



NTNU – Trondheim
Norwegian University of
Science and Technology

iPhone Application for Controlling Wireless Access Point

Xiao Chen

Submission date: December 2013
Responsible professor: Bjørn J. Villa, ITEM
Supervisor: Poul E Heegaard, ITEM

Norwegian University of Science and Technology
Department of Telematics

Abstract

Since internet has become essential part of people daily life, the requirements for accessing internet in both public and residential area are growing magnificently. Then the new issue about how to share and manager the internet environment in both public and residential area become important to the users. This project will implement a prototype iPhone Operation System (IOS) application as service manager client to work with improved existing Wifi hot-spot system which is provided by Raspberry Pi (RPI) and remote central management server.

The main idea about this project is to make a system which can provide internet access allocation and internet filtering controls in wireless network. This project is based on previous student master project, that means this project will use same device and resources which used in that master project. To set up existing access control system made by previous master project, most of the references are taken from the master report[Coo13b] written by Torgeir Pedersen Cook.

This report will include the improved internet access control system and IOS administrator application. The improved access control system now will have the function to block the access request device to connect into wireless network for blocking client user purpose. Moreover, the central management server will have better security login mechanism and sending E-mail notification mechanism. The application contains basic administrate access control function like approve access request and block access request from the client user. And also it will have real-time updating access request list and some security communication mechanism.

This prototype has been tested on test RPI device and test wireless network environment.

This report will also discuss the usability of this prototype in the commercial market and better improvement on the basic working mechanism of the internet access control system.

Acknowledgements

Written by Xiao Chen in Trondheim in December 2013

Thanks for Bjørn J. Villa, Poul E Heegaard and Torgeir Pedersen Cook.

All the source code is published on Github as public repositories for later use.

https://github.com/br1anchen/WifiAccess_RPIServer

https://github.com/br1anchen/WifiAccess_ManagementServer

https://github.com/br1anchen/WifiAccessManager_Android

https://github.com/br1anchen/WifiAccessManager_IOS

Contents

List of Figures	vii
List of Tables	viii
List of Algorithms	ix
List of Acronyms	xi
1 Introduction	1
1.1 Motivation	1
1.2 Related Work	1
1.3 Scope	2
1.4 Report Structure	4
2 System Description	7
2.1 Existing Internet Access Control System	7
2.2 Improvement of Existing System	7
2.3 Analysis of System	9
2.4 User Case	10
3 Raspberry Pi Setting	13
3.1 About Raspberry Pi	13
3.1.1 RPI Hardware for Project	13
3.1.2 RPI Operation System	14
3.2 Applications Using on Raspberry Pi	15
3.2.1 hostapd	15
3.2.2 Dnsmasq	15
3.2.3 Iptables	17
3.3 Scripts on Raspberry Pi	17
3.3.1 RPI Configuration Python Script	18
3.3.2 RPI Client Handler Python Script	19
3.3.3 RPI Bash Shell Scripts	19

4	Central Management Server Improvement	23
5	Mobile Application Development	25
6	System Testing	27
7	Future Work	29
	References	31
	Appendices	
A	Appendix	33
	A.1 RPI Residential Access Point Scripts	33

List of Figures

1.1	Kickstarter Project: Meet Circle	3
2.1	Existing System Architecture	9
2.2	Request Client User Web Request Page	10
3.1	Raspberry Pi used as Residential access point	14

List of Tables

2.1 : Improvement Functions for System	8
--------------------------------------------------	---

List of Algorithms

3.1	hostapd configuration file	15
3.2	dnsmasq configuration	16
3.3	dnsmasq dhcp host format	16
3.4	dnsmasq hosts file	17
3.5	timer method in configserver.py	18
3.6	Original add_static_lease.sh	20
3.7	Improved add_static_lease.sh	20
3.8	block_static_lease.sh	21
3.9	Original reload_dhcp_leases.sh	21
3.10	Improved reload_dhcp_leases.sh	21
3.11	clear_dhcp_lease.sh	22
A.1	network interface configuration file	33
A.2	getClientPermissions function in configserver.py file	34
A.3	main functions in clienthandler.py file	35
A.4	iptables_setup.sh	36
A.5	main functions in iptablesapi.py	37

List of Acronyms

ADT Android Development Tools.

AP Access Point.

ARP Address Resolution Protocol.

DHCP Dynamic Host Configuration Protocol.

DNS Domain Name System.

EAP Extensible Authentication Protocol.

GB Gigabyte.

HTTP Hypertext Transfer Protocol.

IDE Integrated Development Environment.

IEEE Institute of Electrical and Electronics Engineers.

IOS iPhone Operation System.

IP Internet Protocol.

ITEM Telematics.

JSON JavaScript Object Notation.

MAC Media Access Control.

MAC OS Macintosh Operating System.

NTNU Norwegian University of Science and Technology.

RADIUS Remote Authentication Dial In User Service.

RPI Raspberry Pi.

SD Secure Digital.

SDK Software Development Kit.

USB Universal Serial Bus.

WPA Wi-Fi Protected Access.

WPA2 Wi-Fi Protected Access 2.

Chapter 1

Introduction

1.1 Motivation

Based on existing access control system, it is quite promising idea about making normal user have control of the residential area internet access control by using smart filtering, informing and time management. Since the smart phone with the mobile internet access functionality released, more and more sociology research, for instance the paper form Jim McGuigan[McG05], show that there are many weakness of over using mobile phone and internet. The idea of this project is to meet the need of the people who would like to control the daily use of the internet and manager other people access right to some specific wireless network.

Nowadays people are using smart phone to do as much task as they can, because smart phone is easy to carry with and smart phone is the only necessary access people need to have the tons of internet information. Then the requirement of using smart phone to control the existing daily life system such as wireless router, switch and other electronic devices are growing rapidly. This prototype project is to fill the missing part of the previous project to connect the access control system to IOS devices. There are static article[Pro13a] shows that Android has 81.0 percent smartphone share in the third quarter of 2013 and IOS has 12.9 percent, these two mobile operating system shared most mobile operating system in the world. This project will focus on the IOS application to work with the improved access control system. It will make the whole system become more user friendly for normal user to choose different mobile operating system to administrate the system.

1.2 Related Work

There are many security software can do internet control in the market. Such as Norton Family application[Fam13], K9 web Protection[Pro13b], OpenDNS[Sol13] and etc. Most of them can provide block web sites, time restrictions, easy log reports of internet activity and etc. But these kind of software need to install in every access



Figure 1.1: Kickstarter Project: Meet Circle

device in the network. And also it could be uninstalled and broken by accident. They are more like voluntarily joining the internet control policy, which is used greatly on parent and children case but no more other normal cases.

There is a kickstarter[Cir13a] project which has the similar idea as this prototype project. The product name is 'Circle'[Cir13b]. Circle is a device1.1, managed by an IOS app, that enables user to choose how you and your family spend time online by using advanced filtering, time management systems and informing to answer the where, why, and how of your network's internet activity. Although it is just a start-up project, its concept and prototype device are quite promising in the promote video on the kickstarter. The main and unique functionalities Circle has are time management capabilities, device and application notifications, safe, pause and bedtime modes and cost effective for the system.

The prototype of this report is using RPI as the same function as Circle's wireless router. And this prototype system has server side back-end to store client and administrator user database. There are android and IOS applications both work with the internet control system in this project.

At application side, the prototype application of this report will have the same administrator function to approve and block the client internet access request through http request.

For the notification function of the system, the project of this report has the similar idea with another kickstarter project 'NINJA SPHERE'[oYE13]. The idea of NINJA SPHERE is to make the next generation control of your environment with accurate in-home location data and a gesture control interface. Although the project of this report will not cover the advanced way to control the environment of the residential area only the internet access control of the residential area, the idea is

still the same to use mobile application to communicate with the other device and even get notification from other device in the same wireless network area.

The notification of the client request in this project will be sent as notification E-mail. It makes the administrator get updated request information from the internet access control system.

1.3 Scope

The first part of this project will be using the RPI device and the code script from previous student master project report[Coo13b] and previous student Github repository[Coo13a] to set up the internet access control system working. Because the RPI device and Secure Digital (SD) card got from the previous student are without any code and configuration, they should be configured with the reference of previous student master project report.

The second part of this project will be setting up the central management server on the test domain 'apc.item.ntnu.no'(129.241.200.170) from Telematics (ITEM) department of Norwegian University of Science and Technology (NTNU). The work of this report will cover some security concern improvement and some new notification mechanism implement on the remote central management server. And the database structure of the system stored on the remote server would be changed according to the new functionality of the improved internet access control system.

The third part of this project will be implementation of the IOS application intended for end-user to manage and control clients' internet access. The application would be implemented under the IOS 7 Software Development Kit (SDK) and Xcode[Xco13] 5.0 Integrated Development Environment (IDE) on the Macintosh Operating System (MAC OS) 10.8.5 working environment. Since there will be some changes for the central management server, then the android application need to be modified to work with the new back-end server. The changes will be made under Android Development Tools (ADT)[Plu13] 22.3.0 IDE. These two platform application will be tested against central management server and RPI device to make sure all the basic administrator function working well.

The fourth part of this project will be research about how to make the current internet control system more safe and how to implement more advanced internet control function in the current internet control system. The research would be based on articles and some demo testing script using in the current working internet control system.

1.4 Report Structure

In the System Description chapter, it will cover the general information about the previous internet access control system and the improvement form this report project to the current system. In this chapter, some background knowledge of the previous master project[Coo13b] would be mentioned as well.

In the RPI Setting chapter,the main content about the progress to set up RPI internet access control will be presented. Some related modification for previous master project would be mentioned in the chapter. And some background research would be including in this chapter to analyze the performance of the current internet access control system

In the Central Management Server Improvement chapter would have some detail improved code snippet to be discussed why the previous prototype project need to be improved in this way. And it will show database structure changes on the back-end server as well.

In the Mobile Application Development chapter, it will present the basic working process of the application development for this prototype project and some test case to work with the other system components in this internet access control system. The main content of this chapter would be about IOS application development, but also it will have some modification explanation for android application.

In the System Testing chapter, it will show the feedback and analysis from the testing of the prototype project. The analysis will have some future improvement suggestion for later work since there are not enough time to implement the solution to some testing cases.

In the Future Work chapter, it will present some better solution for current project to do the internet access control which can not be implement within such short period of this project working time. And there will some exploring point for the project based on the research of the technical articles.

Chapter 2

System Description

2.1 Existing Internet Access Control System

This project is based on the existing internet access control system. The existing system provides a managed service for the administrator of the system. The Figure 2.1 shows the architecture of the existing system. In the system the manageable residential access point is RPI which has the function as wireless access point and router. The setting progress and changes of this report project will be covered in the chapter 3. The management server in this architecture is host on the remote server to communicate with the other component in the system. It stores system database and provides Hypertext Transfer Protocol (HTTP) communication interfaces. The mobile management application and web management application have the same function in the system which is allowed administrator to manage and control the whole system in more mobile and flexible way. The applications for mobile platform covers IOS and android mobile operating system, since they are main two mobile operating system in the current smart phone shared market. Due to the time limit of his project, the web application development will not be covered in this report although it is easier to make and the working process is the same as other two platform application. The main function of the mobile applications is to manage the client user internet access request. The more detail about this will be covered in the chapter 5.

2.2 Improvement of Existing System

According to the master project report [Coo13b], there are several function requirements mentioned in that report. In this project, the improvement functions of the system shows in the Table

Table 2.1: : Improvement Functions for System

No.	Title	Improvement Function	Importance
1	Block Internet Access	The residential access point can block clients internet access base on MAC and forbid the Internet Protocol (IP) address request	High
2	Bug Fixed for static IP lease	Bug found in the report with wrong script format in static IP lease for residential access point	High
3	Bug Fixed for Missing script	Bug found in the report without essential script file	High
4	Broken SD card changed	Changed broken SD card in the project	High
5	Security Log on service	Implement security log on mechanism for mobile log on request	High
6	Complete Authenticate Log on service	Complete implementation of authenticate log on by user-name and password	High
7	Separate working protocol	Separate client request protocol and mobile management application working protocol on central management server	Medium
8	Set up Database Management Tool	Set up phpmyadmin[Wp13] to manage database on the central management server	High
9	E-mail Notification Mechanism	Implement E-mail notification mechanism on central management server to notify the client user internet access request	High
10	Security Log on	Implement security log on function on both android and IOS mobile application	High
11	Approve and Block Internet access	IOS Mobile Application can approve and block the client internet access request	High
12	Real-time update	IOS Mobile Application can real-time update all the client internet access request	Medium

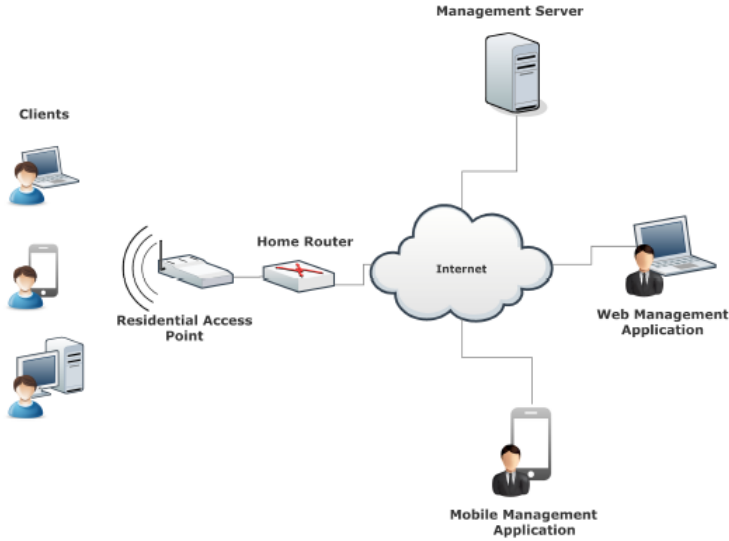


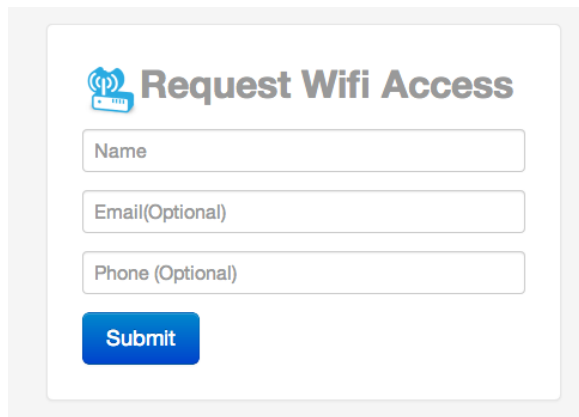
Figure 2.1: Existing System Architecture

2.3 Analysis of System

The system is based on client-server model [mod13], which is a distributed application structure in computing that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Since in the internet access system, the request client users connect with the residential access point (in this project will be RPI), so it is better for the service to provide client-server model communication between different client users and residential access point.

Moreover, there will be different residential access point from different residential areas to connect with the same service provider central management server since the central management server will provide the server back-end service and store the database with information for administrator authentication, different residential Media Access Control (MAC) address and other necessary service data. The main database of the system should be able to be managed and modified by the administrators. Then the client-server model also would be quite fit into this scenario.

According to the communication between mobile application clients and central management server, the client-server model of this architecture is better for central management server to response of the HTTP request and manage the whole service



Request Wifi Access

Name

Email(Optional)

Phone (Optional)

Submit

Figure 2.2: Request Client User Web Request Page

of the system.

More detail about the different component application architecture would be covered in the individual chapter 3, chapter 4 and chapter 5.

Because there is no running system from previous student master project when this project begin, the performance between previous system and current improvement system would not be compared during this report. However, different improvement of the internet access system would be discussed in the later chapters. And all the improvement function would be only based on the master project report[Coo13b].

2.4 User Case

Normal user case for this prototype system will be discussed in this section. There are two children in Adam's family. One is 19-year-old daughter Stella and the other is 15-year-old son Aslak. Usually Adam and his wife Eva live with their younger son, they are using the same wireless network based on internet access control system in their house. However, Aslak likes to play online game too much and sleep very late. Then everyday around 2200, Adam will use his WifiAccess Manager application on his iPhone to block his son's computer internet access to make sure Aslak could sleep at the acceptable time everyday. Some weekend, his daughter come back to their house to enjoy the weekend with them. Stella will make her computer and mobile phone connect with the wireless network, then she use the browser to type any domain name in the address bar, she will be redirected to the same central management server to request internet access like the Figure 2.2. Then she just type her name for the current device to make request. After she made request, there will be a notification e-mail send to Adam's phone since his administrator user on

the WifiAccess Manager application is registered with this e-mail address. Then Adam will use the ios application to approve this request, after that his daughter can connect with the internet this weekend.

More working process will be discussed in the Chapter 6. There will be more detail system testing result in that Chapter.

Chapter 3

Raspberry Pi Setting

3.1 About Raspberry Pi

The RPI is a credit-card-sized single-board computer developed in the UK by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in the schools. [Pi13] The RPI has a Broadcom BCM2835 system on a chip, which includes an ARM1176JZF-S 700 MHz processor (The firmware includes a number of "Turbo" modes so that the user can attempt overclocking, up to 1 GHz, without affecting the warranty), VideoCore IV GPU, and was originally shipped with 256 megabytes of RAM, later upgrade to 512 MB. It does not include a built-in hard disk or solid-state drive, but uses an SD card for booting and long-term storage. The Foundation's goal was to offer two versions.

3.1.1 RPI Hardware for Project

In this project, one Kingston 8 Gigabyte (GB) SD card is used instead of previous Samsung 16 GB SD card in the master project[Coo13b] because the Samsung 16 GB SD card used in the previous project is quite unstable since its root file system has been broken by the voltage changes of the Universal Serial Bus (USB) hub on the RPI during the development of this report project.

In this project, it will use model B as the residential access point because the model B is more powerful on the hardware than the model A, and it has two built in integrated 3-port USB hub. Since the residential access point need provide the Wifi hot-spot function for other client user to connect to, and the normal RPI is not equited with on-board Wifi block component, a compatible D-link DWL-G122 would be used with RPI by the USB connecting.

RASPBERRY PI MODEL B

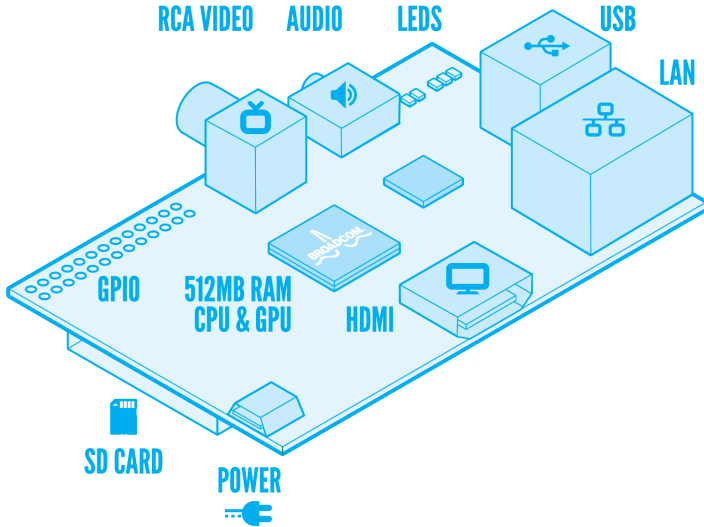


Figure 3.1: Raspberry Pi used as Residential access point

3.1.2 RPI Operation System

The Raspberry Pi Foundation provides Debian and Arch Linux ARM distributions as RPI operating system. Tools are available for Python as the main programming language on the RPI, with support for BBC BASIC (via the RISC OS image or the "Brandy Basic" clone for Linux), C and Perl. In this project the recommended operating system, Raspbian wheezy[Ras13] will be used as RPI operating system. Raspbian is a free operating system based on Debian optimized for the Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make Raspberry Pi run. However, Raspbian provides more than a pure OS: it comes with over 35,000 packages, pre-compiled software bundled in a nice format for easy installation on Raspberry Pi. For this prototype project, this kind of beginner operating system is best choice.

3.2 Applications Using on Raspberry Pi

3.2.1 hostapd

Most using package to provide Wifi hot-spot function on RPI is hostapd[hL13] package. hostapd is an Institute of Electrical and Electronics Engineers (IEEE) 802.11 Access Point (AP) and IEEE 802.1X/Wi-Fi Protected Access (WPA)/Wi-Fi Protected Access 2 (WPA2)/Extensible Authentication Protocol (EAP)/Remote Authentication Dial In User Service (RADIUS) Authenticator. The advantages of using hostapd are that it is compatible well with the RPI and it is easy to manually modified by script configuration file. The set up configuration file shows in Code Snippet3.1.

Code Snippet 3.1 hostapd configuration file

```
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
```

3.2.2 Dnsmasq

For dynamically allocating IP address for connected clients with RPI, the protocol, Dynamic Host Configuration Protocol (DHCP) [Pro13c] is using in this project. DHCP is a standardized networking protocol used on IP addresses and other information that is needed for internet communication. DHCP allows computers and other devices to receive an IP address automatically from a central DHCP server(RPI in this project), reducing the need for a network administrator or a user from having to configure these settings manually. This working process is fit the requirement of prototype project because client users need have a IP address to have the access to post internet access request to the central server then wait for the response of it. And the recommended networking protocol using in this case from the RPI community is DHCP.

In this project, on RPI device, the application named dnsmasq [Dns13] will be used to provide Domain Name System (DNS) forwarder and DHCP server. For this

prototype project, dnsmasq is a lightweight, easy to configure DNS forwarder and DHCP server. It is designed to provide DNS and, optionally, DHCP, to a small network, like the residential area wireless network in this report case.

Code Snippet 3.2 dnsmasq configuration

```
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-range=auth,10.0.0.2,10.0.0.63,2h
dhcp-option-force=auth,1,255.255.255.192
dhcp-option-force=auth,6,8.8.8.8,8.8.4.4

dhcp-hostsfile=/etc/dnsmasq.hosts
```

The configuration file for dnsmasq is shown in Code Snippet 3.2. For this project, we set unauthenticated IP address in the range from 10.0.0.65 to 10.0.0.94, the the lease available time is two minutes. The other hand, the authenticated IP address are in the range from 10.0.0.2 to 10.0.0.62. And for unauthenticated clients, the DNS server would be 129.241.200.170 which is central management server to redirect network traffic for each unauthenticated IP address. For authenticated clients, the DNS server would be the Google Open DNS server to reduce the complexity of the RPI residential access point. Moreover, we set the hosts file to '/etc/dnsmasq.hosts'. this file will store the static lease script. The example script in dnsmasq.hosts would be like Code Snippet 3.4, it includes IP address leased by the client user , MAC address from the client user device and the IP lease available time. The dnsmasq hosts file would be modified when there is any changes from the central management server updating the client user authorization.

Code Snippet 3.3 dnsmasq dhcp host format

```
--dhcp-host=[<hwaddr>][,<ipaddr>][,<lease_time>][,<ignore>]
```

There is a bug in the previous master project report [Coo13b], which is the format of the IP address lease is not correct when there is an authenticated client need to add to static lease for dnsmasq hosts file. The format of the dhcp host should be the format as Dnsmasq dhcp host format 3.3 according to dnsmasq document[doc13].

Then the bash script for adding static lease in the dnsmasq host file need to be fixed because of this bug. There are two files to place the leases for connecting devices managed by dnsmasq. One is to store the static lease info , dnsmasq.hosts (/etc/dnsmasq.hosts), the other is to store the temporary lease info,dnsmasq.leases (/var/lib/misc/dnsmasq.leases). This project is using both to change the IP address for request clients. This will be discuss more detail in the bash shell scripts in the section3.3.

Code Snippet 3.4 dnsmasq hosts file

```
e4:ce:8f:03:7f:e0,10.0.0.6,2h
```

3.2.3 Iptables

After user client gets the IP address from the residential access point, the residential access point need provide a way for user client to get the authenticated IP and set the authenticated IP with the new routing policy. In this project, iptables[Wi13] is used as this purpose.Iptables is a user space application program that allows a system administrator to configure the tables provided by the Linux kernel firewall and the chains and rules it stores. Different kernel modules and programs are currently used for different protocols; iptables applies to IPv4, ip6tables to IPv6, arptables to Address Resolution Protocol (ARP), and ebtables to Ethernet frames.

When a IP packet arives at the residential access point (RPI), it is sequentially checked against the rules in the 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.

3.3 Scripts on Raspberry Pi

On RPI residential access point, the main function to manipulate Iptables and configure dnsmasq is based on the shell scripts on the RPI. And the commands are got from the central management server by the python scripts HTTP post request which running repeatedly in 60 milliseconds interval.Also the proxy server hosting is based on python scripts hosting on the port 8089. Because of the time limit, in this project, it will use the same mechanism on RPIas previous master project [Coo13b], other possible solution will be discussed in Chapter 7. However, there are several bugs in the master project report provided, which is fixed in this project to make RPI residential access point working. In this section,we will discuss these fix changes, the original and correct part from previous master project is not included.

3.3.1 RPI Configuration Python Script

On the RPI residential access point, RPI need to run two important python scripts to make the residential access point work in the whole internet access control system. One is transproxy.py script which is the same transproxy.py python script on the <https://github.com/TorgeirCook/RPIConfigurationServer/blob/master/transproxy.py>, it is developed by Torgeir Pedersen Cook in his master project. This python script is to host the proxy server on port 8089 of RPI to make sure all the redirected network traffic will reach the remote central management server for unauthenticated IP address client user to request internet access.

Another script running on RPI is configserver.py script, the main function getClientPermissions function is shown in Code Snippet A.2 of Appendix A. In this function, it will do the HTTP post request with the current RPI residential access point MAC address to get the correct update client request list in the response returned. Then it will determine the client configuration function by the response result array. The request data list is a JavaScript Object Notation (JSON) array with the different client info JSON object. According to the value of status of the permission parameter in client JSON object, this function will call the right method to manipulate the bash shell scripts. Moreover, the timer for HTTP post request is set in configserver.py script as well, then this main configuration process will be triggered once 60 milliseconds to make sure the access control of the internet is up to date.

Code Snippet 3.5 timer method in configserver.py

```
threading.Timer(60.0, self.sendBindingUpdate).start()
```

In the original scripts from the previous master project, it only call the removeClient and addClient functions in clienthandler.py script (it is shown in Code Snippet A.3 in A, will be discussed more in later) every time for the updated client request devices. Then it did not provide the way to block the client request devices. But in this project, the blockClient function is implemented in clienthandler.py, then it will determine the correct function to approve the request client or to block the request client.

Furthermore, in the previous master project configserver.py script, the shell script reload_dhcp_leases.sh is not working correctly with this python script. So the fixed version of reload_dhcp_leases.sh is implemented in this project along with the corresponding changes in configserver.py script. The content of reload_dhcp_leases.sh script will be discussed in later section.

3.3.2 RPI Client Handler Python Script

Once the response got from the central management server, the corresponding method in `clienthandler.py` shown in Code Snippet A.3 will be called. In addition to the previous master project, the `blockClient` function is implemented, it takes key parameter to block the right MAC address in the lease host file of `dnsmasq` by the shell script `block_static_lease.sh`. And because the possibility of the connected device using the unknown IP address for the client list in `clienthandler.py` (maybe something blocks in the running script `configserver.py` then not register the IP address in client list), so the safe solution in `removeClient` function to clear the DHCP lease file every time for the specific MAC address device. That's reason to have two options in the `removeClient` function to make sure clear the lease every time need to remove client no matter if the client is in the client list or not.

3.3.3 RPI Bash Shell Scripts

There are five shell scripts using in the RPI residential access point. The `iptables_setup.sh` scripts in Code Snippet A.4 in Appendix A is used for setup the iptables in the initial state. On the RPI, the configuration for network interface settings is shown in Code Snippet A.1 in Appendix A which is the same from previous master project. This script will not be discussed in this report since it has not been changed.

The `add_static_lease.sh` script is shown in Code Snippet 3.7, it requires three different input (`$1,$2,$3`), the first input is request client MAC address, the second input is the new IP address and the third input is the lease available time for DHCP. It is used to add new lease in the `dnsmasq.host` to make this MAC address device with the static lease in DHCP. In the code from previous master project, this shell script shown in Code Snippet 3.6 is not correct since it only has one option to add not existing MAC address IP address lease, then there is no way to change the current using lease to different IP address or block this MAC address client device. The improved version is implemented in this project to solve this problem in Code Snippet 3.7.

The `block_static_lease.sh` script shown in Code Snippet 3.8, it is the new function for the previous master project. This script requires one input which is request device MAC address to make the DHCP to ignore this device to require IP address from the current residential wireless network. By using this ignore mechanism in `dnsmasq` application, the blocked device will not get IP address anymore. The reason to block device in this way because usually in the public wireless network, it is reasonable for normal user to require the internet access for some time period using, after certain agreed time, the client device can not access internet any more. But in the residential network, more usual case is that the administrator does not want this blocked device

Code Snippet 3.6 Original 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
```

Code Snippet 3.7 Improved add_static_lease.sh

```
#Add lease to the DHCP lease file of dnsmasq
if ! grep -q $1 /etc/dnsmasq.hosts; then
    #line= grep -q $1 /var/lib/misc/dnsmasq.leases
    dhcphost="$1,$2,$3"
    echo $dhcphost >> /etc/dnsmasq.hosts
    echo "lease added to dnsmasq.hosts: $dhcphost"
else
    echo "dnsmasq.hosts already contains a lease with mac: $1"
    sed -i "\|$1|d" /etc/dnsmasq.hosts
    dhcphost="$1,$2,$3"
    echo $dhcphost >> /etc/dnsmasq.hosts
    echo "lease added to dnsmasq.hosts: $dhcphost"
fi
```

to continue have the IP address in the network since there are not many IP addressed available in such small network, and also more safe way to keep this blocked device can not harm the current residential network anymore because it is used in private residential using, the main point of this kind network is to be managed and observed by administrator not to be free and public for every connect device like the public network.

The reload_dhcp_leases.sh script shown in Code Snippet 3.10 is to reload the dnsmasq to make the new configuration in the dnsmasq.hosts working and remove the current temporary IP address lease. It take one input as MAC address in the lease file to delete corresponding lease information in dnsmasq.leases(the temporary

Code Snippet 3.8 block_static_lease.sh

```
#Block mac address to the DHCP lease file of dnsmasq

dhcphost="$1,ignore"
echo $dhcphost >> /etc/dnsmasq.hosts
echo "blocked mac address added to dnsmasq.hosts: $dhcphost"
```

lease file for dnsmasq). The reason to remove the temporary IP address for the specific device is to force the device to update its new IP address to make sure the access control changing work as quickly as possible. This delete function is not in the Code Snippet 3.9 from previous master project, it is introduced in this project to solve the problem mentioned before.

Code Snippet 3.9 Original 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"
```

Code Snippet 3.10 Improved 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"

sed -i "\|$1|d" /var/lib/misc/dnsmasq.leases
echo "remove current lease to require new lease"
```

The other shell scripts using on RPI residential access point is clear_dhcp_lease.sh file. This file is missing in the previous master project report. It is implemented like Code Snippet 3.11. The function of it is to remove the one corresponding static lease information in dnsmasq.hosts for specific device MAC address. Then this device will have to use the temporary lease which requested again from the RPI residential

access point. It will release the current using authenticated IP address for other new approved device.

Code Snippet 3.11 clear_dhcp_lease.sh

```
#Remove lease from the DHCP lease file of dnsmasq

sed -i "\|$1|d" /etc/dnsmasq.leases
```

These four shell scripts discussed before is used by the in clienthandler.py shown in Code Snippet A.3 to change the DHCP by dnsmasq application. After the lease changes from these shell scripts, there will be corresponding iptables manipulates process in Code Snippet A.5 in Appendix A. This part is implemented in the previous master project, so it will not be discussed in this project report. The main function of it is to remove the corresponding iptable rules for specific IP address and add the new corresponding iptable rule to make the this new authenticated IP address have the internet access control not redirect all the network traffic to the remote central management server any more.

Chapter 4

Central Management Server Improvement

Chapter 5

Mobile Application Development

Chapter 6

System Testing

Chapter 7

Future Work

References

- [Cir13a] Kickstarter-Meet Circle. <http://www.kickstarter.com/projects/304157069/meet-circle>, 2013.
- [Cir13b] Meet Circle. <http://meetcircle.co/>, 2013.
- [Coo13a] Github-Torgeir Pedersen Cook. <https://github.com/torgeircook>, 2013.
- [Coo13b] Torgeir Pedersen Cook. Internet control for residential users. diploma thesis, Norwegian University of Science and Technology, September 2013.
- [Dns13] Dnsmasq. <http://www.thekelleys.org.uk/dnsmasq/doc.html>, 2013.
- [doc13] DNSMASQ documentation. <http://www.thekelleys.org.uk/dnsmasq/docs/dnsmasq-man.html>, 2013.
- [Fam13] Norton Family. <https://onlinefamily.norton.com/familysafety/loginstart.fs>, 2013.
- [hL13] hostapd Linux. <http://wireless.kernel.org/en/users/documentation/hostapd>, 2013.
- [McG05] Jim McGuigan. Towards a sociology of the mobile phone. *Human Technology*, 1(1):45–57, 2005.
- [mod13] Wikipedia-Client-server model. http://en.wikipedia.org/wiki/client-server_model, 2013.
- [oYE13] NINJA SPHERE: Next Generation Control of Your Environment. <http://www.kickstarter.com/projects/ninja/ninja-sphere-next-generation-control-of-your-envir?ref=users>, 2013.
- [Pi13] Wikipedia-Raspberry Pi. http://en.wikipedia.org/wiki/raspberry_pi, 2013.
- [Plu13] ADT Plugin. <http://developer.android.com/tools/sdk/eclipse-adt.html>, 2013.
- [Pro13a] Emil Protalinski. Idc: Android hit 81.0% smartphone share in q3 2013, ios fell to 12.9%, windows phone took 3.6%, blackberry at 1.7%, 2013.
- [Pro13b] K9 Web Protection. <http://www1.k9webprotection.com/>, 2013.

- [Pro13c] Wikipedia-Dynamic Host Configuration Protocol. http://en.wikipedia.org/wiki/dynamic_host_configuration_protocol, 2013.
- [Ras13] Raspbian. <http://www.raspbian.org/>, 2013.
- [Sol13] OpenDNS Parental Control Solutions. <http://www.opendns.com/home-solutions/parental-controls/>, 2013.
- [Wi13] Wikipedia-iptables. <http://en.wikipedia.org/wiki/iptables>, 2013.
- [Wp13] Wikipedia-phpMyAdmin. <http://en.wikipedia.org/wiki/phpmyadmin>, 2013.
- [Xco13] Xcode. <https://developer.apple.com/technologies/tools/>, 2013.

Appendix

Appendix

A.1 RPI Residential Access Point Scripts

Code Snippet A.1 network interface configuration file

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

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

post-up /etc/network/if-up.d/iptables_setup.sh
```

Code Snippet A.2 getClientPermissions function in configserver.py file

```

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", 8080)
    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']
                status =
                client_json ['client'] ['permissions'] ['access'] ['allow']

                if status == True:
                    self.clientlist.removeClient(key)
                    self.clientlist.addClient(client_json)
                else:
                    self.clientlist.blockClient(key)

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

    httpServ.close()

```

Code Snippet A.3 main functions in clienthandler.py file

```

def addClient(self, client_json):
    ip = self.ipaddress_pool.pop();
    client = Client(client_json, ip)

    # Add static lease to DHCP lease file (mac, ip, lease time)
    add_lease_cmd =
        'bash /etc/add_static_lease.sh %s %s 2m' % (client.mac, client.ip)

    iptablesapi.executeCommand(add_lease_cmd)
    self.clientlist[client.mac] = client

def blockClient(self, key):

self.removeClient(key);

block_mac_cmd = 'bash /etc/block_static_lease.sh %s' % key
iptablesapi.executeCommand(block_mac_cmd)

def removeClient(self, key):

    if self.clientlist.has_key(key):
        client = self.clientlist[key]
        ip = client.ip
        self.ipaddress_pool.append(ip)

        #Remove static lease from lease file (mac)
        clear_lease_cmd =
            'bash /etc/clear_dhcp_lease.sh %s' % client.mac

        iptablesapi.executeCommand(clear_lease_cmd)
        #Remove chains and rules from iptables
        client.deleteInitialChainAndRules()
        del self.clientlist[key]
    else:
clear_lease_cmd =
'bash /etc/clear_dhcp_lease.sh %s' % key

iptablesapi.executeCommand(clear_lease_cmd)

def fillIPAddressPool(self):
    #Range for authorized subnet
    for x in xrange(60, 5, -1):
        ip = '10.0.0.%s' % x
        self.ipaddress_pool.append(ip)

```

Code Snippet A.4 iptables_setup.sh

```
#DNS server of unauthorized 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
```

Code Snippet A.5 main functions in iptablesapi.py

```
#!/usr/bin/python
import subprocess

def createNewClientChain(chain_name):
    new_chain_cmd = 'iptables -N %s' % chain_name
    executeCommand(new_chain_cmd)

def deleteClientChain(chain_name):
    delete_chain_cmd = 'iptables -X %s' % chain_name
    executeCommand(delete_chain_cmd)

def jumpFromBuiltInChainRule(builtin_chain_name, custom_chain_name, client_ip):
    jump_rule_cmd1 =
    'iptables -I %s -s %s -j %s'
    % (builtin_chain_name, client_ip, custom_chain_name)
    jump_rule_cmd2 =
    'iptables -I %s -d %s -j %s'
    % (builtin_chain_name, client_ip, custom_chain_name)
    executeCommand(jump_rule_cmd1)
    executeCommand(jump_rule_cmd2)

def deleteJumpFromBuiltInChainRule(builtin_chain_name,
custom_chain_name, client_ip):
    #delete_jump_rule_cmd =
    'iptables -D %s -m mac --mac-source %s -j %s'
    % (builtin_chain_name, client_mac, custom_chain_name)
    delete_jump_rule_cmd1 =
    'iptables -D %s -s %s -j %s'
    % (builtin_chain_name, client_ip, custom_chain_name)
    delete_jump_rule_cmd2 =
    'iptables -D %s -d %s -j %s'
    % (builtin_chain_name, client_ip, custom_chain_name)
    executeCommand(delete_jump_rule_cmd1)
    executeCommand(delete_jump_rule_cmd2)

def acceptAllTrafficFromClient(chain_name):
    accept_traffic_cmd = 'iptables -I %s -j ACCEPT' % chain_name
    delete_drop_rule_cmd = 'iptables -D %s -j DROP' % chain_name
    executeCommand(accept_traffic_cmd)
    executeCommand(delete_drop_rule_cmd)

def blockAllTrafficFromClient(chain_name):
    block_traffic_cmd = 'iptables -I %s -j DROP' % chain_name
    delete_accept_rule_cmd = 'iptables -D %s -j ACCEPT' % chain_name
    executeCommand(block_traffic_cmd)
    executeCommand(delete_accept_rule_cmd)

def initialSetup():
    setup_cmd = 'sh /etc/network/if-up.d/iptables_setup.sh'
    executeCommand(setup_cmd)
```