Stoyan Stoyanov

Integrating online social networks with virtual public networks

Diploma in Computer Science

Robinson College

April 21, 2014

Proforma

Name:	**Stoyan Stoyanov**
College:	**Robinson College**
Project Title:	**Integrating Online Social Networks with Vpun**
Examination:	**Diploma in Computer Science, July 2014**
Word Count:	**TODO[1] (well less than the 12000 limit)**
Project Originator:	Dr. Arjuna Sathiaseelan
Supervisor:	Dr. Arjuna Sathiaseelan

Original aims of the project

To show that social media can be used to make intelligent traffic engineering decisions and to create an open wireless network which access point can be integrated with input from the social media. The dissertation should illustrate how software defined networking(SDN) can be used for that purpose, how an appropriate controller for the network can be developed and how the hardware of the router should be configured. Finally, it aimed to explore the performance differences with a regular wireless network.

Work completed

All of the above work has been completed and presented in the dissertation. The social media that was chosen for integration with SDN is Facebook. The firmware of the router was flashed and configured, a controller was developed and so was a front page allowing the users to connect. Finally, functional testing was carried on and further evaluation of the project was conducted.

[1]This word count was computed by `detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w`

Special difficulties

None.

Declaration

I, Stoyan Stoyanov of Robinson College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed [signature]

Date [date]

Contents

List of Figures

Acknowledgements

Chapter 1

Introduction

In most cases today, all of the devices connected to a computer network attempt to share the resources provided by that network fairly. However, in many situations, we would prefer to possess some notion of priority to these resources and the ability to create more complex sharing policies uniquely tailored to meet the needs of a given network. For example, a coffee shop which offers Wi-Fi access free of charge to all of its customers could offer higher bandwidth capacity for its loyal customers who have “liked” the coffee shop’s Facebook page.

Another desirable feature for networks may be a means of regulating how or when certain users can operate within the network. An example of this could be a private network owned by a family in which the parents set a limit so that their children cannot connect to the home router after 10 P.M. but regain full-speed access during the day to complete their homework assignments.

As wireless Internet access becomes more and more ubiquitous, there will be a higher demand from network owners to be able to manage their networks in more sophisticated ways. One possible solution is to utilise data available from social media networks to create sharing policies, which is the approach taken in this project.

1.1 Aim of the project

The aim of this dissertation is to show that information extracted from social media can be used to make intelligent traffic engineering decisions. In particular, this dissertation describes the design and implementation of a Virtual Public Network (VPuN) [1] which intelligently regulates how and to whom access to network resources is granted based on information provided by social media networks about the users' identification and the users' relationship with the owner of the network.

VPuNs are home networks which are created, deployed, and managed by an evolutionary software-defined networking control abstraction. I provide further detail about the theoretical foundations of this project in the following section and throughout the remainder of this dissertation.

1.2 Theoretical foundations

This project is largely based on the theoretical concepts of software-defined networking (SDN). SDN is an approach to computer networking that originated from research done at Stanford University and the University of California, Berkeley in 2008 [5]. The basic idea of SDN is to decouple the system that makes decisions about how to direct data traffic from the system that performs the task of physically forwarding the data to the desired destination (i.e., splits the data and control plane of a router). SDN is believed to simplify networking because the devices need not understand nor process numerous protocol standards but merely need to follow simple instructions from the SDN controller.

Furthermore, this project was directly inspired by recent work carried out by the Cambridge University Computer Laboratory which shows how software-defined networking has the capability to integrate big data into the decision-making process for how to direct traffic through a network [1]. This paper provided the principal foundation for how the goals of this project could be achieved.

1.3 Similar work

After extensive research, I did not discover any other work with the same core functionality as this project. One recent project created by Fon [2] allows users to log into a computer network using their Facebook credentials. However, the project does not aim to support any procedures for a network owner to create a customised sharing policy or control how capacity is granted to connected users.

Chapter 2

Preparation

This chapter discusses the work completed before beginning the actual implementation of the VPuN. The chapter starts out by providing a list of project requirements and then goes on to justify each of these requirements and the implications they have for the remainder of the implementation, including the various tools and programming languages employed. The chapter ends with a brief discussion on the software development model which was adopted to help structure the implementation process and to manage potential risks.

2.1 Requirements analysis

After careful analysis, I concluded that the project must fulfil the following requirements:

1. The system must integrate data from a suitable source of social media to influence the traffic decisions.
2. The system must use SDN concepts to control the flow of data traffic.
3. The system must allow authorised users to access the network and adhere to the policies enforced by the network owner.
4. The system should provide a simple and intuitive interface for the users.
5. The system should offer quality of service.
6. The system should be reasonably secure.

In the following sections, I present the reasons for these requirements and how they influence the project.

2.1.1 Social media integration

The requirement to integrate data from a social media source to guide the router in making traffic decisions embodies the principal aim of the project. Consequently, one of the most crucial decisions of the project was the choice social media source. This choice would not significantly affect the overall architecture of the project; it would, however, affect the probability that a network owner would elect to deploy such a scheme for their private network. Therefore, desirable qualities in candidate social media networks include a high probability that any given user would have an account with this social media network and good usability (i.e., high ease of use and learnability).

Facebook was a clear frontrunner in this selection as it is the leading social media network today with the greatest number of users by a large margin. Even more, users are fairly accustomed to signing up and logging into other applications using their Facebook credentials, which is exactly the process required for this project.

After appointing Facebook as the social media source for the project, the next imperative decision was which data to pull from Facebook and how to control network traffic based on these data. Observing the structure of Facebook led to the idea of creating sharing policies based on the concept of social relationships, or "friendships" in Facebook, between network owners and other users of the network. I believe that the classes of relationships represented in Facebook (e.g., "close friends," "family," or "acquaintances" classes) are very intuitive for users and that users would be able to easily translate from this Facebook classification scheme to the concept of different classes of users within a network which may receive different resource allocations. By way of example, a group of people who are members of the network owner's "close friends" group on Facebook may receive a higher bandwidth on the owner's network than users who are not "friends" of the network owner on Facebook.

Preparation work associated with the choice of Facebook as the social media source for the project included learning the Facebook APIs and mandatory registration of my project with Facebook.

2.1.2 Software-defined networking

The idea to use SDN concepts in the project initially arose from an interest in exploring the latest research done in the exciting and new field of SDN technology. After further research, I believed an SDN approach was the best available approach to achieve the required functionality for this project.

The specification to use SDN was indeed ambitious due to the relative newness of the field. SDN is not included in the curricula of the computer networking courses, and furthermore the web resources related to SDN are quite scarce. Because I needed a very strong understanding of SDN and associated concepts in order to produce a functioning prototype for this project, I spent the vast majority of the preparation phase learning those theories and approaches in networking.

As stated in section 1.2, the mission of SDN is to split the control and data planes in a router. As a consequence of this division, a new requirement arises for some method of communication between the control plane and the data plane. In this project, that communication requirement is solved using a communications protocol called OpenFlow [3]. OpenFlow is the most commonly employed communications protocol and probably the only stable solution at the time of implementing the project.

Following the approach of SDN, the project requires developing an augmented or intelligent SDN control plane to perform the task of processing the data from Facebook along with any additional information provided. The implementation of the SDN control plane, or controller, is one of the cornerstones of this project because it composes more than half of the implementation for the project and is also the most challenging part from a technical perspective. For these reasons, I provide an in-depth discussion about the precise role of the controller and how it operates.

Controller

The job of the SDN controller is to make traffic decisions and then to send these decisions over to the data plane of the router with the help of OpenFlow. The way this works is, all the packets that are received at the router are classified as being part of something called a “flow” based on the values of their header fields. The constraints of a flow can be very specific such as “all UDP packets from source address 141.12.14.1 sent to port number 2014 at destination address 12.123.12.14,” or they can be very broad such as simply “all UDP packets,” for example.

Once a packet arrives at the router, the router checks its flow tables for the highest priority flow to which this packet belongs. In the case that it finds a match, the router takes the appropriate action corresponding to this flow (e.g., forward the packet or drop the packet). However, it is possible that the packet does not fit within any of the specified flow constraints, and so the packet is placed in a default flow and must be sent to the controller to figure out how to handle the packet in this case.

When the controller receives a packet from the router, it must carefully inspect the fields in the packet header and make an intelligent decision about whether a new flow should be created to include this packet and if so, what specific constraints should be defined in this new flow, or whether it is best to simply drop the packet. For instance, you may want to prohibit your friends using your network to download illegal content, and so you can drop all packets originating from untrusted websites.

In the case of this project, the controller bases its decision to either create a new flow or to drop the packet on information it procures from the social media source. Once it reaches its decision, the controller sends instructions back to the router, and from this point on, the router will continue following the same instructions for handling all packets belonging to this new flow without needing to consult the controller, thus avoiding further latency.

One thing worth mentioning is that every flow has a time-to-live field for how long it may stay in a router's flow table. There is an interesting trade-off between assigning a low time-to-live value which may introduce latency into the system and a higher value which may prevent the system from aptly adapting to changing network conditions.

As a final note, this architecture does not require the router to be computationally very powerful; it just needs to be able to quickly match packets to flows and to support the OpenFlow communications protocol.

Location of the controller

I had the option of implementing the controller locally or remotely. If it were to be implemented locally (i.e., in the same local network as the user), it would cause less latency and would simplify communication with the router. Furthermore, the capabilities of the controller to inspect the local network will increase

On the other hand, a controller implemented remotely would offer the opportunity for a single controller to administer multiple networks and remove the complexity the users having to set up the network themselves. I decided on the latter approach in order to make scalability of such networks simpler and cheaper.

Language of the controller

After considerable research, I came across open source controllers written in many different languages including Java, C, C++, and Python. I preferred to work with the Java implementation because I have the most experience with Java. I also took into account that significant effort would be expended learning about other parts of the project such as SDN and OpenFlow, so I decided that the benefits of implementing the controller in a potentially more efficient language were not worth jeopardising the most essential parts of the project. After completing the project evaluation, I believe that this was a good decision as the results show that Java is efficient enough for the purposes of the project.

Hardware

The hardware required modification in order to support SDN. Not all routers support SDN, and very few have SDN installed and configured by default. Therefore, any hardware I chose would more than likely require modification in order to support SDN. In particular, I would need to flash an OpenWRT [6] operating system, an embedded operating system based on the Linux kernel which is optimized for home networks and supports the option to add modules such as OpenFlow.

My project supervisor helped to select the particular model of hardware for the project which is a model that supports OpenWRT.

The firmware provided for the router was fully operational, and I installed it relatively quickly after some research.

2.1.3 Network access and policies

The third requirement encompasses the basic functionality requirement for the system. The network owner decides on a set of resource allocation policies, and in particular, specifies which users shall be permitted access to the network. These specified users must be authorized with the network and designated a percentage of any existing network resources.

2.1.4 Quality of service

The requirement for offering quality of service was a natural extension of the first and third requirement, because it allows the users to enforce policies that are very observable and simple to comprehend. The quality of service provided in this project is the ability to guarantee bounds on the minimum and maximum bandwidth that certain users of the network receive. For instance, there should be a set upper bound on the amount of bandwidth allotted to the guest users of the network at any given time so as to guarantee that the owner may use his or her network without experiencing poor network performance. I consider this requirement very important, as it is a desirable feature for the potential users since bandwidth is often considered as the most important resource of network from the user perspective.

2.1.5 User interface

Because the users are required to input information into a web page form, the project calls for developing a succinct and easy-to-use interface for the users, bearing in mind that most users will not have a technical background.

The user interaction will consist of new users of the network being automatically redirected to the project's login web page. The users will see an option to redirect to the Facebook login page where they can input their profile credentials. There will also be an option on the project login web page to attempt to join the network without providing Facebook credentials. Once a user has obtained access to the network, he or she will be able to navigate outside of the login pages.

2.1.6 Security

The security requirement is included because every system should protect its users to some degree, and the users of the system should be made aware of that exact degree of security. This allows the users to assess the extent to which they should use and trust the system.

The security of the network is split into two stages: providing secure authentication with Facebook and providing a secure wireless local network. Because I chose to use the secure login protocol provided by Facebook, I was able to greatly reduce the amount of time spent on the first stage if I were to have developed my own protocol. In order to use the Facebook protocol, I needed understand how it functions to ensure it would not pose any problems for the new system or compromise the security on the MAC protocol level. I was particularly concerned that authentication with Facebook would fail due to the fact that the router acts as a smart party and eavesdrops. Fortunately, it appeared that this is not an issue.

2.2 Cross-layer optimization

During the preparation phase, it became evident that the project would require using data from the Application layer of the OSI protocol stack model [4] and pass this data to the Network layer. This clearly violates the layer abstraction, and due to the fact that the Internet heavily relies on this abstraction, layer violation must be performed with extreme caution.

I classified this as one of the high-risk parts of the project, and thus I decided it needed to be resolved in the first prototype. Indeed, I encountered many complications, which are described in the Implementation chapter, as I attempted to devise a way to leak information to the lower layer.

2.3 Software development model

I chose to use a spiral development model. I made this decision because the risks in this project were relatively high and trying to prototype the parts that I am not sure about seemed like the most sensible way to discover any unexpected problems with the hardware or with the technologies that I had to learn.

After analysing the requirements of the project and deciding on the development model, I divided the implementation process into four distinct task modules: modifying the hardware and configuring the router, implementing the controller, developing a simple and intuitive user interface for the network owner and other network users, and integrating with Facebook. Each of these modules is presented in the Implementation chapter which follows.

2.4 Summary

This chapter has presented the requirements analysis and preparation for implementing the VPuN system prototype. I have justified the choice of Facebook at the source of social media data and have outlined the duties of the SDN controller and router. Lastly, this chapter set up the software engineering practices which were exercised during the implementation phase.

Chapter 3

Implementation

This chapter describes the implementation portion of the project. I describe, in detail, what was developed and how it was achieved. I explain and justify the particular steps taken to complete the necessary implementation. For the purpose of clarity, I explain the different parts of the project in separate headings. In actual fact, implementation for each of the four parts largely occurred in parallel in order to rapidly produce working end-to-end prototypes and to reduce risk associated with overall system integration as quickly as possible.

3.1 Tools

For the initial tasks of the project which involved configuring the firmware and the network, the only tools available were the standard command line tools. For the bulk of the remaining implementation process, I used the Eclipse integrated development environment. In addition, I utilized the developer tools provided by Google for the Chrome browser for implementing the web development portions of the project. Finally, one last development tool which proved to be particularly valuable is Wireshark. Wireshark provides a comprehensible user interface for intercepting and inspecting network packets, and I recurrently used this tool for debugging and evaluation throughout the course of the project.

Perhaps most importantly, to avoid losing any part of the project or dissertation, I backed up my content using the cloud storage site Dropbox and additionally stored everything on an external hard drive.

In the following Evaluation chapter, I explain the additional tools used to evaluate the performance of the project.

3.2 Architecture overview

In this section I present a high-level overview of the architecture of the implemented system. It consists of a controller, a captive portal, a web page for authentication, and a physical access point. Whenever, a user tries to use the network, the router with the help of the controller checks if the user has been authenticated. If that is the case the router, the user is granted access and capacity based on the sharing policies created by the owner of the network and implemented on the router. If the user is not authenticated, the captive portal redirects him or her to a web site which allows the user to provide access to his or hers Facebook details. Once this is done, the required data from the Social media is extracted and send to the controller, which finishes the authentication process.

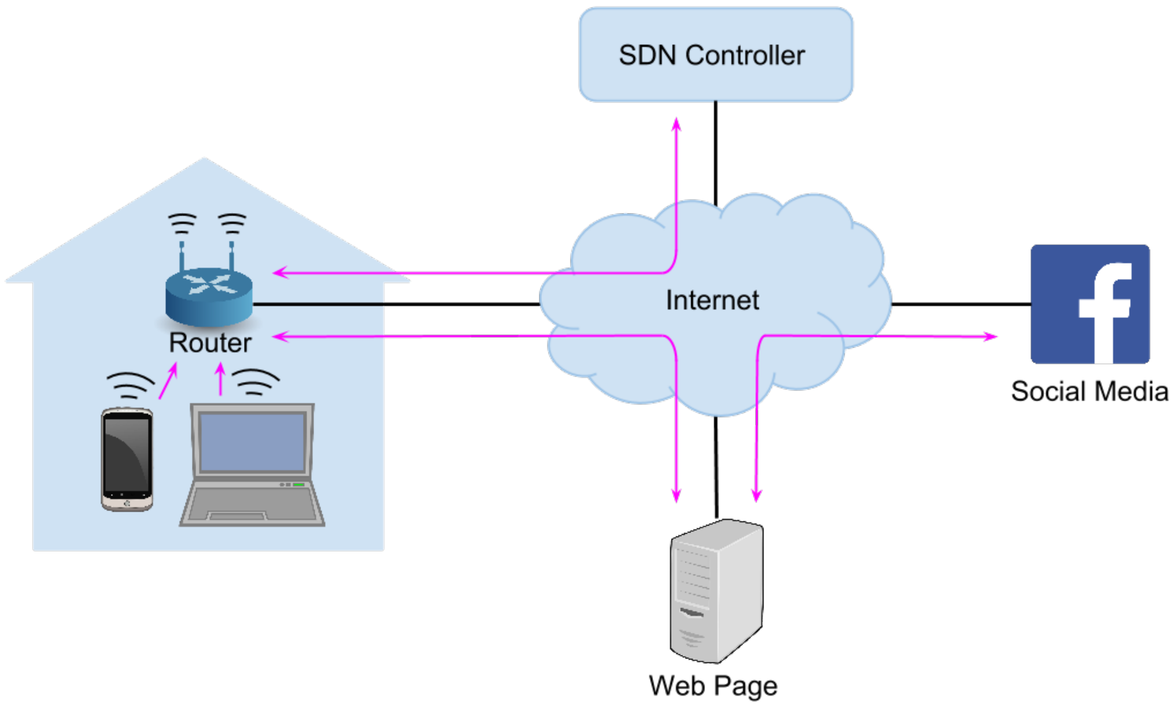

Figure 3.1: Architecture of the project

3.3 Hardware configuration

In this section, I describe the necessary modifications made to the hardware in order to support the functionality for the project. These modifications include flashing the router with the required firmware and setting the special configurations for the network.

As described in the Preparation chapter, I needed to use special firmware for the router because most routers do not support OpenFlow by default. The image of the operating system of the router was provided by my supervisor.

The installation process consisted of first resetting the default settings and connecting to the router through SSH. I then sent the image via atftp, a light protocol used for communications with embedded devices, and restarted the Internet services via the command line.

The next step was to configure the network. For this step, I configured the two main interfaces which the network would support, namely the interface for the owner of the network and the interface for the network guests. I chose to implement the former as a standard password-protected Wi-Fi network. The interface for the guest users, however, needed to be implemented with the support of OpenFlow. This meant that I had to configure the virtual switch on top of physical switch, fortunately OpenFlow allows the creation of virtual ports, bridges, and hosts.

The router had to be administrated as a virtual switch of the network. In order to achieve this, I learned OpenFlow commands such as ovsofctl and ovsvsctl described below which are convenient tools used for communication with the data plane. I also used these commands while implementing the controller. In order to demonstrate what it was used for and how it works I explain some of the commands I used in the lines bellow.

```
ovs-vsctl list Interface
```

This will return various types of information (name, queues belonging to it, link state, link capacity, etc.) for the interfaces supported by the network.

`ovs-vsctl set Interface eth1` $ingress\_policing\_rate = 1000$

The command above will set the ingress policing rate of the Interface eth1 to 1000 kbits per second. This means that the packets coming at the incoming port will start to be dropped if this rate has been exceeded.

3.4 Controller

As described in the preparation chapter, I decided to take advantage of an available open source Java implementation for the base controller, and I ended up electing a controller known as Floodlight. Floodlight is one of the most predominant SDN controllers written in Java and consists of a massive code base which takes care of the meticulous task of managing most of the low-level functionality of the controller such as encoding and decoding the packets, endianness, and processing the headers. It does not offer any higher-level functionality, so I was left with implementing all of the high-level behaviour required for this project.

I began by developing a deep understanding of the structure of Floodlight. Unfortunately, the documentation provided for Floodlight is rather inadequate and even incomplete for many topics, so I ultimately resorted to dissecting the source code. Not only did the source code provide full comprehension of how the controller processes packets and communicates with the router, but it also gave me an idea of what types of errors could be anticipated over the network, what libraries and functionalities were available, and generally how I could efficiently structure and implement my additions to the controller.

Once I understood its key functionality, I began developing on top of Floodlight to achieve the controller functionality required for this project. Recall that the controller receives packets forwarded from the router when the router does not know the correct action to apply to the packet. In other words, the router classifies an arriving packet as belonging to the default flow, which always corresponds to the action of forwarding to the controller. This happens on two occasions: if the packet is the first packet sent from a given device or if the packet belongs to a flow but the entry for that flow has been deleted from the router's flow table after its time-to-live field expired. Therefore, the first functionality required to implement on the controller was to be able to detect arriving packets and track which packets are sent from new devices joining the network. The next step, and perhaps the most complicated procedure of the project, was to match each device to the Facebook ID of that devices owner.

3.4.1 Device tracking

The controller must be able to determine whether an arriving packet is from a device of an already authenticated user or from a device attempting access to the network for the first time because this will govern the subsequent behaviour of the system. All packets sent to or from an unauthenticated device, except those related to the project login page or the Facebook login page, will be dropped, effectively restricting further Internet access for these unauthorized users. Alternatively, if the packet is from a previously authenticated device, the controller must then associate the device with the Facebook credentials of the device owner as described in the following section 3.4.1.

To establish whether or not an incoming packet is from a new device, the controller must somehow identify the sending device of the packet and also keep a list of the devices which have previously sent packets to the router. I chose to uniquely identify each device by its MAC address because typically this value remains the same on a device and also because the probability of two or more devices having conflicting MAC addresses within a single private network is negligible.

The controller stores the previously authenticated MAC addresses in a set structure, which allows for efficient testing for membership. In the case of encountering a new MAC address, the MAC address is stored in a separate set object containing all of the unauthenticated addresses.

In the case of encountering an authenticated MAC address, the controller must determine what flow to create for the packet. Recall that flows in SDN are defined as a set of constraints on the values of a packet's header fields (e.g., protocol, address of the sender, etc.) and an action (e.g., drop, forward, etc.) (Further detail and examples of flows are presented in 2.1.2). Because every link layer packet header includes a source MAC address, I used this value as the main descriptor for the flows. The next step is to determine the action to be matched with the MAC address in the flow, which is where the data from the social network Facebook comes in.

3.4.2 Handling packets from authenticated users

After dropping any unauthorized packets, the controller needs only to consider the packets arriving from previously authenticated devices. To achieve the aims of this dissertation, the system must use information from Facebook to determine what resources to allocate to the authenticated users, and so the controller must be able to match the users with their Facebook identifications. And because network resources can only be allocated to a physical device, the SDN controller specifically needs to be able to associate each Facebook ID with a device in order for the user to gain access to the network.

Ideally, the information about the Facebook ID and the associated MAC address would be directly sent to the controller from the project web page which has all the Facebook information. However, the problem is that the web page does not have the MAC addresses to associate with its Facebook data.

This is because MAC addresses are only used at the link layer of communication and do not necessarily persist over the entire network. Once a packet has left its local network, there is no way to identify its MAC address because when a packet reaches its network gateway and is intended to be forwarded to a device outside the original local network, the gateway substitute the packets original source MAC address with the gateways MAC address in the packet header. Clearly, after a few address substitutions, it is not possible to recover the original MAC address of the sender because the gateways do not communicate the details of the substitution to other nodes.

In particular, this is a problem because simply examining the source MAC address of any packet does not necessarily supply accurate information about from which physical device the packet originated, and so the web server cannot determine how to direct the traffic. Therefore, there must be extra information provided in the communication with the controller.

I discuss the various approaches which were considered for resolving this problem in the following sections.

Exchange MAC addresses

One possible approach is to perform an explicit exchange of the MAC address, for example, by piggybacking the MAC address of the original sender to a packet send to the remote web page. After this, the remote web server will be able to perform the match and place it in a location that is accessible by the controller. However, there is no straightforward way to achieve this without overly complicating the process for the users of the network. This is so because web browsers have a lot of restrictions for security reasons (in this case, the user might not even be able to authenticate only through the web browser which would have made the project pointless).

Using IP addresses

Another approach is to have the web server identify the owner of the packets by IP address rather than by MAC addresses. However, there are various issues caused by the omnipresence of Network Address Translators (NATs). In an effort to solve the IPv4 address exhaustion problem by reusing IP address spaces, NATs will modify source and destination fields in IP packet headers at network junctions which means the controller runs into a problem similar to the one mentioned above with MAC address translation. Any packet arriving at the remote login web page has a sender address which has potentially gone through several NATs and thus cannot be recovered. Of course, the web page can send back the id, but this means that the controller will need to inspect the data content of the packet which will introduce massive overhead during the authentication process rendering this solution impractical.

Another dilemma is that most networks use DHCP for distributing their IP addresses and this can cause collisions at the controller. For example, one device could be authenticated with IP address 192.168.142.100 and a couple days later a new device joins the network and receives the same address assignments so it also obtains access to the network even though it has not authenticated. Solving this issue would require implementing some synchronisation method between the lease times of the DHCP and controller's configurations. Overall, this approach was deemed infeasible and was not ultimately selected.

MAC addresses and NATs

The basic solution which was implemented in this project is to have the web page first determine the relationship between the user and the network owner based on Facebook data and then create a way to communicate this class to the controller using port mappings.

During initial setup of the VPuN, the network owner defined a set of custom classes of users (similar concept to a friend list in Facebook or a circle in Google+) as described in 2.1.2. One example of a set of classes could be a separate class for users who are family, close friends, or acquaintances, with each class in that list receiving less bandwidth than the class before it.

The matching of the classes of users with the MAC addresses is then achieved by establishing a mapping between the classes and port numbers (e.g., family could correspond to port number 12001, close friends to port number 12002, etc.). It is clear that there are enough port numbers as the size of the port field is 16 bits which translates to over 65,000 unique port numbers, even without counting the first 1024 port numbers which are reserved for operating system services or those reserved by other applications. The controller advertises this mapping to the web page, so they are both aware of the scheme.

Once the mapping is established, the way it was implemented on low level is by having the device attempt to open a TCP connection to the router at its dedicated port. As soon as the router receives the SYN packet sent from the device, the controller is able to examine the SYN packet's port number field and categorize the packet into the proper user-defined class. At this point, the packet is dropped and the connection is not instantiated, which means that we were able to discover the class of the device only with the exchange of few packets.

In summary, the controller only needs to look at the destination port number in the packet header and the source MAC address to determine the action for the flow. The flow is then added to the router's flow table, and the router can now direct any packets accordingly.

3.5 Captive portal

According to the project specification, a user must authenticate through the project web page before they may be allocated any network resources. To satisfy this requisite, I implemented a captive portal on the router. Captive portals intercept or "capture" web page requests seen by the router and forces the requesting HTTP client to first navigate to a particular web page before gaining access to the Internet.

3.5.1 Special requirements

The captive portal developed for this project is slightly more complicated than a standard one which might be encountered when accessing an Internet connection, for example, at a coffee shop. Normally, captive portals bring users to a web page to view an advertisement from a sponsor or ask the users to read and agree to a set of terms and conditions. For the purposes of this project, however, the captive portal must allow users to authenticate via their Facebook profile credentials.

Another difference is that captive portals most often force users to web pages which are hosted on the same router as the captive portal, and I opted to instead host the authentication web page on a remote server. Hosting the web page remotely takes away the burden on the network owner of hosting the web page themselves. It also improves scalability since one page can serve numerous networks rather than having an extra web page for every network.

Considering the above special requirements for this implementation of a captive portal and the resources which were available, I decided to utilize a basic open source implementation as a starting point and develop additional features on top of it, including linking it to the remote authentication web page developed for the project. The open source captive portal implementation I selected is Nodogsplash because it functions well under OpenWRT, the operating system of the private networks router.

3.5.2 Implementation

The first step was to set the URL to which the captive portal redirects every user before they have been authenticated. In this case, the captive portal redirects users to the projects login page, which in turn redirects to the Facebook login page if the user chooses to log in with his or her Facebook account. Therefore, I first needed to permit the user secure access to Facebook at port 443 (the port for HTTPS). I accomplished this by transmitting a simple DNS request from the router to add the IP address of the Facebook server to a “whitelist” containing the only addresses which are made accessible to unauthenticated users.

At this point, I encountered some complications because, due to the size Facebook and its large number of users, Facebook uses multiple servers to store its content including, for example, items from the style sheet of the login page. Accordingly, loading the content for the Facebook login page requires downloading from various server locations. This means that users not only need access to the server hosting the main login page but also to every other server storing any of the requested objects. I resolved this issue by using the Wireshark tool to inspect the failed requests from the web page, identifying each server address which needs to be made accessible, and finally also adding these addresses to the whitelist allowed for unauthenticated users.

Once the user has authenticated with his or her Facebook credentials and the account is classified as previously described in 3.4.1, he or she finally must authenticate on the MAC level. This involves informing the captive portal that HTTP requests originating from this MAC address should not be intercepted and redirected. Keep in mind that at this point, the captive portal and the controller are not communicating with each other.

To authenticate users on the MAC level, I used an exchange of parameters via GET requests, as it is the only mechanism provided by the captive portal. Admittedly, this mechanism does leave authentication vulnerable to certain types of security attacks (e.g spoofing the MAC address), but we have to keep in mind that the purpose of this authentication is simply to keep account of who has been redirected to the web page which grants extra bandwidth. Furthermore, even if an attacker circumvents the captive portal, he or she will still have the lowest level of bandwidth (which can be accessed anyway if the user chooses not share his or hers Facebook details) since it is the controller who grants the capacities not the captive portal. This means that an attacker will not get any advantage if he chooses to attack this part of the system.

3.6 Web Page

In this section, I discuss the development of the web page to which the users are redirected using the captive portal technique described in the previous section.

Provided that I had no previous experience in the area of web development, the first thing that I needed to do was to do basic research on the subject and to understand how different web development technologies are used.

After gaining a basic understanding about web development and associated technologies, I needed to choose a language in which to implement the website. Ultimately, I used the languages Javascript and PHP because the project required both server-side and client-side programs.

The server-side program is needed because at the point in time the user reaches the login web page, he or she is not yet authenticated and thus has only been granted access to that web page and the Facebook login page. Therefore, the user is not permitted to execute any of the necessary calls to the Facebook APIs to perform authentication themselves, and the server must execute the commands on the users behalf.

However, the server is used by multiple routers and clients, so whenever a packet is received at the server, it is hard to determine which device originally sent it. Because the client knows the location of the router and the server does not, this naturally led to the introduction of a client-side program to execute any communication with the router, while the server simply performed the authentication with Facebook.

Although the above scheme provides a good design pattern with a clear decoupling of the client-side and server-side implementations for the web page, it did require additional work to establish a way of communicating between the two separate parts. This communication was implemented by passing PHP variables from the web page server to the client browser.

In the following subsections, I describe the two sub-parts of the web page implementation in further detail.

3.6.1 Server-Side Implementation

For the server-side, I used the Facebook SDK to perform login with the users Facebook credentials and registered the URL of the web page with Facebook.

I first selected which data fields I would request from each users profile. In view of the fact that users are becoming increasingly cautious about sharing their personal data, I chose to request the absolute minimal amount of information mandatory for a working system. In other words, I only request access to data which will be directly used for making decisions in the most commonly used resource sharing policies. This decision may be restrictive to some owners of networks who might want to enforce more intricate policies which demand additional information such as a policy in which bandwidth is only allocated to users who reply to messages from the network owner within two hours, However, the scheme I chose offers a higher sense of security to the average user, which I deemed more important for the success of this project.

The next step is to pull the values of the desired fields from the users Facebook profile. Recall that when a user attempts to connect to the network, he or she is immediately redirected to a Facebook login page. Upon successful authentication, the web page obtains the user identification data plus an access token, an opaque string that identifies the user, application, or web page and contains information about when the token will expire and which application generated the token. Using this information and some additional information about my registered application, I make calls to the Facebook Graph API. The Graph API provided by Facebook allows various types of queries about the users and their profiles (e.g., photos, posts, likes, etc.). For example, in order to obtain a list of the friends of the recently-authenticated user, I made the following query:

```
/me/friends/ownerId
```

The data from this API is returned in a JSON format which I parse using PHP. Not surprisingly, in order to evaluate any sort of social relationship between two users, I not only needed profile information about the user attempting to access the network but also profile information about the network owner (specifically, his user ID). Fortunately, this information was actually already acquired from the owner of the network when he or she initially set up the network and defined the custom classes of users.

Finally, I apply the data returned from the Graph API calls to determine the social relationship between the user and the network owner and categorize the user into the appropriate class. This involved writing a fairly simple algorithm in PHP for classification. This information about to which class the user belongs is sent back to the client, and from this point on, the client-side of the application is left with the task of completing the authentication process and granting the user access to the network.

3.6.2 Client-side implementation

At this stage, the client web browser is aware of the class to which the user belongs, and so the immediate next step was to implement the mapping technique described in 3.4.1. I utilized a protocol called WebSockets to establish a communication channel over a TCP connection between the web browser and the controller so that the client can communicate the class of the new device to the controller. Because the message is sent from the client to the router and both live in the same link layer network, the MAC address of the packet will not have changed and so I am able to uniquely match each devices true MAC address with the correct class.

Now that the controller is aware of the class associated with the device, it can finally allocate the proper network resources to the user. The user can now operate within the network and access the Internet.

3.7 Summary

In this chapter, I have described at high-level the implementation of this project. The sections above describe the four of the most important parts of my project: the controller, the hardware configuration, the interface for users, and the integration of the project with Facebook. The major difficulties with the project especially the controller and the captive portal have been explained as well as the approaches used to overcome them. I have also specified the tools that were used.

Chapter 4

Evaluation

In this chapter, I present how the system implementation was evaluated and show that the project met the success the criteria. The implementation meets the specification requirements described in the Preparation chapter and achieves the aims of the dissertation.

This chapter begins by describing the functional testing of the system components, the network scenarios which were tested, and all of the results. In the subsequent sections, it focuses on evaluating the features of the system and comparing the system with the similar projects. Finally, I compare the performance of the developed system against that of a standard network. I believe obtaining performance comparable to a normal network is very important because adoption of a system like the one developed for this dissertation depends on being able to meet the expectations of the users.

4.1 Functional testing

For the purposes of functional testing of the system, particularly the parts which involve using the Facebook APIs, I utilized my own personal Facebook account and created a few additional dummy accounts. The dummy profiles were used to simulate other users without the need for involving actual human users and their real personal data. Throughout the entire duration of the project, the application was in sandbox mode, so no real users were able to access the system at any time. . The fake profiles were used for testing common scenarios of interacting with the system and simulating the possible social relations between the users.

Automated testing made up a significant portion of the testing phase for this project. Since most of the code was written in Java, I was able to use JUnit for testing. Among the things that were tested were the establishment of communication between the router and the controller, the parsing of the command line commands send to the router and their respective results. Writing the automated tests for the controller required more work because I needed to simulate the packets sent to the router and also mimic the behaviour of a router upon receiving these packets. Once this was done, I tested the resultant behaviour of controller by verifying that packets were dealt with in an expected manner.

4.1.1 Rate limiting

Part of the functional testing involved confirming that my project can perform rate limiting. In addition to using the numerous speed testing applications available on the web, I performed a test in my local network. The way I achieved this is by using the tool iperf, which allows me to measure the bandwidth between two points in the network. The output from the test is presented bellow:

```
TCP window size: 85.3 KByte:
[4] local 192.168.142.220 port 5001 connected with 192.168.142.167 port 47334
[4] 0.0-12.3 sec 1.38 MBytes 935 Kbits/sec
```

This was a sample taken in the case of the two nodes belonging to the default class, where the TCP window size is 85.3 KBytes. On the last line it is shown that the average bandwidth is 935 Kbits/sec, which is what is expected, because my default limit is 1000 Kbits/sec. Even though this value is slightly different from 1000 this is not surpising, because of the ways TCP congestion control and rate limiting works. Once the limit is reached packets are being dropped which leads to rapid reduction in the TCP congestion window size, which in turn results in a slightly lower average bandwidth over time.

4.2 Mininet

I used a tool called Mininet for further testing system behaviour and proving that certain OpenFlow commands and configurations operate as expected. This tool allowed me to create a realistic virtual network running a real kernel as well as switch and application code on a single machine. I created switches that support OpenFlow and virtual nodes to connect to the network. I was then able to simulate how the network would behave with many nodes and verify that the policies enforced by the commands still hold. For example, I tested the creation of flows and the behaviour of the network once they were set up. Another huge advantage for this environment was that it enabled measurement of the bandwidth and latency on isolated traffic, which is impossible to measure in a real-world network.

4.3 Latency of the produced system

One of the major concerns for the system was that the processing time required for the controller would drastically slow down the network performance. There are two major causes for a user to perceive a network as slow: low bandwidth and high latency. Low bandwidth is not an issue in this case, because the communication between the router and the controller is limited to only a few kilobytes per second. However, an increase in latency compared to a standard network is unavoidable because latency is introduced in the communication between the controller and the router. In theory, this latency should not be too great because only at certain periods single packets are being examined by the controller. However, there was a risk of using to small time-to-live fields of the flows or having flow tables which are too larger and introduce latency because of their complicated processing. In the following subsections, I will describe how I gathered data for the latency, how I analysed the data, and what my findings and conclusions are.

4.3.1 Gathering and analysing the data

I evaluated the latency of the system using round-trip time samples returned by the ping networking utility tool between two computers connected to my access point. I input the resulting round-trip times of a sample size of 1000 packets into a Java program I implemented to extract and evaluate the results of my measurements. I applied the following exponential weighted moving average formula from the Part IB Computer Networking course [4] to estimate the round-trip time:

$$EstimatedRTT = \alpha * EstimatedRTT + (1 - \alpha) * SampleRTT$$

SampleRTT is the newest measurement, and EstimatedRTT is the existing smoothed round-trip time estimate. The value used for alpha was 0.875 as suggested by the Karn/Partridge Algorithm (reference). The newly estimated values by my program which implement the mentioned algorithm were plotted in Figure 4.1 using Matlab.

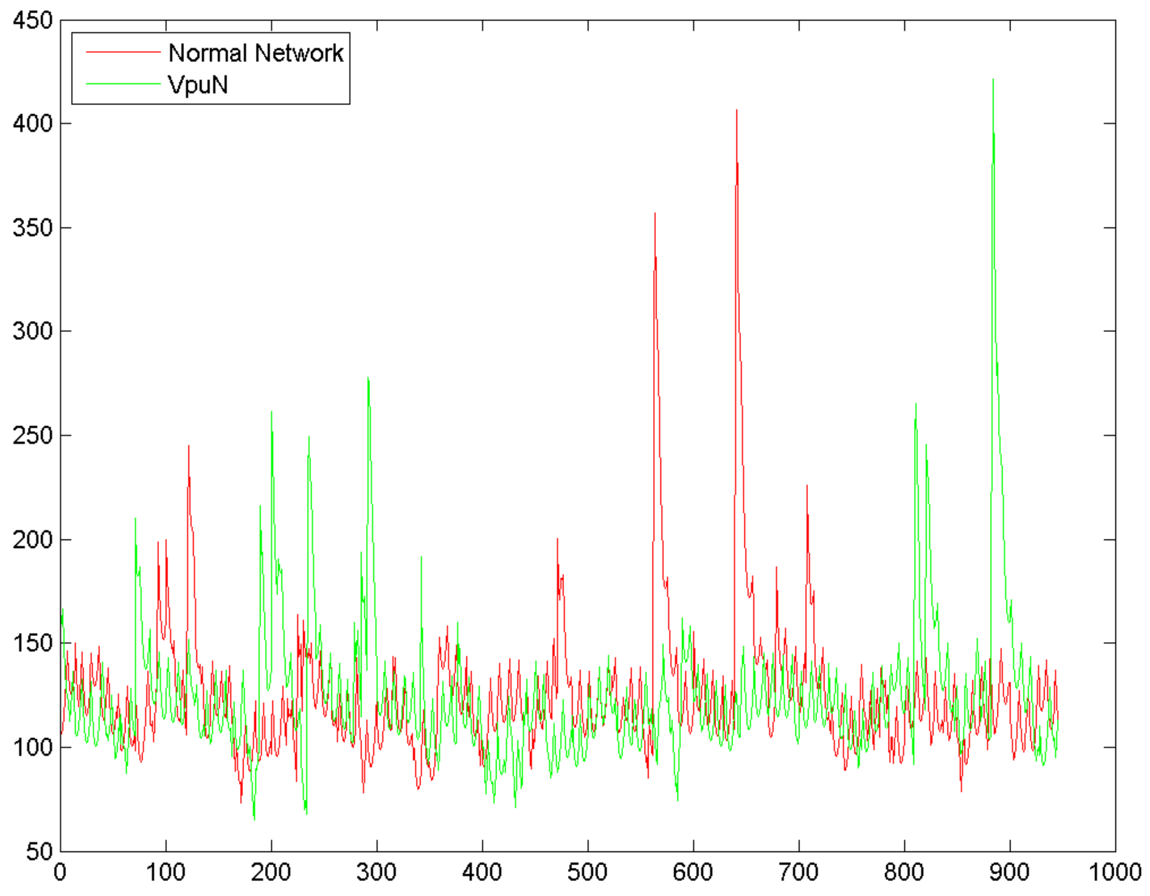

Figure 4.1: Latency comparison

4.3.2 Results and conclusions

In Figure 4.1 above, above, the resulting round-trip time estimates for the virtual public network I implemented are plotted (green) against the resulting estimates for a standard network (red). The time in milliseconds on the vertical axis and the packet sample number on the horizontal axis.

The peaks observed in the graph signify lost packets. By computing the rate of packet loss for each network, it can be seen that the chance of a lost packet did not increase from in the new system. The loss of a packet is quite random and it can be caused by variety of reasons including signal degradation, congestion, or corrupted packets. Because the lost packets skew the data, I only compared the stable regions of the graph in which no packet loss occurs. In such regions, it can be seen that the latency of the two network does not differ by more than a few percent. Thus I was able to conclude that the new system can be used without perceiving any additional latency.

4.4 Comparison with the Fon project

In this section, I present a simple feature comparison with the most similar project to the one described in this dissertation.

	Integration with Facebook	Isolation of the Networks	QoS
FON's Project	Yes	No	No
My Project	Yes	Yes	Yes

As it can be seen from the table above, this project offers separation of networks and quality of service on top on of what the FON's project offers. The former provides comforts to the owner of the network because it better protects his or her privacy and guarantees access to the Internet even when there are numerous other users connected to the access point. The quality of service (this is affected by the latency and the bandwidth and both of those can be managed from the controller) is also an important feature as it prompts more complicated schemes of sharing the resources and will be a major incentive for deploying a network like this. An advantage of the FON's project is that it already works on more platforms than my currently project.

Chapter 5

Conclusion

This dissertation aimed to prove the feasibility of incorporating data from social media sources such as Facebook into the process of making traffic routing decisions within a privately owned network. Using the SDN abstraction, it has been shown that the access points of the network can perform relatively complex operations without significantly affecting the performance of the sample network.

I have implemented a fully functional end-to-end system which employs SDN concepts to relay decisions from an intelligent controller to the forwarding router, thus meeting my success criterion. The new system provides a network owner the ability to create custom classes of users in Facebook and to distribute the network resources to each group in turn as he or she desires. New users connecting to the network may authenticate via Facebook credentials and are then able to use the network in compliance with the policies set by the network owner.

5.1 Lessons learnt

If I had to do this project again, I would aim for smaller increments with more prototype iterations in order to realize the obstacles I encountered sooner such as issues caused by layer violation. Another thing which I would have done differently is to have a more active preparation phase because I felt I did not experiment enough with the technologies and the tools, which ended up hindering me later on in the implementation phase.

5.2 Future development

An immediate extension to the current system which would perhaps increase its relevance to Internet users today is to include other popular sources of social media such as Twitter, Google+, or Instagram and allow the owner of the network to select some subset of sources to influence the flow of data traffic. These additional sources could provide the necessary data to build more suitable sharing policies. On the other hand, there will be an expected limit to how much data the controller can process without introducing significant latency, which would be an interesting limit to test.

Advancement in software-defined networks with customised resource sharing policies could lead to a massive increase in the accessibility of the Internet. For example, owners of private networks could open their network resources free to the public at times when they do not expect to be online, thereby connecting people who may not normally have access. There is a strong movement toward expanding access to the world of information offered by the Internet, and I believe people are increasingly interested in utilizing the currently wasted resources to achieve this.

There remains much further work to be done before it can be determined if such a system could be released for market use, but the results of this project suggest that such an approach has remarkable potential to change the way wireless networks are managed in the future.

Bibliography

[1] C.S Sriram D. Trossen P. Papadimitriouy J. Crowcroft A. Sathiaseelan C. Rotsos. Virtual public networks. In *2nd IEEE European Workshop on Software Defined Networking (EWSDN)*, Berlin, Germany, 2013.

[2] Fon Wireless Ltd. Fon launches world's first wifi router for home with facebook login, October 2013.

[3] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, April 2008.

[4] Andrew Moore. Computer networking. University of Cambridge.

[5] ONF White Paper. Software-defined networking: The new norm for networks, April 2013.

[6] Wikipedia s. Openwrt, 2014.

Appendix A

refs.bib

```
@INPROCEEDINGS{VPuN,
   author = " A. Sathiaseelan C. Rotsos, C.S Sriram
D. Trossen P. Papadimitriouy J. Crowcroft",
   title = "Virtual Public Networks",
   booktitle = "2nd IEEE European Workshop on Software Defined Networking (EWSDN)",
   year = 2013,
    address = "Berlin, Germany",
}

@Booklet{SDN,
title = "Software-Defined Networking: The New Norm for Networks",
author ="ONF White Paper" ,
month = "April",
year = 2013,

}

@Misc{FON,
title = "Fon Launches World's First WiFi Router For Home with Facebook Login",
author = "Fon Wireless Ltd.",
month = "October",
year = 2013,

}

@Article{opneflow,
author = "Nick McKeown and Tom Anderson and Hari Balakrishnan and Guru Parulkar and Larry Peterson and Jennifer Rexford
title = "OpenFlow: enabling innovation in campus networks",
journal = "ACM SIGCOMM Computer Communication Review",
year = 2008,

volume = 38,
number = 2,
pages = "69-74",
month = "April",
}

@Misc{Openwrt,
author = "Wikipedia s",
title = "OpenWRT",
year = 2014,
}

@Booklet{partIB,
title = "Computer Networking",
author = "Andrew Moore",
address = "University of Cambridge",
OPTyear = 2012,
}
```

Appendix B

Project Proposal

Diploma in Computer Science Project Proposal

Integrating Online Social Networks with Virtual Public Networks

Stoyan Stoyanov,Robinson College

19th October 2013

Project Originator: Dr. Arjuna Sathiaseelan

Project Supervisor: Dr. Arjuna Sathiaseelan

Director of Studies: Dr. Alastair Beresford

Project Overseers: : Prof. Peter Robinson & Dr. Robert Watson

Introduction and Description of the Work

The notion of crowdsharing home broadband networks with the public have gained popularity (e.g. FON). With the advent of Software Defined Networking (SDN), there are now more opportunities for network operators as well as home broadband users to manage such crowdshared networks. Recent work carried out at the Computer Laboratory demonstrated how SDN could be used with big data (such as information from Facebook, Twitter, etc.) to make intelligent traffic engineering decisions . SDN can be used for creating and managing open wireless networks called virtual public networks . Such networks can be more efficient and provide access to users who would not be able to afford it otherwise. Furthermore, in the case of an emergency, home networks could be open to the public.

Starting Point

While implementing this project, I will mainly rely on my understanding of computer networks and through the Part IB course in Computer Networking and Part II Principles of Communication course. Moreover, there may be need to research additional advanced concepts in this field whenever my current knowledge is insufficient. I also have moderate experience working with social media APIs, which will be helpful at certain stages of the project.

Substance and Structure of the Project

I will have to create, deploy, and manage a virtual public network [1], like one you might use in the home, and integrate its access points with the social media (Facebook, Twitter, Google). In order to do that, I will have to implement middleware using Openflow/OpenVSwitch, which will allow me to communicate with the router and allow the enforcement of policies for sharing the network and the amount of traffic. From the social media, I will start implementing simple policies that grant a user a certain capacity based on the friendship status with the owner of the network. Once this is done, I will explore more dynamically generated policies based on multiple factors derived from the social media.The project requires significant modifications to both the home router firmware (using OpenWRT with OpenVSwitch) and the Floodlight controller. There are a few options for languages in which I can implement the controller (the most probable being C++ and Java), and I will make that decision after further reading.

Success Criterion

The first and most important criterion for success will be a working end-to-end system. I will test the system with multiple users and provide information from broadband tests. I will also need to assess how the system behaves under changing social relationships. This will not include actual human subjects, because I will create a bunch of fake profiles and relations between them. Another thing I will assess is benchmarking and performance, with special attention given to the latencies involved when the router and controller talk to each other.

Timetable and Milestones

My intentions are to work throughout the year (including the holidays) and to be done with the dissertation two weeks before the deadline, so that I have time to revise and also in case there are delays in my project.

1. **18th October - 31st October** In this period I will focus on understanding the basic concepts behind SDN and virtual public networks. Another thing, I will do is check how to establish authentication between a Facebook server, the controller,the user and the data about the owners of the home network.

2. **1st November - 14th November** I will research any additional tools that I will need for the project, and I will install and start modifying the controller. I will also install and experiment with the basic firmware supporting OpenWRT(I already have access to a Netgear WNDR3800 flashed with OpenWRT and OpenVSwitch). Another important task for this period will be setting up a good, reliable testing environment.

3. **15th November - 19th December** This will be a crucial implementation period. I will create most of the middleware and implement simple integration with Facebook (e.g granting more capacity to your close friends). This will be an important milestone for the project because it should effectively produce a working demo.

4. **20th December - 3th January** In this phase, I will fix all outstanding issues with the current implementation, and I will implement more dynamic and complex algorithms for generating policies for sharing the network.

5. **4th January - 4th February** I will do further testing, and then I will focus on analysing the produced system. In this phase, I will be able to

measure the behaviour of the system under changing conditions and explore its limits and performance under different levels of load.

6. **5th February - 31st March** I have left some free time during this period in case there is some unanticipated delay in the previous stages. If everything has gone according to schedule, I will work on the dissertation and one or two extensions.

7. **1st April - 21st April** I will finalize the dissertation, and I will try to get as much feedback on it as possible and amend as necessary. This will include sending a copy to my supervisor and director of studies and integrating their feedback in the thesis.

8. **22nd April - 2nd May** This is the time for final small modifications. Finally, I will print and submit the thesis and focus on exam revision..

Resources Required

The project will require me to create a controller for a home network router, for which I will make use of my personal laptop. I will also need a home router, which has already been provided by my supervisor. I will use a gitHub repository for work and backup. A significant challenge will be the creation of testing environment, which will be very useful for both evaluation and development. I will create this testing environment with the help of VMs. Besides those mentioned, I do not require any other special resources.