseCode(504)
        self.originalRequest.finish()

class ProxyRequest(http.Request):
    def __init__(self, channel, queued, reactor=reactor):
        http.Request.__init__(self, channel, queued)
        self.reactor = reactor

    def process(self):

        ip = http.Request.getClientIP(self)
        rpi_mac = util.getMacAddress('eth0')
        client_mac = util.getMacAddressFromIP(ip)
        url = self.uri
```

```
log.msg("Request IP : %s, Request MAC : %s , RPI MAC: %s,
Request URL: %s" % (ip, client_mac , rpi_mac, url))
host = self.getHeader('host')
if not host:
    log.err("No host header given")
    self.setResponseCode(400)
    self.finish()
    return

port = 80
if ':' in host:
    host, port = host.split(':')
    port = int(port)

self.setHost(host, port)
self.content.seek(0, 0)
postData = self.content.read()
postData += '&client_mac=%s&rpi_mac=%s'
% (client_mac, rpi_mac)

factory = ProxyClientFactory(self.method, self.uri,
postData, self.requestHeaders
    .getAllRawHeaders(),
    self)
self.reactor.connectTCP(host, port, factory)

def processResponse(self, data):
    return data

class TransparentProxy(http.HTTPChannel):
    requestFactory = ProxyRequest

class ProxyFactory(http.HTTPFactory):
    protocol = TransparentProxy

reactor.listenTCP(8089, ProxyFactory(), interface='0.0.0.0')
reactor.run()
```

Repository The Python code is published to the the public repositories hosted by GitHub. The code can be viewed either through a web browser or retrieved to your computer by following the procedure described in Appendix D.4.2. Repository URL: <https://github.com/TorgeirCook/RPIProxyServer.git>.

Appendix **B**

Management Server

Description The management server contains:

- Web page for clients requesting Internet access
- Web for managing clients Internet permissions
- Functionality to respond to Raspberry Pi polling requests
- Functionality to support the mobile application

Repository The PHP, HTML, CSS and JavaScript code is published to the the public repositories hosted by GitHub. The code can be viewed either through a web browser or retrieved to your computer by following the procedure described in Appendix D.4.2. Repository URL: <https://github.com/TorgeirCook/ManagementServer.git>.

Appendix

Mobile Application

Description Android application to manage clients Internet permissions.

Repository The Java code is published to the the public repositories hosted by GitHub. The code can be viewed either through a web browser or retrieved to your computer by following the procedure described in Appendix D.4.2. Repository URL: <https://github.com/TorgeirCook/WifiAccessManager.git>.

Appendix **D**

Development Aid

The purpose of the Development Aid is to ease further of the system. If you have any questions please contact me at torgeirpc@gmail.com.

D.1 Raspberry Pi

The Debian Wheezy system image used on the RPI is provided on request.

D.1.1 Commands

```
# login to rpi
username=pi
password=pcontrol

# start-stop-restart hostapd
sudo /etc/init.d/hostapd start-stop-restart, started on boot

# start-stop-restart dnsmasq
sudo /etc/init.d/dnsmasq start-stop-restart, started on boot

# run initial set-up of iptables, the rules in this bash script are added # to ipt
sudo /etc/network/if-up.d/iptables_setup.sh

# view all rules in iptables
sudo iptables -L -v -n

# run add_static_leas bash script
/etc/add_static_lease.sh

# run reload_dhcp_leases bash script
```

```
sudo /etc/reload_dhcp_leases.sh

# run python script
sudo python example.py

# view all running processes
ps aux | less

# edit crontab jobs. crontab is used to schedule time based tasks
crontab -e

# view non-commented lines of a file
grep ^[~#] /etc/file
```

D.1.2 Directories

```
# hostapd configuration file
/etc/hostapd/hostapd.conf

# dnsmasq configuration file
/etc/dnsmasq.conf

# network interfaces configuration file
/etc/network/interfaces

# dnsmasq dhcp dynamic lease file
/var/lib/misc/dnsmasq.leases

# dnsmasq static dhcp lease file. leases added to this file are
# added to dnsmasq on dnsmasq restart
/etc/dnsmasq.hosts

# directory of the python code used in this project
/etc/network/server/
```

D.2 Management Server

```
# login to management server
username=torgeir
```

```
password=pedersen
```

D.2.1 Apache Tomcat

```
#start-stop-restart apache tomcat server
sudo /etc/init.d/tomcat6 start-stop-restart

# apache tomcat log file
/var/log/tomcat6/catalina.out

# deployed web applications directory
/var/lib/tomcat6/webapps/

# web interface to manage and deploy web applications
# username=admin
# password=pcontrol
http://apc.item.ntnu.no/manager/html

# root web application url
http://apc.item.ntnu.no
```

D.2.2 MySQL

```
# login to mysql server
username=root
password=pcontrol

# start mysql database interface
sudo mysql

# mysql command: use database
use WifiAccess
```

D.3 Mobile Application

To enable Android development install the Android Software Development Kit and Tools available at: <http://developer.android.com/sdk/index.html>.

D.4 Software Tools

D.4.1 Eclipse

Eclipse was used as the IDE for all development in this project. Eclipse is available at: <http://www.eclipse.org/downloads/>. Eclipse provides plug-ins for all the programming languages used in this project. Plug-ins can be installed through *Help->Eclipse Marketplace* in the Eclipse menu bar.

Eclipse is the default IDE for Android development. An Android application is installed on your the device by running the application in Eclipse.

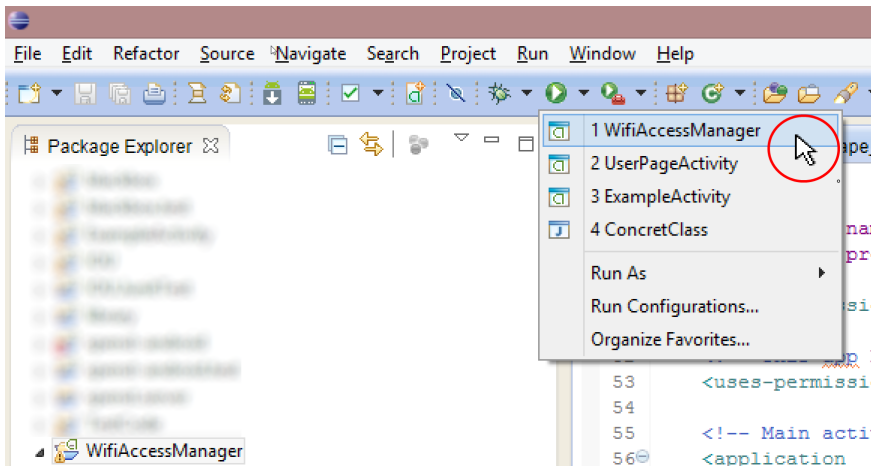


Figure D.1: Install Android Application

D.4.2 Git

The revision control technology used in this project was *git*. All the code written in this project is available at the public repositories hosted by *GitHub*. The URL to the different repositories is given in the Appendix. To gain access to the code, first install *git* on your Linux system using the following command.

```
sudo apt-get install git
```

To retrieve the the code from the repositories given in Appendices, run the following *git* command in the command line.

```
git clone URL
```



```
# example, get mobile application
git clone https://github.com/TorgeirCook/WifiAccessManager.git
```

Git clients are available for other operating systems at: <http://git-scm.com/downloads/guis>. To gain a more comprehensive understanding of *git* consult: rogerdudler.github.io/git-guide/.