Dynamically Routed VPNs with Policy Distribution

Supervisor: Dr Stephen Hailes
Group members: Kai Duan
Ying Shi
Qian Yang
Oliver Priest

For the Degree of Master of Science

Department Of Computer Science
University College London
Gower Street, London.

September 2004

Acknowledgements

The project group would like to thank Dr Stephen Hailes and Manish Lad for their support and valuable assistance over the course of our project. Their opinions have helped guide us clearly over the course of the project.

The project members would also like to thank their family and friends for their ongoing love and support that helps so very much in the learning process.

Contents

1. INTRODUCTION 5

1.1. OVERVIEW OF VPNS 5
1.2. PROJECT PURPOSE 6
1.3. MOTIVATION 6
1.4. SCOPE 7
1.5. REPORT OVERVIEW 7

2. BACKGROUND 10

2.1. DIFFERENT VPN TECHNOLOGIES 10
2.2. EXTENSIBLE MARKUP LANGUAGE (XML) 12
2.3. ROUTING INFORMATION PROTOCOL (RIP) 12
2.4. RELATED PROJECT: AKENTI 13

3. PROJECT OBJECTIVES 14

3.1. MANAGEMENT SUMMARY 14
3.1.1. SOFTWARE COMPONENTS 14
3.1.2. SECURITY POLICIES 14
3.1.3. DYNAMIC ROUTING 14

4. DESIGN 16

4.1. DESIGN OBJECTIVES 16
4.2. DESIGN ASSUMPTIONS AND CHOICES 16
4.3. INITIALISATION & UPDATING 17
4.4. THREATS ANALYSIS 20
4.5. POLICY SPECIFICATION 23
4.6. SECOND PHASE DESIGN CONSIDERATIONS 25

5. IMPLEMENTATION 28

5.1. SOFTWARE LIBRARIES USED 28
5.2. MODULE: XML PARSER & GENERATOR 32
5.3. MODULE: SCHEDULER 34
5.4. MODULE: VIRTUAL INITIALISATION SERVER 36
5.5. POLICY HANDLER MODULES 38
5.6. MODULE: COMMUNICATION MONITOR 41

6. TESTING 45

6.1. UNIT TESTING 45
6.2. INTEGRATION TESTING 48
6.3. OVERALL SYSTEM TESTING 51

7. EVALUATION AND FUTURE WORK 56

7.1. EVALUATION OF RESEARCH METHODS 56
7.2. EVALUATION OF PROJECT OBJECTIVES 57
7.3. EVALUATION OF PROJECT MANAGEMENT TECHNIQUES 57

7.4. EVALUATION OF DESIGN, IMPLEMENTATION AND TESTING 59
7.5. SECURITY EVALUATION 60
7.6. FUTURE WORK 61

8 CONCLUSION 64

APPENDIX A: PROJECT MANAGEMENT 65

1. TEAM CONSTITUTION AND STRUCTURE 65
2. WORK BREAKDOWN STRUCTURE 66
3. EXTREME PROGRAMMING OVERVIEW 68
4. SCHEDULING 70
5. PROJECT DOCUMENTATION 72
6. RISK MANAGEMENT 73

APPENDIX B: XML SCHEMA 75

1. XML SCHEMA FOR ACCESS CONTROL POLICY 75
2. XML SCHEMA FOR INITIAL POLICY 76
3. XML SCHEMA FOR UPDATING POLICY 77

APPENDIX C: TECHNICAL USER GUIDE 78

1. FILES AND DESCRIPTION 78
2. INTERFACE SPECIFICATION 79
3. BUILDING PROCESS 82

1. Introduction

1.1. Overview of VPNs

Secure computer networking has risen in popularity over the last 20 years seeking to address the security concerns of individuals and organisations alike. With the introduction of advanced encryption and decryption algorithms and the decreasing cost of powerful microprocessors, the Virtual Private Network (VPN) has become a popular choice for decision makers who own and maintain secure computer networks. The term "Virtual Private Network" can be broken down into 3 separate words each of which has a specific meaning in the IT world. "Virtual" implies that something is in fact simulated and performs the functions of something that doesn't fully exist. In the case of VPNs, a virtual network service will be in operation over a real existing network infrastructure. The traffic flowing over a real network is actually partitioned into separate discrete secure tunnels each of which has no interaction with the other. "Private" simply means that any data that flows over the logical tunnels between two endpoint devices may be considered secret in some way. One of the best features of VPNs is the fact that it is possible to send data to intermediate nodes that have no need to examine the contents of the data. They simply have to forward the data to the next device in the tunnel. Although the data that transits a VPN is private the infrastructure used to transport the data can be public. This allows for large tunnels to be created over the Internet through service providers and third party networks. Private data allows the VPN to use its own network addressing and routing mechanisms. "Network" refers to the simple fact that a VPN is in fact a set of interconnected devices that communicate with each other in some predefined manner often transmitting and sharing data.

VPNs have now started to mature and the technology now exists on numerous platforms in many different guises. Various standards have also been drafted to help support the rapid development of VPN software and hardware and much progress has been made in the area. Good research, coupled with viable commercial products has allowed VPNs to become a popular enabling technology providing security solutions to individuals and organisations. VPN technology is also moving at a staggering pace and developments and improvements to the technology are occurring extremely frequently.

1.2. Project purpose

For our group project, we have decided to focus on dynamically routed VPNs. Our group project will attempt to look at the area of dynamic routing and policy distribution mechanisms in more detail providing a software framework that deals with some of the shortcomings in current VPN technologies and implementations. Offering dynamic routing in overlay models using traditional VPN standards and VPN software is not feasible for a number of reasons that will be discussed in detail in another part of this report. The only real solution is to break the mechanism and propose a new novel way of solving the issue. As well as analysing the requirements of our software, the group needed to design, implement and test the software using a variety of project management techniques.

1.3. Motivation

Current VPN implementations are effectively based on fixed static routing. The problem also becomes worse when the use of fully meshed network endpoints introduces an N squared scaling problem. As the number of VPN endpoints increases, so too does the number of VPN tunnels to the power of the endpoints. A fully meshed VPN with 5 endpoints requires 10 VPN tunnels to create a fully meshed network while a fully meshed network with 20 endpoints requires 190 separate tunnels to create the same fully meshed infrastructure. Obviously, there are serious scaling issues with fully meshed VPNs. Partially meshed VPN overlay networks is one viable alternative that provides partial connectivity between selected endpoints in the VPN. A traditional approach to dealing with the problem requires the use of backbone endpoints that connect major nodes in the VPN that carry more data. The rest of the endpoints effectively form tunnels with one or more of the backbone nodes. Routing table information allows data to be passed through a VPN tunnel towards the backbone. Once the data arrives, dynamic routing can occur and the packet can make its way to a destination indirectly without the need for a directly connected VPN tunnel to the endpoint. Providing a dynamic routing service to VPNs is the primary motivation for the whole project. However there are other important secondary motivations. Providing more granular VPN security policies is another key motivation for the group. Allowing high-level security policies to be defined and applied to an overlay network is a particularly difficult challenge. Although, the group did not fully implement high level policy enforcement we were able to construct distributed access control policies based upon XML that were applied to VPN endpoints.

One important motivation for the group is to help to come up with new ideas at solving the above problems and to allow our ideas to be used in any future research work. The motivation for producing dynamically routed VPN software stems from the fact that the group is passionate about the subject and would like to see the technology grow.

1.4. Scope

The scope of the project was dictated by a list of project objectives and goals that the group had initially set at the start of the project. The group decided that they would develop a VPN framework that can support numerous types of security policies. Data communication and routing and security updates would be passed through the software framework. One project deliverable was to produce software that incorporates a dynamic routing mechanism for VPNs. Another was to test this software with a number of computers that would act as VPN nodes. The scope of the project also included distributing access control policy information in the second iteration of the design process. The group did not attempt to create a brand new VPN standard but instead chose to use existing technologies and APIs. In the course of the project the group created various XML structures that could be transmitted and parsed. The XML data structures were used to store routing information and security policies in an efficient manner.

1.5. Report overview

The rest of the group report will be broken down into the following sections:

Chapter 2 – Background

This chapter gives an overview of the currently existing standards and VPN technologies available at present in both the academic and business arenas. A brief summary of the technology that has influenced the project group is listed here.

Chapter 3 – Project objectives

This chapter examines what exactly the project is trying to achieve. The objectives will be discussed in detail.

Chapter 4 – Design

This chapter covers software design and provides an overview of the software components and how they work together. Reasoning for the system design is also covered in detail.

Chapter 5 – Implementation

This chapter covers software implementation including any important novel software coding features that are of interest.

Chapter 6 – Testing

This chapter covers unit testing, integration testing and overall software testing. Testing methodologies and techniques that the group used will also be mentioned along with the results of the tests.

Chapter 7 – Evaluation and future work

This chapter provides a critical evaluation of every aspect of the group project. Future follow up work will also be discussed.

Chapter 8 – Conclusion

The final chapter covers the project conclusion and a summary of what was achieved throughout the lifetime of the project.

Appendix A – Project management

This appendix covers the group project management strategy, allocation of responsibilities, team communication and software engineering methodology.

Appendix B – XML Schema

This appendix covers details of the XML schemas that the security policies use.

Appendix C – Technical User Guide

This appendix covers the technical user guide and includes important information about using the software.

2. Background

The background section gives a basic overview to some of the most important VPN technologies and standards that currently exist. The purpose of including these technologies is to give an indication of the influence they had on our project. By examining existing successful technologies we were subsequently able to evaluate the pros and cons of each technology and look for potential improvements therefore making our research more valid and useful.

2.1. Different VPN technologies

2.1.1. IPSec

IPSEC (Internet Protocol security) is a set of VPN standards that have emerged over the last nine years to create secure IP networks that can guarantee data confidentiality and integrity. The technology is popular with many software and hardware vendors adopting the standards creating products and implementations. IPSec is particularly complex and cumbersome. The software implementations that exist in academia and industry never fully utilise all of the proposed features proposed in the standard. However, there is strong interoperability between competing software products allowing for a mixed software environment when creating VPN tunnels between endpoints and gateways that can often reside on different operating system and machine architecture. At the heart of IPsec are a set of protocols that handle data encapsulation and the exchange of public key data. These include the Authentication Header Protocol (AH) and the Encapsulating Security Payload protocol (ESP). Both protocols effectively create security headers providing security and tunneling information that are appended to the original IP datagrams. Key distribution is handled by the OAKLEY and ISAKMP protocols (Internet security association key management protocol). IPSEC also makes use of two small databases that reside on a VPN endpoint. These are the security association database (SAD) used to monitor secure traffic flows and the security policy database (SPD) used to apply and enforce specific security policies at VPN endpoints. IPSec operates in two different modes when transmitting data. The first mode is tunnel mode that effectively creates a brand new IP header and appends it to the existing IP packet effectively masking the original IP address information along with any other fields in the header and data payload. This technique is similar to the Generic Routing Encapsulation standard.

(GRE). The second mode, known as transport mode allows the original IP header to be used while the security information is inserted between the IP header and the data payload. Both modes have the own purposes while tunnel mode is generally required when creating a tunnel from/to a VPN gateway.

IPSec is particularly important because our project is focused on some important shortcomings found in traditional IPsec VPNs. Although the IPSec standards allow routing protocols to be carried across the tunnels the security association databases actually have the "final say" about where the data is routed. Therefore routing protocols carried across the tunnels become useless and the security associations and their security policies act like static routes. Another problem exists concerning the amount of time that a security association takes to be established or changed when compared to a routing protocol update. The difference is a few orders of magnitude making security association updating seem inappropriate for large complex network designs.

Overuse of static routes in large computer network designs can lead to many different kinds of networking problems. Static routes make the network unscalable and prone to configuration and failover problems. Providing an adaptive secure network is a key requirement for many organisations and static routing in one form or another does help to meet these requirements. Although IPSec has many novel features, the lack of dynamic routing support is a big problem that needs to be addressed in the long term.

2.1.2. SSL VPNs

Secure Socket Layer VPNs are beginning to appear over the last few years and present a different approach to VPNs. Whereas an IPSec VPN works at the IP layer (network layer), SSL VPNs operate at the application layer effectively providing secure connectivity to remote applications. At the heart of SSL VPNs is the SSL API that utilises a public key certificate system to transmit encrypted data over TCP sockets. Just like IPSec, SSL is designed to provide data confidentiality and integrity. SSL technology is available as open source software libraries so the project group instantly took an interest in the availability of APIs and other open source SSL projects. Because SSL is relatively simple when compared with IPSec the project group began to look at ways in which SSL could play a part in our software design. Because SSL uses a socket based application interface, it is possible to design and code applications based upon a client/server framework. Such

a simple framework turned out to be a good starting block for our project providing us with some stable starting components.

2.2. Extensible markup language (XML)

XML has become the de facto standard for storing structured information in documents that can be transmitted between different software entities. The language makes use of text-based documents that can be read by humans and computers. XML is a flexible and popular language used by many different types of organisations to store structured information. One of the best features of XML is the software engineer's ability to create their own unique structured information. The project group looked at many different ways of representing structured information and felt that XML provided an excellent framework for transmitting our routing and policy information messages between VPN nodes. XML can be easily parsed using a wide variety of open source libraries making the technology available to meet our needs.

2.3. Routing Information Protocol (RIP)

Routing Information Protocol is one of the oldest and most popular dynamic routing protocols used on the Internet in the 1980 and 90's. The routing protocol is a traditional distance vector protocol where reachability information is passed between routing devices. Each router maintains a routing table of all the IP networks it learns from its connected interfaces and from routing updates received from neighbours. The routes are usually based on some kind of hop-based metric. A typical RIP implementation has a routing table that contains destination addresses and next hops. The protocol has seen numerous improvements over the years. The second version of the protocol contained numerous fixes, the most important of which was to make the protocol handle classless routing by allowing the routing table to contain subnet mask information used to make routing decisions based on the longest match of an IP address instead of the matching by class of address. Multicasting was also incorporated into the second version eliminating broadcasting. Because RIP is well documented and available as open source code, the project group felt that the protocol was particularly suited to our small-scale project. RIP is suitable for small computer networks because it is simple to implement. The algorithm can be easily modified and used in routing engine software and this made the protocol interesting to the project group.

2.4. Related Project: AKENTI

AKENTI is a research project based upon distributed access control in distributed computer network environments. The project aims to remove the concept of centralised access control and replace it with a system that achieves the same level of expressiveness but allows system stakeholders to enforce their access control requirements independently of other stakeholders. Access control is maintained by digital certificates that contain user information, resource usage requirements, user attribute assertions and delegated authorisation information. The AKENTI project looks at novel ways of controlling access control. The project enabled the group to look at different novel approaches to providing access control in our VPN project.

3. Project objectives

3.1. Management summary

The overall goal of the project is to design and implement an expandable software infrastructure that will support dynamic routing and policy distribution within a VPN network containing numerous endpoints.

The project goal can be further broken down into following three aspects:

3.1.1. Software components

- The design and creation of a set of software daemons used for VPN communication between a set of VPN endpoints.

    1. To analyse the system requirements and create a software design that allows VPN endpoints to send data in a secure, reliable and efficient manner.
    2. To create a suitable software implementation based on the above design.
    3. To fully test all software implementations in a real world environment.
    4. To produce detailed user documentation to assist with understanding and using our software.

3.1.2. Security policies

- The design and creation of low-level security policy structures used to enforce security policies with the VPN.

    1. To design a suitable policy structure that will enable to VPN to transport and enforce a wide variety of VPN security policies.
    2. To allow the policy structure to be easily expandable and modifiable.

3.1.3. Dynamic routing

- The implementation of a dynamic routing mechanism used to maintain reachability information between VPN endpoints.

1. To design a routing mechanism that can be used in a small scale VPN.
2. Where possible use an existing implementation of a routing protocol to save project time.
3. Implement the code in a software module.
4. Fully test the routing algorithm in with a small amount of VPN endpoints.

4. Design

4.1. Design objectives

As mentioned previously in the project objectives section of the report, the group aims to develop a framework including a set of daemons that can be installed and used on selected computers that act of VPN endpoint nodes to accomplish dynamic routing and security policy distribution within a VPN domain.

Dynamic routing in a VPN setting is achieved when a VPN network contains more than 2 endpoints. Data can be routed dynamically along an indirect logical tunnel instead of being passed directly to the destination endpoint. The data traffic may pass through a certain number of endpoints before reaching its destination. While at first glance it may appear that this kind of routing is inefficient it ultimately allows VPN tunnels to scale when a large number of endpoints exist within the VPN. Dynamic routing can help to avoid a fully meshed VPN by allowing VPN nodes at the edge of the VPN domain to maintain a smaller number of logical tunnels with central VPN nodes.

Policy distribution provides granular control of network behaviour and resources. Policies can be applied to a VPN domain that affect how data is forwarded between VPN endpoints, whether dynamic routing is used or not, what security coverage levels are being enforced, what access controls are applied to the VPN, what routing and security algorithms are used etc. Policies allow for tighter control within the computer network and can allow a network to operate under more stringent security requirements.

4.2. Design assumptions and choices

Before the group could begin to design our software prototype several assumptions were made. In our first prototype, it was assumed that all the VPN nodes were known by the network administrator before the VPN was established. We also had to make the assumption that the size of our VPN network was relatively small. Therefore the logical topology can be manually designed by administrators and is easy to manage and maintain.

Our software prototype would be fully independent from any kind of VPN standard however the project will be based on a SSL VPN. The design reason for this is based

upon the simple library interfaces that SSL provides. Preliminary testing has also confirmed that SSL scales quite well in a software based environment. (Please examine the testing section on SSL establishment in section 7) VPN endpoints can be connected up quite easily, which avoids the complicated configuration work at the VPN establishment stage. The design will also make use of SSL tunnels to distribute our security policies and we make the assumption that SSL tunnels are secure enough for this purpose. The last and the most important assumption that the project group makes is that all the participants are entirely trustworthy. (This assumption may raise problems like internal security threats, which is addressed in the threats analysis section.)

4.3. Initialisation & updating

The software prototype consisted of two main parts; the Initialisation component in charge of the establishment of policy distribution infrastructure and the distribution of initial policies. The design makes use of an XML file that contains the logical layout of the VPN. This is read by the component upon start up. After the VPN is established and a logical topology had been established, the Updating component is started to receive and send real-time security policies. During the first iteration of the design, the update policies consisted of dynamic routing information however as the design was refined the group were able to send access control security policy information.

4.3.1. Topology design

Before the VPN can be established, the layout and topology needs to be created and stored in some way. The group decided that it would be the simplest solution if the topology of the VPN were constructed manually**.** The topology information contained in the initial policy file should be distributed from a centralised host. Therefore, a table contains the entire topology table of the whole network and is stored in a central node within the VPN network. This node can be any one of the endpoints but there must be the only one central node.

An example of the table is shown over page:

IP Address	128.16.9.78	128.16.9.217	128.16.9.216	128.16.9.215	128.16.9.77
128.16.9.78	-	Y	N	Y	Y
128.16.9.217	Y	-	Y	N	N
128.16.9.216	N	Y	-	Y	N
128.16.9.215	Y	N	Y	-	Y
128.16.9.77	Y	N	N	Y	-

Table 4.1: Sample of topology information table

4.3.2. Initialisation

When initialisation occurs, the initial policy XML file is read and parsed. SSL tunnels are then established between the central node and the other nodes waiting to receive the initial policy. Information about every tunnel connection in the VPN is included in the initial policy and is distributed from the central node.

We decided to choose a centralised method to distribute initial policies based upon a "push" model for simplicity. The initial policy is sent from the central node when a system administrator runs the initialisation component. As figure 4.1 shows, SSL tunnels are used to distribute policies securely.

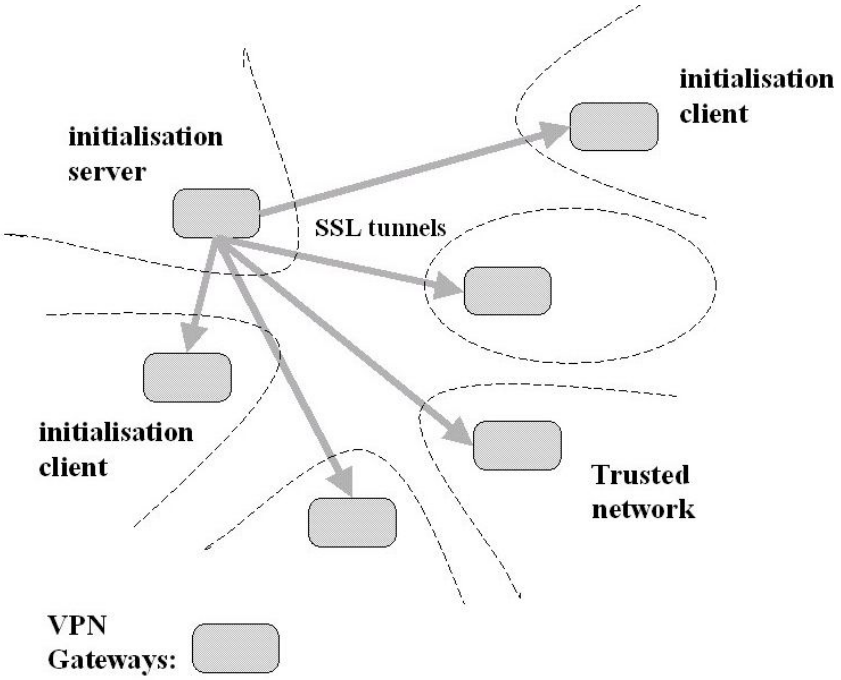


Figure 4.1: Initialization using SSL tunnels

After the initialisation, VPN has been established according to a logical topology, as figure 4.2 shows.

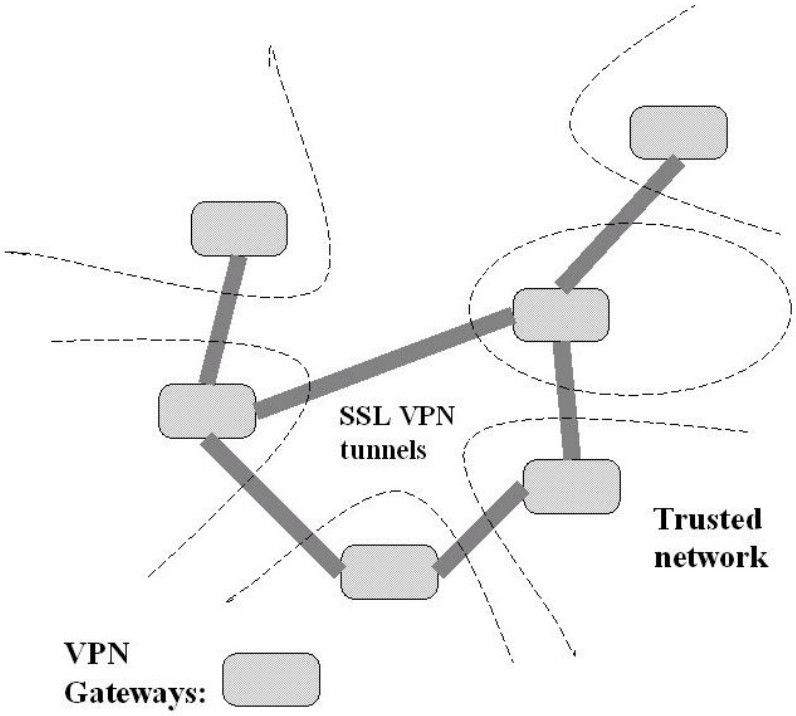


Figure 4.2: Logical topology of VPN

4.3.3. Updating

After the initialization stage, updating of routing and policy information occurs on a regular basis. The updating policies contain routing and security information that is specified later under the policy specification section. The project group chose the RIP routing algorithm to maintain the routing information. As figure 4.3 shows over page, SSL tunnels were used distribute update policies.

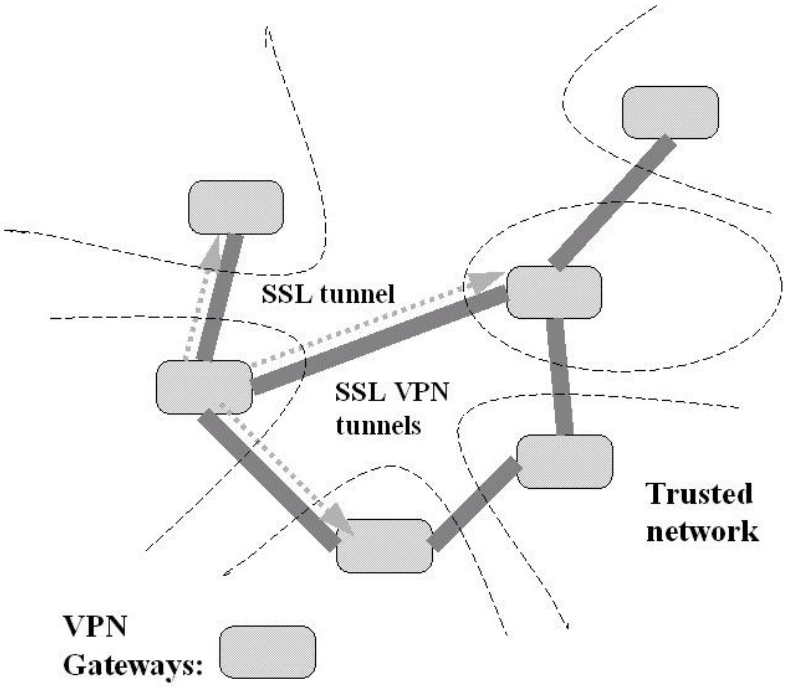


Figure 4.3: Updating policy distribution

4.4. Threats analysis

4.4.1. Generic threats analysis

Any security policy distribution mechanism is the subject of numerous attacks and threats. This next section of the report aims to highlight some of the potential threats the software will face along with potential solutions to mitigate and remove the threats.

Security policy distribution will need to be secured against passive and active tap based attacks intent on reading and modifying potentially sensitive security policy material from the wire/airwaves. If an attacker is able to "snoop" the network traffic then s/he may be able to gain a tactical security insight into the types of security policies a virtual private network is using. This risk is mitigated using a mixture of traditional shared key and public key cryptography techniques to secure any sensitive communication between a set of endpoints. The level of encryption is an important point in modern networks

because if the shared key length of too low then it may be possible to use a brute force attack to obtain sensitive policy information.

A second security threat can also be identified. An attacker may be able to modify the contents of a policy to his/her own specific needs. It may be that a security policy could be modified to specifically create security loopholes thus allowing an attacker to exploit a potential opening in a VPN. This kind of attack could be based in a man in the middle attack where by a legitimate host transmits a policy which is then intercepted by a third party, modified and then sent on its way to appear as if it comes from another host. The best way to mitigate or remove this kind of threat is to use some kind of hash function to provide evidence that the data has not been modified in transit.

Replay of secure policy information is a further potential threat within the design of the system therefore steps must be taken to provide security mechanisms to mitigate the threats.

Denial of service (DOS) attacks must also be considered in our policy distribution mechanism. We must look at the way that the protocol works so that it cannot be manipulated to do more harm than good. In the wrong hands our mechanism could be misused to repeatedly send out policy information to a host that has no requirement to receive such policy information. Because secure policy exchange will be based on public key cryptography it is feasible that the calculations required in policy exchanges could overwhelm the processor resources of a VPN endpoint. It is therefore vital that steps are taken to secure the mechanism against DOS attacks using some kind of cookie based system so that non existent requesting hosts are able to force the receiving host into heavy calculations. Although a single calculation does not present much of a challenge, receiving thousands of policy requests is likely to heavily burden a receiving host's resources.

Secure policies affect how routing occurs within a VPN therefore there are a large amount of potential threats pertaining to routing based attacks are particularly valid in the context of our work. Some of the most serious threats can come in the form of default routing issues and internal routing attacks. Misrouted data could be directed over an unprotected path and it may be possible for an attacker to intercept traffic while it is on its way to its destination. Another potential problem is for an attacker to redirect traffic inside the VPN to a computer host that has already been compromised by a cracker in

some way. Securing routing information using hash functions is a small step in helping to secure routing information running within a VPN. Configuration of routers and firewalls to protect secure traffic from leaving the VPN is also an important step in securing the VPN correctly.

Policy information must also be stored in some way. Many implementations spend a large amount of time securing the transmission across the network but forget about storing any policy information in a secure manner on a computer host. A skilled cracker will have no trouble accessing the file system holding sensitive information if they are able to gain some kind of insight into where the data is located. Perhaps one of the best ways to mitigate this threat is to encrypt the contents of the policy store itself with a suitable file encryption algorithm. That way even if the cracker is able to locate the policy information, s/he will not be able to make any sense of the information.

So far we have discussed intentional threats based on traditional attacks. However, some of the serious threats are caused by mistakes in policy configurations. This can lead to erroneous low-level security policies being applied to security devices potentially opening security holes in the VPN. One way to handle this kind of mistake-based threat is to have some kind of security configuration hierarchy allowing specific users access to configuration-based privileges. This will help to mitigate mistakes and allow only authorised individuals to create or modify security policies. Policies can generally be deemed to be correct if they do what they are supposed to do.

4.4.2. Specific project related threats

4.4.2.1. Initial policy storage

The storage of the initial policy and any other subsequent policies is a particular valid security issue in the context of our project. Although policies may be transmitted using SSL between hosts it is of paramount importance that the policy files (associated with any kind of security policy) be stored in a secure manner on a hosts native file system. This will help to protect the integrity of the VPN. In theory a good hacker will be able to gain access to regular file systems and therefore be able to read the contents of files pertaining to the security of the VPN. If an attacker is able to gain some understanding of the logical topology of the VPN, the type of encryption used, the length of the keys, and any other type of lower level security information then it will a lot easier to gain access to

the VPN. The storage of policy files must take into account some form of file encryption used to protect sensitive policy files.

4.4.2.2. SSL tunnel creation DOS attack

Creating SSL tunnels between a small number of endpoints does not stress a modern computer too much due to the high performance processors that most PC's use. However, scaling a VPN to support a large number of endpoints may start to pose processing problems if there are a sufficiently large number of tunnels. Also of importance is the fact that it may be possible to manipulate the initial policy mentioned above to attempt to establish a tunnel to a non-existent host by modifying the initial policy. On a less powerful host this may lead to resource starvation as the host attempts to open such a large amount of tunnels. One way to deal with the DOS problem would be to implement some kind of tunnel throttling mechanism to control the number of connections in our software implementation.

4.4.2.3. Man in the middle SSL disruption attack

A traditional man-in-the-middle attack can be used to disrupt SSL communications with the intent of causing unnecessary retransmissions by intercepting and slightly modifying SSL payloads causing the integrity check to fail. The result is a packet drop followed by a retransmission. If the attacker continues to intercept SSL packets and modify them as they make their way to their actual destination then it is possible to disrupt tunnel communication between two endpoints effectively isolating an endpoint from the rest of the VPN. It is worth noting that this kind of attack is difficult to defend against but is also highly unlikely to occur.

4.5. Policy specification

As described previously, the two policy distribution stages (initialisation and updating) involve two different types of policies. These are the initial policy and updating policy. The initial policies purpose is to provide the necessary topology information for VPN establishment. The updating policy is for policy information updates and has sub-types such as routing update policies and access control policies.

Policy distribution is achieved through SSL secure connections. As SSL includes authentication mechanism and data encryption, secure policy distribution is ensured. We decided to express our security policies in XML format as it is extensible and flexible. There are ready-to-use XML parser libraries such as eXpat, which helped to simplify our work on policy processing.

4.5.1. The initial policy

For simplicity the group decided to adopt a centralised approach for the initialisation process in our software design. As the main purpose of the initial policy is to support VPN establishment, we needed to distribute topology information inside the initial policy. Based on our previous design assumptions that the administrator knows about all the participants in the VPN and the associated connection topology, it is possible to distribute neighbour information to each host allowing every node in the VPN to know whom it is going to connect to. This neighbour information also constitutes the basis of successful routing. An example of the Initial Policy is shown below. Inside routing_topology there is a list of neighbours, which have two elements *id* and *ip*.

```
<policy type="initial">
        <routing_topology>
            <neighbor>
                <id></id>
                <ip></ip>
                </neighbour>
        </routing_topology>
 </policy>
```

Figure 4.4 Initial policy XML file.

4.5.2. The updating policy

The updating policy is used during the second phase of policy exchange, after VPN establishment, to update information on each host. The update information may take the form of routing information or other security-related policy information. Updates may take place periodically or just when it is policy change is necessary.

A routing update is implemented in a similar manner to a RIP update. Every VPN host sends and receives routing update information from and to their neighbours. In fact, a routing update policy imitates the contents of RIP header according to the RIP specification. Below is an example of routing_update policy. The *protocol* element indicates the routing protocol that is in use, which is RIP for the duration of our project. The protocol field can make use of other protocols allowing our design to be future proofed. The *addr_family* indicates the type of address being specified, and the *cmd* corresponds to the command in RIP header. The *metric* attribute is normally a measurement of number of hops, but here it is possible to assign it some security-related values to facilitate security-based forwarding decision.

```
<policy type="routing_update">
     <protocol></protocol>
     <version></version>
     <addr_family></addr_family>
     <cmd></cmd>
     <route_entry>
         <ip></ip>
         <metric></metric>
     </route_entry>*
</policy>
```

Figure 4.5 Routing update XML file.

4.6. Second phase design considerations

Once the initial software design was completed and implemented the group were able to look at ways to improve our design further. After a lot of discussion the group came out up with a list of improvements to the original design. Please examine a summary list of potential enhancements to our software prototype.

4.6.1. Access control policies

The first design had no specific XML structure for access control policies. This was quickly addressed and a module to handle access control policies was designed and

implemented. Please view a sample XML file for an access control policy below. This policy can be used to give VPN node the ability to specify its routing and forwarding preferences. For example, node A may not want any traffic from node B, thus any traffic node B wishes to pass through node A can only be forwarded through another node, which could be node C. If node C is connected to both A and B and also is permitted by A to forward its traffic. In order to enable these software functions to be carried out we needed to extend the software parser and generator modules to add a process for the access control policies. The group also needed to design a separate component to handle access control. This was responsible for generating access control policies and maintaining a separate access control table. This access control table will be consulted before any traffic is going to be forwarded out of the VPN nodes network interface.

```
<policy type="access_control">
      <subject>128.16.80.111</subject>
            <allow />
            <deny />
</policy>
```

Figure 4.6 Access control policy XML file

4.6.2. Application interface development

Our program should provide an interface to other applications, which can be easily used to transfer data inside VPN without having to know about the VPN topology and routing mechanisms. A dedicated module will be responsible for handling the application data. It will look up both the access control table and forwarding table to make final forwarding decision and set up secure SSL connection to next hop and forward the application data.

4.6.3. VPN visual monitor

At the suggestion of our project supervisor the group began designing a visual software program that could show the VPN overlay network links graphically as they were established between VPN nodes in real time. The purpose of this tool was to give a clear presentation of what the network looked like from a graphical viewpoint. VPN nodes would be represented with filled circles and lines would represent VPN links. The lines

simply connect to the circles to show logical links between nodes. While the software is quite basic it could be used as a starting point for a visual network management tool. The purpose of the tool in the context of our project is to give a simple visual illustration of the VPN logical network at a given point in time. This helps an individual not accustomed to the project to gain an insight into what is going on from a technical perspective.

5. Implementation

The implementation section lists the software implementation approaches and techniques that the group used throughout the course of the project. Special attention has been given to an explanation of novel implementation decisions along with detailed module information relating to the unique operation of our software.

5.1. Software libraries used

5.1.1. OpenSSL

OpenSSL is based on the excellent SSLeay library developed by Eric A. Young and Tim J. Hudson. It is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library managed by a worldwide community of volunteers that use the Internet to communicate, plan, and develop the OpenSSL toolkit and its related documentation. The group made use of the OpenSSL library to enable us to create secure VPN tunnels for our SSL VPN implementation.

The following APIs were used:

- *SSL_CTX *SSL_CTX_new(SSL_METHOD *method)*
  It creates a new SSL_CTX object as framework to establish TLS/SSL enabled connections.

- *int SSL_CTX_use_certificate_file(SSL_CTX *ctx, char *file, int type)*
  It lets the peer use certificate.

- *int SSL_CTX_use_PrivateKey_file(SSL_CTX *ctx, char *file, int type)*
  It lets the peer use private key..

- *SSL *SSL_new(SSL_CTX *ctx)*
  It creates a new SSL object.

- *int SSL_set_fd(SSL *ssl, int fd)*
  It sets a descriptor to make it under monitoring.

- *int SSL_accept(SSL *ssl)*
  It waits for a TLS/SSL client to initiate a TLS/SSL handshake.

- *int SSL_connect(SSL *ssl)*
  Initiate the TLS/SSL handshake with an TLS/SSL server

- *int SSL_read(SSL *ssl, void *buf, int num)*
  Read bytes from a TLS/SSL connection

- *int SSL_write(SSL *ssl, const void *buf, int num)*
  Write bytes to a TLS/SSL connection

It's necessary to set up TCP connection before establishing SSL connection. When that's done, a SSL connection can be set up like this:

```
ssl = SSL_new (ctx);
CHK_NULL(ssl);
SSL_set_fd (ssl, sd);
ret = SSL_connect (ssl);
if (ret == -1)
{
      SSL_shutdown (ssl);
      close(sd);
      SSL_free(ssl);
      continue;
}
```

And an example showing how to transmit data with this SSL connection:

```
ret = SSL_write (ssl, buf, size+1);
if (ret == -1)
{
      SSL_shutdown (ssl);
      close(sd);
      SSL_free(ssl);
      continue;
}
```

5.1.2. Genx

Genx is a library, written in the C language, for generating XML. Its goals are high performance, a simple and intuitive API, and output that is guaranteed to be well-formed; the output is also guaranteed to be Canonical XML, suitable for use with digital-signature technology. As XML played a key role in our policies distributed, the library was essential.

The following APIs were used:

- *genxWriter genxNew(void * (*alloc)(void * userData, int bytes), void (* dealloc) (void * userData, void * data), void * userData);*
  Creates a new instance of *genxWriter*.

- *genxStatus genxStartDocFile(genxWriter w, FILE * file);*
  Prepares to start writing an XML document, using the provided FILE * stream for output.

- *genxStatus genxEndDocument(genxWriter w);*
  Signals the end of a document.

- *genxStatus genxStartElementLiteral (genxWriter w, constUtf8 xmlns, constUtf8 type);*
  Start writing an element.

- *genxStatus genxEndElement(genxWriter w);*
  Close an element, writing out its end-tag.

- *genxStatus genxAddAttributeLiteral(genxWriter w, constUtf8 xmlns, constUtf8 name, constUtf8 value);*
  Adds an attribute to a just-opened element.

- *genxStatus genxAddText(genxWriter w, constUtf8 start);*
  Write some text into the XML document.

This is a typical example of how to create a simple XML element:

```
genxStartElementLiteral(writer, NULL, "version");
sprintf(temp, "%d", rpPtr->version);
genxAddText(writer, temp);
genxEndElement(writer); /*version*/
```

5.1.3. eXpat

eXpat is an XML parser library written in C. It is a stream-oriented parser in which an application registers handlers for things the parser might find in the XML document (like start tags). It is widely used by Mozilla and other applications. Thus we chose it for XML parsing purpose.

The following APIs were used:

- *XML_Parser XML_ParserCreate(const XML_Char *encoding);*
  Construct a new parser.

- *Void XML_ParserFree(XML_Parser p);*
  Free memory used by the parser.

- *XML_Status XML_Parse(XML_Parser p, const char *s, int len, int isFinal);*
  Parse some more of the document.

- *void XML_SetUserData(XML_Parser p, void *userData);*
  This sets the user data pointer that gets passed to handlers.

- *XML_SetElementHandler(XML_Parser p, XML_StartElementHandler start, XML_EndElementHandler end);*
  Set handlers for start and end tags with one call.

- *XML_SetCharacterDataHandler(XML_Parser p, XML_CharacterDataHandler charhndl)*
  Set a text handler.

- *int XML_GetCurrentLineNumber(XML_Parser p);*
  Return the line number of the position.

- *const XML_LChar * XML_ErrorString(int code);*
  Return a string describing the error corresponding to code.

Below is an excerption showing how to register related functions and data with eXpat:

```
XML_SetUserData(parser, array);
XML_SetElementHandler(parser, StartHandler,EndHandler);
XML_SetCharacterDataHandler(parser,CharacterHandler);
```

After being registered, those functions need to be implemented by programmer to specify how to handle the data parsed from xml input. The *UserData* is used for communication between different handlers.

5.2. Module: XML Parser & Generator

Before we give out the descriptions of each modules we developed in details, the overall structure of the software components is as shown below:

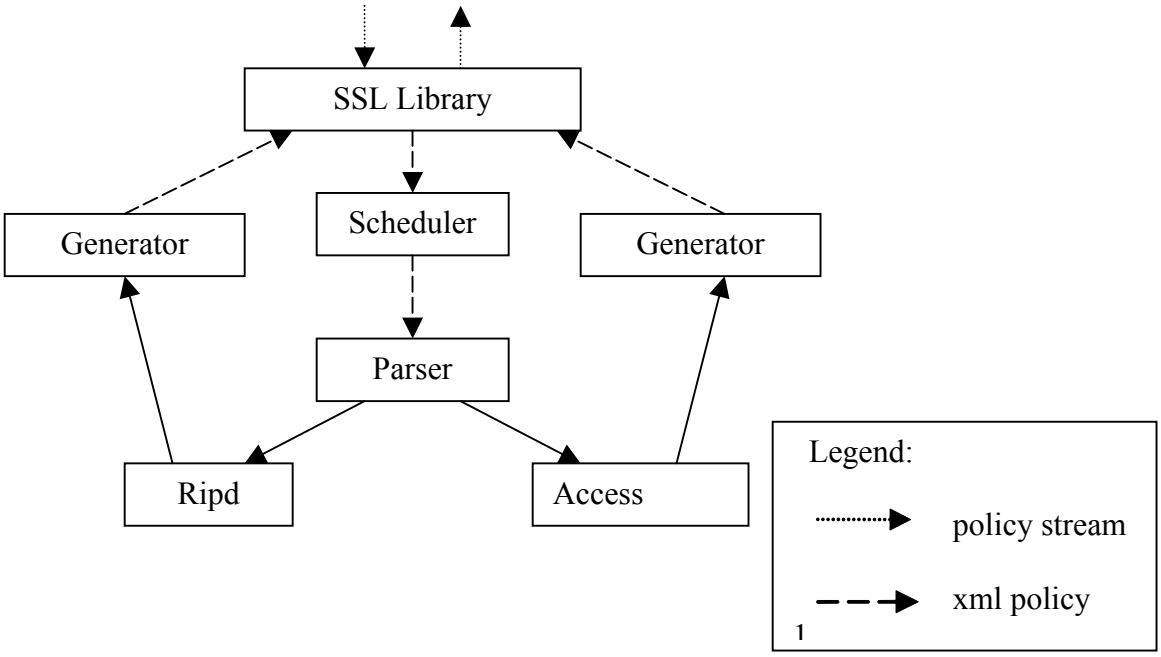


Figure 5.1 overall stucture of software components

5.2.1. Module description

This module aims to provide unified interface for network transmission. Using XML format to transmit data can guarantee the consistency of data and provide good portability. The parser and generator are implemented in API mode. The program that needs them just calls the functions and passes in the appropriate arguments. The third party libraries, Genx & eXpat, are used.

5.2.1.1. Parser

The parser only provides a single entry for all kinds of policy. Firstly, it works out the correct policy type, then the xml policy is parsed in a type-specific fashion, which means every policy has a specific xml tag handler and character handler. The output is an object that includes detailed policy information, which is globally stored.

int ParsePolicy (char* buf, int count)
Description The only function for the user program to invoke. The incoming buffer could contain any type of XML policy.
Arguments: buf: Input buffer which contains policy XML text. count: Length of buf.
Return value: int: Policy type indicator.

Table 5.1 Method description of Parser

5.2.1.2. Generator

There are several generator functions, each for a particular type of policy. Because the group adopted the Genx library, the file descriptor is used by Genx to output the XML content.

int GenerateInitialPolicy (CInitPolicy* ipPtr, FILE* xmlPolicy)
Description Generate XML initial policy from CInitPolicy and put it into file xmlPolicy.
Arguments: ipPtr: Policy object. xmlPolicy: File descriptor to put xml context in.
Return value: int: 0 if successful, otherwise –1.
int GenerateUpdatePolicy (TRipPacket* rpPtr, FILE* xmlPolicy)
Description Generate XML routing update policy from TRipPacket and put it into file xmlPolicy.
Arguments: rpPtr: Policy object. xmlPolicy: File descriptor to put xml context in.
Return value: int: 0 if successful, otherwise –1.
int GenerateAccessPolicy (AccessList* alPtr, FILE* xmlPolicy)
Description Generate XML access control update policy from AccessList and put it into file xmlPolicy.
Arguments: alPtr: Policy object. xmlPolicy: File descriptor to put XML context in.
Return value: int: 0 if successful, otherwise –1.

Table 5.2 Method descriptions of Generator

5.3. Module: Scheduler

5.3.1. Association among Modules

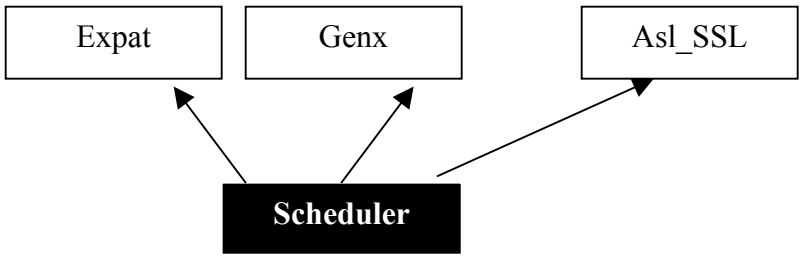


Figure 5.2 Scheduler modules association

5.3.2. Software operation stages

5.3.2.1. Initialisation process

The scheduler's initialisation process includes several aspects:

- ✓ **Networking preparation**
  Set up sockets and secure sockets and prepare buffers. For secure sockets, an encryption key or certificate may be needed.

- ✓ **System preparation**
  This includes global variable initialisation and structure creation. Also, all the components and policy handlers should be started at this step.

- ✓ **Initial policy reception**
  There is a global flag in the scheduler indicates whether the system is initialised or not. If it is not, the scheduler only waits for initial policy and ignores all other policies. When an initial policy arrives, the scheduler configures locally according to the initial policy and enters the next stage.

5.3.2.2. Dispatch process

After initialisation, the scheduler enters a simple dispatching loop. It waits for an incoming policy packet and checks its policy type after parsing it. The scheduler then dispatches the policy to corresponding handler.

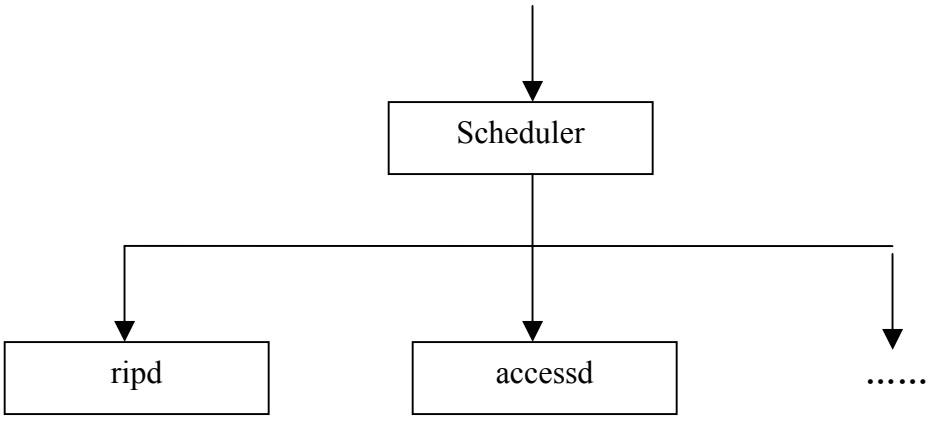


Figure 5.3 Scheduler dispatching to separate policy handlers

5.3.3. Specification

System status The global flag indicates the system state. The system can be either initialised or not initialised.

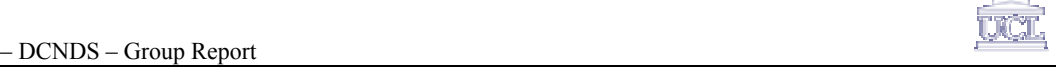
Socket operation The scheduler maintains a single socket for listening and accepting connections. The connections should originate from other node's policy handlers, used to send policy messages. This socket for incoming packet always exists, while outgoing sockets are set up as needed and torn down thereafter.

Daemon startup The scheduler is also the central controller for the system. It is in charge of launching the daemons and handlers. It is also in charge of managing the sub-threads' status. The state of sub-thread can be detachable.

5.4. Module: Virtual Initialisation Server

5.4.1. Module description

This module runs only once each time before the whole VPN environment is set up and aims to collect initial configuration information, such as routing topology and access control information, and distribute it into the VPN nodes group.



- ✓ **Information collection**
  We assume that the configuration file is created in advance and we get the file name. Currently, the file only contains topology information, which is composed of element entry list. Each entry includes a node IP address and the node's neighbour list.

- ✓ **Information distribution**
  After information collection, the server module encapsulates information of each node separately in XML messages and sends them out.

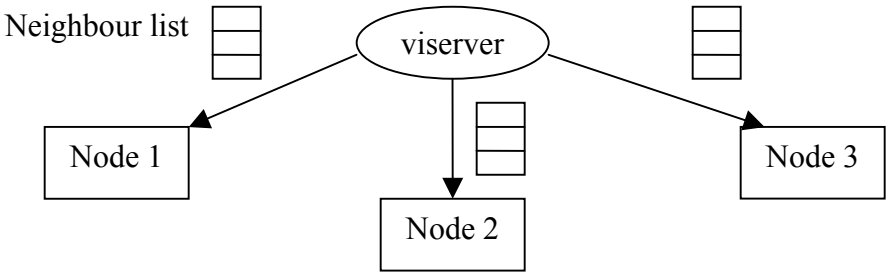


Figure 5.4 XML message distribution between VPN nodes.

- ✓ **Partial Set-up**
  The VPN network may be set up partially, that is, some nodes may fail or may be deactivated at the initial stage so that server cannot connect to them. Under this network condition, the VPN can work by ignoring the failed nodes as long as there are only few nodes failed compared to the total number of nodes.

5.4.2. Input file requirement

Currently, we assume that the initial VPN topology is set up manually and information is stored in a configuration file.

```
<node> <neighbour_num>
<neighbour_1> < neighbour_2> …
<node> <neighbour_num>
<neighbour_1> < neighbour_2> …
…
```

Figure 5.5 Topology configuration file

The file consists of entries, each of which contains the nodes own ip <node> and number of neighbour <neighbour_num> and a neighbour ip list <neighbour_1> <neighbour_2>…

5.5. Policy handler modules

5.5.1. Overview

Policy processing is the major part of the overall system. The effect of policy distribution is shown at each local node where different policies are applied. To achieve maximum independency and extensibility, we designed different policy handlers for different type of policies. The handlers may be implemented in different types, such as threads, dynamic-linked libraries in order to facilitate using.

5.5.2. Module: Routing Policy Handler

The routing policy handler component is an independent component that is in charge of sending out self routing information and processing incoming routing information. It is subject to RIPv1 protocol.

5.5.2.1. Workflow description

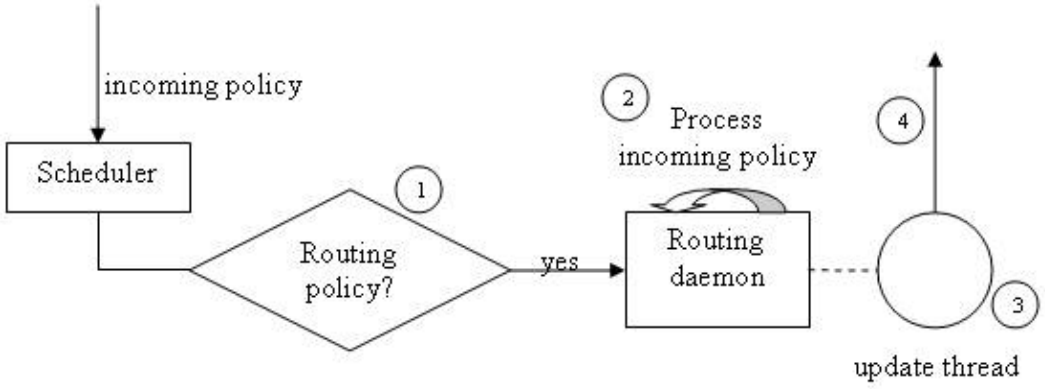


Figure 5.6 Routing policy processing path

1) The incoming routing policy first reaches the scheduler and is recognised by the scheduler. The scheduler calls the interface of the routing daemon and passes the policy object to the routing daemon.

2) The main thread of routing daemon processes the policy and updates local routing table.

3) There are also several affiliated threads which generate outgoing routing policies. The policy includes information about the local routing table. The threads send the policies regularly according to the RIP protocol specification.

5.5.2.2. Function specification

*void rip_response_process (TRipPacket *aPacket, addr_t aFrom)*
Description This is the only function exposed to scheduler. The scheduler invokes it to pass the routing update packet to the handler.
Arguments: aPacket: The routing update packet received. aFrom: Source IP address from which the packet is sent.

Table 5.3 Method description of RIPd

5.5.2.3. Process independence

To achieve maximum process independence, a good implementation aims to make the routing daemon a separate process that only communicates with the scheduler through a simple interface. The interface could be a function or better be some mechanism like network message.

The routing policy handler may use a real 3rd party routing daemon internally. It just needs to fill in a routing update packet with content of the policy object and pass it to the daemon. This is shown below.

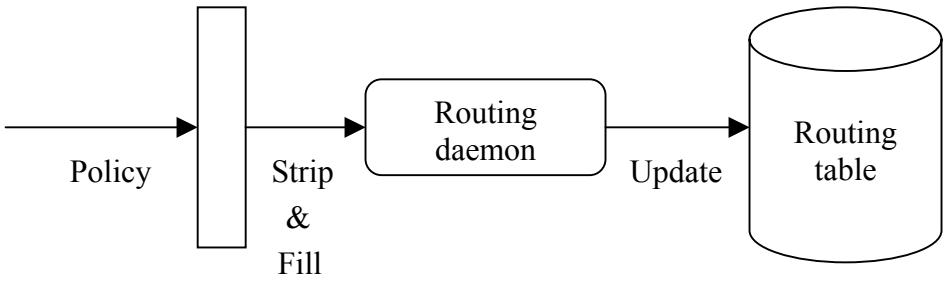


Figure 5.7 Independent process routing update example

5.5.3. Module: Access Control Policy Handler

5.5.3.1. Access control policy

The access control policy handler processes access control policies. This type of policy specifies the access permission for other nodes. It may deny or allow the transmission and reception of packets from certain nodes. Each policy structure contains a subject IP address and two lists; one deny list and one allow list. Packets from node in the deny list will be denied. Packets from node in the allow list can be accepted. Packets from a node that is in neither of the lists will be accepted by default.

```
struct AccessList
{
        addr_t subject;
        std::vector<addr_t> allow_list;
        std::vector<addr_t> deny_list;
};
```

Figure 5.8 Access control policy structure

5.5.3.2. Specification

Internally, ACH (Access Control policy Handler) maintains a table of ACP (Access Control Policies).

void AccessInitial (const CInitPolicy& initP)
Description: Initialize local ACP table from initial policy.
Arguments: initP: the initial policy object
void ReceiveAccessPolicy (const AccessList& al)
Description The function called by scheduler when there is an incoming ACP. ACH uses it to update local ACP table.
Arguments: al: the ACP object
void SendAccessPolicy (addr_t own)
Description Send own ACP to all the neighbours.
Arguments: own: Own IP address

Table 5.4 Method descriptions of Access Control

5.6. Module: Communication Monitor

5.6.1. Module description

The monitor module could be used for administration or debug purpose. It displays a policy communication activities at real time by drawing a line connecting two dots to represent a connection between two nodes. The module is implemented using Java. It communicates with other software modules via UDP enabling messages to be loosely coupled. This allows for software portability.

5.6.2. Message format

Policy Information Message contains policy information such as policy type, source IP address and destination IP address. Class *information* defines all the related attributes, as below:

Information
int iType
int iMinorType
int iSrcIp
int iDestIp

Table 5.5 Class Information

Currently available policy types are:

Version	Type
1.0	Initial policy
2.0	Routing update policy
3.0	Access control policy

Table 5.6 Policy types

5.6.3. Module components

There are three main components in this module: *Monitor*, *MonitorNet* and *MonitorFrame*:

- **Monitor**
  Monitor is used for information storage and acts as a coordinator. It also defines some facility classes. When *MonitorNet* receives Policy Information Message, it stores the messages in *Monitor*. The *Monitor* component also provides a mechanism to update the view of *MonitorFrame* in real time.

- **MonitorNet**
  MonitorNet is a network component that listens to the network in another thread waiting for incoming packets.

- **MonitorFrame**
  MonitorFrame is a software component which gives user a view of what is happening visually during VPN policy exchanges.

5.6.4. Component association and interaction

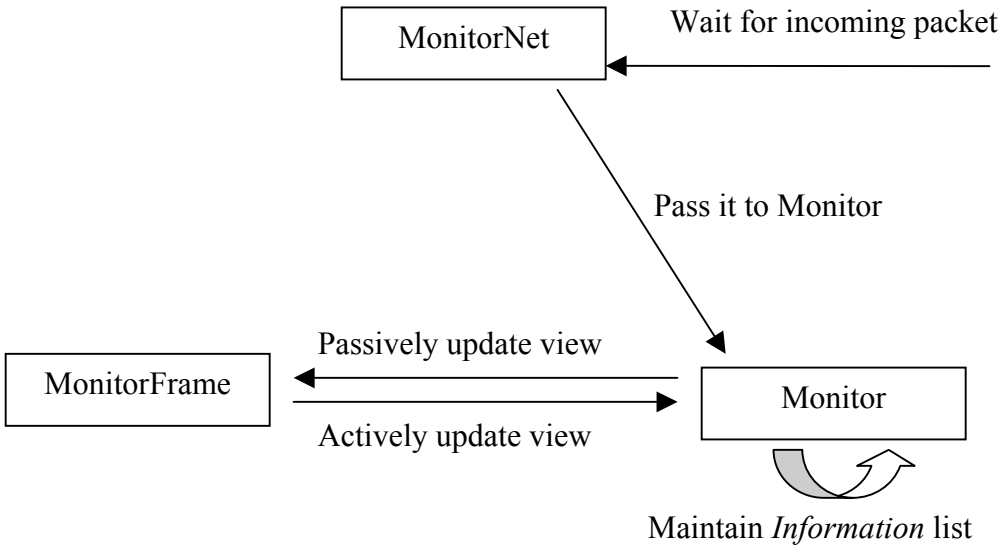


Figure 5.9 Illustration of component interaction for the communication monitor

The three components are designed in a Model-View-Controller pattern. *MonitorFrame* is the view and *Monitor* is the controller. The data is also stored in *Monitor*. When the controller *Monitor* starts to run, it creates *MonitorFrame* and *MonitorNet* objects. It lets the *MonitorNet* thread start and *MonitorFrame* show. *MonitorNet* keeps receiving communication reports and puts them into *Monitor's* report list. This behaviour also makes *Monitor* update the view as necessary while *MonitorFrame* regularly updates its view. The update action is to draw lines, each of which represents an entry of the report list.

5.6.5. Status of *MonitorNet*

The status of *MonitorNet* is placed in *Monitor*. The status should be changed according to situation. *MonitorNet* has to check its status within each loop so that it can handle different events, such as exit and halt, correctly.

5.6.6. *InfoEntry* & associated timer

The *Monitor* class has a member vector of *InfoEntry* objects. *InfoEntry* class has a member of *Information* object and a member of *Timer* object. Because the *Information*

object is only used to show communication situations on the screen, an object is useful only for a short time. An associated *Timer* object removes the *Information* object from the vector when the timer expires. The expiration time can be a fixed.

5.6.7. Updating the view

Updating the view of GUI is achieved in two ways:
Firstly, *Monitor* has a method *UpdateView()* to update the view, that is, to let *MonitorFrame* repaint itself.
Secondly, In *MonitorFrame's* method *paint(),* it gets the *InfoEntry* vector from *Monitor* and draw lines for each entry.

5.6.8. MonitorNet

MonitorNet does its work in a while loop. If its status is normal, it waits for incoming UDP packet for a fixed time interval, and loops again if the socket timeouts. Otherwise, it

has received a packet and adds an entry at the end of the *InfoEntry* vector. It checks the status once each loop to ensure that it terminates properly.

5.6.9. MonitorFrame

This is a typical *JFrame* derived class. It overrides the *paint()* method to display the communication situation on the screen.

6. Testing

Testing is very important during the software development process. The group tested our software thoroughly in order to make that our code functions in the manner in which we expect it to do. Software testing can be split into four separate phases:

1) Modeling the software's environment
2) Selecting test scenarios
3) Running and evaluating test scenarios
4) Measuring testing progress

The scope of the first phase involves modeling the software's environment which encompasses a mixture of separate tests. These comprise of unit tests, integration tests and overall system tests.

Our testing plan followed that structured approach. First we completed unit testing on the most important components to make sure they can worked correctly as individual components. Once unit testing was complete, integration testing was started combining several components together to make sure the communication between them is working correctly. Finally we began overall system testing (simulation) to make sure the whole software fulfills our design expectations.

6.1. Unit testing

Unit testing tests individual software components or a collection of components. Testers define the input domain for the units in question and ignore the rest of the system. Unit testing sometimes requires the construction of throwaway driver code and stubs and is often performed in a debugger.

Although XP requires unit testing for every class/ function, due to time constraints, we chose the most important functions in each module to be tested. We decided to write testing code by ourselves, not using the ready-to-use unit testing tools. That way we had much tighter control of the overall testing process as well as information to be displayed as a result.

The software has been divided into several sections and over page the corresponding unit testing sections will be introduced.

6.1.1. SSL link up test:

- **Testing file:**
  serv.exe, cli.exe, server_req.pem, server_key.pem

- **Function tested:**
  SSL connection

- **Testing approach:**
  The serv.exe program is run first followed by the cli.exe. The duration of the SSL establishment is then recorded and logged.

6.1.2. Ripd test:

- **Testing file:**
  ripd_testing.cpp

- **Function tested:**
  RTable rip_initial_process (CInitPolicy);
  TRipPacket rip_output_packet (addr_t);*

- **Test approach:**
  Make up a neighbor list and send it to rip_initial_process to initialise routing table. Then print out the initialized routing table to check whether it has been initialized correctly. Use rip_output_packet to create rte structure, based on the content of initialized routing table, and then print out the rte structure content to check whether the filling-in process is correct.

6.1.3. Generator test:

- **Testing file:**
  generator_test.c

- **Functions tested:**
  int GenerateInitialPolicy (CInitPolicy, FILE*);*

- **Test approach:**
  As GenerateInitialPolicy, GenerateUpdatePolicy and GenerateAccessPolicy are very similar, we choose to test just one of them. A simple CInitPolicy is made up and then sent to GenerateInitialPolicy function of generator.c. If the task is finished successfully, "Done!" will be printed on the screen and an XML file (here named "initp.xml") should be created which should contain the initial policy created based on the CInitPolicy object. We can check the content of that XML file to make sure the right XML policy is generated.

6.1.4. Parser test:

- **Testing file:**
  parser_test.c

- **Functions tested:**
  int ParsePolicy(char , int);*
  *static void XMLCALL StartHandler (void *, const char *, const char **)*
  *static void SetDepthForInitialPolicy (void *, const char *)*
  *static void XMLCALL CharacterHandler (void *, const XML_Char *, int)*
  *static void HandleInitialPolicy (void *, const XML_Char *, int)*
  *static void XMLCALL EndHandler (void *, const char *)*

- **Test approach:**
  The XML initial policy generated in the generator testing, which has been proved to be consistent to the definition, is passed to the ParsePolicy function of parser.c. Due to the structure of eXpat parser, a series of functions are also involved in order to parse this xml file. If the XML initial policy is correctly parsed, a return value will indicate the type of the policy (Initial Policy, Update Policy, Access Policy). During the test, some additional "printfs" were added inside parser to examine the element contents that had been parsed was correct and the CInitPolicy object was correctly filled in. (Those additional printfs were deleted after testing.)

6.1.5. Access control test:

The access control module was added in during the second iteration of the design cycle. The *parser*, *generator* and *scheduler* all needed to be modified to include this feature. However, as previously mentioned, the *parser* and *generator* all passed unit testing successfully, the modification just needed to follow the already established structure. The access control daemon is relatively simple and straightforward. As a result, no unit testing is carried out on this part, however, integration testing has been done, which will be covered next.

6.2. Integration testing

Integration testing tests multiple components that have each received prior and separate unit testing. In general, the focus is on the subset of the domain that represents communication between the components.

6.2.1. First iteration integration testing

After unit testing, the group was satisfied that SSL links could be set up successfully and data can be transmitted correctly. Because the transmission part is relatively easy to control during integration testing, we focused on the integration between parser, generator and ripd. Communication between all these components is located in the system testing section.

A diagram showing the integration testing flow is given in the next page:

We designed a four-step integration testing plan based on the structure of our software.

Step 1. Generate the initial policy. This part is the same as the generator test done in the unit testing (generator_test.c). However, as it is important to the integrity of the integration testing, we also include the test here. The result of this step is generating an XML file that includes the initial policy. As we ignore network communications during this test, we conclude that the neighbour receives the XML policy.

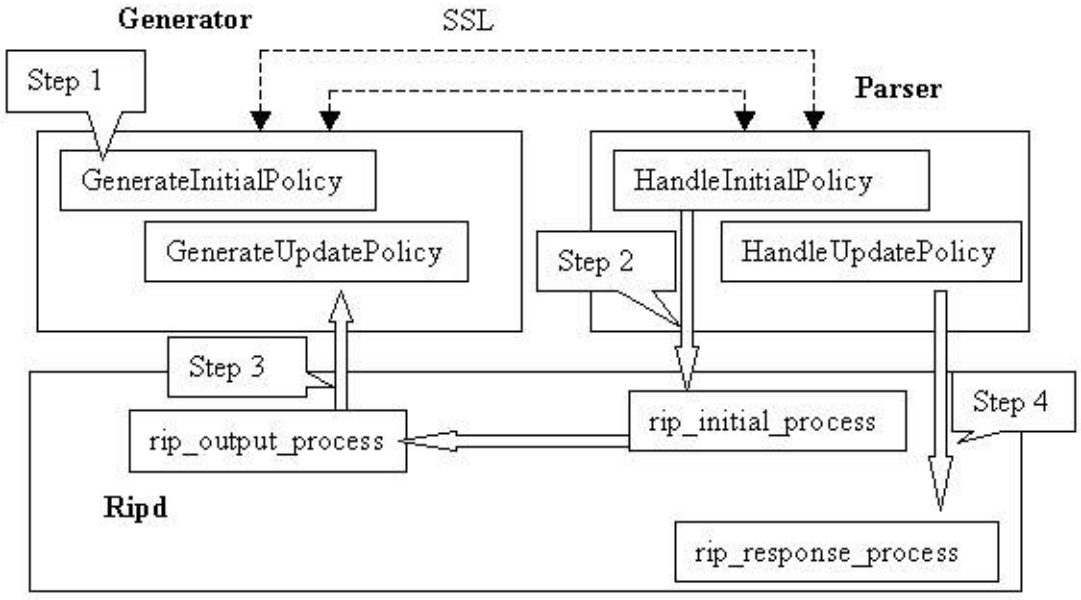


Figure 6.1 Integration Testing Plan Demonstration

Step 2 The testing program (test.c) will read the (initial policy) XML file that generated in step 1 to simulate receiving the initial policy. It passes the XML policy to parser (through invoking *ParsePolicy* function), in which function *HandleInitialPolicy* will be responsible for the actual handling of initial policy. (Although other functions will also be involved such as *SetDepthForInitialPolicy* etc.) Next, the testing program will initialise the local routing table based on the *CInitPolicy* object content prepared by parser, by invoking *rip_initial_process*. If successful, the content of the initialized routing table will be displayed on the screen.

Step 3 Since the routing table has already been initialised, we can send update policy to neighbors. In the testing program (test1.c), the *rip_output_process* function is invoked, in which *GererateUpdatePolicy* is invoked to generate update policy xml files based on the content of routing table. Also, *rip_output_process* is responsible for setting up SSL connections to its neighbors to send the created update policy. Again, in this testing the group ignored SSL transmissions, so the correct result should be the successful creation of several (depending on number

of neighbors) update policy XML files into local file storage, and the content of those XML files can be examined to make sure related functions work properly.

Step 4 Once again, we suppose those update policies have been successfully transmitted through SSL connections, so in the testing program (test2.c), one of those XML files is read and passed to the *ParsePolicy* function where *HandleUpdatePolicy* is going to prepare a TRipPacket object based on the content of the XML policy. After that, *rip_response_process* will maintain the local routing table according to the TRipPacket passed to it. If successful, the updated routing table will be displayed on the screen.

6.2.2. Second iteration integration testing

The previous section on integration testing applied to our first software prototype. In the second prototype, access control functionality was added so a small integration testing section on the communication between access control component, parser and generator was required. As the procedures of access control and rip are similar, so we can just adapt the testing design of the first prototype for the testing of access control. This testing is consists of two parts, part one roughly corresponds to the *Step 1*, *Step 2* and *Step 3* of the first integration testing, and part two corresponds to the *Step 4*.

The first part testing program (test_a.c) gets the CInitPolicy object from the usual routine (xml input -> HandleInitialPolicy), then uses the CInitPolicy object to initialize the local access table by *AccessInitial* function. Once successful, the initialised access table will be displayed. In the next stage, we made the process slightly simpler by just using the initialised access table (which only has empty *allow_list* and *deny_list*) as an updated access table. The SendAccessPolicy now creates an updated access policy by invoking the *GenerateAccessPolicy* function using the initialised access table and the content of the updated access policy we checked to make sure all functions work properly.

The second part of the testing program (test4.c) will use the access control policy files generated by the first part testing program (test_a.c) as input, and pass this XML policy to the *HandleAccessPolicy* function of *parser* to get the AccessList structure. Then with this AccessList structure, the *ReceiveAccessPolicy* function of access daemon (access.c) can update/maintain the local *access_table*.

6.3. Overall system testing

Overall system testing tests a collection of components that constitutes a deliverable product. Usually, the entire domain must be considered to satisfy the criteria for a system test. In system testing, we integrate every component and follow the full procedure test.

The procedure can be expressed as below:

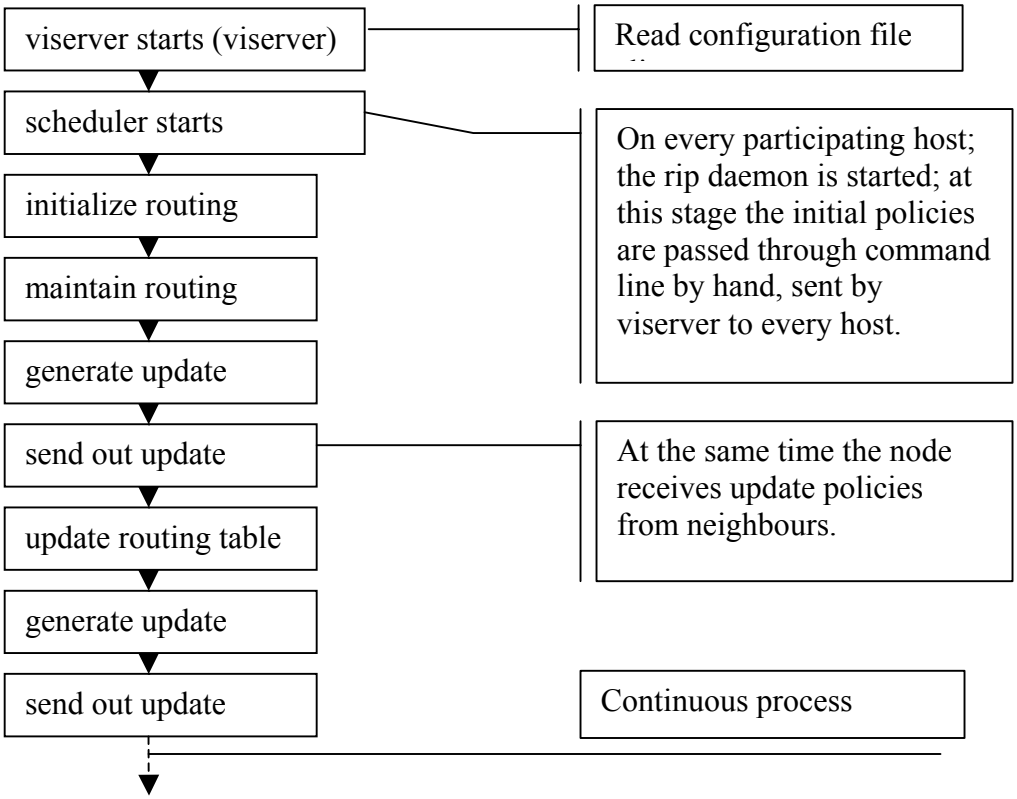


Figure 6.2 System testing flow

6.3.1. Testing environment

We chose a simple but generic topology for system testing which contains 5 hosts. The topology is shown below.

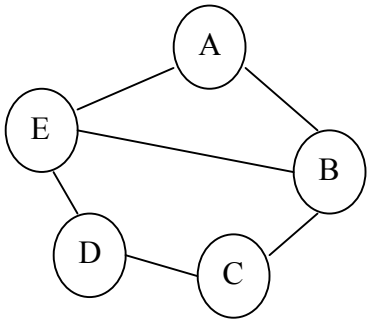


Figure 6.3 Testing Topology Demonstration

Host A is responsible for running viserver. While in practice there will be a certain rule to decide which participant will take this responsibility (perhaps the machine with the lowest IP address).

We have a dedicated demonstration program "Monitor" to monitor the execution of the software. Its main function is to show the connections and communication between VPN hosts. Upon reading topology information from a configuration file (which should be the same as the one that viserver gets), the monitor draws several circles on the screen each representing a host. When a SSL connection is established (no matter for what reason, initial policy, update policy or payload traffic), a line will be drawn between the corresponding two circles indicating the communication activity.

6.3.2. Testing result

Fragments of the testing output (result) with brief explanation are shown below.

```
enter server cert
cert
key
pwd
Start routing daemon...0
Waiting for initial policy...
routing daemon routine 1887508
```

rip daemon started and wait for initial policy.

```
TCP socket accept a new socket: 4
Entering sub thread...
SSL_read: 378
<policy
type="initial"><port>80</port><routing_topology><neighbor_num>3</neighbor_num>
<neighbor><neighbor_id>0</neighbor_id><ip>128.16.9.165</ip></neighbor><neighbor
><neighbor_id>1</neighbor_id><ip>128.16.9.178</ip></neighbor><neighbor><neighb
or_id>1</neighbor_id><ip>128.16.9.182</ip></neighbor></routing_topology></policy>
Calling xml parser...
initial policy got.
Neighbour 0: 2148534693
Neighbour 1: 2148534706
Neighbour 2: 2148534710
```

Initial policy received and parsed.

```
RIP routing table initialized!
            ip              metric  nexthop                 signal
Routing table:  128.16.9.165  1       128.16.9.165    0
Routing table:  128.16.9.178  1       128.16.9.178    0
Routing table:  128.16.9.182  1       128.16.9.182    0
Leaving sub thread...
```

Routing table initialized according to initial policy.

```
TCP socket accept a new socket: 4
Entering sub thread...
SSL_read: 214
<policy
type="routing_update"><protocol>RIP</protocol><version>1</version><cmd>2</cmd>
<entry_num>1</entry_num><route_entry><addr_family>2</addr_family><ip>128.16.9.
178</ip><metric>1</metric></route_entry></policy>
Calling xml parser...
routing policy got.
**********************************
rte size is : 1
```

Update policy received and parsed.

Table updated! Case 1: new entry added
Leaving sub thread...
routing daemon routine 1689321
Neighbor list:
128.16.9.165
128.16.9.178
128.16.9.182
Routing table:

	ip	metric	nexthop	signal
Routing table:	128.16.9.165	1	128.16.9.165	1
Routing table:	128.16.9.178	1	128.16.9.178	1
Routing table:	128.16.9.182	1	128.16.9.182	1
Routing table:	128.16.9.177	2	128.16.9.182	1

Outgoing packet's rte size: 3

From information provided by update policy, new entry is added into routing table. Routing table is maintained. And new update policy is created (outgoing packet's rte size:3).

6.3.3. Problem solving

During the testing process, we encountered a lot of problems. However, the unit testing and integration testing that we did helped a lot to reduce the number of possible errors and narrow the scope of a problem. Here are some typical examples of the problems that the group faced and how we solved them.

- **Problem 1**

Problem description: Segmentation fault
Reason: Used wrong function to delete vector after using it
Solution: Use *vector::clear()* instead of *vector::empty()*

- **Problem 2**

Problem description:	The contents and lengths of XML update policies were not correct.
Reason:	All hosts ran the program in the same directory, and also the name of the created XML update policies by each host are the same, so when the hosts write, they write to exactly the same file. This caused the file length to increase and the content to be incorrect.
Solution:	Prepare different executing directories for every host.

- **Problem 3**

Problem description:	Repeated content in one policy file.
Reason:	File not closed after writing to it, the next time the file is opened the file data is appended.
Solution:	Close the file and re-open for the next operation.

7. Evaluation and future work

The group project lasted for approximately eighteen weeks and in that time a lot was achieved. The project was a completely collective effort bringing together a group of individuals, each with unique skills, qualities and attributes. The evaluation section will be broken down into various subsections relating to an appraisal of the following project components; evaluation of research methods, evaluation of project objectives, evaluation of project management techniques and evaluation of software design, implementation and testing. A security evaluation will also be included. The final section relates to future work and discusses the other directions that the project could move in.

7.1. Evaluation of research methods

Initially, the group carried out detailed research in many different areas of computer network security. Different types of VPNs were examined in great detail to familiarise ourselves with developments within the academic and commercial arenas. The group worked and communicated well in the early stages of the project often sharing documents and web links with the intention of building up a solid knowledge base from which to work from. From a strategic point of view our research was well organised yet in hindsight it would have been better if we had searched on a more focused area saving us valuable time examining unrelated material. On the other hand, reading such a wide variety of written material helped us to examine our problem from different angles and identify some current solutions to specific security problems. Because our research area is fairly new within academic circles we did not utilise our research time in the most appropriate way. The project group made use of popular search engines such as Google along with journal websites such as Citeseer to obtain much of their initial research documents. Initial research into IPSec was quite successful. However, the group soon realised that IPSec was a very complicated set of VPN standards. The group then began to refine our search of other VPN technologies that were emerging over the last year. SSL VPNs were considered new so the group refocused some of our research efforts into getting a better understanding of SSL. This research was particularly successful once the group realised that the technology was well documented and developed yet considerably simpler. Research techniques were considered successful overall with good communication between the project group members. Research occurred throughout most of the project with a bulk of it being carried out in the early stages. Our ever changing

project objectives meant that good research methods were required in order to obtain the information we required for our changing software design. The team delivered well on the research of the project overall. If we were to work on the project again we would make our research even more focused and try harder to keep to our work schedule.

7.2. Evaluation of project objectives

The project group composed a list of three main objectives to be completed by the end of the project. In many respects we believe we did an excellent job in addressing each of the main project objectives as they changed frequently at the beginning as the project developed. As already mentioned, it was clear from the outset that attempting to modify a particularly complex VPN standard was out of the question due to time constrictions imposed by the deadline. The group quickly realised that existing SSL technology was the way forward and hastily began work on a working prototype. The group proceeded to design and implement our initial software daemons within a month. This gave the group a lot of extra confidence and boosted overall morale. The first iteration of the software design included many features which were well aligned with the project objectives. The group was also successful at designing a suitable policy structure that could be expanded to add new types of security policies. This helped to future proof the system. The routing protocol used in the SSL VPN was also implemented fully and functions extremely well. The group achieved all of their three main objectives within the specified time frame and this could be seen as a major project success.

7.3. Evaluation of project management techniques

Project management played a particularly important role in shaping how the project was run and how the project ultimately developed from a brain storming session right the way through to a finished product. The group had researched good project management techniques prior to the start date and we felt confident that we could use an effective modern project management strategy to achieve of project objectives on time and within specification. The group realised that good project management meant constant planning and ongoing revision on a regular basis. Because many of us were unfamiliar to group projects the learning curve was particularly steep. The group approached project management with a small amount of skepticism as there were many different control

documents and checks in place to make sure that the project succeeded. Many members felt this was excessive. However, within the first few weeks of the project getting

underway we realised just how important and necessary such documentation was. The group applied project management techniques particularly well. Our work breakdown structure was accurate and we tried to follow an Extreme Programming methodology throughout the course of the project. Schedules and time frames slipped considerably and the group often missed specific milestone deadlines. However we recovered well once we addressed the cause of the delays and implemented contingency plans and schedule changes.

Good project management planning also meant that our project workload was well distributed. There were incidents where one member of the group would be slightly overworked in a particular section of the project. For example, Ying Shi, our lead programmer was slightly overworked during the implementation phase while Kai felt under pressure producing a design proposal. Overall, the work load was split fairly evenly and well managed. Any problems were handled efficiently by assigning another member of the team to provide an assisting role.

Extreme programming techniques were used successfully throughout the project and pair programming was common practice. The tests that the project group prepared and wrote were of a bespoke nature. On the whole the group embraced the XP methodology well often going out of their way to make sure that coding was implemented to match current project objectives and integration testing was frequent and well controlled.

Risk management techniques employed by the group helped to mitigate the potential risks which could have severely hampered the group's efforts. During the beginning of May, Hyo Sim Moon quit the course leaving the project a person short. Because we had anticipated the risk, countermeasures were already in place to distribute the workload between the remaining team members. This can be seen as a successful response to a serious problem. The group also anticipated other major risks relating to loss of data and software. Risk mitigation involved backing up their work via CVS on a daily basis to prevent versioning problems and loss of data.

7.4. Evaluation of design, implementation and testing

The software design that the group originally produced contained too many assumptions and lacked technical clarity. Gradually over time a good final design document was drafted which involved every member of the team. Because every person had input in the

design of the software, the result was a well thought out document that fulfilled the project objectives. The involvement of every single person in the design process can also be seen as an advantage providing each group member with specific detailed knowledge about what was being created. This helped to create a shared vision for the project group. Technically, our software design had numerous strengths and weaknesses. The design was well thought out and achieved each of the project objectives. However, there were some assumptions that had to be made regarding the level of trust in the VPN. The topology of the VPN needed to be known beforehand and potential VPN nodes needed to listen on particular port number before they were connected to the VPN overlay network. This resulted in a lack of auto-configuration in the design because the network nodes had to be prepared beforehand defeating the purpose of providing a dynamic VPN join mechanism.

Implementation was very successful due to the strong design effort mentioned above. The group members worked in teams on separate software modules often communicating on a daily basis. Because our software design was clear, implementation went particularly smoothly albeit slightly delayed. Software bugs were dealt with efficiently and because the project group had done their research well, they were familiar with the software libraries available to them. On the whole the implementation of software can be considered successful.

Testing was carried out in a thorough and efficient manner which resulted in a stable implementation. The group designed their own testing programs which were used to validate program functions and interaction. While the group did not test every single aspect of the software, a conscious effort was made to test the most important functions with a high degree of success.

7.5. Security Evaluation

With our software prototype now complete the group can begin a critical evaluation of the security aspects of our project. Within the design section of this report, the group highlighted numerous potential security issues that could cause our software to become vulnerable or comprised by different kinds of attack. Many of these vulnerabilities still propose some kind of threat to the system because the group did not have sufficient time to make the software prototype more robust.

The project group spent a lot of time actually making the software work within specification and because our software iterations were quite tight it meant that no time was set aside to enhance the security features of the software. In hindsight, it would have been more prudent to allot more time to develop mechanisms used to counter threats and if the project group were to complete the project over again then a greater effort would have been made in addressing these issues.

The software itself is well structured and coded and is not vulnerable to any kind of buffer overrun attacks however the question of DOS and man in the middle attacks still exists. It is easily possible for an attacker to port scan a VPN host and find out which port it is actively listening on. Once this is discovered, an attacker can launch a DOS or DDOS attack effectively interrupting VPN communications between VPN endpoints.

Much like many other software implementations, the software is also vulnerable to SSL man in the middle attacks and unfortunately this type of attack is very difficult to defend against. There is nothing to stop a third party attempting to modify packets either secure or insecure as they transit the network. The goal may not be the modification of packets but the retransmission of packets that fail a hash function. As it stands the project software is just as vulnerable to man in the middle attacks as any other software that currently exists.

The matter of an encrypted file system for storing XML policies was noted and discussed in detail once the group had completed the first software design iteration. The group was well aware of the importance of encrypting policies as they resided on system hosts. Without such encryption, an attacker can read the files to gain inside information about the topology of the VPN from a node perspective. An active attack is also possible thereby allowing an attacker the opportunity to deliberately modify an XML policy to

introduce some kind of false topology information into the VPN, perhaps to include hosts that are not fully secure or to misdirect traffic flows in some way. Each of the above problems poses absolutely huge security problems and the group was well aware of the importance of encryption in our software implementation. Unfortunately, time constrictions were such that we simply ran out of project time to implement the encrypted file system feature into the second software iteration but we also recognise that if we had more time then there is no reason why this work could not have been carried out.

The mechanism used for the distribution of our policies is secure. Using SSL, policies and data routed between VPN endpoints is confidential and integrity is guaranteed. Although the availability of data cannot be guaranteed due to DOS threats mentioned earlier, the chances of losing availability is statistically very slim.

The group has also made excellent progress is designing a security based policy infrastructure that is both flexible and extensible. The system can be upgraded to support future security based policies as they are designed for a specific purpose.

On the whole the project group made a good attempt at implementing basic security features in our software prototype. However, there was room for further improvement and given extra time, many of the threats listed above could have been mitigated or removed entirely.

7.6. Future work

With our project finished, the group can now look at possible future follow up work which can improve and expand our original software design and move our research forward in different directions.

7.6.1. Additional policies

The group managed to implement routing policies and security access control lists within our software however there is potential to expand the types of policies even further to encompass aspects of Quality of Service (QOS) mechanisms and high level security policies. These could be provided as optional services that could be used at the network administrator's request. QOS guarantees across overlay VPNs is a particular exciting area to examine further.

7.6.2. Further development of the API

The development of an API that can be used by other applications for data transmission occurred near the end of the project and was only a first attempt at integrating our software framework with other program modules. Further development of this API is a particularly interesting area for further study. At present the API is very basic at there is much more room for added functionality.

7.6.3. File encryption

As well as receiving the initial policy, every node will periodically generate some update policies that contain routing update information to be sent to all its neighbors. Those update policies are stored in plain XML on the native file system of the host. The operating system was Solaris, known for its excellent security mechanisms. However, despite the fact that the file should not be accessible to individuals or groups who do not have the correct security privileges, it would be prudent that any security related XML file be encrypted in some fashion making the security information of the VPN even more secure. The Open SSL library that we use can offer basic encryption function therefore we are going to use it to encrypt those xml files locally on the native file system. When the file needs to be transmitted in is possible to write code that will decrypt the file prior to transmission over a secure channel. There is a small period of insecurity during this phase as the file resides in system memory on the host. However we believe that future work in this area will help to bring a significant security improvement to the system.

7.6.4. Additional routing protocol support

The support of other distance vector and link-state routing protocols would be an excellent improvement to the system which would involve creating slightly new XML structures. Giving the system administrator a choice on what type of routing protocol to use on the VPN would create flexibility allow the system to be customised to suit different network environments.

7.6.5. Graphical user interface support

Providing a GUI would allow the software to become much more user friendly. Configuration of the VPN could then be completed less skilled network administrators. A GUI also simplifies application use and allows for simpler fault finding.

7.6.6. Authentication

The authentication of users and hosts could be integrated more tightly with the operating system effectively providing an additional layer of security determining what privileges a user has in creating SSL tunnels and distributing policies.

8 Conclusion

The group set out with the goal of designing a set of software daemons that could incorporate dynamic routing across VPN tunnel endpoints. Other project objectives included a policy distribution mechanism and the creation of a routing protocol. All three objectives played an important part of the design process. The group worked hard to achieve a stable implementation that incorporated all of the above features.

The group has tried to take a new approach to designing dynamically routed VPN software using SSL libraries. While the system has numerous issues that need to be addressed, the ideas produced during the lifetime of this project merit further study and work.

There will always be security vulnerabilities in the best VPN software and the group is well aware that much more work is required for an application to achieve a secure status in a production environment.

Partially meshed VPN tunnels and dynamic routing in overlay networks has an exciting future within the Internet and the use of policy driven networks will no doubt create a new breed of applications that can make use of secure network overlays.

Appendix A: Project management

1. Team constitution and structure

As the project was particularly small the group decided that each team member would be given specific focused roles through out the course of the project. Initially the project team consisted of five members however Hyo Sim decided to withdraw from the course in May. The group moved quickly to make sure her responsibilities were distributed to the rest of the group equally.

Kai Duan was designated as the group coordinator and was responsible for the overseeing the projects overall progress. This was a unanimous decision for the group based upon the personal and professional qualities that Kai possessed. Kai also spent a large portion of the project helping to design the software and was responsible for coding the routing module.

Hyo Sim was assigned to documentation and agenda writing early on in the project. During weekly team meetings Hyo Sim was instrumental in ensuring that group communications were well documented and up to date. Hyo Simm would have played an active role in the design of the software as well as assisting in software testing if she had remained on the course.

Oliver Priest was responsible for the group's initial research efforts. The group project was focused on a particularly new area of research and Oliver played an active role collecting and disseminating the vast amount of data that the group collected from white papers, journals, books and websites. As lead researcher he was responsible he deciding what information was important enough for the group to read and take an interest in. Oliver also played a technical role in many of the design decisions the group took over the course of the project. As well as assisting in the software design, Oliver maintained the website and handled the coordination of the group report overseeing the overall layout and proof reading.

Ying Shi was responsible for the main design of the software prototype and also acted as the lead programmer for the project. Ying Shi is easily the strongest programmer in the group and one of the most highly respected coders on the DCNDS course. The group felt

it was only natural for him to oversee most of the coding for the project. One of Ying Shi's main responsibilities was ensuring that the various software modules that the group coded worked together as a single application. Ying Shi is also the author of the system scheduler module.

Qian Yang was responsible for the designing software for the project. In addition to this she was responsible for coding the parsing module and was the lead software tester within the group. Qian Yang designed and wrote many of the software test programs that were used to test the software modules. Qian Yang also maintained the projects web log online and wrote the meeting agendas and minutes.

Below is a summary list of each individual's responsibilities:

- ✓ **Kai Duan**
  Responsible for software design, coding, group coordinator.

- ✓ **Hyo Sim Moon (quit)**
  Responsible for software design, agenda writing, software testing, and documentation.

- ✓ **Oliver Priest**
  Responsible for software design, project Webmaster, documentation, lead researcher.

- ✓ **Ying Shi**
  Responsible for software design, lead programmer.

- ✓ **Qian Yang**
  Responsible for software designing, coding, testing and documentation.

2. Work breakdown structure

Please see the work breakdown structure below that lists the different work modules that the project was broken into. All parts were divided equally between the various project members often with one member in charge of specific parts of the project. Sometimes more than one individual was working on a specific area of the project.

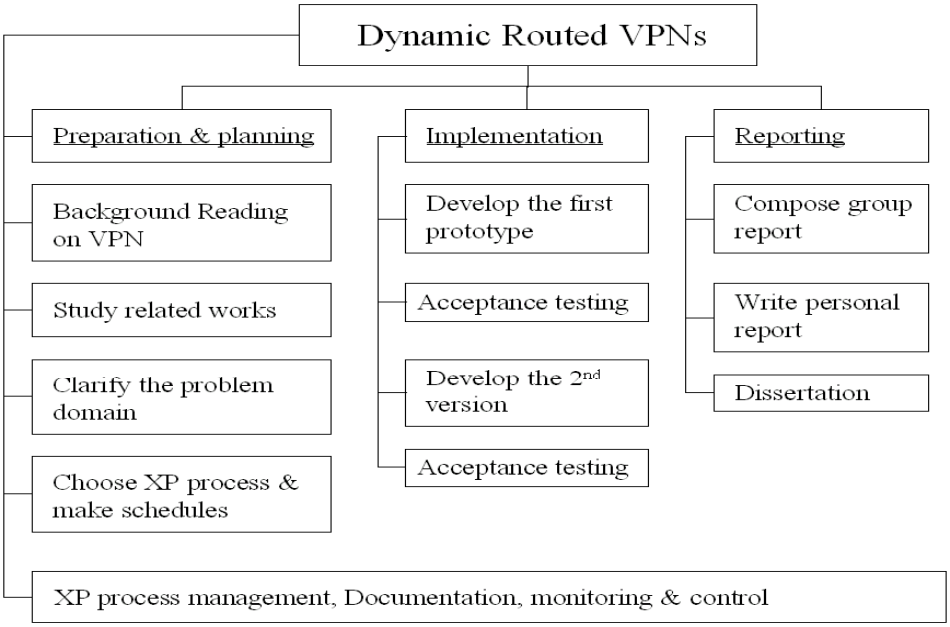


Figure A.1: WBS structure diagram

Once we had listed all of the work that needed to be completed we were able to assign tasks to the different group members. The table over page shows our division of work. Please note the tasks and roles are changeable as we chose the Extreme Programming methodology (discussed later).

Name	**Background reading (May, June)**	**1st phase of implementation (June)**	**2nd phase of implementation (July)**	**Report writing (August)**
Kai Duan	Research reading, designing	Coding for RIP daemon	Document collection & report writing	Report writing
Hyo Sim Moon (quit)	Research reading, documentation	Quit	-	-
Oliver Priest	Research reading, designing	Designing	Overall testing, report writing	Report writing
Ying Shi	Research reading, designing	Coding for Schedule module	Coding for Interface module	Report writing
Qian Yang	Research reading, designing	Coding for XML parser, documentations	Coding for Access Control module, documentations	Report writing

Table A.1: Work division table

3. Extreme programming overview

3.1. Extreme programming principles

The project group decided to choose Extreme Programming (XP) for our software implementation methodology. This was because XP suited our project needs when we compared it with other more formal software engineering methodologies that were found to be overly complex. Our project could be considered fairly small and simple. XP gave us the flexibility to achieve our goals while maintaining some kind of structured approach to software engineering. Because the software was constantly evolving we needed an agile methodology that was simple and effective.

Some of the most fundamental software implementation techniques involve the concept of coding in pairs with one member writing code while the other member observing and commenting on the work that was being carried out. XP allowed the group to work on the separate software modules more efficiently. Compilation errors and software bugs well generally dealt with far more quickly when two people were assigned to a software task compared with a single person attempting to solve all of the problems on their own. Because this software approach was successful in the early stages of coding, the group continued the code pairing approach throughout the course of the entire implementation. It is important to note that a single individual had formal responsibility for a single module and the second observer would often play a support role in the problem solving.

XP allowed the group to design a simple application at first that solved a few of the project goals. Once the application was deemed stable, new modules and features were added. This allowed our software to effectively evolve as our project goals changed. This resulted in flexibility in our software design that was free of unnecessary complexity and complications. Creating a stable working software prototype was one of the early goals that the group needed to achieve and putting a minimal system into “production” quickly. This would allow the group to have some kind of working foundation from which to build and improve on. One of the other benefits in producing working software early on in the project was a much needed confidence boost for all concerned knowing that we were making real progress.

XP tends to emphasise software design and implementation that involves staying in contact with the customer. Because our project had no formal customer we often

visualised that our projects customer was our project supervisor. This meant frequent discussion with Steve Hailes on a weekly basis informing him of the direction that our software design and implementation was going.

Another important concept in XP is the use of advanced testing to validate the software design. The group decided to create separate unit tests to make sure that our software modules were fully functioning. Integration testing was also used rigorously to make sure that module integration was smooth and effective. XP also calls for short software development cycles. With this in mind the group aimed to complete a software prototype that had two separate iterations. The group felt that time was particularly limited and that two iterations would allow us to design and implement stable software in the first iteration leaving the second iteration to implement added software features and improvements. XP should involve keeping software design and implementation simple and the group went to great lengths to keep everything well documented and understandable. Although the overall project remained particularly complex, breaking down our design goals and segmenting our workload clearly helped to keep the project design simple and manageable.

XP also emphasizes on coding for current needs. Initially the group made design decisions at the start of the project that were totally different to decisions made in the middle of the project. Coding tasks therefore changed considerably as the project goals changed. The group was particularly good at adapting to these changes and we were often able to anticipate what direction our coding would take after a long discussion at a group meeting with our supervisor.

3.2. Key reasons for choosing XP

XP was chosen over other more formal software engineering methods because it is seen as agile and can quickly adapt to sudden changes in project direction. XP was created in response to problem domains whose requirements change. This described our project perfectly. The group experienced many of these changes throughout the course of the project and in hindsight we believe we chose a particularly suitable methodology. XP implements a simple, yet effective way to enable “groupware” style of development focusing very much on strong communication and collaborative development. We found that this approach to software design suited our needs best which is why XP was chosen. Other benefits of XP over traditional software implementation methodologies included

enhanced team communication, (especially during implementation) and enhanced feedback loops that allowed the group to locate and deal with problems quickly and effectively leading to greater confidence in our ability to achieve our project goals.

XP is also particularly suitable for lightweight teams such as ours where each person has a variety of different roles that can change quickly on a weekly basis. Agile software methodologies such as XP are most suited to small closely knit groups.

XP is generally used for commercial projects rather than research projects however the group felt that the advantages of an agile methodology far outweighed the lack of "user stories" in the academic environment. Because XP is popular in the commercial sector there is usually some kind of customer for which the software is intended. In our case, the customer could be seen as our project supervisor and the university itself.

XP was also chosen because of the strict scheduling guidelines and enhanced documentation that the methodology required. This suited our project perfectly giving us specific control documents and a control system that would ensure project success. This allowed the project coordinator and the team members to know exactly what had to be achieved on a given day within a given timeframe.

4. Scheduling

Although we chose the XP process to develop our software, our project involved a considerable amount of background research and report writing. We needed a strict schedule to achieve all our project goals so that we managed our available time efficiently. All the project activities were planned according to the schedule. This was modified as changes within the project occurred. The project schedule actually resembles the timeframe of a waterfall model software engineering approach. The following Gantt chart below shows the schedule for the whole project.

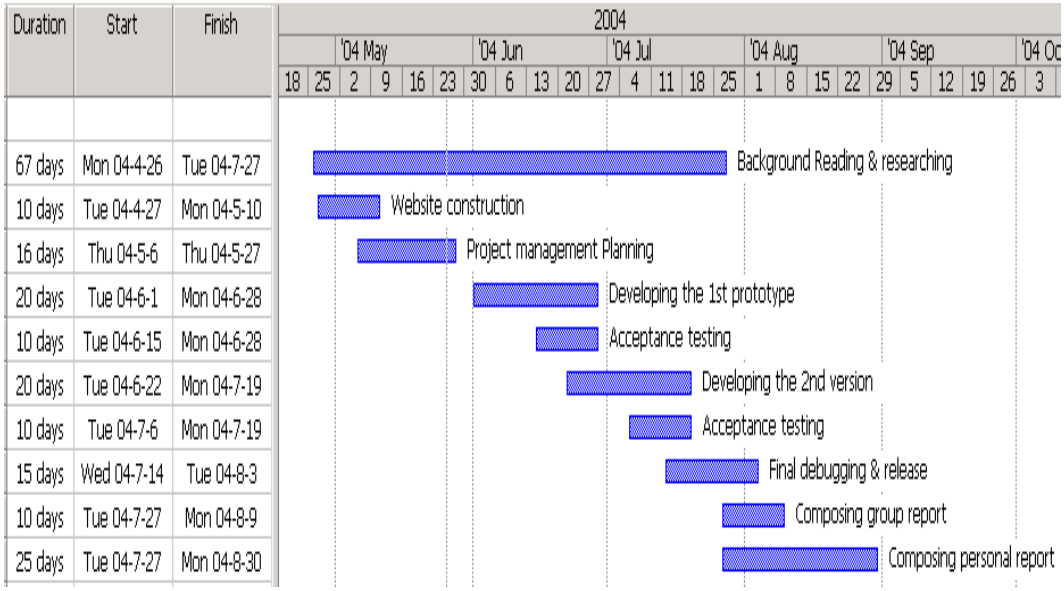


Figure A.2: Gantt chart for the project

Furthermore, we made schedules for every coming month. This included the work that each member was participating in. The monthly schedules gave us more detailed instructions used for monitoring the current progress of the project. Below is an example of a monthly schedule.

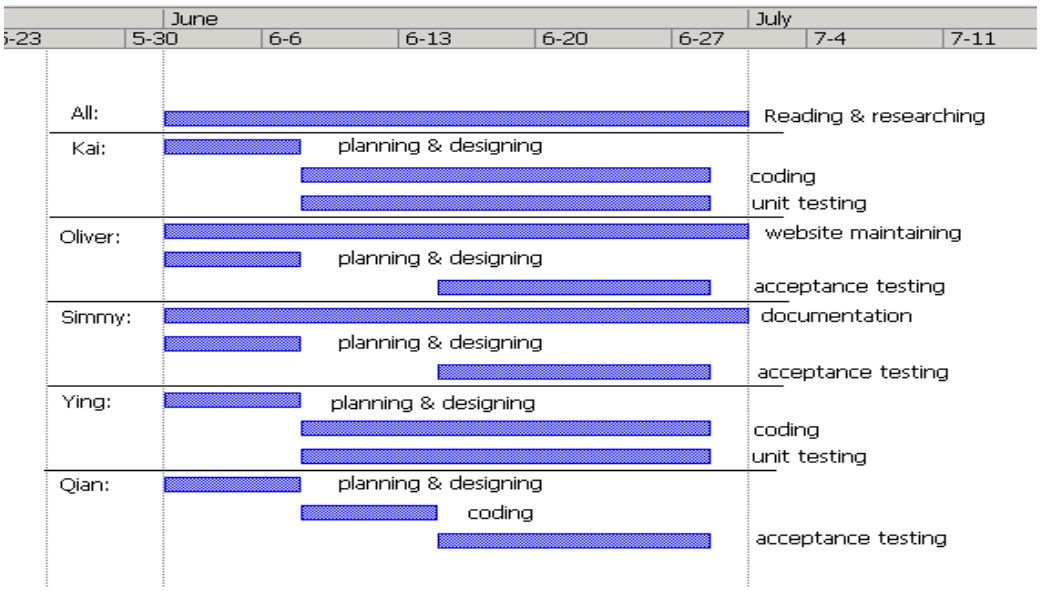


Figure A.3: Example for monthly schedule

5. Project documentation

Throughout the course of the project the group produced a large amount of documentation. Most documentation was used to monitor the progress of the project. Certain documents were used to inform our project supervisor of our progress and some documents were used to provide information to individuals external to the project. Please see a summary list of documents below.

- ✓ **Project schedule**
  The project schedule was used as a blueprint for the group activities over the course of the project listing all of the project milestones.

- ✓ **Individual schedule**
  The individual schedule was used to show what work an individual team member was involved in at a given point in time.

- ✓ **Meeting minutes**
  Meeting minutes were used to log what was discussed over the course of a team meeting. Meeting minutes were also copied to the web log.

- ✓ **Weekly progress report**
  The weekly progress report listed all the work that had been carried out successfully over the course of the week in relation to the project objectives.

- ✓ **Issue log**
  The issue log was produced on a weekly basis summarising all of the potential problems and issues that the group faced on a weekly basis. The project coordinator was responsible for clearing the issue log by delegating the issues to team members.

- ✓ **Design proposal**
  The design proposal was submitted to the project supervisor once the initial software design process was complete. This allowed the supervisor to review our design progress and comment on our design decisions offering valuable feedback.

- ✓ **Code documents**
  Coding documents were produced regularly with full comments so that the

implementation could be examined and evaluated objectively. Coding documents were often exchanged between various group members to improve software module integration.

- ✓ **Testing report**
  A testing report was produced once all testing was completed documenting all unit tests, integration tests and overall application testing.

- ✓ **Project website**
  The project website was created early on in the project to act as an information portal for our work. The site also contained links to relevant research work and individual team member listings.

- ✓ **Project Web log**
  The web log was used as a discussion board and information repository for the project. The team meeting agenda and minutes were listed there on a weekly basis. []

6. Risk management

6.1. Potential risks

Over a five-month period there are many potential risks that a group of four people completing a research project will face. Therefore, a risk management strategy was something that needed to be taken seriously from the onset of the project. The group analysed all the possible risks especially risks that related to our project in the very early stages before we had really made a start. The following table contains the most significant risks.

Risk Type	Possible Risks	Countermeasure
People	Team member/s ill and absent at critical times.	Organise the team so there is overlapping work coverage and team members are aware of each other's work.
	Team member withdraws from the course.	Each member of group keeps logs of their work and is aware of each other's work.
Technical	Team members do not possess the required skill sets.	Perform training in the early stages to improve skill levels.

Managerial	Underestimating project development time.	Hold weekly status meetings and include slack time in project plan to compensate for unexpected circumstances.
	Inability to have all team members present at meetings.	Keep meeting records and upload the records onto the project website.
	System does not meet client acceptance criteria.	Communicate with client on a regular basis.

Table A.2 Potential risks

6.2. Real problems met and solved

Over the course of the entire project we met many kinds of problems and dealt with each efficiently and effectively using a good risk management strategy. Some of the most important problems that the group faced included:

- ✓ **Hyo Sim Moon leaving the course at the end of May**
  We distributed Hyo Sim's responsibilities and tasks between the other group members. Cecile took over the role of recording meeting minutes and Oliver handled the documentation of the project.

- ✓ **A long period of background researching time**
  We took a long time to make the objectives of the project clear and becoming familiar with the related technologies was more difficult than first anticipated. The group dealt with this by first acknowledging the problem and rescheduling our work so we finished within our deadline.

- ✓ **Underestimating the development time**
  The group quickly realised that the development cycles took much longer than previously thought. This was addressed quickly and the group worked hard to make sure that we had enough time to test and document our work. If the group were to complete the project again then we would anticipate that the software development time would be much larger in hindsight.

Appendix B: XML Schema

1. XML schema for Access Control policy

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!-- definition of simple elements -->
<xs:element name="subject" type="xs:string"/>
<xs:element name="allow_ip" type="xs:integer"/>
<xs:element name="deny_ip" type="xs:integer"/>

<!-- definition of attributes -->
<xs:attribute name="type" type="xs:string"/>

<!-- definition of complex elements -->
<xs:element name="allow">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="allow_ip" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="deny">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="deny_ip" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="policy">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="subject"/>
   <xs:element ref="allow"/>
   <xs:element ref="deny"/>
  </xs:sequence>
  <xs:attribute ref="type" use="required"/>
 </xs:complexType>
</xs:element>

</xs:schema>
```

2. XML schema for Initial policy

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!-- definition of simple elements -->
<xs:element name="port" type="xs:integer"/>
<xs:element name="neighbor_num" type="xs:integer"/>
<xs:element name="neighbor_id" type="xs:integer"/>
<xs:element name="ip" type="xs:integer"/>

<!-- definition of attributes -->
<xs:attribute name="type" type="xs:string"/>

<!-- definition of complex elements -->
<xs:element name="neighbor">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="neighbor_id"/>
   <xs:element ref="ip"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="routing_topology">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="neighbor_num"/>
   <xs:element ref="neighbor" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="policy">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="port"/>
   <xs:element ref="routing_topology"/>
  </xs:sequence>
  <xs:attribute ref="type" use="required"/>
 </xs:complexType>
</xs:element>

</xs:schema>
```

3. XML schema for Updating policy

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!-- definition of simple elements -->
<xs:element name="protocol" type="xs:string"/>
<xs:element name="version" type="xs:integer"/>
<xs:element name="addr_family" type="xs:integer"/>
<xs:element name="cmd" type="xs:string"/>
<xs:element name="entry_num" type="xs:integer"/>
<xs:element name="ip" type="xs:string"/>
<xs:element name="metric" type="xs:integer"/>

<!-- definition of attributes -->
<xs:attribute name="type" type="xs:string"/>

<!-- definition of complex elements -->
<xs:element name="route_entry">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="addr_family"/>
   <xs:element ref="ip"/>
   <xs:element ref="metric"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="policy">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="protocol"/>
   <xs:element ref="version"/>
   <xs:element ref="cmd"/>
   <xs:element ref="entry_num"/>
   <xs:element ref="route_entry" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute ref="type" use="required"/>
 </xs:complexType>
</xs:element>

</xs:schema>
```

Appendix C: Technical User Guide

1. Files and Description

File	Description
def.H	Global type and macro definitions.
asl_SSL.H	Header file for secure link library module.
asl_SSL.C	Secure link library which provides interface to use SSL connection facility.
viserver.C	Initialisation server module file.
sched.C	Scheduler module file.
parser.H	Header file for xml parser module.
parser.C	Xml parser module file.
generator.H	Header file for xml generator module.
generator.C	Xml generator module file.
ripd.H	Header file for routing policy handler module.
ripd.C	Routing policy handler module file.
access.H	Header file for access control policy handler module.
access.C	Access control policy handler module file.
monitor.H	Header file for monitor reporter.
monitor.C	Monitor reporter. This is module in C++ giving a way to communicate with Monitor in Java.
monitor\Monitor.java	Monitor java class and other definitions for monitor module use.
monitor\MonitorNet.java	Network component of monitor module.
monitor\MonitorFrame.java	GUI component of monitor module.

2. Interface specification

2.1. asl_SSL

```
enum EPeerType
{
    kServer,
    kClient
};
```
Enumeration type indicates this network peer is server or client.

```
SSL_CTX* initialize_ctx(EPeerType aType, const char* certfile,
const char* keyfile);
```
Initialize an SSL context object and return it.

Parameters:	
aType:	type of network peer
certfile:	name of certificate file
keyfile:	name of private key file
Return value:	
Initialized SSL context object.	

```
void destroy_ctx(SSL_CTX* ctx);
```
Destroy SSL ontext object.

Parameters:	
ctx:	SSL context object to be destroyed

2.2. parser

```
extern CInitPolicy i_policy;
```
Global initial policy object in current use of parsing process.

```
extern TRipPacket r_packet;
```
Global routing policy object in current use of parsing process.

```
extern AccessList a_list;
```
Global access control policy object in current use of parsing process.

```
int ParsePolicy(char* buf, int count);
```
Parse xml format policy to policy object in global variable.

Parameters:	
buf:	Buffer in which xml policy is put
count:	Length of buf

Return value:
Policy type.

2.3. generator

```
int GenerateInitialPolicy (CInitPolicy* ipPtr, FILE* xmlPolicy);
```
Generate xml format initial policy from object.

Parameters:	
ipPtr:	Pointer of initial policy object
xmlPolicy:	Output file descriptor of xml format policy
Return value:	
Policy type.	

```
int GenerateUpdatePolicy(TRipPacket* rpPtr, FILE* xmlPolicy);
```
Generate xml format routing policy from object.

Parameters:	
rpPtr:	Pointer of routing policy object
xmlPolicy:	Output file descriptor of xml format policy
Return value:	
Policy type.	

```
int GenerateAccessPolicy(AccessList* alPtr, FILE* xmlPolicy);
```
Generate xml format access control policy from object.

Parameters:	
alPtr:	Pointer of access control policy object
xmlPolicy:	Output file descriptor of xml format policy
Return value:	
Policy type.	

2.4. monitor

```
class Information {
public:
  int type;
  int minortype;
  int destip;
};
```
System communication information class.

Members:	
type:	Type of the information
minortype:	Minor type of the information
destip:	Destination ip address of this communication action.

```
void reportMonitor (int type, int minortype, int destip);
```

Report a communication action to the monitor.

Parameters:	
type:	Type of the information
minortype:	Minor type of the information
destip:	Destination ip address of this communication action.

2.5. Monitor (Java)

2.5.1. class Monitor

Public Methods
SetNetStatus()
ResetNetStatus()
Run()
UpdateView()
Exit()

2.5.2. class Information:

Public Fields
IType
IMinorType
ISrcIp
IDestIp

2.5.3. class InfoEntry extends Object:

Public Methods
GetInfo()
GetTimer()

Schedule()

2.5.4. class TimeoutTask extends TimerTask

Public Methods
Run()

2.6. MonitorFrame (Java)

2.6.1. class MonitorFrame extends javax.swing.Jframe

Public Methods
paint()

2.7. MonitorNet (Java)

2.7.1. class MonitorNet

Public Methods
init()
run()

2.7.2. class NetThread extends Thread

Public Methods
run()

3. Building process

1) Edit source files in src\ directory
2) Run batch files in bin\ directory to build.
   bv.sh is file for building viserver module.
   bs.sh is file for building scheduler module.
3) Run executable files in out\ directory to test