INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

PERFORMANCE ANALYSIS OF L2TP TUNNELING THROUGH SIMULATION

By Jianxue Huang

**A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of**

**Master of Science
in
Information and System Science**

Department of Computer Engineering

**Carleton University
Ottawa, Ontario
April 29, 2002**

**© copyright
2002, Jianxue Huang**

**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-72089-6

**The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis**

**PERFORMANCE ANALYSIS OF L2TP
TUNNELING THROUGH SIMULATION**

**Submitted by Jianxue Huang
in partial fulfillment of the requirements for
the degree of Master of Science**

Thesis Supervisor

Chair, Department of Computer Engineering

Carleton University

April 29, 2002

ABSTRACT

The use of Virtual Private Network (VPN) technologies is a hot topic in the networking industry. L2TP is a new VPN protocol, which is becoming popular. The thesis investigates the behavior and performance of L2TP protocol in different cases by simulation. First, the categories of VPNs, and the components of L2TP and its implementation for VPN use are reviewed and discussed. A simulation model of L2TP tunneling (i.e., the establishment of control connection) is constructed for performance analysis purpose. The simulation is performed on the Network Simulator (*ns*) environment developed at Berkley. For each simulation experiment, confidence intervals are computed. The L2TP performance for different scenarios is investigated in the research. According to the performance analysis results, the packet retransmission feature in L2TP is a major factor that has a strong impact on the system performance. In order to improve the performance, a new adaptive retransmission algorithm is proposed. The simulation results show that the proposed algorithm can notably improve the L2TP performance.

ACKNOWLEDGMENTS

I would first like to give my thanks to Dorina Petriu, my advisor and thesis supervisor, for her kind guidance and support throughout my study in M.S. program, especially during the research for this thesis.

I would also like to give my thanks to the Department of Computer Engineering, Carleton University for providing an ISS program to us. This gives me an opportunity so that I can study in this field and fulfill my M.Sc. degree.

Thanks also go to staff in the computer lab in the Department for their help in software installation and problem solving during my working on my thesis.

TABLE OF CONTENTS

ACCEPTANCE SHEET	ii
ABSTRACT	iii
ACKNOWLEDGEMENTS	iv
TABLE OF CONTENTS	v
LIST OF FIGURES	viii
LIST OF TABLES	x
LIST OF ACRONYMS	xi
CHAPTER 1 INTRODUCTION	1
1.1 Background	1
1.2 Motivation and goals	1
1.3 Methodology	3
1.4 Contributions	4
1.5 Contents	4
CHAPTER 2 VIRTUAL PRIVATE NETWORK (VPN) AND LAYER TWO TUNNELING PROTOCOL (L2TP)	7
2.1 Virtual Private Network (VPN)	7
2.1.1 The concept of VPN	7
2.1.2 VPN issues and categories	8
2.1.3 VPN methods	11
2.2 Layer 2 Tunneling Protocol (L2TP)	14
2.2.1 L2TP access models	14
2.2.2 L2TP tunnel structure	16
2.2.3 Protocol stack	17
2.2.4 L2TP header and message format	18
2.2.5 Protocol operation	20
2.2.6 L2TP implementations for VPN	22
CHAPTER 3 THE NS ENVIRONMENT AND SIMULATION	

DESIGN PROCESS	24
3.1 Network Simulator (<i>ns</i>)	24
3.1.1 Overview	24
3.1.2 The structure of <i>ns</i>	25
3.1.3 The network topology in <i>ns</i>	27
3.1.4 Routing	29
3.1.5 Transport and application agents	30
3.2 The selection of L2TP Features in the simulation	31
3.2.1 The establishment of control connection	31
3.2.2 Tunnel authentication	33
3.2.3 Control message format and packet size	34
3.3 Simulation design process	36
3.3.1 Problem formulation and objectives	37
3.3.2 Model conceptualization and translation	37
3.3.3 Model verification and validation	38
3.3.4 Production runs and analysis	39
3.3.5 System improvement	40
CHAPTER 4 SIMULATION MODELING	41
4.1 The development of L2TP agents	41
4.1.1 The structure and functionalities of the agents	41
4.1.2 Agent implementation	42
4.1.3 Packet retransmission	47
4.1.4 CPU resources competition mechanism	48
4.1.5 The outputs of agents	51
4.2 Inter-network topology modeling	52
4.2.1 Transit-Stub model	52
4.2.2 Topology generation	54
4.2.3 Converting an SGB file to <i>ns</i> format	55
4.3 Background traffic generation	58
4.4 Simulation scenarios	60
4.4.1 The specifications and implementation of scenarios	60

4.4.2 Parameter determination	62
CHAPTER 5 IMPLEMENTATION AND OUTPUT ANALYSIS	65
5.1 Case study: the implementation of scenario 9	65
5.2 Output data analysis of other scenarios	73
5.3 System performance analysis	79
CHAPTER 6 PERFORMANCE IMPROVEMENT OF L2TP	88
6.1 Problem formulation	88
6.2 New adaptive algorithm design for retransmissions	90
6.3 Algorithm implementation	94
6.4 Result analysis and comparison	97
CHAPTER 7 SUMMARY AND CONCLUSION	103
BIBLIOGRAPHY	107

LIST OF FIGURES

Figure 2-1	Intranet VPN, Extranet VPN and Virtual Private Dial Network	11
Figure 2-2	L2TP topologies: Remote Access and LAC Client Access with L2TP	15
Figure 2-3	L2TP tunnel components	16
Figure 2-4	The L2TP protocol stack	17
Figure 2-5	L2TP message header	18
Figure 2-6	A typical sequence of L2TP data session establishment triggered by an incoming call	21
Figure 3-1	The linkage between OTcl objects and C++ objects	25
Figure 3-2	Partial <i>ns</i> class hierarchy	26
Figure 3-3	A typical process of three-message exchange in L2TP control connection establishment	32
Figure 3-4	ACHAP-like authentication	34
Figure 3-5	The format of the L2TP Attribute Value Pair (AVP)	35
Figure 4-1	Logic working flow of agent L2tp_lac	43
Figure 4-2	Logic working flow of agent L2tp_ins	46
Figure 4-3	The semaphore for CPU competition	50
Figure 4-4	Example of Internet domain structure	53
Figure 4-5	N54_1 (the number of nodes: 54; node degree: 2.19)	56
Figure 4-6	N54_2 (the number of nodes: 54; node degree: 3.63)	57
Figure 4-7	N108_1 (the number of nodes: 108; node degree: 2.19)	57
Figure 4-8	N108_2 (the number of nodes: 108; node degree: 3.60)	58
Figure 5-1	The average length of tunnels in different networks	80
Figure 5-2	The average tunneling time with different tunnel lengths (hops) in Scenario 9	81
Figure 5-3	The average tunneling time with different number of retransmissions in Scenario 9	82

Figure 5-4	Tunneling time under different traffic rates in the network topology with 54 nodes	83
Figure 5-5	Tunneling time under different traffic rates in the network topology with 108 nodes	84
Figure 5-6	Tunneling time under different traffic rates in the network topology with node degree 2.19	85
Figure 5-7	Tunneling time under different traffic rates in the network topology with node degree 3.6	86
Figure 6-1	Retransmission ambiguity 1: the retransmission's ACK is wrongly regarded as the original transmission's ACK	92
Figure 6-2	Retransmission ambiguity 2: the original transmission's ACK is wrongly regarded as the retransmission's ACK	92
Figure 6-3	Sequence diagram for the communication among the LAC agent, Tcl interfaces, and the LAC node	97
Figure 6-4	The comparison of tunneling time from L2TP tunneling and improved L2TP tunneling	100
Figure 6-5	The comparison of tunneling time at different rtx times by using L2TP rtx algorithm and using improved algorithm	101
Figure 6-6	The comparison of tunneling time at different hops by using L2TP rtx algorithm and using improved algorithm	101

LIST OF TABLES

Table 4-1	properties of real inter-network topologies	63
Table 5-1	Ensemble batch means and cumulative means, averaged over 10 runs	68
Table 5-2	The smallest sample size required for scenario 9 ($R_0 = 10$)	71
Table 5-3	The examination of sample size based on 15 runs	72
Table 5-4	Mean values with first two batches being removed in Scenario 9	72
Table 5-5	The values of simulation items for Scenario 1	73
Table 5-6	The values of simulation items for Scenario 2	74
Table 5-7	The values of simulation items for Scenario 3	74
Table 5-8	The values of simulation items for Scenario 4	75
Table 5-9	The values of simulation items for Scenario 5	75
Table 5-10	The values of simulation items for Scenario 6	76
Table 5-11	The values of simulation items for Scenario 7	76
Table 5-12	The values of simulation items for Scenario 8	77
Table 5-13	The values of simulation items for Scenario 10	77
Table 5-14	The values of simulation items for Scenario 11	78
Table 5-15	The values of simulation items for Scenario 12	78
Table 5-16	The simulation results of 12 scenarios	79
Table 6-1	The structure of table <code>table_RTT</code> and <code>table_COUT</code>	91
Table 6-2	Ensemble batch means and cumulative means, averaged over 10 runs	98
Table 6-3	The smallest sample size required for scenario 9_1	99
Table 6-4	Mean values in of 10 runs in Scenario 9_1	99

LIST OF ACRONYMS

AVP	Attribute Value Pair
CBR	Constant Bit Rate
CHAP	PPP Challenge Handshake Authentication Protocol
FCS	Frame Check Sequence
ICCN	Incoming-Call-Connected
ICRQ	Incoming-Call-Request
ICRP	Incoming-Call-Reply
ISP	Internet Service Provider
ISDN	Integrated Services Digital Network
L2F	Layer Two Forwarding
L2TP	Layer Two Tunneling Protocol
LAC	L2TP Access Concentrator
LNS	L2TP Network Server
MPPE	Microsoft Point-to-Point Encryption
NAS	Network Access Server
<i>ns</i>	Network Simulator
OSI	Open Systems Interconnection
PPP	Point-to-Point Protocol
PPTP	Point-to-Point Tunneling Protocol
PSTN	Public Switched Telephone Network
RADIUS	Remote Authentication Dial In User Service
RTT	Round-Trip-Time
SCCCN	Start-Control-Connection-Connected
SCCRP	Start-Control-Connection-Reply
SCCRQ	Start-Control-Connection-Request
VPDN	Virtual Private Dial Network
VPN	Virtual Private Network
ZLB	Zero-Length-Body

CHAPTER 1: INTRODUCTION

1.1 Background

The communication through computer-mediated data networks has grown rapidly and has become a business necessity in the last years. The Internet, as a global public network, provides an ideal backbone for data communication due to its low-cost and ubiquitous access. Nowadays, more and more companies and agencies are using VPN (Virtual Private Network) technology to build secure networks over the Internet for their private use. Internet Service Providers (ISPs) also provide VPN services to their customers, and allow customers to conduct secure communication through the public networks.

VPN is based on a broad collection of technologies. Several protocols can be used to provide VPN services. L2TP (Layer 2 Tunneling Protocol) is one of them. L2TP is a new protocol, which became a standard in the middle of 1999. L2TP represents a merging of another two popular VPN protocols (Point-to-Point Tunneling Protocol and Layer 2 Forwarding) and combines their best features, so it is expected that many VPNs will be based on L2TP in the near future.

L2TP can tunnel a stream of Point-to-Point Protocol (PPP) sessions to remote access servers. This allows the PPP sessions to be performed over multiple links. Using L2TP tunneling, an ISP or other access service can create a virtual tunnel to link customer remote sites or remote users with a corporate home network. PPP is a standard means in the industry to define a point-to-point encapsulation protocol. L2TP benefits from tunneling the feature-rich PPP, and inherits from PPP its set of supported protocols. This makes L2TP a powerful tunneling tool and a popular VPN protocol.

1.2 Motivation and Goals

With the growth of Internet traffic, the performance becomes a critical consideration for the design and implementation of a protocol. Performance analysis is

a very important means in protocol evaluation and improvement. Usually, it is difficult to conduct the performance analysis of a protocol through measurements in a real network. The networks that are large enough for the analysis are expensive and difficult to control. Therefore, they are rarely available or suitable for experimental purposes.

Another approach is to build a model of the network, and to study its performance either analytically or by simulation. In a simulation model, for example, we can create an environment that mimics the real-world network and emulates the situations of the real communication. In such an environment, we can perform the protocol implementation in a dynamic manner. From the simulation, a large amount of data can be produced. Through analyzing these data, the performance of a system in some specific conditions can be evaluated.

As mentioned before, L2TP is a new and a very important VPN protocol. Conducting simulation experiments to study the performance of L2TP tunneling in different situations (such as in different sizes of networks, for different connectivity levels, and at different background traffic rates) is very significant. Through such a simulation study, we can investigate the factors that have a negative effect on the system performance, and then improve the features of the protocol which are responsible for it.

The goals of this research include:

- i. Conducting the performance analysis of L2TP tunneling in different situations, so that the factors that negatively affect the system performance can be located and investigated. The performance analysis is carried out through simulation. A simulation model of L2TP tunneling in a Remote Access model will be designed for this purpose. Since L2TP is a feature-rich protocol. It is impossible for this research to simulate the whole protocol. The research will focus on the establishment of the control connection (i.e., the tunnel generation). The simulation is performed in the Network Simulator (*ns*) environment [FV00, GreNS]. Multiple results that are related to the system performance will be collected by simulation.
- ii. Based on the result of performance analysis, modifying the policy or algorithm of the L2TP Protocol that has serious negative influence on the performance so as to

improve the system performance. This includes designing and implementing an adaptive retransmission algorithm, and conducting the performance comparison between the original L2TP model and the improved L2TP model.

1.3 Methodology

In this research, the simulation is designed and implemented by using the *ns* environment. The methodology employed for the research can be described by the following steps:

- Design and develop L2TP agents to implement the establishment of L2TP control connection. In *ns*, a protocol is performed by a corresponding protocol agent. In this research, a couple of agents need to be developed to simulate the process of L2TP tunneling. The verification and the validation of the agents are done throughout the development of agents.
- Construct proper network topologies. Several network topologies of different sizes at different connectivity levels will be used in the simulation to generate L2TP tunnels over them. This allows the comparison of L2TP tunneling in different situations.
- Generate background traffic. Traffic is brought into the networks during the simulation, in order to emulate the situation of the real-world network. It can also help to investigate the behavior of the system in a dynamic context.
- Perform the simulation in different situations, i.e., different sizes of network, different connection levels and different traffic rates. A corresponding simulation scenario needs to be generated for each situation.
- Implement statistical analysis on output data, such as removing the initialization bias and determining the sample size of each scenario so as to meet the requirement of the given precision.
- Conduct performance analysis, investigate the system behavior in different situations, and find out the factors in the protocol that badly lowers the system performance.

- Improve the L2TP protocol for obtaining a better performance. This involves designing a new algorithm, revising the L2TP agents and bringing a new interaction mechanism among L2TP agents and some *ns* components.
- Implement a simulation based on the improved L2TP protocol, and then evaluate the result.

1.4 Contributions

The contributions of this research can be summarized as follows:

1. Performance study of L2TP by simulation: understanding what factors have impact on the performance and how some of them can be improved. As far as we know, there are not other similar studies in literature, probably due to the fact that L2TP is quite new.
2. Improving the retransmission mechanism of L2TP protocol. According to the performance analysis, the retransmission of packets is a major factor that reduces the system performance in L2TP. A new adaptive retransmission algorithm is designed and implemented. Using the new algorithm, the system performance can be notably improved.
3. Designing and developing L2TP agents to be added to the *ns* set of agents. The protocol agent is a core in *ns* for the implementation of a protocol. A couple of agents are designed and developed according to the L2TP protocol. They are used to achieve the establishment of L2TP control connections. The agents will be posted on the web and made available to others for reuse.

1.5 Contents

The thesis comprises seven chapters. These chapters are organized as follows:

Chapter 1: This chapter gives the introduction of the thesis. The outline of the research is described in this chapter. It includes: the background, motivation and the goals, the methodologies used in the research and the main contributions of the research. At the last, there is the structure of the thesis.

Chapter 2: This chapter is the review chapter. The concepts of VPN and L2TP tunneling are introduced in this chapter. The categories and methods of VPN are discussed in section 2.1. In section 2.2, the L2TP protocol is reviewed. The review focuses on the general concepts of the protocol, such as L2TP tunnel structure, protocol stack, L2TP header format and the data session establishment. The implementations of L2TP for VPN uses are briefly discussed.

Chapter 3: The simulation environment (i.e., the *ns*) is introduced in section 3.1. The introduction of *ns* focuses on those features and components that are closely related to the simulation in this research. In section 3.2, the establishment of L2TP control connection is described in detail. This component is the part that will be simulated in the research. The general design of the simulation is given in section 3.3.

Chapter 4: This chapter is one of the key chapters in the thesis. In this chapter, the simulation model is designed. The simulation modeling comprises four basic parts: the design and development of L2TP agents (it is discussed in section 4.1); the selection and generation of network topologies (section 4.2); the generation of background traffic (section 4.3); the simulation scenario design, construction and the component integration (section 4.4).

Chapter 5: This is another key chapter. The simulation implementation is described in this chapter (section 5.1). The simulation is performed in 12 different scenarios. The output data analysis is given in section 5.2. In this section, Scenario 9 is used for the case study. The process of data analysis for Scenario 9 is described in detail. For other scenarios, only the simulation results are presented. The performance analysis is conducted in section 5.3. The protocol factor that lowers the performance of the system is determined.

Chapter 6: This chapter describes the improvement made on the L2TP protocol. Firstly, the problem is formulated in section 6.1. In problem formulation, the protocol feature (i.e., the retransmission algorithm) that needs to be improved is addressed. Then, a new algorithm for packet retransmission is designed and implemented. The new retransmission mechanism is described in section 6.2. Finally, the performance of the

new system that is based on the improved L2TP protocol is examined and compared with the previous one's.

Chapter 7: This chapter gives the summary and the conclusions. Some topics that need to be further studied are briefly addressed.

CHAPTER 2: VIRTUAL PRIVATE NETWORK (VPN) AND LAYER TWO TUNNELING PROTOCOL (L2TP)

Virtual private networking has been a hot topic among IT professionals lately. It is popularly associated with the data communication industry. Based on the Internet, a popular low-cost backbone infrastructure, a Virtual Private Network (VPN) provides users with a mechanism for secure data communication, which is efficient both in cost and connectivity.

There are several protocols being used in VPNs, Layer Two Tunneling Protocol (L2TP) being one of them [L2TPcisco]. L2TP defines an IP tunneling mechanism. An L2TP tunnel provides a virtual point-to-point connection to a remote network, so that the peers involved in the communication can conduct point-to-point operations.

This chapter covers the reviews on the VPN, including the concept, categories and the methods of the VPN (Section 1), and the L2TP protocol (Section 2). In Section 3, the VPN implementations in L2TP will be discussed.

2.1 Virtual Private Network (VPN)

2.1.1 The Concept of VPN

There are many different technologies for implementing Virtual Private Networks [Kos98, Ent02]. In this thesis, the research and discussion are limited to the Internet-based VPN.

A VPN is simply defined as the 'emulation of a private Wide Area Network (WAN) facility using IP facilities' (including the public Internet, or private IP backbone) [GLHAM00]. It is called "virtual" because it depends on the use of virtual connections, i.e., temporary connections that have no physical presence, and simulates some of the properties of a private network. It provides a mechanism for delivering data in a secure manner so that it allows users to use a public network as if it were private.

In today's business world, a secure communication network is a must in every organization in order to pass digital and voice information. Traditionally, accessing the network from distant locations that are not physically connected to the network can be done either through extending the private line or through leased lines to implement point-to-point communication for keeping information private. These lines are very expensive to build and maintain [Pha00].

A VPN system addresses this problem by creating a "virtual" private network, so that remote users who belong to this network can communicate freely and safely between the remote networks through the untrusted public networks. Typically, a VPN uses the Internet as the infrastructure to establish secure connections with business partners or distant branches. The advantages of using a VPN include:

- Data privacy and Lower cost of development. The VPN provides data privacy by encrypting the data and/or tunneling the data over the public network [Pha00]. The characteristics and integrity of communication services within one closed environment are isolated from all other environments which share the common underlying infrastructure. With a VPN, there is an initial setup and implementation cost, but they are not as expensive as the leased lines [TB99].
- Ubiquitous access. VPNs provide access from anywhere the Internet reaches. Nowadays, Internet points of presence are available worldwide, providing VPN connection points in nearly every country and in most of the major cities in the world [VPN98].
- Flexibility. VPNs enable rich, flexible communication with remote branches, regional offices, customers and business partners, as long as those are already connected to the Internet [VPN98].
- Scalability. The VPN solutions are highly scalable due to their simplicity and ease of development. True any-to-any communication can be deployed with good scalability [Pha00].

2.1.2 VPN Issues and Categories

Security is the central consideration for VPN applications. To implement a secure communication, three basic issues must be taken into account. One issue is data privacy.

It means to protect the data in transit so that hostile or curious third parties in the middle are not able to intercept and read the transmission. This goal can be fulfilled through data encryption. The typical protocols used for this purpose include Microsoft Point-to-Point Encryption (MPPE), Secure IP (IPSec) [Pri98] etc.

The second issue is Spoof-proof. This is the mechanism of authentication, i.e., both parties in a communication must know with confidence that they are talking to the right people. The user authentication prevents one user from masquerading as another. For example, when A wants to communicate with B, A should be certain that he is talking to B rather than anyone else, and vice versa. PPP Challenge Handshake Authentication Protocol (CHAP) and Remote Authentication Dial In User Service (RADIUS) are the typical protocols used in user authentication [Sim96, RRSW97].

The third issue is the concept of data integrity. During the communication, both A and B need to be able to protect the messages from being altered by any third party, such like the insertion of many messages to prevent A from reaching B. Several existing message digest algorithms can be used for this purpose [SWE99, Bal93].

The concept of VPN has a broad meaning. Attempts to categorize the wide range of technologies have been proven difficult. Different technologies have provided different services yet all manufactures called these services VPN. Since VPN services can be viewed from different aspects, they can be classified in different ways.

From the platform point of view, VPNs can be classified into two types: *hardware-based VPN* and *software-based VPN*. In hardware-based VPN products, the VPN is implemented either in standalone devices or the routers and servers in which some cards are added. In software-based VPNs, the main processor of the router or server usually needs to be involved in the VPN process, such as the data encryption [VPN98].

Most hardware-based VPN systems are encrypting routers. They can deliver the best performance since they don't waste processor overhead in running an operating system or applications. However, they may not be as flexible as software-based systems.

Software-based VPNs are ideal in situations where the endpoints of the VPN are not controlled by the same organization (typically for client support requirements or business partnerships) or when different routers are used within the same organization. Software-based solutions are best suited for lower-volume connections at small and medium size

companies that have lower security requirements. Some VPN operations, such as encryption, are processor intensive. Therefore, high volumes of traffic with maximum security solutions require dedicated designed hardware.

From the network structure point of view, VPN implementations typically fall into three basic categories: *Intranet VPN*, *Extranet VPN*, and *Virtual Private Dial Network (VPDN)*.

Intranet VPNs, also called LAN-to-LAN VPN [Pha00, Opp98, CohR00] or WAN VPN [Jai99] in the literature, connect multiple sites within an enterprise. When a company wants to extend its internal network to connect remote regional or branch offices (e.g. connecting the offices in east coast with the offices in west coast), an Intranet VPN can provide an efficient solution. Those offices can be connected over public networks, such as the Internet. The Intranet traffic is either encrypted or tunneled through the Internet using VPN technology to maintain data privacy and integrity.

Extranet VPNs connect one enterprise to multiple enterprises. They are the extension of Intranet VPNs. The term 'extranet' is commonly used to refer to a scenario whereby two or more companies have networked access to a limited amount of each other's corporate data. Extranet VPNs can be used among two or more partners who are corporate in their business. For example, an Extranet VPN is used by a manufacturing company to allow its suppliers to query databases for the prices and the availability of the products, and then, to order and track the status of outstanding orders. The same technologies used in creating Intranet VPNs are used in Extranet VPNs, but in different context. A key feature of an Extranet VPN is the control of who can access what data. This is essentially a policy decision [GLHAM00].

A VPDN, also called secure remote access VPN [TB99], provides dial access to intranet and extranet applications via Public Switched Telephone Network (PSTN) or Integrated Services Digital Network (ISDN) links. Typically, this is done through the deployment of Network Access Servers (NASs) at one or more central sites. The user dials into such an NAS, and then, is connected to a public IP network (e.g., the Internet). User packets are tunneled across the public network to the desired site, giving the impression to the user of being 'directly' connected to that site. In a VPDN, the user authentication is a prime requirement since anyone could potentially attempt to gain

access to such a site using a switched dial network. The VPDN typically provides the remote LAN access for the roaming users, such as mobile executives and employees who are telecommunicating to work.

The Intranet VPN, Extranet VPN and Virtual Private Dial Network (VPDN) are illustrated in Figure 2-1.

Figure 2-1: Intranet VPN, Extranet VPN, and Virtual Private Dial Network
(From Phaltankar [Pha00])

This review cannot exhaust all images of VPNs. There are other views of VPNs. For example, from the OSI (Open Systems Interconnection) layers point of view, VPNs can be classified into Network Layer VPNs, Link Layer VPNs, Transport and Application Layer VPNs [FH98], and so on. They will not be discussed in this review.

2.1.3 VPN Methods

As discussed in section 2.1.2, the central issue of the VPN is the security of the communication. There are several technologies used for VPNs to protect data traveling

across the public networks. The most important concepts are firewall, authentication, encryption, and tunneling [SWE99, KA98].

Firewall: An important element of the VPN concept lies at the gateways between the private network and the public network. When one of the local hosts sends data (or a request) to another host in a remote network, the data must first pass from the private network through the protecting gateway device, travel through the public network, and then pass through the gateway device that is protecting the host in the remote network at the receiving end. The internal network of an enterprise is kept private through strict access control from external networks. The firewall techniques provide such an access control. They are used to protect an entire internal network (at different levels) at its gateway routers [CohS00, Ake00].

Although most VPN packages themselves do not implement firewall directly, they are an integral part of VPN. The firewall is used to keep unwanted visitors from entering one's network, while allowing VPN users through. Usually, the firewall is the first line of protection in the fabric of a VPN.

Different strategies have been employed in firewalls. The most common one is perhaps the packet filtration. The packet filtration strategy is to block specific users (based on userID or hostID) or specific IP services (based on IP address, sub-network address or specific port number) from crossing the gateway router. Restrictions can also be based on the type of access (email, Telnet, FTP, etc.), contents of the data accessed, and so on [SWE99].

Authentication [SWE99, McD00]: The central concept of the VPN is to ensure the secure communication between two distinct networks over public mediums. The LAN is perhaps being shared from either end during their communication. Therefore, the firewall techniques only cover half of the question. The firewall offers only a perimeter protection. It either allows or denies traffic based on the source or destination, but once the traffic makes it into one's network, the disciplines of authentication and encryption are needed for the further protection by secure the conversation.

Authentication technologies are used to make sure that the two ends involved in the communication are the true parties who want to communicate to each other. Most VPN

authentication systems are based on a shared key system. The keys are run through a hashing algorithm, which generates a hash value. The other party holding the keys will generate its own hash value and compare it to the one it received from the other end. If these two values are equal to each other, the peer is acceptable. Otherwise, the attempt on the communication of the peer will be rejected. PPP Challenge Handshake Authentication Protocol (CHAP) is a common protocol used in the user authentication.

Encryption [Kos98, Smi93]: Encryption is a cryptographic function that ensures data privacy. Using this function, we can transfer the readable data into a non-readable format which can be recovered using a special key. Thus, when a packet is going to be transmitted, the data portion of the packet payload is encrypted, and then the packet is encapsulated and sent out. The encrypted data cannot be read by intermediary observers in an untrusted network. After the intended recipient receives the packet, he (she) can decrypt the data using an appropriate key.

There are two primary encryption techniques used in VPNs: private key encryption (also known as the symmetric key encryption) and public key encryption (also known as the asymmetric key encryption). In private key encryption, the same key is used to encrypt and decrypt the data. In this case, there must be a shared secret password known to all parties who need to access to the encrypted information. Therefore, this method usually well fits to the small workgroup. For the large group, the secure exchange of the encryption key will become unmanageable.

In public key encryption, two separate keys are used to encrypt and decrypt the data: a private key and a public key. Each communication end has a pair of keys (a private key and a public key). The public key is made available to the public. The sender uses a combination of his private key and the receiver's public key to encrypt the data. The receiver decrypts the data using his private key and the sender's public key.

Tunneling: Tunneling is the encapsulation of point-to-point transmission inside IP packets [VPN98]. Using tunneling technologies, data packets are transmitted across a public routed network, such as the Internet or other commercially available networks, in a private "tunnel" that simulates a point-to-point connection. This approach enables

network traffic from many sources to travel via separate tunnels across the same infrastructure. It allows network protocols to traverse incompatible infrastructures.

The major point-to-point tunneling protocols used by VPN are the Layer 2 Tunneling Protocol (L2TP) and Point-to-Point Tunneling Protocol (PPTP) [Ham99]. L2TP will be described in section 2.2 and in Chapter 3 (section 3.2).

The VPN methods discussed above can either be used individually or be used in combinations, depending on the security demands of the communication. Usually, the data encryption is the strictest individual method in secure data transmission. It is also the most processor intensive process. Therefore, the good design of a VPN should meet the requirements for security of the application, and at the same time, gain a good system performance as much as possible.

2.2 Layer 2 Tunneling Protocol (L2TP)

L2TP is one of the key VPN protocols that is developed to carry Point-to-Point Protocol (PPP) traffic over non-point-to-point networks. The L2TP protocol was created from the combination of Microsoft Point-to-Point Tunneling Protocol (PPTP) and Cisco Layer-2 Forwarding protocol (L2F) [VLK98]. It offers the best features of these two tunneling protocols as well as additional features. Today, L2TP has become a standard for Layer 2 tunneling. It can be used in any situation where one may use PPTP or L2F. L2TP provides virtual remote access by tunneling PPP applications. Tunneling is the encapsulation of one type of protocol within another protocol [LucTech99]. In L2TP, PPP frames are encapsulated in IP packets.

2.2.1 L2TP Access Models

As we know, a data link protocol (also called a data link control) operation is limited to one individual link. That is, the link control is responsible only for the traffic between adjacent machines on a link. Using L2TP, a PPP operation can be extended to permit it to operate not just two PPP peers on one link, but between different devices in separate networks and multiple links.

An L2TP tunnel starts from an L2TP Access Concentrator (LAC), and terminates at an L2TP Network Server (LNS). An LAC is a device attached to a switched network fabric (such as PSTN or ISDN) or co-located with a PPP end system capable of handling the L2TP protocol. The LAC may tunnel any protocol carried within PPP. It is the receiver of incoming calls and the initiator of outgoing calls. An LNS is a PPP server with L2TP capabilities. It is the end point of the session, and handles the server side of the L2TP tunnel.

The LAC sits between an LNS and a Remote System. A Remote System is an end-system attached to a remote access network (such as a PSTN). It is either the initiator or recipient of a call. The connection from the LAC to the Remote System is either local or a PPP link. Two typical L2TP scenarios are illustrated in Figure 2-2.

Figure 2-2. L2TP topologies: Remote Access and LAC Client Access with L2TP

In Figure 2-2, Scenario 1 shows the Remote Access model, where the Remote System initiates a PPP connection across the PSTN to an LAC. The LAC then tunnels the PPP connection across the Internet to an LNS whereby a private network can be accessed. The Remote Access method is also known as compulsory tunneling, since a tunnel is created without any action from the user and the remote user's computer does not control the tunnel.

Scenario 2 illustrates the LAC Client Access model. An LAC Client is a host which runs L2TP natively. In this model, the host that contains the LAC Client software already has connection to the Internet. Usually, it accesses to the Internet through a cable data network. The LAC Client can create a tunnel to the LNS without use of a separate LAC. This model is also called voluntary tunneling. In this model, the tunnel is created by the user. The user can even decide which traffic will be sent in to the tunnel.

2.2.2 L2TP Tunnel Structure

An L2TP tunnel is comprised of two basic components: a single control connection and a set of zero or more L2TP data sessions. Figure 2-3 illustrates the structure and the components of the L2TP tunnel in the Remote Access model.

Figure 2-3: L2TP tunnel components (from She00)

The control connection for the tunnel is the component of the tunnel which is responsible for establishing, maintaining, and releasing data sessions, as well as the tunnel itself. Those issues are dealt with through the exchange of different control messages between the LAC and the LNS. Control messages are carried over a reliable control channel within L2TP to guarantee delivery. Either an LAC or an LNS can initiate

a control connection. Once the control connection is established, the tunnel is said to be established (because there is a control connection and zero or more sessions – [She00]). In this research, the process of the control connection establishment will be simulated. The method and issues for establishing the control connection will be discussed in detail in Chapter 3 (section 3.2).

An L2TP data session is a state of a PPP connection between a Remote System and the LNS. When an end-to-end PPP connection between a Remote System and the LNS is established, an L2TP data session is created between the LAC and the LNS. Each data session corresponds to a single PPP stream between the LAC and the LNS. In each session, the PPP frames are encapsulated by data messages and carried over the tunnel. The data (PPP streams) are transported through an unreliable data channel within L2TP. When packet loss occurs, data messages are not retransmitted.

The control connection is the initial connection between an LAC and an LNS. After the control connection is established, individual sessions may be created.

2.2.3 Protocol Stack

Figure 2-4 shows the L2TP protocol stack. This stack tells us how the L2TP packets are transported over an IP network.

Figure 2-4: The L2TP protocol stack

As mentioned in section 2.2.2, two types of messages are used in L2TP: control messages and data messages. Figure 2-4 illustrates the relationship of control messages over the control channel and the PPP frames over the data channel. L2TP runs on the top of UDP. When a PPP packet comes to the LAC from a Remote System, it is encapsulated first by an L2TP header, and then a UDP header, and an IP header. After the packet is encapsulated, any router or machine that encounters it from that point (i.e., the LAC) on will treat it as an IP packet. Thus, the packet is delivered as a common IP packet along the path created according to the IP routing rules. The advantage of IP encapsulation is that it allows many different protocols to be routed across an IP medium, such as the Internet.

2.2.4 L2TP Header and Message Format

L2TP uses a common header for both the control channel and the data channel transmissions. This means both control messages and data messages begin with the commonly formatted L2TP header. The structure of the header is shown in Figure 2-5.

0	1	2	3	4	5	6	7	8	...	12	13	...	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
T	L	x	S	x	O	P	x	Version						Length																
Tunnel ID														Session ID																
Ns														Nr																
Offset Size														Offset Pad																

Figure 2-5: L2TP message header

The meaning of each field can be explained as following.

- **T bit:** message type.

The T bit is used to specify whether a message is a control message or a data message.

0: data message.

1: control message.

- **L bit:** length present. Optional

The L bit specifies whether or not the length field of the L2TP header is present. If the L bit is set (i.e. the value is 1), the length field is present. Control messages must have this bit set. If the L bit is clear, the length field is not present.

- ***x bit:*** Reserved for future extension. All reserved bits must be set to 0 on outgoing messages and ignored on incoming messages.
- ***S bit:*** Sequence present.
The S bit is used to specify whether or not the Ns and Nr fields are present. If it is set (the value is 1), the Ns and Nr fields are present. Control messages must have this bit set.
- ***O bit:*** Offset present.
If the O bit is 1, the Offset Size field is present. The O bit must be set to 0 for control message.
- ***P bit:*** Priority.
If P bit is set (i.e. the value is 1), this data message should receive preferential treatment in its local queuing and transmission. The P bit must be set to 0 for all control messages.
- ***Version:*** 3 bits.
Indicates the L2TP protocol version. It must be set to 2 for the current version of L2TP.
- ***Length:*** 16 bits. This field indicates the total length of the message in bytes.
- ***Tunnel ID:*** 16 bits. Tunnel ID indicates the identifier for the control connection. L2TP tunnels are named by identifiers that have local significance only. That is, the same tunnel will be given different tunnel IDs by each end of the tunnel. Tunnel ID in each message is that of the intended recipients, not the sender.
- ***Session ID:*** 16 bits. It indicates the identifier for a session within a tunnel. L2TP sessions are named by identifiers that have local significance only. That is, the same session will be given different session IDs by each end of the session. Session ID in each message is that of the intended recipients, not the sender.
- ***Ns:*** 16bits. Optional. Ns stands for "Next Sent" and indicates the sequence number for this data or control message, beginning at zero and incrementing by one for each message sent.
- ***Nr:*** 16bits. Optional. Nr stands for "Next Received" and indicates the sequence number expected in the next control message to be received. Thus, Nr is set to the Ns of the last in-order message received plus one.
- ***Offset Size:*** 16 bits. Optional. Specifies the number of bytes past the L2TP header at which the payload data is expected to start. Actual data within the offset padding is undefined. If the offset field is present, the L2TP header ends after the last byte of the offset padding. The offset field is not present in the control message.
- ***Offset Pad:*** Variable length. Optional. It pads the bits to an even 32-bit alignment.

L2TP employs different formats to form the data packet and the control message packet. Both the data packet and the control message packet begin with a common L2TP header. For the L2TP data packet, it is formed by an L2TP header, followed by the data payload (a PPP frame). The L2TP control message uses Attribute Value Pair (AVP) to exchange information. Typically, a non-zero set of AVPs is coded behind the L2TP packet header to be exchanged between the LAC and the LNS. The control message and the AVP format will be discussed in Chapter 3 (section 3.2.3).

2.2.5 Protocol Operation

The L2TP specification is mostly written to describe the Remote Access model. In this research, the simulation is designed based on the Remote Access model as well. Therefore, the discussion in this subsection will focus on the Remote Access model. Some differences in operation between the Remote Access model and the LAC Client model will be mentioned.

In order to tunnel a PPP session, two steps need to be done. The first is to establish a control connection for a tunnel between the LAC and the LNS. The second is to establish a data session that is triggered by an incoming call or an outgoing call. The order of these two steps cannot be ignored. The tunnel (i.e. the control connection) must be established before the data session is setup. The data session must be setup before the tunnel can begin to tunnel PPP session. Once the PPP session has been tunneled in an L2TP data session, the LAC can pass packets forth and back between the Remote System and the LNS.

In an L2TP tunnel, only one control connection exists, and multiple data sessions can be created and be operated over that tunnel, each data session for a single PPP stream. Usually, a single L2TP tunnel exists between an LAC and an LNS. If more users want to tunnel their PPP sessions between this LAC and LNS pair, they can just create their own data session within the existing L2TP tunnel. In the circumstance that different policies need to be taken for different quality of service (QoS), multiple tunnels may be generated between an LAC and LNS pair.

The things in an LAC Client Access model are little different from that in the Remote Access model. In an LAC Client Access model, the LAC Client exists in the

same software stack as the remote system PPP (i.e., the LAC Client software and the remote system PPP are on the same device). In this case, there will be only one tunnel between this device (the LAC) and a given LNS, and there will probably be only one data session going on each time in the tunnel.

When a PPP session is completed, the associated data session is brought down. The tunnel may be released after (and only after) all data sessions within it have been brought down. Either the LAC or the LNS can decide that the tunnel between them is no longer needed.

The establishment of the control connection will be discussed in Chapter 3 (section 3.2). After control connection is successfully established, individual data sessions may be created. The establishment of a session may be triggered either by an incoming call or by an outgoing call. For an incoming call, the LAC will initiate a session and request the LNS to accept the session. For an outgoing call, the LNS will initiate a session and request the LAC to accept the session.

When an incoming call comes to the LAC from a remote system, the LAC firstly detects whether or not there is already a tunnel up between it and the LNS to which the call is to be tunneled. If no tunnel exists yet, the control connection between them will first be established. Otherwise, a data session will be brought up. A data session is brought up through a three-message exchange. The typical sequence of data session establishment triggered by an incoming call is illustrated in Figure 2-6.

Figure 2-6. A typical sequence of L2TP data session establishment triggered by an incoming call

Where

- *ICRQ*: Incoming Call Request;
- *ICRP*: Incoming Call Reply;
- *ICCN*: Incoming Call Connection;
- *ZLB*: Zero Length Body. The ZLB is just used as an acknowledgement. It is not included in the three-message exchange.

Those messages are all control messages, and are carried through the control channel. After the LNS receives the valid ICCN, the data session is setup. The process of session establishment triggered by an outgoing call is similar to that shown in Figure 2-6, but the initiator is the LNS.

Once the data session is setup, the PPP session can be tunneled. As PPP frames are received by the LAC on the connection to the remote system, the LAC strips of any linked framing or transparency bytes, and checks the FCS (Frame Check Sequence). Then, the PPP frames are encapsulated in L2TP and sent to the LNS over the tunnel using the registered UDP port 1701 (the entire L2TP packet, including payload and L2TP header, is sent within a UDP datagram). The LNS accepts these L2TP packets, strips the L2TP encapsulation, processes the incoming frames for the appropriate interface, and then passes the PPP frames to the end receiver.

2.2.6 L2TP Implementations for VPN

As mentioned in previous sections, L2TP protocol is a virtual extension of PPP protocol across IP network infrastructure. It extends PPP to permit it to operate in separate networks over multiple links. L2TP architecture inherits all of the flexibility and breadth of protocol support present in PPP but it does not have to redefine or redesign this support. PPP provides a standard method for transporting multi-protocol datagrams over point-to-point links. It is ideally suited for VPN related applications because its functionality already mimics the behavior of what VPN would need: a point-to-point tunnel. Therefore, L2TP need only correctly tunnel PPP data between the L2TP peers to enable the multi-protocol VPN.

The core point of VPN is the security of the communication. L2TP provides two aspects of security concepts in the control connection establishment [Sri00]. One aspect is the authentication. When a tunnel is generated between a LAC and LNS pair, an optional authentication can be performed. This will help to make sure that a party is talking to the right partner. The other aspect is the ability to hide AVP value fields in control messages. In a control message, individual AVPs can be encrypted according to a shared secret between the L2TP peers. This process is known as AVP hiding [She00]. This feature can be used to hide sensitive control message data, such as user password or user IDs.

There is no security protection mechanism in L2TP for the PPP data being tunneled (of course, the L2TP tunnel can be secured by the transport being used, such as through using IPSec, but this is beyond the L2TP protocol). This leaves security to the PPP protocol security features. PPP provides two mechanisms to protect the communication. One is the connection authentication of point-to-point connections. This can be achieved by using CHAP (PPP Challenge Handshake Authentication Protocol), RADIUS [AZ00], or EAP (Extensible Authentication Protocol) [BV98]. The other mechanism is data encryption. This can be done using Microsoft Point-to-Point Encryption (MPPE) protocol.

The security mechanisms both in L2TP or PPP are optional. It is not necessary for each communication to use the strictest security method. Strict security means long process time and poor performance. Therefore, the key point is to choose the proper method according to the requirement of the application.

In this chapter, the VPN concept and L2TP protocol are reviewed and discussed, except for the process of L2TP control connection establishment. The simulation environment, the establishment of L2TP control connection and the simulation design will be discussed in the next chapter.

CHAPTER 3: THE NS ENVIRONMENT AND SIMULATION DESIGN PROCESS

In this research, the simulation of L2TP tunneling is performed on a platform named Network Simulator (*ns*), which provides us with an environment in which various protocols and algorithms can be simulated. In this chapter, the structure and principle of *ns* will be briefly described (section 3.1). The description will focus on those components that will be used in or related to the research in this thesis. The overview of L2TP protocol was described in Chapter 2, where the description on the process of control connection establishment was left. In this chapter, this process, which will be simulated in this research, will be discussed in detail (section 3.2). The design of the simulation model will be given in section 3.3.

3.1 Network Simulator (*ns*)

3.1.1 Overview

The Network Simulator (*ns*) is a discrete event simulator. It began as a variant of the REAL network simulator in 1989, where REAL is a network simulator originally intended for studying the dynamic behaviour of flow and congestion control schemes in packet-switched data networks. Since then, *ns* has been evolved substantially over the past few years. In 1995, *ns* development was supported by DARPA (the Defense Advanced Research Projects Agency) through the VINT (Virtual InterNetwork Testbed) project. The project is a joint effort by people from UC Berkeley, USC/ISI, LBL, Xerox PARC and researchers from other agencies.

Currently, *ns* development is supported through DARPA with SAMAN (Simulation Augmented by Measurement and Analysis for Networks) and NSF with CONSER (Collaborative Simulation for Education and Research). During the *ns* development, substantial contributions have also come from other researchers. At present, *ns* is still not a finished product, but the result of an on-going effort of research and development.

The version used in this research is ns-2.18b [FV00].

3.1.2 The structure of *ns*

ns is an object oriented simulator, which targets networking research. It provides users with a platform for simulating basic and more complicated network behaviour. It allows users to create an event scheduler, create network models, compute routes, create connections among nodes, create traffic, trace events and so on.

The *ns* structure consists of two parts: the base elements written in C++ and the front-end interpreter written in OTcl. The elements in C++ part work at the lower level. They are responsible for dealing with the data, such as the detailed simulation of protocols, packet action and so on. The front-end interpreter is responsible for control issues, such as periodic or triggered actions, and provides users with an interface. Users can create simulation scenarios, schedule events and perform their simulation at the OTcl level. They do not need to do any C++ programming work unless they want to add new functionality to *ns*.

ns supports a class hierarchy in C++ (named compiled hierarchy) and a similar class hierarchy in the OTcl interpreter (named interpreted hierarchy). The two hierarchies are closely related to each other. From the user's perspective, there is a one-to-one correspondence between a class in C++ side and one in OTcl side. Through `tc1cl` (the OTcl and C++ linkage, all of the code for these uses is given in a separate directory, `~tc1cl/`), *ns* provides a mechanism to make objects and variables appear on both sides. Figure 3-1 illustrates this relationship.

Figure 3-1. The linkage between OTcl objects and C++ objects

For establishing the linkage between an instance variable on one sides and the corresponding peer on the other side, the user can create a bi-directional binding. Thus, both the compiled instance variable and interpreted instance variable will access the same data. The binding is created in the constructor of a C++ class, such as:

```
L2tpAgent_lac::L2tpAgent_lac() {
    bind("packetSize_", &size_);
    bind("off_l2tp_", &off_l2tp_);
}
```

In `tclcl`, a class named `Tcl` is defined. It encapsulates the actual instance of the `OTcl` interpreter, and provides methods to access the interpreter. Through some methods defined in this class, such as `tcl.eval()` and `tcl.result()`, users can also call an interpreted procedure (either in the `Tcl` script or in `OTcl` classes) or access an interpreted variable from the compiled side. Correspondingly, compiled classes use a specific method named `command()` to provide a gate to `Tcl/OTcl` interpreter. Thus, users can accomplish the communication between compiled classes and interpreted classes.

The class hierarchy consists of a bunch of classes. The root of this hierarchy is class `TclObject`. It is the base class for most of the other classes in the interpreted and compiled hierarchy. A partial class hierarchy (the view in C++ side) is illustrated in Figure 3-2.

Figure 3-2. Partial *ns* class hierarchy (modified from Hua99)

In this hierarchy, class `Simulator` provides the general interface to users. From this interface, the user can configure and implement a simulation. The OTcl class of `Simulator` is defined in directory `~ns/tcl/lib/`. Each simulation requires a single instance of the class `Simulator` to control and operate the simulation. Through a `Simulator` instance, the user can create a scheduler, change a scheduler, schedule events, trigger the scheduler to start the simulation and so on. Some examples for the uses of this interface are the following:

```

set ns [new Simulator]      # create a scheduler
$ns use-scheduler Heap      # change the default scheduler
                             # into Heap scheduler
$ns at <time> <event>       # schedule a event
$ns run                     # start scheduler

```

3.1.3 The Network Topology in *ns*

In order to implement a simulation, a network topology must be generated. From the user's perspective, a network topology consists of a set of nodes and links. Two nodes are connected through a link.

The OTcl `Node` is a standalone class. However, most of the components of the node are themselves `TclObject`. The structure of a node is shown in the *ns* manual [FV00]. The simplest node is the unicast node. The components included in a unicast node are two `TclObjects`: an address classifier and a port classifier. A number of agents can be attached to a node. The function of these classifiers is to distribute incoming packets to the correct agent or outgoing link. When receiving a packet, the node examines the packet's fields, usually the field for destination address. If the node in question happens to be the destination of the packet, the packet will be passed to the corresponding agent through the port classifier. Otherwise, the node maps the field value of the packet to an outgoing object that is the next downstream recipient of this packet.

The node is created through the `Simulator` interface. The method for creating a node is as follows:

```

set ns [new Simulator]
set n1 [$ns node]

```

Those commands will create a node object and put it in the variable `n1`.

The class `Link` is a standalone class in OTcl as well. The class `SimpleLink` is a subclass of class `Link`. It provides the ability to connect two nodes with a point-to-point link. *ns* provides the instance procedure `simplex-link{}` to form a unidirectional link from one node to another. The instance procedure `duplex-link{}` constructs a bi-directional link from two simplex links. A link between two nodes is generated through the Simulator interface:

```
set ns [new Simulator]
set n1 [$ns node]
set n2 [$ns node]
$ns duplex-link $n1 $n2 <bandwidth> <delay> <queue_type>
```

Three parameters are involved in the generation of a link: `bandwidth`, `delay` and `queue_type`. `bandwidth` is the speed of the link in bits/sec. `delay` is the time (in second) used for a bit to travel through the link. `queue_type` specifies the type of the queue used in this link. The queues provided in *ns* include DropTail, RED, CBQ, FQ, SFQ, and DRR. Usually, we use DropTail queue in the simulation. DropTail queue implements FIFO scheduling and drop-on-overflow buffer management, which are typical in most present-day Internet routers.

The time used to send a packet from one node to another can be calculated. Typically, this time includes three parts: transmission delay, delivery delay and queuing delay. The transmission delay relates to the bandwidth of the link and the packet size. It can be calculated as s/b , where s is the packet size in bit, b is the bandwidth in bits/sec. The delivery delay relates to the distance between two nodes. These two delays are static, and must be specified when a link is generated. Both of them are handled in class `Linkdelay`. The queuing delay is dynamic. It depends on the instant traffic level of the link. If the link is busy when a packet arrives, the packet will be put into a queue until the link become free. The queuing delay is calculated in class `queue` or the particular derived queue classes.

In *ns*, the node acts as a starting point or an ending point of the link. The mechanism of competing for the CPU of the node is not present in *ns*.

3.1.4 Routing

Two types of routing can be performed in *ns*: unicast routing and multicast routing. They are implemented through different types of nodes. As described in 3.1.3, the node for unicast routing contains only one address classifier and a port classifier. To implement multicast routing, the nodes must be extended with extra components. Those components include a multicast classifier and several replicators. The multicast classifier is used to classify packets according to both source and destination group addresses. It maintains a table mapping source/group pairs to slot numbers. The replicator is responsible for producing additional copies of the packet, one for each entry in the group. Thus, a copy of packet can be delivered to each of the nodes listed in the classifier's table.

In a typical routing implementation, three function blocks may be involved: routing agent, route logic and classifier. Routing agent (rather than transportation protocol agent or application agent) is responsible for exchanging routing packet with neighbours. Route logic uses the information gathered by routing agents to perform the actual route computation. Classifiers inside a node use the computed routing table to perform packet forwarding.

ns employs three routing strategies: static, session and dynamic. The static route computation is the default mode in *ns*. It uses Dijkstra's all-pairs SPF algorithm. In static mode, the route algorithm is run exactly once prior to the start of the simulation. The routes are computed using an adjacency matrix and link costs of all the links in the topology.

The session routing mode uses the same algorithm and method as that used in the static mode, and runs the routing algorithm prior to the start of the simulation as well. However, it will also run the same algorithm to compute routes in the event that the topology changes during the course of the simulation.

The dynamic routing mode implements Distributed Bellman-Ford (i.e. Distance Vector) algorithm. The implementation sends route update periodically. If the topology changes or an agent at the node receives a route update, the new route will be recomputed and installed.

3.1.5 Transport and Application Agents

The Transport and Application layer features are implemented through a series of agents. Agents represents endpoints where network layer packets are constructed or consumed, and are used to implement the protocols. The main Transport agents defined in *ns* are a set of TCP agents and UDP agents. The typical Application agents are FTP agent, Telnet agent and a set of traffic generators, such as Exponential (generate traffic according to an exponential on/off distribution), Pareto (generate traffic according to a Pareto on/off distribution) and CBR (i.e. Constant Bit Rate, which generate traffic according to a deterministic rate. There is a CBR agent in the Transport layer as well, which may cause confusion sometimes).

All Transport agents are derived from class `Agent`, which is implemented partly in C++ and partly in OTcl. The class `Agent` supports packet generation and reception. From Figure 3-2 it can be seen that class `Agent` is derived from class `Connector`. In class `Connector`, a mechanism for sending packets is defined. After a packet is generated in an agent, the `send()` method in class `Connector` is called to send the packet downstream one hop. The Application agents are derived from class `Application`, which is a subclass of `TclObject`.

Agents can be created from the Simulator interface. The Transport agents should be generated in pairs (one is the sender, and the other is the receiver). After the sender agent and the receiver agent are generated, they must be attached to a pair of nodes respectively: the sender to the source node, and the receiver to the destination node. These two agents, then, are connected to each other for delivering packets. Application agents are attached to Transport agents. The following is the example to create, attach and schedule agents.

```
set ns [new Simulator]
set n1 [$ns node]
set n2 [$ns node]
$ns duplex-link $n1 $n2 5Mb 30ms DropTail

set tcp [new Agent/TCP]
set sink [new Agent/TCPSink]
$ns attach-agent $n1 $tcp
$ns attach-agent $n2 $sink
```

```

$ns connect $tcp $sink
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 1.0 "$ftp start"

```

If a user wants to implement a protocol for which an agent does not exist yet, the user has to develop a new agent and bring it into *ns*. Since *ns* did not contain an L2TP protocol agent, we have to develop one for the thesis.

3.2 The Selection of L2TP Features for the Simulation

The L2TP protocol was reviewed in Chapter 2. From this review it can be seen that two basic components are included in the protocol. This research will focus on one of them, the tunnel generation (i.e., the establishment of control connection). It is the key component of the L2TP protocol. In this section, the process and the method of tunnel generation are described, and the implementation of it is discussed.

3.2.1 The Establishment of Control Connection

The existence of a tunnel between two peers is tightly linked to the existence of a control connection between two peers. Therefore, the terms *tunnel* and *control connection* are sometimes used interchangeably [She00]. A control connection is established through a series of message exchanges between an LAC and an LNS. Either an LAC or an LNS can initiate a control connection. Once the control connection is established, the tunnel is said to be fully established.

The messages used to handle (establish, maintain, and release) control connection are called control messages. The control messages are delivered through a control channel. The control channel provides a reliable delivery mechanism to the control message between two L2TP peers (the LAC and the LNS). There are many types of control messages, in which three of them (SCCRQ, SCCRP and SCCCN) are used to generate the tunnel. Besides these three messages, a special message (ZLB) is used as an acknowledgement. The four messages involved in the process of a control connection establishment are described as follows:

- *Start-Control-Connection-Request (SCCRQ)* is used to initialize a tunnel between an LAC and an LNS. It is usually sent by the LAC, but can also be sent by the LNS to start the tunnel establishment process.
- *Start-Control-Connection-Reply (SCCRP)* is sent in reply to a received SCCRQ message. The SCCRQ is used to indicate that the SCCRQ was accepted. In this case, the process of the tunnel establishment should continue.
- *Start-Control-Connection-Connected (SCCCN)*: The SCCCN message is sent in reply to an SCCRQ. It completes the tunnel establishment process.
- *Zero-Length-Body (ZLB)*: A ZLB message is a special message that is used solely to send acknowledgement for control messages.

A three-message exchange is employed to setup a control connection. A typical process of message exchange is illustrated in Figure 3-3.

Figure 3-3. A typical process of three-message exchange in L2TP control connection establishment

When an LAC receives an incoming call from the remote system that needs to be tunneled to an LNS, if there is not a tunnel between this LAC and LNS pair, the LAC will create a SCCRQ message based on its local policy for the tunnel, and then, send it to the LNS. After the LNS receives the SCCRQ, it takes actions to look up information in its storage and compare it with the parameters found in the received SCCRQ. If the SCCRQ is acceptable, the LNS responds the LAC with a SCCRQ message. The LAC, then,

examines the received SCCRP message. If everything is fine, it sends a SCCCN message to the LNS. If the received SCCCN is acceptable for the LNS, the tunnel is successfully generated. Otherwise, establishment of the tunnel must be disallowed [Tow99].

Since the establishment of the control connection is a reliable delivery, the LNS must send an acknowledgement for the SCCCN to the LAC, or the LAC will keep re-transmitting the SCCCN message. The acknowledgement is a ZLB message. Though the ZLB message is not included in the three-message exchange, it is involved in the process of the tunnel generation. This is why the ZLB sending is included in Figure 3-3.

During the three-message exchange, the mutual authentication of the L2TP endpoints can be performed. The information for the authentication is carried by three messages. The tunnel authentication will be discussed in section 3.2.2.

L2TP employs an exponential back-off interval algorithm to handle the packet retransmission. The retransmission mechanism will be discussed in section 4.1.3.

3.2.2 Tunnel Authentication

L2TP incorporates a CHAP-like [Sim96] tunnel authentication system during control connection establishment. CHAP is a strong authentication protocol. It uses secret keys between the peers. To participate in tunnel authentication, a single shared secret must exist between the LAC and the LNS. The authentication can be performed either in one-direction or mutually, depends on the requirement of applications. In this research, a mutual authentication is performed.

The authentication is accomplished as follows:

1. The LAC randomly generates a challenge, puts it in the SCCRQ, and sends the SCCRQ to the LNS.
2. The LNS receives the SCCRQ. After determining the shared secret, the LNS calculates the response based on the challenge from the SCCRQ. Then, the LNS generates its own challenge. Both the response and the challenge are put in the SCCRP, then, the SCCRP is sent to the LAC.
3. After receiving the SCCRP, the LAC verifies that the peer agrees on the shared secret based on the response carried in the SCCRP. If the value received in the response is equal to the expected value, the LAC calculates its response based on

the challenge carried in the SCCRP. The response is put in the SCCCN, and sent to the LNS.

4. After receiving the SCCCN, the LNS verifies the response in the SCCCN message. If the value received in the response is equal to the expected value, the LNS sends an acknowledgement (a ZLB message) to the LAC. The tunnel is successfully generated.

The response value is the one-way hash calculated over a stream of octets consisting of the identifier, followed by (concatenated with) the “secret”, followed by the challenge value [Sim94]. The length of the response value depends upon the hash algorithm used (128 bits in L2TP tunnel authentication). The process of the mutual authentication is illustrated in Figure 3-4.

Figure 3-4: ACHAP-like authentication (modified from Black[Bla00])

3.2.3 Control Message Format and Packet Size

The L2TP control message consists of the L2TP packet header followed by a non-zero set of Attribute Value Pair (AVP) instances. Instead of defining control message to consist of static fields, the L2TP protocol defines control message to comprise a list of AVPs. The format of the L2TP AVP is shown in Figure 3-5.

Figure 3-5: The format of the L2TP Attribute Value Pair (AVP)

Where

- **Mandatory (M) bit:** Controls the behavior required of an implementation which receives an AVP which it does not recognize. If the M bit is set on (= 1), a receiver that does not recognize this AVP must terminate the tunnel. If the M bit is not set, an unrecognized AVP must be ignored.
- **Hidden (H) bit:** Identifies the hiding of data in the Attribute Value field of an AVP. This capability can be used to avoid the passing of sensitive data as clear text in an AVP.
- **Length:** Encodes the number of octets contained in this AVP. This field itself is 10 bits, permitting a maximum of 1023 octets of data in a single AVP. The minimum Length of an AVP is 6 octets.
- **Vendor ID:** The IANA assigned "SMI Network Management Private Enterprise Codes" [RP94] value.
- **Attribute Type:** The Attribute Type field is used with the Vendor ID field to uniquely identify the AVP instance.
- **Attribute Value:** The actual value of the attribute. It follows immediately after the Attribute Type field, and runs for the remaining octets indicated in the Length (i.e., Length minus 6 octets of header).

An AVP is an extensible format for containing a message field. By using AVPs, the L2TP control messages can easily support optional fields simply by including or not including the AVP in the message. If extensions need to be done on the L2TP control message in the future, we simply define a new AVP and add it into the message.

There are several types of AVPs. They are used for different purposes. Some of them are applicable to all control messages, and some of them are optional. According to RFC 2661, The following AVPs must be present in the SCCRQ and the SCCRP:

- Message type AVP
- Protocol version AVP
- Host name AVP
- Frame capabilities AVP
- Assigned tunnel ID AVP

In the SCCCN, only the Message Type is the required AVP. In addition, there are several AVPs optional for the control messages. In order to determine the packet size which can meet the requirement of all control messages in the simulation, the AVPs that may be used in the control messages are listed as follows:

- Message type AVP: 8 bytes (6 bytes for the header, 2 bytes for the values)
- Protocol version AVP: 8 bytes
- Host name AVP: 10 bytes
- Frame capabilities AVP: 10 bytes
- Assigned tunnel ID AVP: 8 bytes
- Challenge AVP: 8 bytes
- Challenge response AVP: 22 bytes

The octets used for others:

- L2tp header: 12 bytes (offset field is not present in the control message)
- UDP header: 8 bytes
- IP header: 20 bytes

The number of octets required by the items listed above is 114 in total. This means that the minimum packet size for the L2TP control message in this simulation is 114 bytes. In order to reserve enough room for the potential use of some optional messages, the size of the L2TP packet used in this simulation is defined as 200 bytes.

3.3 Simulation Design Process

Simulation is a complex process in which many details are involved. It imitates the operation of a real-world process or system over time. In order to get a good result, the simulation process should be carefully designed. The following subsections give the

procedures and designs of the simulation that will be conducted in this research, which are based on the general steps of simulation modeling [BCN96, Law91, Sha75].

3.3.1 Problem Formulation and Objectives

The real-world system being simulated in this research is a group of networks with L2TP tunneling components running on them. The objective of the simulation is to determine the tunneling time and the values of some other important parameters in different networks under different conditions, and if possible, to improve the policy or algorithm that influences the system performance in a negative way. The following factors were varied in the simulation experiments:

- Network size. The simulation will be conducted for networks in two different sizes. One group of networks consists of 54 nodes, and the other group consists of 108 nodes.
- Node degree (or connectivity level, see section 4.2 for an explanation). For each network size, two topologies are constructed. One has a low node degree (of 2.19), and the other a high node degree (about 3.60).
- Traffic rate. In order to emulate the real-world network, traffic should be brought into the network to compete for the network resources with the L2TP tasks. The simulation will be conducted under three traffic rates (i.e. the sending rate of traffics during "On" times): low level traffic (at 100Kb/sec.), middle level traffic (at 500Kb/sec.), and high level traffic (at 1000Kb/sec.).

Besides tunneling time, the number of hops, the number of retransmissions, and the CPU waiting time (this feature needs to be developed, see 4.1.4 for the details) for each tunnel will be determined by simulation and used in performance analysis.

3.3.2 Model Conceptualization and Translation

Modeling a network system and implementing a simulation on it need many components to be involved. *ns* just provides the users with a general simulation platform. In this research, a model for the simulation of L2TP tunneling was built. The model comprises four basic building blocks:

- **L2TP agent:** The development of L2TP agents is the core task in modeling the system. A pair of L2TP agents needs to be developed. One is an `L2tp_lac` agent, and the other is an `L2tp_lns` agent. The `L2tp_lac` agent will be attached to an LAC node, and it is responsible for making a request to initialize a tunnel. The `L2tp_lns` agent resides in an LNS node, and it acts as the receiver. These two agents work together to perform tunnel establishment. The design and the development of these two agents are described in section 4.1.
- **Network topology:** Four network topologies corresponding to the conditions described in 3.3.1 were generated for implementing the simulation. There is a tool kit named GT-ITM associated with the *ns* package. It can be used to generate Transit-Stub network topologies. The generation of network topologies is discussed in section 4.2.
- **Traffic generation:** The background traffic was generated according to an Exponential On/Off distribution at three different sending rates (100 Kbps, 500 Kbps and 1000 Kbps). This is discussed in section 4.3.
- **Simulation scenarios:** After all of the components were developed, they were combined together. A scenario, which itself is a building block, acts as a frame which integrates all building blocks and makes everything ready to work. According to the combinations of all experimental factor values given in 3.3.1, twelve scenarios were constructed for the simulation. They are described in section 4.4.

After the conceptual models were created, each was implemented either in C++ or in Tcl/OTcl. The L2TP agents are programmed in C++. The network topologies and traffic are implemented in Tcl/OTcl. The scenarios are represented by a set of Tcl scripts in which the events are scheduled according to determined policies.

3.3.3 Model Verification and Validation

After the simulation model is developed, it needs to be verified and validated. Verification pertains to the computer program prepared for the simulation model. The purpose of verification is to examine if the computer program performs properly. This

includes the examination of input parameters (the system configuration) and the examination of logical structure of the model. The verification may be done in different stages, i.e., after each component is developed, it should be debugged and tested thoroughly. In this research, the logical examination was done in two ways: functional examination and structural examination.

The functional examination means to trace each functional feature individually, from its starting point to the ending point. This can help us to make sure the functions of the feature are fully fulfilled. For example, in order to examine the retransmission (abbreviated as rtx) mechanism, we had to trace the entire rtx process, from rtx packet regeneration to timeout triggering, from the LAC sending rtx packet to the LNS receiving the packet, from dealing with the reception of duplicated rtx packets to the cancellation of the rtx timer etc. Functional examination guarantees that an individual feature works properly.

In structural examination, the model was examined as a whole. The purpose of structure examination is to check whether the model contains all features required in the simulation, and whether those components can work together (interact with each other) correctly. The structural examination can help us to make sure that the model works properly as a whole.

Validation is concerned with building the right model. It is the determination that the model is an accurate representation of the real system. During the development of the model, when each feature (or component) is completed, it must be validated. Finally, the whole model must be validated. The validation can help us to make sure the model will meet the requirements laid in the problem formulation. A component named *nam* in *ns*, which is a graphic interface, can be used to monitor a running model and its results.

3.3.4 Production Runs and Analysis

Production runs and their subsequent analysis are used to estimate measures of performance for the system designs. In this research, for the normal case, the length of each simulation run (also called a replication) is 2000 tunnels (note: the run length means the number of successfully created tunnels in a run. This does not include the failed tunnels). The result of the simulation in a run was divided into 20 batches. Thus, each

batch includes 100 tunnels. For the simulation of each scenario, a number of runs may need to be conducted for getting a satisfactory result. The number of runs required for each simulation scenario depends on the convergence of the output data. If the output data are convergent fast, few runs can satisfy the given criterion. Otherwise, more runs are required. In order to get an acceptable accuracy, the confidence interval and half-length error [BCN96, LK82] are employed to evaluate the sample data. The simulation will be repeated in run until a 95% confidence interval with a 6.5% half-length (h.l.) width is met.

For some scenarios, 2000 tunnels in a run may be not enough, and the 6.5% h.l. width may be difficult to reach. In this case, a longer run will be employed, and a slight bigger error (such as 7.5% h.l. width) is accepted.

The simulation implementation and output analysis are described in Chapter 5.

3.3.5 System Improvement

Through the analysis of the simulation results, the main factors that have a negative impact on system performance can be found. Consequently, we will attempt to change the respective algorithms and policies in order to improve system performance. This will be discussed in Chapter 6.

In this chapter, the simulation environment is described. The features of L2TP covered in the simulation are discussed. The simulation method, procedures and experimental factors have been described. Chapter 4 will describe the simulation model.

CHAPTER 4: SIMULATION MODEL

This chapter is the implementation of simulation modeling. As discussed in section 3.3, the components involved in the simulation in this research are grouped into four basic building blocks: L2TP agents, network topology, traffic generation and simulation scenarios. The modeling of the four building blocks is described in this chapter.

4.1 The Development of L2TP Agents

Packet sending and receiving in *ns* are handled by a series of protocol agents. In order to generate an L2TP tunnel, three control messages plus a ZLB message need to be carried over the network. This task will be achieved by the L2TP agent, which is different from other protocol agents in *ns*, which only deal with a one-way or two-way sending-receiving. Once an L2TP agent is triggered, it will start to send SCCRQ to create the tunnel until it receives the ZLB.

4.1.1 The Structure and Functionalities of the L2TP Agents

The L2TP agent is a compound agent, written in C++. It comprises three parts: a packet struct (C++ struct), an L2tp_lac agent and an L2tp_lns agent. The packet struct is used to define and generate packets for L2TP control message. It contains a set of variables that are used to pass information during the tunnel establishment. Agent L2tp_lac is the LAC side agent. It plays the role of tunnel initiator. Agent L2tp_lns is placed on the LNS side, which is the ending point of the tunnel. The L2tp_lac agent and the L2tp_lns agent cooperated with each other to generate tunnels over the network. The main functionalities of agent L2tp_lac can be summarized as follows:

- Generating and sending the SCCRQ packet, starting the tunnel generation.
- Re-transmitting the SCCRQ packet until the ACCRP is received.

- Receiving and processing the SCCRP packet.
- Generating and sending the SCCC� packet.
- Re-transmitting the SCCC� packet until the ZLB packet is received.
- Receiving and processing the ZLB packet.
- Handling the protocol execution and authentication in the LAC.
- Implementing the CPU competition mechanism.
- Collecting and distributing information for adaptation purposes in the improved agents (see Chapter 6 for the detail).
- Terminating the process if receiving an invalid message.
- Communicating with *ns* to schedule the next tunnel.

The main functionalities of agent `L2tp_lns` can be summarized as follows:

- Receiving and processing the SCCRQ packet.
- Generating and sending the SCCRP packet.
- Receiving and processing the SCCC� packet.
- Generating and sending the ZLB packet.
- Handling the protocol execution and authentication in the LNS.
- Implementing the CPU competition mechanism in the LNS.
- Outputting the statistic information (such as tunneling time, hops, the number of retransmissions etc.)

4.1.2 Agent Implementation

To generate a tunnel, we need to create an instance for both `L2tp_lac` and `L2tp_lns`, and attach them to a pair of nodes (one is the source, and the other the destination). Then *ns* schedules an event to trigger the `L2tp_lac` agent to start the tunnel generation. In order to generate a number of tunnels in a run (2000 tunnels in the normal case, see the designs in section 3.3.4) for the statistic purpose, agents must keep working until the end of a simulation run is reached. Each time a tunnel is completed, the corresponding pair of agents (an `L2tp_lac` and an `L2tp_lns`) will be detached from the nodes they were attached. Then, *ns* re-schedules a new event to re-attach them to

another pair of nodes, and to start a new tunnel. This process is repeated until the pre-determined number of tunnels is reached.

The actions and the working flow of an L2tp_lac agent are illustrated in Figure 4-1. In this diagram, end states are shown as thick circles; the middle states are shown as thin circles; events triggered by other agents are shown as ellipses; actions are shown as rectangles; Boolean decisions are shown as diamonds. The meanings of the states are:

Figure 4-1: Logic working flow of agent L2tp_lac

- *Idle.* This is the initial and the end state. Before the agent is triggered and after a tunnel is established, the agent enters the idle state.
- *Prepare the SCCRQ packet.* When an LAC agent begins to establish a tunnel, it firstly prepares a packet for sending the SCCRQ. This is a complex action. It includes three sub-actions. The first is to get the current time and assign it to an instance variable for the use of tunneling time computation. The second is to set the semaphore to ensure the mutually exclusive operation on the CPU (see section 4.1.4 for details). The third is to set a timer to emulate the occupation of CPU. The CPU is occupied by the current agent to execute L2TP, UDP and set a challenge to the SCCRQ. During this period of time, the agent "falls asleep". When the timer expires, the agent is waken up and goes to the next step.
- *Max rtx.* The value of this factor is the maximum number of times that the control message will be retransmitted. According to RFC 2661 [Tow99], the default value of maximum retransmission times is 5.
- *Fail.* If no peer response is detected after 5 retransmissions, the generation of this tunnel fails.
- *Send SCCRQ.* The LAC sends the SCCRQ to the LNS.
- *Set rtx timer for SCCRQ.* Set a retransmission timer (rtx timer) for the SCCRQ retransmission purpose.
- *Timer valid?* This is a Boolean value. If the rtx timer is canceled, it becomes invalid. Otherwise, when the timer expires, it will trigger the SCCRQ retransmission.
- *Receive SCCRP.* The LAC receives the SCCRP that comes from the LNS.
- *Valid pkt?* When receiving the SCCRP, the L2tp_lac agent will examine the packet. If this is the first time the agent receives the SCCRP for creating the current tunnel, and if the packet comes from the right source node and gets to the right destination node, the packet is valid. Otherwise, the packet is invalid.
- *Discard packet.* Through the examination given above, if the received packet is an invalid packet, it is discarded.
- *Cancel timer.* If the received packet is valid, the L2tp_lac agent cancels the rtx timer. The rtx timer becomes invalid.

- *Set semaphore.* If the packet is valid, the agent applies for the CPU to process the packet. It firstly checks the CPU semaphore. If the semaphore shows that the CPU is occupied, the agent has to wait until the semaphore expires. The agent then sets a new semaphore and occupies the CPU. See section 4.1.4 for details of semaphore handling.
- *Process SCCRP.* The CPU processes the SCCRP packet. It includes: executing UDP twice (one for receiving the SCCRP, one for sending the SCCCEN), executing L2TP twice (same as UDP), examining the authentication response made by the LNS, calculating the response for the LNS challenge and put it into the SCCCEN.
- *Send SCCCEN.* The LAC sends the SCCCEN to the LNS.
- *Set rtx timer for SCCCEN.* After sending out the SCCCEN, the agent sets a timer for the SCCCEN retransmission purpose. When the timer expires, the agent will retransmit the SCCCEN packet.
- *Receive ZLB.* The LAC receives the ZLB packet from the LNS. The succeeded steps, i.e. checking packet, canceling timer and setting semaphore, are similar to that for the SCCRP.
- *Process ZLB.* The CPU processes the ZLB packet. It includes: executing UDP, executing L2TP. After the LNS receives the SCCCEN and accepts it, the tunnel is considered fully established. The ZLB packet only acts as an ACK. It just tells the L2tp_lac agent to cancel the SCCCEN rtx timer. Therefore, the step *Process ZLB* may not be implemented in the simulation.
- *Call scheduler for next tunnel.* Once the current tunnel is established, the agent calls the *ns* to schedule next tunnel establishment. A new pair of nodes will be selected and acts as the LAC and the LNS. See section 4.4 for details.

The actions performed by the L2tp_lns agent are much simpler than that of the L2tp_lac agent. They are illustrated in Figure 4-2. The meaning of those actions can be described as follows:

- *Receive SCCRQ.* The LNS receives the SCCRQ from the LAC.
- *Set semaphore.* This action is the same as that done by the L2tp_lac agent. But it works on the LNS.

- *Process pkt.* The LNS CPU processes the SCCRQ packet. It includes: executing UDP twice (one for receiving the SCCRQ, one for sending the SCCRP), executing L2TP twice, calculating the response for the LAC's challenge, setting an authentication challenge to the LAC.
- *Send SCCRP.* The LNS sends the SCCRP to the LAC, in which a challenge is included.
- *Wait for event.* After the LNS sends a packet to the LAC, it waits for the next packet from the LAC. The next event is either receiving packet, or the tunnel is torn down by the LAC due to a failure.

Figure 4-2: Logic working flow of agent L2tp_lns

- *Receive SCCCEN.* The LNS receives the SCCCEN from the LAC.
- *Set semaphore & process pkt.* These two actions are similar with the previous ones. But processing the SCCCEN packet contains different contents. They are: executing UDP (only for receiving the SCCCEN, the time for sending the ZLB is not included in the tunnel generation), executing L2TP, and examining the authentication response made by the LAC.
- *Send ZLB.* The LNS sends the ZLB packet to the LAC. This is just for the acknowledgement use.

- *Output data.* Once the LNS accepts the SCCCN, the tunnel is successfully established. The agent, then, writes the statistic data about the tunnel generation into a file. See section 4.1.5 for details of the output data.
- *Complete?* Each time a tunnel is generated, the L2tp_lns agent will check the output file to make sure whether the number of tunnels in total hits the pre-determined number (e.g. 2000 tunnels in a run) already. If yes, stop the simulation. Otherwise, wait for the next event.
- *End.* Ending the simulation.

4.1.3 Packet Retransmission

The control connection in L2TP is a reliable delivery mechanism. Therefore, packet retransmission is a very important feature in L2TP tunneling. According to the L2TP protocol (RFC 2661[Tow99]), the timeout value of retransmission begins at 1 second. Each subsequent retransmission of a message employs an exponential back-off interval. Thus, the second retransmission occurs after 2 seconds, then 4 seconds, etc. The maximum interval between retransmissions is specified to be 8 seconds. Therefore, the retransmission intervals should be 1, 2, 4, 8 and 8 seconds.

The maximum number of times that the LAC will attempt to retransmit a control message is 5. If no peer response is detected after 5 retransmissions, the effort of tunnel establishment fails.

The packet retransmission is handled by the L2tp_lac agent. A timer (rtx timer) for retransmission purpose is defined in the L2tp_lac agent. After the agent sends out the SCCRQ, it sets the rtx timer with a corresponding interval value. When the timer expires, if no acknowledgement is received, the L2tp_lac agent will retransmit the SCCRQ packet. The retransmission will continue until either an acknowledgement (i.e. the SCCRP packet) is received or the number of times for retransmissions is over 5. Once the agent receives the SCCRP, it cancels the rtx timer.

Before the SCCRQ packet is transmitted, it takes a while for the LAC's CPU to prepare the packet (see Figure 4-1). After the packet is prepared, a copy of it can be used in the retransmission. Therefore, no more CPU work is needed in the subsequent retransmissions.

In LNS side, every received SCCRQ packet is treated equally by the L2tp_lns agent. They all must be processed by the CPU, no matter whether it is an original packet or a retransmitted packet. The agent will create a new SCCRP packet for each received SCCRQ packet, and sends it to the LAC as the acknowledgement.

Once the L2tp_lac agent receives the SCCRP, it cancels the rtx timer immediately. No more SCCRQ retransmission occurs after that. The first received SCCRP packet is considered valid. It will be processed. All other subsequently received SCCRP packets are considered invalid, and will be discarded.

The retransmission of the SCCCN is similar to that of the SCCRQ. When the L2tp_lac agent sends the SCCCN out, it makes a copy of it for the retransmission use. This copy is held in an instant variable of the agent. When the rtx timer expires, it will be taken out and used in the retransmission.

To examine the validity of the received packet, we need to make sure that the packet comes from the right source node and gets to the right destination. This problem may happen if some retransmitted packet (SCCCN packet in particular) is still on the way when the agents have been detached from the current nodes, and been reattached to another pair of nodes after the tunnel generation is completed. In this case, the packet is still sent to the original destination node. If a new agent has been attached to this node to the same port as the previous agent, it will receive a weird packet.

In order to deal with this problem, two instance variables, `nodeID_` and `destID_`, are defined in both L2tp_lac agent and L2tp_lns agent. They contain the LAC node's ID and the LNS node's ID respectively. When a pair of nodes is selected to carry agents, the IDs of the nodes are respectively assigned to `nodeID_` and `destID_` of both agents. Thus, when the agents receive packets, they can compare the values of `nodeID_` and `destID_` with the values carried in the corresponding fields of the packet. If they do not match, the packet is considered to be invalid and will be discarded.

4.1.4 CPU Resources Competition Mechanism

In the real network, current generation routers can perform routing directly between interfaces without involving the main CPU. It is said that routing happens in the "data plane" (or in the interrupt level) of the router. For the routers on the end points of a

tunnel, such as the LAC and the LNS, some tasks need to be done in the so-called “control plane” above the IP layer. These tasks consume CPU time. Examples of such tasks are: packet encapsulation with protocols, setting challenge, authentication and so on. In general, the developers of software are intended in the efficiency of operations both in the data and the control plane.

As mentioned before, *ns* is a network simulation platform. Though node in *ns* represents router in a real network, it does not have any mechanism to compete and consume CPU resources. In order to emulate the conditions of a real network, and get a more accurate result in the simulation, we need to take the competition for processor resource into account (i.e. more users may compete for the CPU resource introducing additional delays). In this research, a semaphore mechanism is designed and developed to emulate the mutually exclusive operation on the CPU resource.

In the real operating system, a semaphore is an integer. It simply acts as a signal. The signal operation of a semaphore removes one process from the queue and puts it in the list of ready processes [SG98]. This mechanism will not work in the simulation since the simulation works in a different way from the real system. In a discrete-event simulation, each event must be explicitly scheduled. The simulation goes on by following the schedule table. Each event occurs according to its scheduled time. In order to simulate the CPU execution time, we use a semaphore that can be acquired by one task at a time. The semaphore holding time will represent the CPU execution time. An instance variable `semaphore_` is added to class `node` (OTcl class, defined in `ns-node.tcl`):

```
Node set semaphore_ 0
```

Then, two member procedures are defined in the node. They are used as the interface for the semaphore operation.

```
Node instproc get_semaphore {} {
    Node instvar semaphore_
    return $semaphore_
}

Node instproc set_semaphore {value} {
    Node instvar semaphore_
    set semaphore_ $value
}
```

Besides the changes in node class, both L2tp_lac and L2tp_lns agents must provide the mechanism to operate the semaphore. At the same time, the scenario script must provide an interface so that the agents and the nodes can communicate with each other. The principle of the "CPU" semaphore is illustrated in Figure 4-3. In Figure 4-3 two possible cases are illustrated. They are:

- *Case 1:* Suppose *agent1* attempts to use the CPU. It comes at time $t_1 = 1$ (the current time of the system clock), and will occupy the CPU for $\Delta t (= 5 \text{ ms})$. It firstly calls procedure `get_semaphore{}` to get the current semaphore value S . Currently this value is 0. Then, it compares its coming time t_1 with the semaphore value S . Since $t_1 (= 1) > S (= 0)$, the CPU is not occupied. *agent1* is allocated the CPU, and at the same time, the semaphore is set with $S = t_1 + \Delta t = 1 + 5 = 6$.

Figure 4-3: The semaphore for CPU competition

- *Case 2:* Suppose *agent2* attempts to use the CPU at t_2 and will occupy it for 5ms. It reads first the system clock to get the current time. Suppose the current time is 4, i.e. $t_2 = 4$. The current value of the semaphore is $S = 6$. Since $t_2 < S$, the CPU is occupied. *agent2* has to wait. The waiting time is $t_w = S - t_2 = 6 - 4 = 2(\text{ms})$. *agent2* sets the semaphore with $S = t_2 + \Delta t + t_w = 4 + 5 + 2 = 11$. It will be allocated the CPU in 2ms.

4.1.5 The Outputs of Agents

Once the tunnel is successfully established, i.e. the LNS has received the SCCCN message and has processed it, a series of statistic data about the tunnel will be output to a file by the L2tp_lns agent. The content of the output file is as follows:

Rec	start_t (sec)	hops	rtx	ps_wait (ms)	time (ms)
1	30.027	3	0	9.12	260.75
2	30.016	4	0	5.49	373.54
3	30.034	5	0	8.75	362.21
4	30.264	4	0	11.28	279.72
5	30.122	5	0	19.16	492.78
6	30.224	4	0	3.79	414.20
:	:	:	:	:	:
:	:	:	:	:	:
1994	208.383	12	0	0.00	1141.85
1995	209.146	4	0	0.00	436.39
1996	209.358	4	0	7.73	337.27
1997	209.246	6	0	3.22	665.59
1998	209.417	7	0	0.05	530.98
1999	208.096	10	1	0.00	1940.88
2000	209.424	7	0	0.00	629.49

where:

Rec: record number. For the normal cases, the record number ranges from 1 to 2000. This means that 2000 tunnels will be established in a run.

start_t: start time of the tunnel establishment.

hops: The number of hops of the tunnel. The values in this column show the path length of tunnels.

rtx: the number of times of retransmissions involved in a tunnel establishment. It is the sum of the SCCRQ retransmissions and the SCCCN retransmissions. Only the valid retransmissions are counted.

ps_wait: process waiting time. It is the sum of all process waiting time during the tunnel establishment. See section 4.1.4 for the details.

time: The time used to generate a tunnel. It is the time from the L2tp_lac agent preparing the SCCRQ packet to the L2tp_lns agent completing the SCCCN packet processing.

4.2 Inter-network Topology Modeling

To simulate the behavior of a protocol, the topology of the network needs to be generated. The topology is an abstraction or a model of the actual network structure. The topology of a network is typically modeled using a graph in which nodes represent switches or routers, and edges represent direct connections (links) between switches or routers. The selection and generation of a proper topology is very important in a simulation since different topologies may lead to different results.

A variety of topology models can be found in literature. Zegura *et al.* [ZCB96] surveyed and analyzed the properties of graph models commonly used in studies of network. They also described a new hierarchical mode -- Transit-Stub model (it is also described in [CDZ97]). In this research, the Transit-Stub model is used to perform the simulation.

4.2.1 Transit-Stub Model

In order to explain why the Transit-Stub model is selected in this simulation, let's first take a glance at the simplest model -- random graph model, which is commonly used in studies and simulations at present. In a random model, the nodes in the network are placed at random points in a two dimensional grid. Links are added to the network by considering all possible pairs of nodes and then deciding whether a link should exist according to a probability function involving how far apart the two nodes are and how many links are expected to be in the whole network [CDZ97]. The most serious drawback of this model is that the networks created randomly may not reflect any real network. There is no backbone, no hierarchy, and the paths are usually longer than that in the specially designed networks. At the same time, there is no guarantee that the random model will produce a connected network.

The Transit-Stub model can overcome those drawbacks to a certain degree. This model accurately reflects a wide range of the properties of real inter-networks, including hierarchy and locality. In this model, a set of parameters is employed to construct

network topologies, so that the topology can be specifically designed through selecting proper values of the parameters.

An example of the Transit-Stub topology is illustrated in Figure 4-4.

Figure 4-4: Example of Internet domain structure (from Calvert *et al.* [CDZ97])

The Transit-Stub model is a domain structure. It consists of transit domains and stub domains. Stub domains generally correspond to campus networks or other collections of interconnected LANs, while transit domains are almost always wide- or metropolitan-area networks (WANs or MANs) [CDZ97]. A primary characteristic of domain structure is routing locality, i.e. the path between any two nodes in a domain stays entirely within the domain. Some other characteristics of the Transit-Stub model can be summarized as follows:

- A stub domain carries only the traffic that originates or terminates in the domain, i.e. the nodes in a stub domain are either the original senders or the final receivers.
- The responsibility of the transit domain is to interconnect stub domains efficiently.

- A stub domain can be either a single-homed domain, which connects to only one transit domain, or a multi-homed domain, which connects to more than one transit domains.
- Two stub domains can be connected directly via a stub-stub edge.
- The shortest path between node u in stub domain U and node v in stub domain V goes from U through one or more transit domains to V , and does not pass through any other stub domains.
- For two stub domains that are connected via a stub-stub edge, the path between two nodes on the two domains may (but need not to) go along that edge and avoid any transit domain.

4.2.2 Topology Generation

A tool kit has been developed for the generation of different topologies, in which a tool for generating Transit-Stub topologies is included. The tool kit is provided together with the *ns* all-in-one package. It is named GT-ITM (Georgia Tech Inter-network Topology Models) topology generator. This generator is built on the top of the Stanford GraphBase (SGB). The SGB is a platform of data structures and routines for representing and manipulating graphs.

To generate a topology, we firstly need to build a specification file. The specification file contains a set of parameters. Those parameters are used to specify the model and the topology [UCB]. The syntax of the specification file is:

```
File format: <method keyword> <number of graph> [<initial seed>]
"ts" parameters <# stubs/transit> <# t-s edges> <# s-s edges>
Method parameters <n> <scale> <edge_method> <alpha>
```

Where:

<method keyword>: "ts" for Transit-Stub graph.

<number of graph>: the number of graphs of specified type to generate.

<initial seed>: initial random number seed; optional.

<# stubs/transit>: the average number of stub domains attached per transit node.

<# t-s edges>: the number of extra transit-stub edges.

<# s-s edges>: the number of extra stub-stub edges.

<n>: the number of nodes in graph.

<scale>: one-side dimension of space in which nodes are distributed.

<edge_method>: method for generating edges; 3 for pure random.

<alpha>: random graph parameter ($0.0 \leq \alpha \leq 1.0$). It is the probability to add an edge between a pair of nodes.

As designed in section 3.3, the simulation will be conducted over 4 different network topologies in this research. Two of them contain 108 nodes, from which one has a low node degree [1] (around 2.2), and the other has a high node degree (around 3.6). In these two topologies, each has two Transit domains that consist of 12 nodes. The remaining 96 nodes form several Stub domains. The other two topologies have 54 nodes with different node degrees (2.19 and 3.63 respectively). These two topologies have one Transit domain that consists of 6 nodes. The remaining nodes are Stub nodes.

4.2.3 Converting an SGB File to *ns* Format

Using the GT_ITM tool with topology specification file as the argument, we can create an SGB format file. The SGB file defines the structure of the network topology. This file must be converted into an *ns* format file. Then, we can use it to build the simulation scenarios.

A conversion program is provided in the *ns* tool packet. This program simply converts an SGB file into a plain *ns* format file. In this *ns* format file, the transit nodes are not distinguished from the stub nodes. The bandwidth of all links, regardless of the links between transit nodes or the links between stub nodes, is represented by a unique variable (users have to assign a value to this variable). All links in the network topology

[1] Suppose a graph consists of n nodes and m edges. The average node degree of this graph is defined as $2m/n$.

can only have the same bandwidth. Thus, the topology loses its hierarchy property in terms of bandwidth.

In this research, we need to identify the stub nodes for attaching agents, examining the validity of packets, scheduling events and so on. The transit links and the stub links will be assigned with different bandwidth values. For these purposes, a new conversion program is developed. In this program, the transit nodes can be distinguished from the stub nodes, and the bandwidths are randomly assigned. The range of bandwidths for transit links is from 45 Mbps (which corresponds to the T-3 line.) to 155 Mbps (which corresponds to the OC-3 (Optical Carrier-3 level).). The bandwidth range of stub links is from 1.5 Mbps (which corresponds to T-1 line) to 10 Mbps [PH01].

Four network topologies were created for the simulation experiments. They will be used to build simulation scenarios. Topology N54_1 is illustrated in Figure 4-5, N54_2 illustrated in Figure 4-6, and topology N108_1 and N108_2 in Figure 4-7 and Figure 4-8 respectively.

Figure 4-5: N54_1 (the number of nodes: 54; node degree: 2.19)

Figure 4-6: N54_2 (the number of nodes: 54; node degree: 3.63)

Figure 4-7: N108_1 (the number of nodes: 108; node degree: 2.19)

Figure 4-8: N108_2 (the number of nodes: 108; node degree: 3.60)

4.3 Background Traffic Generation

In order to emulate the situation of a real network, we need to bring traffic into the simulation, rather than only conduct the establishment of L2TP tunnels in a free network. This traffic, together with L2TP packet delivery, shares the routers (nodes), links, and compete for the network resources with each other.

ns provides a traffic generator to users. It allows users to select different modes (EXPOO_Traffic, POO_Traffic and CBR_Traffic) and use different rates to generate and send traffic. In this simulation, the EXPOO_Traffic mode is used. The EXPOO_Traffic mode generates traffic according to an Exponential On/Off distribution. Packets are sent at a fixed rate during “On” periods, and no packets are sent during “Off” periods. Both On and Off periods are taken from an exponential distribution. Packets are constant in size.

An Exponential On/Off object can be created from OTcl. It is embedded in the OTcl class Application/Traffic/Exponential. Its instance variables are:

- `packetSize_`: the constant size of the packets generated.
- `burst_time_`: the average “on” time for the generator.
- `idle_time_`: the average “off” time for the generator.
- `rate_`: the sending rate during “on” time.

By initializing those instance variables with proper values, users can configure the traffic generator and design their specific traffic pattern. The values used for this simulation are:

- `packetSize_`: 532 bytes. It corresponds to the Maximum Transmission Unit (MTU) of the Point-to-Point network.
- `burst_time_`: 500 ms
- `idle_time_`: 500 ms
- `rate_`: 100 Kbps, 500 Kbps and 1000Kbps. These three sending rates are for three types of scenarios respectively. Rate 100 K is for the low traffic scenarios, rate 500 K for the middle and rate 1000 K for the heavy traffic scenarios, respectively.

The EXPOO_Traffic mode starts from the idle state (i.e. the Off state). The length of idle time is a random value with exponential distribution. The average value of this exponential distribution is 500 ms. The random values are created by the instance method `Offtime_.value()`. Where, `Offtime_` is an instance of class `ExponentialRandomVariable`. `value()` is a method of this instance, which is used to create exponential distributed random values.

After an idle period, there is a burst period. In the burst period, the generator creates packets and sends them to the destination. The creation of the burst period is different from the idle period. Instead of using a period of time, the length of the burst period is defined as the number of packets that can be transmitted in this period. Firstly, based on the given average time (500 ms in this research), the average number of packets that can be transmitted in this period of time is computed:

```
interval_ = (double) (size_ << 3) / (double) rate_
burstlen_.setave(ontime_ / interval_)
```

Then, taking this number as the average, an exponentially distributed random value is computed by the instance method `burstlen_.value()`. This random value is the number of packets that will be transmitted during this burst period.

In each burst period, at least one packet is transmitted. If the given average “On” time is too short to transmit a packet, or the random value calculated is less than 0.5 (equal to or greater than 0.5 is rounded to 1), one packet will be sent in this burst period.

After the burst period, there will be another idle period. These processes are repeated until the end of the simulation is reached.

4.4 Simulation Scenarios

After the basic building blocks were developed, they have to be put together to form the simulation system. A simulation scenario, which itself is a basic building block, is responsible for this task. All components used in the simulation are integrated in the scenario so that they can come into play. A set of scenarios is created in this research. They are used in simulations to examine the performances in deferent situations. The specification and implementation of those scenarios is discussed in 4.4.1. In addition, a set of parameters is used to define the behaviors of the system. The determination of the parameters' value is discussed in 4.4.2.

4.4.1 The Specifications and Implementation of Scenarios

There are 12 simulation scenarios that are developed to perform the simulations. The specifications of these scenarios are:

Scenario 1 - 3:
 node: 54
 node degree: 2.19
 transit domain: 1
 transit node: 6
 stub node: 48
 traffic rate: 100 Kbps (for Scenario 1)
 500 Kbps (for Scenario 2)
 1000 Kbps (for Scenario 3)

Scenario 4 - 6:

node: 54
 node degree: 3.63
 transit domain: 1
 transit node: 6
 stub node: 48
 traffic rate: 100 Kbps (for Scenario 4)
 500 Kbps (for Scenario 5)
 1000 Kbps (for Scenario 6)

Scenario 7 - 9:

node: 108
 node degree: 2.19
 transit domain: 2
 transit node: 12
 stub node: 96
 traffic rate: 100 Kbps (for Scenario 7)
 500 Kbps (for Scenario 8)
 1000 Kbps (for Scenario 9)

Scenario 10 - 12:

node: 108
 node degree: 3.60
 transit domain: 2
 transit node: 12
 stub node: 96
 traffic rate: 100Kbps (for Scenario 10)
 500Kbps (for Scenario 11)
 1000Kbps (for Scenario 12)

To perform the simulation of a scenario, a simulator object must be firstly created. This simulator object will handle all simulation activities. Let's take Scenario 9 as an example to describe the process of the simulation implementation. The network topology of Scenario 9 consists of 108 nodes, 96 of them are stub nodes and the other 12 nodes are transit nodes. The topology is created using GT-ITM.

Ninety-six pairs of traffic agents (a pair of traffic agents consists of a source agent and a sink agent) are created. The responsibility of these agents is to create the background traffic. For each pair of agents, we randomly select a pair of stub nodes and attach the agents to them. Since there are 96 stub nodes and 96 pairs of traffic agents, each node, on the average, has the same opportunity to play the role of both the source node and the destination node at the same time. The source agents and the sink agents are connected in pairs. Once the simulation starts, they begin to generate and send traffic packets in an exponential mode.

Ninety-six pairs of L2TP agents are also created. Each agent pair contains an L2tp_lac agent and an L2tp_lns agent. They are used to generate L2TP tunnels. Similar to the traffic agents, we randomly select a pair of stub nodes and attach the L2TP agents to them. Each pair of L2TP agents is scheduled to start at a random time that is calculated from an exponential distribution with average 1 second. In order to reduce the influence of the traffic bursting at the beginning of the simulation, L2TP agents are scheduled in 30 seconds after the simulation starts:

```
set time [$rng_1 exponential]
set start_t [expr $time + 30]
$ns at $start_t "$lac_agent($i) start"
```

When L2TP agents are attached to the selected nodes, the nodes' IDs are passed to the agents. The IDs will be used to examine the validity of the received packets.

```
$lac_agent($i) add_nodeID $s_node $r_node
$lns_agent($i) add_nodeID $r_node $s_node
```

After a pair of L2TP agents completes a tunnel, they are detached from the current nodes, and re-attached to another randomly selected node pair. *ns*, then, reschedules the agents according to an exponential distributed time value. This process will be repeated until a given number of tunnels (for normal cases, 2000 tunnels) is reached.

For each stub node, there is a pair of L2TP agents being attached to (this is the situation on average. Some nodes may have more agents, and some nodes may not have any agent at some moment). These two agents will share the router processor. Thus, the CPU competition mechanism will come into play.

4.4.2 Parameter Determination

Many parameters are involved in the simulation. The values of these parameters need to be determined before the simulation implementation. The determination of some parameters' values (such as the bandwidth and delay of links) has been described in previous sections. The others are discussed in the following.

Node degree of the topology: In order to examine the performance of L2TP tunneling in different connectivity levels of the network, two node degrees are used in

this research: a low degree (2.19) and a high degree (around 3.60). The node degree at 2.19 corresponds to some real networks. See Table 4-1.

Network	Nodes	Ave. Degree
ARPAnet	47	2.9
CERFnet	63	2.1
Sesquinet	94	2.1
PGF	137	2.2
Ga. Tech	30	3.9

Table 4-1: properties of real inter-network topologies (from Zegure *et al.* [ZCB96])

Node degree 3.60 is selected since we also want to conduct the simulation in a network with high connectivity. From Figure 4-6 and Figure 4-8 it can be seen that the network topologies at this degree possess very high connectivity (the stubs are already close to a fully connected mesh).

Protocol execution time (UDP & L2TP): When a user application comes to an LAC, the LAC will create an L2TP control message, then encapsulate it with an UDP header. This process takes some time for executing the protocols. The protocol execution time is difficult to estimate since it is hard to find literature about this, and at the same time, it is different from product to product. Therefore, we can only estimate a reasonable range. Fortunately, the protocol execution time is very short. It does not badly influence the examination of the system performance.

In order to measure the L2TP execution time, we can create a pair of connection – oriented stream sockets (a server socket and a client socket) to test the performance of TCP in a Linux environment. Using a very small message (e.g. one byte), we can reduce the transmission time to almost zero. Then, we send the message to the same host (i.e. using one computer as both the client and the server at the same time), and compute the RTT (Round-Trip-Time) of TCP. Since the sender and the receiver of the message is the same host, the latency (i.e. the delivery time) and the routing time can be ignored. Thus, the RTT is mainly formed by the TCP execution time.

Through 100 times test, the average RTT value has been measured. The value of 0.393 (ms) was obtained on a Pentium II computer under Linux. Therefore, the TCP execution time for the sender is $0.393 / 2 = 0.197$ (ms). The L2TP protocol is similar to TCP in features. We can take this time as the L2TP execution time. Since in our simulation the topology generator produces link delay with an accuracy of 1 ms, we decided to set the overall parameters accuracy to 0.5 ms. Thus, we take 0.5 ms as the L2TP protocol execution time.

Similar to the measurement of TCP performance, the UDP execution time is set to 0.5 ms as well.

Setting challenge: To set a challenge, the LAC does a lookup for tunnel configuration to the peer to make sure there is a shared secret. If the shared secret exists, it is set to the SCCRQ message through a challenge AVP. According to Ahmed [Ahm99], this type of operation takes no more than 0.5 ms. Therefore, the challenge setting time is set to 0.5 ms.

Response calculation: The response value is a one-way hash calculated over a stream of octets consisting of the identifier, followed by the "secret", followed by the challenge value (RFC 1994 [Sim96]). The length of the response value is 16 octets (128 bits). According to the measurement from Shamus Software Ltd. [SSoftware], the calculation of a 128 bits hash value takes about 2.5 ms. Therefore, 2.5 ms is taken as the response calculation time in this research.

Response examination: The operation of response examination is similar to the challenge setting operation. Therefore, 0.5 ms is taken as the response examination time.

Up to now, the four basic building blocks of the simulation have been modeled and developed. Integrating those building blocks into a whole, we can build up the simulation system. The different scenarios of the system are configured through assigning proper values to the parameters. After that, we can implement the simulation. In next chapter, the simulation implementation and the output analysis will be discussed.

CHAPTER 5: IMPLEMENTATION AND OUTPUT ANALYSIS

In Chapter 4, the simulation system has been modeled. After the model is translated into computer programs (in C++, Tcl and OTcl), we can start to perform the simulation experiments. In this chapter, the simulation implementation, the output data analysis and the performance analysis will be described. There are 12 simulation scenarios in total. For these scenarios, the procedures of simulation and data analysis are similar. In order to avoid duplicate description, one of those scenarios is selected as a case study. The selected case (Scenario 9) will be described in detail in section 5.1. The remaining scenarios will be discussed in section 5.2, and only the final results of analysis are described for them. The performance analysis of the system will be given in section 5.3.

5.1 Case Study: the Implementation of Scenario 9

Scenario 9 is selected as a case study because it is typical for the whole set of scenarios. In Scenario 9, the network topology consists of 108 nodes, with the node degree of 2.19. The traffic-sending rate is 1000 Kbps (see section 4.4 for the topology specification). The design of the simulation implementation was given in section 3.3.4.

In this scenario, the run (replication) length of the simulation is 2000 tunnels. A run is divided into 20 batches, each batch containing 100 tunnels. The purpose for dividing a run into batches is to remove the initialization bias and to estimate the steady state of the system (it can also be used for other purpose, but that will not be discussed here). The number of implemented runs for a simulation scenario depends on the accuracy of the output data. The procedures of the implementation and the output data analysis include the following steps:

Step 1: Running the Tcl program of Scenario 9 to perform the simulation:

```
ns scenario_9.tcl
```

While the simulation is performed, the results are written into two files: `data` and `data_fail`. The items in file `data` include: the start time of a tunnel, the number of hops of a tunnel path, the number of times of retransmissions, process waiting time, and the tunneling time. The tunneling time is the major measure for the performance of a scenario. The others will be used to portray the condition of the network and investigate the performance of the system. In the next steps, the output data analysis will be done based on the tunneling time.

The format of the output file `data` and the explanation of each item in `data` were given in section 4.1.5. The content of file `data_fail` is the information about the tunnels that fail in establishment because the maximum number of retransmission has been reached. The failure information can be used to evaluate how busy the network is. It will help us choose proper traffic-sending rates in scenario design.

In order to estimate the sample size required for getting the desired accuracy, an initial sample needs to be observed. The recommended size (the number of runs) of the initial sample is at least 4 or 5, whereas 10 or more are recommended [BCN96], depending on the accuracy of the sample data and the final size required. In this scenario, ten statistically independent runs are taken as the initial sample (i.e. $R_0 = 10$). They can be obtained by running the Tcl program `scenario_9.tcl` 10 times.

Step 2: Calculating batch means of the tunneling time.

In each run of the simulation, the mean value of the tunneling time of each batch is calculated. The batch mean is a simple sample mean. It is defined as:

$$\bar{Y}_{rj} = \frac{1}{m} \sum_{i=1}^m Y_{rji} \quad (5-1)$$

Where:

r represents run.

j represents batch.

i is i^{th} tunnel in a batch.

Y_{rji} is the observation (i.e. the generation time of tunnel i) in batch j , run r .

m is the batch size, $m = 100$.

$\bar{Y}_{rj}$ is the batch mean (sample mean).

Since the run length is 2000 tunnels, there are 20 batches in a run. The following numbers are the batch means in run 1:

738.51	2544.18
1671.26	1734.47
2353.17	2197.54
4845.46	2080.23
2936.75	2606.19
1801.73	3318.73
2104.93	3582.52
1827.92	2660.84
1628.52	2144.57
3292.77	2095.14

Step 3: Reducing the initialization bias.

The initialization bias is introduced by the initial condition of the simulation. When the simulation begins, all L2TP agents are scheduled to start at different time points from idle state. The first scheduled agents will establish tunnels with less resources (CPU and network) competition. Thus, the generation time of those tunnels will be shorter than that in the steady state of the system. This causes the biases from the true values.

The other influence comes from the packet retransmission. In the output file (data), the first generated tunnels are first recorded. Since the tunnels in which there are some messages being retransmitted will take a longer time to establish than those without retransmission involved, these tunnels will come and be recorded later. Therefore, a number of tunnels that are recorded near the beginning in the output file will have shorter generation time. This may be the most significant factor to cause the initialization bias.

In order to remove the initialization bias, the ensemble averages need to be calculated and examined. The ensemble average is the average of the corresponding batch means across runs. For each batch j , the ensemble average across all R_0 runs is defined to be:

$$\bar{Y}_{.j} = \frac{1}{R_0} \sum_{r=1}^{R_0} Y_{rj} \quad (5-2)$$

Where $R_0 = 10; j = 1, 2 \dots 20$.

The ensemble averages are displayed in the third column of Table 5-1. From this column, it can be seen that the ensemble averages of first two batches (729.79 and 1601.76) are less than all other ensemble averages. This is because the influence of the initial condition.

Run Length T	batch j	Average batch mean $\bar{Y}_j$	Cumulative Average $\bar{Y}_{..(j,0)}$ (no delete)	Cumulative Average $\bar{Y}_{..(j,1)}$ (delete 1)	Cumulative Average $\bar{Y}_{..(j,2)}$ (delete 2)	Cumulative Average $\bar{Y}_{..(j,3)}$ (delete 3)
100	1	729.79	729.79	—	—	—
200	2	1601.76	1165.78	1601.76	—	—
300	3	3049.82	1793.79	2325.79	3049.82	—
400	4	3078.71	2115.02	2576.76	3064.27	3078.71
500	5	2543.76	2200.77	2568.51	2890.76	2811.24
600	6	2576.71	2263.43	2570.15	2812.25	2733.06
700	7	2707.13	2326.81	2592.98	2791.23	2726.58
800	8	2609.96	2362.21	2595.41	2761.02	2703.26
900	9	2710.02	2400.85	2609.74	2753.73	2704.38
1000	10	3151.62	2475.93	2669.94	2803.47	2768.27
1100	11	2579.86	2485.38	2660.94	2778.62	2744.72
1200	12	2558.19	2491.44	2651.59	2756.58	2724.00
1300	13	2986.14	2529.50	2679.47	2777.45	2750.21
1400	14	2797.33	2548.63	2688.54	2779.10	2754.49
1500	15	2785.77	2564.44	2695.48	2779.62	2757.10
1600	16	2964.15	2589.42	2713.40	2792.80	2773.03
1700	17	3130.93	2621.27	2739.49	2815.34	2798.59
1800	18	2468.03	2612.76	2723.52	2793.63	2776.55
1900	19	3076.68	2637.18	2743.14	2810.28	2795.31
2000	20	2643.07	2637.47	2737.88	2800.99	2786.36

Table 5-1: Ensemble batch means and cumulative means, averaged over 10 runs

The fourth column in Table 5-1 is the cumulative average sample mean without deleting any batch. The cumulative average at a row is the average of all ensemble batch means before it. Column five, six and seven give the cumulative average sample means after deleting one, two and three batch means from the beginning respectively. They are defined to be:

$$\bar{Y}_{..(n,d)} = \frac{1}{n-d} \sum_{j=d+1}^n \bar{Y}_j \quad (5-3)$$

Where:

$\bar{Y}_j$: ensemble average batch mean.

d : the number of deleted observations.

n : representing observations, $n = d + 1, \dots, 20$.

From Table 5-1, it can be seen that a downward bias is present. After the first two observations are deleted, this initialization bias can be removed. Therefore, we delete the first two batches from the output data, and use the remained 18 batches for the performance analysis (i.e. $d = 2$) in this scenario.

Step 4: Calculating the sample mean $\bar{Y}$ and the sample variance S_0 of ten runs.

Firstly, the sample mean of each run with the remaining 18 batches is calculated using formular 5-4.

$$\bar{Y}_r = \frac{1}{(b-d)} \sum_{j=d+1}^b \bar{Y}_j \quad (5-4)$$

Where

$\bar{Y}_r$: the sample mean of r^{th} run.

b : the number of batch in total in a run. In this scenario, $b = 20$.

d : the number of deleted batches in a run. $d = 2$ for this scenario.

$\bar{Y}_j$: the batch mean of j^{th} batch in r^{th} run (see formula 5-1).

The sample mean of each run is:

$\bar{Y}_1$:	2541.98
$\bar{Y}_2$:	3153.44
$\bar{Y}_3$:	2549.60
$\bar{Y}_4$:	3118.44
$\bar{Y}_5$:	2897.34
$\bar{Y}_6$:	2954.58
$\bar{Y}_7$:	2727.73
$\bar{Y}_8$:	2945.01
$\bar{Y}_9$:	2437.35
$\bar{Y}_{10}$:	2684.47

Secondly, calculate the general sample mean of R_0 runs ($R_0 = 10$). The sample mean of 10 runs is given by:

$$\bar{Y} = \frac{1}{R_0} \sum_{r=1}^{R_0} \bar{Y}_r \quad (5-5)$$

The value of the general sample mean of 10 runs is: $\bar{Y} = 2800.99$.

Thirdly, calculate the sample variance of R_0 runs. The sample variance is defined to be:

$$S_0^2 = \sum_{r=1}^{R_0} \frac{(\bar{Y}_r - \bar{Y})^2}{R_0 - 1} \quad (5-6)$$

The sample variance is $S_0^2 = 62229.79$.

Step 5: Estimating the sample size R .

Given a half-length for a confidence interval, we can estimate the sample size required for meeting this criterion. In Scenario 9, a confidence interval 95% for a half-length error, 5% of the average tunneling time, is set for the simulation implementation. This means that it is desired to estimate the tunneling time to within $\pm 5\%$ error with probability 0.95. Based on this criterion, we first roughly estimate, and then, accurately compute the sample size through analyzing the initial sample. The rough estimate for R is given by

$$R \geq \frac{(z_{\alpha/2}, S_0)^2}{\varepsilon^2} \quad (5-7)$$

Where

$\phi(z_\alpha)$: cumulative normal distribution of confidence intervals. The value of z_α can be found in Table A.3 in Banks *et al.* [BCN96].

$1-\alpha$: confidence interval, $1-\alpha = 0.95$, $\alpha = 0.05$.

ε : the half-length error, $\varepsilon = 0.05 * \bar{Y} = 0.05 * 2800.99 = 140.05$.

Bring the values of above parameters into inequality (5-7). The estimate of the sample size can be obtained:

$$(Z_{0.025} * S_0 / \epsilon)^2 = (1.96)^2(62229.79)/(140.05)^2 = 12.19$$

Therefore, the final sample size must be at least as large as 13.

Step 6: Determining the sample size R .

The sample size calculated in Step 5 is just a estimate. It gives a lower limit of the possible value of the sample size. We can take this value as the start point to compute the true value of the required sample size. Inequality (5-8) gives the smallest sample size that satisfies the given criterion.

$$R \geq \frac{(t_{\alpha/2, R-1} S_0)^2}{\epsilon^2} \quad (5-8)$$

Where

$t_{\alpha/2, R-1}$ is the percentage point of the Student t distribution with $R-1$ degree of freedom. The value of t can be found in Table A.5 in Banks *et al.* [BCN96].

The results calculated from inequality (5-8) are shown in Table 5-2.

R	13	14	15
$(t_{0.025, R-1} * S_0 / \epsilon)^2$	15.08	14.80	14.53

Table 5-2: The smallest sample size required for scenario 9 ($R_0 = 10$)

From Table 5-2, it can be seen that $R = 15$ is the smallest integer satisfying inequality (5-8). Therefore, $R - R_0 = 15 - 10 = 5$ additional runs are needed.

Step 7: More runs.

Repeat Step 1 to perform simulation 5 more times, so that the number of runs reaches 15 in total. The sample size determined in Step 6 is based on the first 10 runs' output data. This means that if the accuracy of the output data from additional runs is as good as the data from the first 10 runs, the sample size should be at least 15. Since there is no guarantee that the data from additional runs are as good as the previous observations, we cannot ensure whether or not this sample size is enough to satisfy the given criterion.

Therefore, after performing R ($= 15$) run simulations, the results should be examined again. Repeat Step 2, 4 and 6 to examine the sample size. The results are shown in Table 5-3.

R	14	15	16
$(t_{0.025, R-1} * S_0 / \epsilon)^2$	13.38	13.13	13.01

Table 5-3: The examination of sample size based on 15 runs

From Table 5-3, it can be seen that 15 runs of simulation can satisfy the requirement of the accuracy criterion.

Step 8: Computing the means of all 15 runs.

The mean values of all items in the output file for Scenario 9 are calculated (with the first two batches being removed in each run) based on the 15 runs. The results are shown in Table 5-4.

Run	Hops	rtx	ps_wait (ms)	Time (ms)
1	6.81	0.78	2.12	2541.98
2	6.86	0.96	1.63	3153.44
3	6.84	0.77	1.81	2549.60
4	6.72	0.96	1.72	3118.44
5	6.95	0.86	1.73	2897.34
6	6.94	0.84	1.52	2954.58
7	6.89	0.85	1.87	2727.73
8	7.02	0.89	1.88	2945.01
9	7.01	0.74	2.29	2437.35
10	6.76	0.78	1.84	2684.47
11	6.82	0.86	1.65	2886.19
12	6.72	0.88	1.50	2793.40
13	6.84	0.72	2.00	2382.15
14	6.98	0.79	2.02	2602.35
15	6.89	0.83	1.77	2853.78
Mean	6.87	0.83	1.82	2768.52

Table 5-4: Mean values with first two batches being removed in Scenario 9

In order to indicate how busy the network is under this scenario, the number of tunnels that are failed in establishment is counted. The average number of failed tunnels for each run is 379.8. It means that $379.8 / (379.8 + 2000) = 16\%$ of all tunnels are failed in their generation. The reason for the failure is that some message has been retransmitted 5 times during tunneling, but no response from the peer was received. This percentage shows that the network is quite busy. That is why 1000 Kbps is taken as the high level rate of traffic sending.

5.2 Output Data Analysis of Other Scenarios

The procedures for analyzing the output data of other scenarios are the same as the procedures described in section 5.1. In this section, therefore, only the results of the analysis for the other 11 scenarios are presented. The detail process will be omitted. The parameters and the results of Scenario 1 to 12 (except Scenario 9) are displayed and briefly explained in the following.

Scenario 1: The parameters and results for Scenario 1 are shown in Table 5-5. Five runs of simulation are performed for meeting the 98% confidence interval for 5% half-length mean tunneling time. In order to remove the initialization bias, the first one batch in each run is deleted. The mean tunneling time of this scenario is 368.52 ms.

Items	Value
Nodes	54
Node degree	2.19
Traffic sending rate (Kbps)	100
Run length (tunnels)	2000
Batch size (tunnels)	100
Batches / run	20
Average failure (tunnels) / run	0
Confidence interval	98%
Half-length error (h.l.)	5%
Runs of simulation (R)	5
Deleted batches (d) / run	1
Sample variance (S^2)	26.26
Sample mean of the scenario (ms)	368.52

Table 5-5: The values of simulation items for Scenario 1

Scenario 2: The parameters and results of Scenario 2 are displayed in Table 5-6.

Items	Value
Nodes	54
Node degree	2.19
Traffic sending rate (Kbps)	500
Run length (tunnels)	3000
Batch size (tunnels)	100
Batches / run	30
Average failure (tunnels) / run	14.3
Confidence interval	95%
Half-length error (h.l.)	7.5%
Runs of simulation (R)	26
Deleted batches (d) / run	3
Sample variance (S^2)	23414.60
Sample mean of the scenario (ms)	845.22

Table 5-6: The values of simulation items for Scenario 2

In this scenario, three batches in each run are deleted for removing the initialization bias. Therefore, a longer run (with 3000 tunnels) is taken in the simulation. With the increase of the traffic-sending rate, the tunneling time becomes scattered. In this case, more observations are needed to satisfy the pre-determined criterion. According to the calculation, 26 runs are required in the simulation of this scenario.

Scenario 3: The parameters and results are shown in Table 5-7.

Items	Value
Nodes	54
Node degree	2.19
Traffic sending rate (Kbps)	1000
Run length (tunnels)	2000
Batch size (tunnels)	100
Batches / run	20
Average failure (tunnels) / run	275
Confidence interval	95%
Half-length error (h.l.)	6.5%
Runs of simulation (R)	24
Deleted batches (d) / run	1
Sample variance (S^2)	103320.78
Sample mean of the scenario (ms)	2270.25

Table 5-7: The values of simulation items for Scenario 3

Scenario 4: The parameters and results of Scenario 4 are shown in Table 5-8. Due to the low traffic-sending rate (100 Kbps) in this scenario, the output converges very fast. Therefore, only few runs are required.

Items	Value
Nodes	54
Node degree	3.63
Traffic sending rate (Kbps)	100
Run length (tunnels)	2000
Batch size (tunnels)	100
Batches / run	20
Average failure (tunnels) / run	0
Confidence interval	98%
Half-length error (h.l.)	5%
Runs of simulation (R)	5
Deleted batches (d) / run	1
Sample variance (S^2)	6.53
Sample mean of the scenario (ms)	258.85

Table 5-8: The values of simulation items for Scenario 4

Scenario 5: The results are shown in Table 5-9. Like the situation in Scenario 2, the sample variance is very large in this scenario. It looks that when a middle level traffic is brought into the network, there will be more differences among the tunneling time than

Items	Value
Nodes	54
Node degree	3.63
Traffic sending rate (Kbps)	500
Run length (tunnels)	3000
Batch size (tunnels)	100
Batches / run	30
Average failure (tunnels) / run	1.3
Confidence interval	95%
Half-length error (h.l.)	6.5%
Runs of simulation (R)	31
Deleted batches (d) / run	2
Sample variance (S^2)	5169.25
Sample mean of the scenario (ms)	405.78

Table 5-9: The values of simulation items for Scenario 5

that in a low level or in a high level traffic. According to Kelton [Kel86], there is no much value in performing more runs after more than 25 runs. Therefore, a longer run length (3000 tunnels) is used in the simulation. The number of runs is 31.

Scenario 6: The results of analysis for scenario 6 are shown in Table 5-10.

Items	Value
Nodes	54
Node degree	3.63
Traffic sending rate (Kbps)	1000
Run length (tunnels)	2000
Batch size (tunnels)	100
Batches / run	20
Average failure (tunnels) / run	139.8
Confidence interval	95%
Half-length error (h.l.)	6.5%
Runs of simulation (R)	30
Deleted batches (d) / run	2
Sample variance (S^2)	77784.92
Sample mean of the scenario (ms)	1640.87

Table 5-10: The values of simulation items for Scenario 6

Scenario 7: The parameters and results of scenario 7 are shown in Table 5-11.

Items	Value
Nodes	108
Node degree	2.19
Traffic sending rate (Kbps)	100
Run length (tunnels)	2000
Batch size (tunnels)	100
Batches / run	20
Average failure (tunnels) / run	0
Confidence interval	98%
Half-length error (h.l.)	5%
Runs of simulation (R)	5
Deleted batches (d) / run	1
Sample variance (S^2)	24.73
Sample mean of the scenario (ms)	584.69

Table 5-11: The values of simulation items for Scenario 7

Scenario 8: In this scenario, the traffic rate is in the middle level (500 Kbps). Like in Scenario 2 and Scenario 5 (they have the same traffic rate), a big number of runs are needed for obtaining a satisfied result. The results of output analysis for Scenario 8 are shown in Table 5-12. Twenty-five runs of simulation are performed.

Items	Value
Nodes	108
Node degree	2.19
Traffic sending rate (Kbps)	500
Run length (tunnels)	3000
Batch size (tunnels)	100
Batches / run	30
Average failure (tunnels) / run	42.44
Confidence interval	95%
Half-length error (h.l.)	6.5%
Runs of simulation (R)	25
Deleted batches (d) / run	3
Sample variance (S^2)	38477.84
Sample mean of the scenario (ms)	1265.49

Table 5-12: The values of simulation items for Scenario 8

Scenario 10: Table 5-13 displays the parameters and results of Scenario 10.

Items	Value
Nodes	108
Node degree	3.60
Traffic sending rate (Kbps)	100
Run length (tunnels)	2000
Batch size (tunnels)	100
Batches / run	20
Average failure (tunnels) / run	0
Confidence interval	98%
Half-length error (h.l.)	5%
Runs of simulation (R)	5
Deleted batches (d) / run	1
Sample variance (S^2)	22.15
Sample mean of the scenario (ms)	413.59

Table 5-13: The values of simulation items for Scenario 10

Scenario 11: The traffic rate in Scenario 11 is 500 Kbps. The sample variance of the outputs is not large. This is because when the size of the network and the node degree get larger, it can accommodate more traffic. From the item "Average failure / run", it can be seen that there is no failure in this scenario. This indicates that the rate 500 Kbps is no longer a middle level traffic for this network, but a pretty low one. Therefore, the outputs are not very scattered. The results of output analysis are shown in Table 5-14.

Items	Value
Nodes	108
Node degree	3.60
Traffic sending rate (Kbps)	500
Run length (tunnels)	2000
Batch size (tunnels)	100
Batches / run	20
Average failure (tunnels) / run	0
Confidence interval	95%
Half-length error (h.l.)	5%
Runs of simulation (R)	13
Deleted batches (d) / run	1
Sample variance (S^2)	1281.01
Sample mean of the scenario (ms)	455.54

Table 5-14: The values of simulation items for Scenario 11

Scenario 12: The results of Scenario 12 (in Table 5-15) can verify the discussion

Items	Value
Nodes	108
Node degree	3.60
Traffic sending rate (Kbps)	1000
Run length (tunnels)	6000
Batch size (tunnels)	200
Batches / run	30
Average failure (tunnels) / run	135.3
Confidence interval	95%
Half-length error (h.l.)	6.5%
Runs of simulation (R)	18
Deleted batches (d) / run	3
Sample variance (S^2)	19331.03
Sample mean of the scenario (ms)	1118.07

Table 5-15: The values of simulation items for Scenario 12

given in Scenario 11. With the size and the node degree of the network getting larger, the criterion of traffic rate shifts up. 1000 Kbps is no longer an high traffic rate for Scenario 12, but acts as a middle level rate. The tunneling time becomes very scattered as that occurred in other scenarios with the middle traffic rate (Scenario 2, 5 and 8). Therefore, a longer run (6000 tunnels) and a larger batch size (200 tunnels) are employed in this scenario 12.

5.3 System Performance Analysis

The measurement of the system performance in this research is tunneling time. The mean values of tunneling time in 12 scenarios have been calculated in the preceding sections. Those results present the performance of networks of different sizes and in different conditions. In order to analyze the performance of the system, variables that may have influence on or help to study the system performance need to be investigated. Several variables have been measured in the simulation. Those include the average

Scenario	Hops (/tunnel)	Rtx times (/tunnel)	CPU waiting time (ms) (/tunnel)	Tunneling time (ms) (/tunnel)	Failure (%) (/run)
Scenario 1	5.48	0.0	6.19	368.52	0.0
Scenario 2	5.47	0.21	3.55	845.22	0.5
Scenario 3	5.40	0.73	1.07	2270.25	12.1
Scenario 4	4.24	0.0	7.35	258.85	0.0
Scenario 5	4.25	0.08	5.94	405.78	1.0
Scenario 6	4.24	0.54	1.54	1640.87	6.5
Scenario 7	7.00	0.0	17.91	584.69	0.0
Scenario 8	7.00	0.29	6.78	1265.49	1.4
Scenario 9	6.87	0.83	1.82	2768.52	16.0
Scenario 10	5.31	0.0	30.59	413.59	0.0
Scenario 11	5.27	0.03	27.46	455.54	0.0
Scenario 12	5.28	0.31	6.94	1118.07	2.2

Table 5-16: The simulation results of 12 scenarios

number of hops, the average number of retransmissions, the average CPU waiting time in a tunnel generation, and the failure percentage of tunnel generation in a run. The general results of all those variables are displayed in Table 5-16.

Based on the data displayed in Table 5-16, we can study the system performance in different situations, and investigate the relationship between the system performance and related variables. In following paragraphs, the system performance will be discussed from different aspects through comparing system variables in different ways.

Discussion 1: The relationship between the tunnel length and the average node degree. Firstly, let's observe the average tunnel length in different topologies. Figure 5-1 shows the average number of hops of a tunnel in different network topologies.

Figure 5-1: The average length of tunnels in different networks

From Figure 5-1, it can be seen that for networks with the same number of nodes, the average tunnel length is smaller in the network at a higher node degree. Since a high node degree means more connections among nodes, it is more likely in a high node degree network to find a shorter path between a pair of nodes than that in a low node degree network.

Discussion 2: Tunneling time with different tunnel lengths. Generally speaking, if a tunnel consists of more hops, the tunnel will have a long path. The tunnel with a long

path needs longer time to be established since packets have to travel through a long way to go forth and back from the LAC to the LNS to pass messages. The relationship between the number of hops and the tunneling time in Scenario 9 is illustrated in Figure 5-2.

Figure 5-2: The average tunneling time with different tunnel lengths (hops) in Scenario 9

Just like expected, the line illustrated in Figure 5-2 shows that the tunneling time varies almost linearly according to the number of hops. With the number of hops of a tunnel increasing, the generation time of the tunnel steadily goes up.

Discussion 3: Tunneling time with different number of retransmissions. Again, the Scenario 9 is taken as an example in this discussion. The rise in tunneling time caused by the increase of retransmissions is illustrated in Figure 5-3. The figure shows that with the number of retransmissions increasing, the tunneling time goes up rapidly.

Comparing Figure 5-3 with Figure 5-2, we can find that the values of tunneling time (situated on Y axis) in Figure 5-3 are much greater than that in Figure 5-2. This means that the influence on the tunneling time made by retransmissions is much significant than that made by the number of hops of the tunnel. It proves that the retransmission is a very important factor that can influence the system performance.

Figure 5-3: The average tunneling time with different number of retransmissions in Scenario 9

As discussed in section 4.1.3, an exponential back-off interval is employed in the retransmission of L2TP packet, i.e. the retransmission intervals are 1, 2, 4, 8 and 8 seconds respectively. With the number of times increasing, the value of interval gets larger. This property leads to the tunneling time growing rapidly, especially for the 4th and 5th retransmissions since their intervals are both 8 seconds. This will produce a big jump in tunneling time at these two points. After 5 times of retransmissions, there must be at least one retransmission of a new message being involved (because the maximum number of retransmission for a message is 5). The interval for re-transmitting this new message starts from 1 second. Therefore, the rate of increase of the tunneling time at 6th retransmission should slow down. In Figure 5-3, this trend can be clearly observed: the tunneling time at 4th and 5th retransmission rises sharply, and then, the curve gets flat at 6th retransmission.

Discussion 4: Tunneling time in the network with 54 nodes. In order to investigate the influence upon the performance made by the connectivity level of the network, let's compare the tunneling time in two networks that consist of the same number of nodes, but with different node degrees. Figure 5-4 illustrates the tunneling time in two networks that consist of 54 nodes, with node degree 2.19 and node degree

3.63 respectively. The lines in the figure show the variance of the tunneling time with different traffic rates. The six points on two lines correspond to the tunneling time of Scenario 1 through Scenario 6.

Figure 5-4: Tunneling time under different traffic rates in the network topology with 54 nodes

In Figure 5-4 it can be seen that with the traffic rate getting larger, the time used to generate a tunnel goes up. When the traffic is heavy (1000 Kbps), the tunneling time goes up sharply. This is because when the network gets busy, more packets are put into the link queues and wait there. This results in a longer tunneling time and more retransmissions.

Under the same traffic rate, the tunneling time in the network with node degree 3.63 is shorter than that in the network with degree 2.19. One reason is discussed in Discussion 1. The average length of a tunnel in the network with node degree 3.63 (4.24 hops) is less than that in the network with node degree 2.19 (5.45 hops). The other reason is that when the traffic is sent in the same rate, the link utilization in the network with high node degree will be lower than that in the low degree network. Therefore, the network with higher node degree can accommodate more traffic.

Discussion 5: Tunneling time in the network with 108 nodes. The tunneling time in the networks that consist of 108 nodes, with node degree 2.19 and node degree

3.60 respectively, is illustrated in Figure 5-5. The six points on two lines correspond to the tunneling time in Scenario 7 through Scenario 12.

Figure 5-5: Tunneling time under different traffic rates in network topologies with 108 nodes

For Figure 5-5, we can derive the similar conclusion to the previous discussion. In these two networks, it is more apparent that with the traffic rate increasing, the tunneling time in the network with high node degree grows slowly.

Discussion 6: The tunneling time in the network with node degree 2.19. The analysis given in the preceding two discussions are about the influence upon performance made by the average node degree. Now, let's see the question from another side: how the size of the network affects the performance. Though two networks with different sizes may not be fully comparable due to their different structures, we can still investigate the general behavior of the system. To do so, we can reorganize the data and compare the average tunneling time in two networks: one consists of 54 nodes, and the other consists of 108 nodes. Both of them have the same node degree 2.19. Figure 5-6 illustrates this comparison.

From Figure 5-6 we can see that for each traffic-sending rate, the average tunneling time in the network with 108 nodes (abbreviated to network₁₀₈) is longer than that in

the network with 54 nodes (abbreviated to network_54). Two lines in the figure are almost parallel to each other. The reason for this result is straightforward: the average length of a tunnel in a large network is greater than that in a small network. From Table 5-16 it can be seen that the average length of a tunnel in network_108 is 6.96 hops (Scenario 7 through Scenario 9). In network_54, the average length is 5.45 hops (Scenario 1 through Scenario 3). The difference in tunnel length between two networks is 1.51 hops.

Figure 5-6: Tunneling time under different traffic rates in the network topology with node degree 2.19

Another reason is the number of retransmissions (abbreviated to rtxs). The average rtxs per tunnel in Scenario 2 and 3 (corresponding to network_54) are 0.21 and 0.73 respectively. The rtxs per tunnel are 0.29 and 0.83 in Scenario 8 and Scenario 9 (corresponding to network_108) respectively. This means that when there is a middle or a high rate traffic in the network, the retransmissions involved in a tunnel generation in network_108 is greater than that in network_54.

The influence from the CPU waiting time is less important.

Discussion 7: The tunneling time in the network with node degree 3.6. When the node degree is 3.6, two lines are no longer parallel to each other (displayed in Figure

5-7). When there is a heavy traffic in the network (1000 Kbps), the tunneling time in network_108 is unexpectedly less than that in network_54. From Table 5-16, it can be found that the average tunnel length in network_108 is 5.29 hops (Scenario 10 through Scenario 12). In network_54, the average length is 4.24 hops (Scenario 4 through Scenario 6). The difference of tunnel length between two networks is 1.04 hops. Though this difference is less than that in the networks with node degree 2.19 (where it is 1.51, see Discussion 4), the average length of a tunnel in network_108 is still greater than that in network_54. Therefore, the influence of tunnel length is not the answer for this question.

Observing the rtxs listed in Table 5-16, we can see that the average rtxs per tunnel in Scenario 6 is 0.54, while in Scenario 12, it is only 0.31. There are much more retransmissions involved in network_54 than in network_108. Obviously, the number of retransmissions, in this case, becomes the most significant factor. Its influence on the system performance is over the length of the tunnel. Figure 5-7 indicates that the larger network can accommodate more traffic. This implies that the relation between the average node degree and the capacity of the network (see Discussion 4 and 5) is not linear. The size of the network influences the capacity as well.

Figure 5-7: Tunneling time under different traffic rates in the network topology with node degree 3.6

In this chapter is described first the collection of simulation results in order to insure adequate accuracy. Based on the output data, the system performance is investigated. Several variables that are closely related to the system performance are collected from the simulation. Their influence on the performance is investigated from different aspects. The investigation shows that retransmission is a very significant factor for the system performance. Based on this conclusion, an improvement on the protocol will be made in the next chapter with the purpose of performance improvement.

CHAPTER 6: PERFORMANCE IMPROVEMENT OF L2TP

The simulations of 12 scenarios were implemented in Chapter 5. Through the performance analysis it can be seen that the system performance is influenced by several factors. Some factors come from system intrinsic features, and some are determined by the protocol. In this chapter, the effort will be made on the improvement of the performance through developing a new algorithm for packet retransmission to replace the algorithm used in L2TP protocol.

6.1 Problem Formulation

The discussion given in section 5.3 shows the factors that influence the system performance. These include the number of retransmissions, the traffic rate, the size and the node degree of the network, the length of the tunnel, and the CPU waiting time. Between these factors, the retransmission mechanism is a protocol feature, and the others are the system intrinsic properties. The system intrinsic properties are hardware related and network specific. The protocol features are determined by the designer. They are universal to all networks. According to the discussion made in Chapter 5, the packet retransmission is a very significant factor for the system performance. Therefore, the effort for the improvement of the system performance will be made on the improvement of the L2TP retransmission mechanism.

The algorithm used for the retransmission in L2TP protocol employs an exponential back-off interval (see section 4.1.3 for the details). Five retransmission intervals are 1 second, 2 seconds, 4 seconds, 8 seconds and 8 seconds respectively. The performance analysis indicates this is not a good scheme because the same intervals are used for all tunnels, regardless whether they are long or short.

Let's continue to take Scenario 9 as an example. For the tunnels in which there is no retransmission involved in the tunnel generation, their average generation time is 589 ms. The tunnel generation is a process of three-message exchange. The average RTT

(Round-Trip-Time) can be calculated as $(589 / 3) * 2 = 393 \text{ (ms)} = 0.39 \text{ second}$. Compared to this value, the retransmission intervals employed in L2TP protocol appear too long. Therefore, the retransmission will have a serious influence on the tunneling time. The influence was clearly displayed in Figure 5-3.

A trade-off in the estimation of the timeout period is the followings: if the timeout period is too short, there will be too many retransmissions. Otherwise, if the period is too long, it will lead to inefficient use of network bandwidth and result in a very long tunneling time. The standard for "short" or "long" totally depends on the length of the tunnel. Therefore, it is impossible for a fixed interval to fit all tunnels.

An adaptive algorithm is used in TCP's timeout and retransmission [ISI81, Stev94, Tab95]. The fundamental of this algorithm is to measure the RTT experienced on a given connection, and modify its timeout using this measured value. After an RTT of this connection is measured, the estimated RTT is determined by

$$\text{new_RTT} = \alpha * \text{old_RTT} + (1 - \alpha) * \text{measured_RTT}$$

Where α is a smoothing factor with a recommended value of 0.9 [Stev94].

Given this smoothed estimator (new_RTT), the retransmission timeout is set to

$$\text{timeout} = \beta * \text{new_RTT}$$

Where β is a delay variance factor, and $\beta > 1$ [Tab95]. In RFC 793 [ISI81], the recommended value of β is 2.

In L2TP tunnel establishment, the situation is a little different from TCP. The L2TP control message is quite small in size (200 bytes in this research). Here we have reserved a large spare room for potential uses. In normal practice, the packet size of the control message should be smaller than this. Due to this fact, a few more retransmissions will not add much traffic to the network.

On the other hand, the message exchange during the tunnel generation is a send-and-wait mechanism. After a message (say, the SCCRQ) is sent out from the LAC, the LAC will wait for the acknowledgement (i.e. the SCCRP) from the LNS. The sliding window mechanism is not applicable at this moment. In this case, we should not worry

about much traffic to be added to the network. Therefore, we can pay more attention to the efficient use of the network bandwidth. Based on this point of view, an adaptive algorithm is developed, in which we use a lower value as the delay variance factor ($\beta = 1.5$) so as to reduce the tunneling time (see section 6.2).

In order to compare the results from the new algorithm with that from the old one, a new scenario, named Scenario 9_1 is generated based on Scenario 9. They are exactly the same except Scenario 9_1 is using the new retransmission algorithm.

6.2 New Adaptive Algorithm Design for Retransmissions

Like TCP retransmission, the new algorithm uses an adaptation mechanism to collect RTT (Round-Trip-Time) in order to adjust the retransmission interval. The RTT for each tunnel will be collected from the generation of this tunnel and will be used for the follow-up tunnel generation between these two nodes. Instead using a fixed value (in TCP), the smoothing factor is also adapted according to the collection times of the RTT. In order to store these information, two tables are created in each ending router (here in this research, the stub node): `table_RTT` and `table_COUT`. In `table_RTT` there are RTTs for the tunnels that start from this node. The RTT will be used as the knowledge to arrange packet retransmissions. In `table_COUT` are the numbers of times the RTT is collected recently for each tunnel.

In Scenario 9_1, there are 96 stub nodes. From each stub node, 95 possible tunnels can be generated, each to one of the destination stub nodes. Therefore, 95 entries are defined in `table_RTT` and `table_COUT` respectively, each entry for a possible destination node (in a real network, it is a VPN partner of this LAC or LNS). The destination node ID is used as the index of the entry. All entries in both `table_RTT` and `table_COUT` are initialized with 0.

Table 6-1 illustrates the structure of `table_RTT` and `table_COUT`.

The L2TP agents are responsible for collecting the RTT, and passing it to the node. The node will update the existing value, if it exists, with the new value. The process of RTT collecting and updating can be described as follows:

table_RTT										
Dest. node-ID	1	2	3	4	5	6		95		
RTT	0	0	0	0	0	0		0		

table_COUNT										
Dest. node-ID	1	2	3	4	5	6		95		
count	0	0	0	0	0	0		0		

Table 6-1: The structure of table table_RTT and table_COUNT

When a pair of agents is attached to a pair of stub nodes, the L2tp_lac agent firstly accesses the table_RTT of the LAC to look up the value of the corresponding entry. If this value is greater than 0, the L2tp_lac agent will use it as the base to compute timeout interval for all retransmissions in this tunnel generation. If this value is 0, it means that there is no knowledge available for this path yet. In this case, the original L2TP retransmission algorithm (defined in RFC 2661) will be used as the default one.

While a pair of L2TP agents generates a tunnel, the L2tp_lac agent will collect the RTT of the packet (i.e., the time span from sending the SCCRQ to receiving the SCCRP). This value is then, passed to the LAC node. The LAC node updates the corresponding entry in its table_RTT and table_COUNT table using the value:

```
new_rtt = (old_rtt * count + rtt) / (count + 1);
count = count + 1;
```

Where

new_rtt: the new average value that will be used to replace the existing one.

old_rtt: the existing average RTT value before update.

rtt: the RTT value newly collected by the L2tp_lac agent.

count: the number of times that the RTT has been collected recently.

If there is no existing data in this entry (in this case, count = 0), the new collected RTT will be assigned as the entry value. If the existing RTT is the average of n times'

collection, the influence of the new collected RTT is $1 / (n+1)$. This will avoid that a specific RTT value has too much influence on the choice of retransmission interval. Only the 10 most recent RTT values are considered important for the decision.

When the L2tp_lac agent is going to retransmit a packet, it accesses `table_RTT` to get the knowledge, and sets the timeout value with

$$\beta * \text{new_rtt}$$

Where $\beta = 1.5$ (see the discussion in section 6.1).

When measuring the RTT value, the *retransmission ambiguity problem* must be paid attention to, or the agent may obtain a wrong data. Two ambiguities are illustrated in Figure 6-1 and Figure 6-2.

Figure 6-1. Retransmission ambiguity 1: the retransmission's ACK is wrongly regarded as the original transmission's ACK

Figure 6-2. Retransmission ambiguity 2: the original transmission's ACK is wrongly regarded as the retransmission's ACK

In Figure 6-1, the ACK for the original transmission may be lost. The agent wrongly takes the ACK for the retransmission as the ACK of the original transmission. In Figure 6-2, the ACK for the original transmission comes after the retransmission occurs. The L2tp_lac agent wrongly regards the original transmission's ACK as the retransmission's ACK. In both situations, the agent learns incorrect knowledge.

In order to avoid this problem, a variable named `time_stamp` is defined in the packet struct. When an L2tp_lac agent is going to send a packet, it reads the current system time, and stamps this time on the packet. This time is stored in variable `time_stamp`. When the LNS receives this packet, it takes the time out, and then, put it into the ACK packet. Thus, when the LAC receives the ACK, the L2tp_lac agent will calculate the RTT by

$$\text{RTT} = \text{now_time} - \text{stamp_time}$$

Where

`now_time`: the current system time.

`stamp_time`: the time stored in the variable `time_stamp`.

This process is applied to all SCCRQ packets, no matter whether it is the original packet or a retransmitted one. Since each packet carries its own sending time, its RTT can be calculated individually. Therefore, no ambiguity will arise.

There are 96 stub nodes in Scenario 9_1 in total. From each stub node, 95 tunnels may be generated. On average, it needs 95 samples of simulation for a stub node to obtain the `new_rtt` for each entry. For 96 stub nodes, we need $95 * 96 = 9120$ samples of simulation to fill the knowledge table on average. This will take a long simulation time. We should optimize it somehow.

In this simulation, the links in networks are duplex-links, i.e. the delay and the bandwidth of a link are identical in both directions. If the RTT for sending a packet from node A to node B is m , the RTT for sending from B to A is also m . Thus, when an L2tp_lac agent passes the new value to the LAC node, we can let it pass this value to the LNS node as well (the L2tp_lac agents possesses the ID of the destination node, i.e. the

LNS node in its instance variable `destID_`. See section 4.1.3 and 4.4.1 for details). Using this value, the LNS node updates the entry of its `table_RTT` and `table_COUT` which is indexed by the LAC node's ID.

In this way, the number of samples for filling the knowledge table can be cut by half. Theoretically, we only need $9120 / 2 = 4560$ samples to build the table. Therefore, the number of tunnels included in a run in Scenario 9-1 is defined as 7000 tunnels, in which the first 5000 tunnels are used to build the retransmission knowledge table, and the last 2000 tunnels are used as the observations for the system analysis.

6.3 Algorithm Implementation

In order to implement the new algorithm designed in section 6.2, a series of changes must be made in the L2TP agent, the stub node and the Tcl script of the scenario. A number of new elements need to be added in those programs. The new added elements and their functionalities are described as follows.

Changes in L2TP agent: In agent `L2tp_lac`, two methods are added:

```
double L2tpAgent_lac::get_table_RTT() { ... }
void L2tpAgent_lac::pass_knowledge(double rtt) { ... }
```

Method `get_table_RTT()` is used to access knowledge table `table_RTT` in the LAC so as to get the retransmission information for a given destination node. When the `L2tp_lac` agent arranges the retransmission timer, this method is called. The RTT value returned by this method is used to set the timer. If no information exists for the path to this destination node, the original algorithm defined in the L2TP protocol will be used to deal with the retransmission.

After getting the retransmission information for this tunnel, a method named `set_rtran_timer()` is called to handle the retransmission timer:

```
void L2tpAgent_lac::set_rtran_timer() {
    int default_value, sID, rID;
    double knowldg_value, rtx_t;
```

```

        knowldg_value = this->get_table_RTT();
        if(knowldg_value < 0.01) {
            rtx_t = (double)this->default_rtx_interval();
            rtran_timer_.resched(rtx_t);
        }
        else {
            rtx_t = knowldg_value * 1.5;
            rtran_timer_.resched(rtx_t);
        }
    }
}

```

Where `default_rtx_interval()` is an instance method in the `L2tp_lac` agent in which the exponential back-off interval algorithm is defined.

After the new RTT for this path is measured, method `pass_knowledge()` is called to pass this value to the LAC node.

Changes in node: The new algorithm is implemented at the OTcl node. At this level, users can get the whole logical view of the node. Firstly, an instance procedure for the table initialization needs to be defined:

```

Node instproc init_table {trn_n total_n} {
    $self instvar table_RTT
    $self instvar table_COUT

    for {set i $trn_n} {$i < $total_n} {incr i} {
        set table_COUT($i) 0
        set table_RTT($i) 0
    }
}

```

In this procedure, both `table_RTT` and `table_COUT` are defined as node instance variables. When a node is created by `ns`, this procedure is called from the Tcl script to initialize all entries in `table_RTT` and `table_COUT` with value 0. Beside this, two other procedures are defined. One is named `get_table_RTT{dest_id}`. It is used to retrieve the entry values in the knowledge table. The other is named `set_table_RTT{dest_id rtt}`. It is used for passing the new RTT value to the table and updating the corresponding entry in the table.

```

Node instproc get_table_RTT {dest_id} {
    $self instvar table_RTT
    return [set table_RTT($dest_id)]
}

```

```

Node instproc set_table_RTT {dest_id rtt} {
    $self instvar table_RTT
    $self instvar table_COUT

    set count [set table_COUT($dest_id)]
    if {$count >= 10} {
        set count 9
    }
    set old_rtt [set table_RTT($dest_id)]
    set new_rtt [expr ($old_rtt * $count + $rtt) /
        ($count + 1)]
    set table_RTT($dest_id) $new_rtt

    incr table_COUT($dest_id)
}

```

Changes in Tcl script: The scenario of a simulation is represented by a Tcl script. The Tcl script is the bridge between the node (Otcl view) and the agent. L2TP agents communicate with nodes through Tcl procedures provided in the Tcl script. Those procedures act as interfaces to both L2TP agents and stub nodes.

When the nodes are created, *ns* calls the node instance procedure `init_table {}` to initialize the knowledge tables:

```

for {set i 0} {$i < $total_node} {incr i} {
    set n($i) [$ns node]

    # Call node's instproc to initialize the RTT table
    $n($i) init_table $tran_node $total_node
}

```

When an L2tp_lac agent is going to retransmit a packet, it firstly access the knowledge table of the LAC node to retrieve the information through a Tcl procedure `get_RTT{}`. The procedure `get_RTT{}` is defined as follows:

```

proc get_RTT {host_id dest_id} {
    global n

    set rtt_v [$n($host_id) get_table_RTT $dest_id]
    return $rtt_v
}

```

After the L2tp_lac agent measures the new RTT of the path, it pass this value to both the LAC node and the LNS node through the Tcl procedure `set_RTT{}`.

```

proc set_RTT {host_id dest_id rtt} {
    global n

    # Modify the host's knowledge table
    $n($host_id) set_table_RTT $dest_id $rtt

    # Modify the peer's knowledge table
    $n($dest_id) set_table_RTT $host_id $rtt
}

```

Through these interface procedures, the communication can be carried out among components. Thus, those components are integrated into a simulation scenario. The communication procedures and the relationships among the L2tp agent, the Tcl interface, and the stub node can be represented using a UML sequence diagram. The diagram is illustrated in Figure 6-3.

Figure 6-3: Sequence diagram for the communication among the LAC agent, Tcl interfaces, and the LAC node.

6.4 Result Analysis and Comparison

The implementation of the simulation and procedures of the output analysis are the same as that in Scenario 9. As described in section 6.2, the number of tunnels included in a run is 7000, in which the first 5000 tunnels are used for building up the knowledge table.

These 5000 tunnels will be removed from the samples and they will not be included in the data analysis. The remained 2000 tunnels are used as the observations. Therefore, the run length in this scenario is still 2000 tunnels.

The Ensemble batch means and cumulative means of 10 runs are calculated. They are displayed in Table 6-2. Observing the average batch mean (column 3) and the cumulative average without batch deleted (column 4) in the Table 6-2 we can see that the initial condition almost has no influence on the remained samples. Therefore, no batch needs to be deleted.

Based on those 10 runs of simulation, the smallest sample size for getting a 95% confidence for 6.5% half-length is calculated. The result is displayed in Table 6-3. This is not an estimate, but an accurate calculation. The result indicates that 10 runs of simulation are able to meet the requirement of the given criterion. Therefore, no more runs are needed.

Run Length T	batch j	Average batch mean $\bar{Y}_j$	Cumulative Average $\bar{Y}_{..(j,0)}$ (no delete)
From 5000			
5100	1	1894.26	1894.26
5200	2	1883.31	1888.78
5300	3	2073.95	1950.50
5400	4	1825.10	1919.15
5500	5	1959.16	1927.15
5600	6	1858.04	1915.64
5700	7	1974.94	1924.11
5800	8	1765.33	1904.26
5900	9	2205.98	1937.78
6000	10	1800.91	1924.10
6100	11	1960.72	1927.43
6200	12	1814.02	1917.98
6300	13	1796.11	1908.60
6400	14	1846.07	1904.14
6500	15	2001.41	1910.62
6600	16	1873.23	1908.28
6700	17	1903.98	1908.03
6800	18	1836.49	1904.06
6900	19	1844.04	1900.90
7000	20	2090.42	1910.37

Table 6-2: Ensemble batch means and cumulative means, averaged over 10 runs

R	8	9	10
$(t_{0.025, R-1} * S_0 / \epsilon)^2$	9.37	8.98	8.59

Table 6-3: The smallest sample size required for scenario 9_1

The means of all items are calculated based on the outputs of 10 runs' simulation. They are displayed in Table 6-4. Comparing them with the means of Scenario 9 (Table 5-4), we can see that the average tunneling time of Scenario 9_1 (1910.37 ms) is much shorter than that of Scenario 9 (2768.52 ms). This means that the new retransmission algorithm can greatly improve the performance of the system.

Run	Hops (per tunnel)	rtx (per tunnel)	ps_wait (ms) (per tunnel)	Time (ms) (per tunnel)
1	6.70	0.72	2.04	1760.99
2	6.84	0.78	2.76	1937.05
3	6.81	0.62	2.60	1742.13
4	6.82	0.78	4.22	1855.80
5	6.83	0.93	2.39	2116.84
6	6.97	0.84	2.83	1797.53
7	6.89	0.81	2.48	2052.24
8	6.88	0.88	2.43	2187.57
9	6.77	0.68	4.03	1746.47
10	6.71	0.76	2.52	1907.09
Mean	6.82	0.78	2.83	1910.37

Table 6-4: Mean values in of 10 runs in Scenario 9_1

The average number of retransmission of Scenario 9 is 0.83 per tunnel. The number of times is 0.78 in scenario 9_1. In order to make them more comparable, we can calculate the tunneling time based on the same number of retransmission. The tunneling time of Scenario 9_1 with number of retransmission 0.83 is

$$(1910.37 / 0.78) * 0.83 = 2033 \text{ (ms)}$$

The improvement of the tunneling time is:

$$(2768.52 - 2033) / 2768.52 = 0.2657$$

This means that the performance of the system is improved by 26.57% when the average number of times of retransmission is 0.83 per tunnel. Figure 6-4 shows the comparison of tunneling time from L2TP tunneling and improved L2TP tunneling. Since the retransmission intervals defined in the original L2TP protocol grow exponentially, it is obvious that with the increase of the number of retransmissions, the performance improvement will be more remarkable.

In order to show this conclusion, the average tunneling time at different numbers of retransmission is calculated. The comparison of the average tunneling time under original L2TP algorithm and the adaptive retransmission algorithm is illustrated in Figure 6-5. In this figure, the curve of Scenario 9_1 (using the adaptive retransmission algorithm) is much more flat than the curve of Scenario 9. With the number of retransmission getting bigger, the difference between two curves becomes bigger. Therefore, when the network is very busy, the new algorithm will greatly improve the system performance.

Figure 6-4: The comparison of tunneling time from L2TP tunneling and improved L2TP tunneling

Figure 6-5: The comparison of tunneling time at different rtx times by using L2TP rx algorithm and using adaptive algorithm

Figure 6-6 shows the difference of tunneling time between two scenarios at different numbers of hops. Though there is no relation between the new algorithm and the hops for the tunnel, we can see that the average tunneling time in Scenario 9_1 is improved at different lengths.

Figure 6-6: The comparison of tunneling time at different hops by using L2TP rx algorithm and using adaptive algorithm

The percentage of failures to generate tunnels should be compared between the two scenarios. This can tell us if the new retransmission mechanism results in more failed

tunnels. In Scenario 9, the average number of failed tunnels in a run is 380. The failure percentage is $380 / (380 + 2000) = 16\%$, where the number 2000 is the run length. In Scenario 9_1, the average number of failed tunnels in a run is 430. The failure percentage is $430 / (430 + 2000) = 17.7\%$. The percentages are very close. Considering the influence of the initial condition in Scenario 9, it is reasonable to believe that the new retransmission mechanism does not cause more failures.

The system that implements the new retransmission algorithm will need more memory resource since two tables need to be generated on each node. Therefore, there is a trade-off between memory and the performance improvement. In this research, there are 95 entries in table `table_RTT`. Each entry can be a double variable in 8 bytes. The table will takes up 760 bytes in storage. Each entry in table `table_COUT` can be an integer in 4 bytes. This table will need 380 bytes. Two tables will need 1140 bytes in total. Of course, the tables should be declared larger than this for the potential use. However, it is not a big deal for a modern network switch or router.

In this chapter, a new algorithm for the packet retransmission is designed and implemented. The output analysis shows that the new algorithm can significantly improve the performance of the system, especially when the network is very busy. The trade-off is memory versus improved time. The memory required by the tables (the table in an `Otcl` node can be implemented either as an `Tcl` array or as a list) is relatively unserious.

CHAPTER 7: SUMMARY AND CONCLUSION

The simulation and performance analysis of a network protocol is a complex task. In order to perform a simulation, the simulation system has to be firstly designed and built up. Usually, a number of components may be included in the system. After that, a series of follow-up research work needs to be done, such as the simulation implementation, data analysis, system improvement and so on. The achievement of the simulation needs multiple aspects of efforts to be involved in. It is quite similar to the process of producing a product: we have to build up the workshop first, design and construct the manufacture line, and only then, the product can be produced. After the product is built, it needs to be tested, and further, to be improved. Though our final goal is to obtain the product, we have to go through all those procedures to reach the goal.

In this research, efforts have been made in multiple aspects around the simulation design, implementation, data analysis and protocol improvement. The research work can be summarized as follows:

- The review and discussion of the VPN concept, categories and methods. VPN is a very broad category. Different technologies and methods have been used to fulfill its central goal: taking the public network as the private use, so as to pursue a secure communication. In this research, the Remote Access model is taken to perform the simulation of L2TP tunneling.
- The review of the L2TP protocol and the discussion about the implementations of L2TP in VPN uses. L2TP consists two components: control connection and data session. The task of control connection is to establish, maintain and terminate the tunnel. The control connection is a reliable connection. An L2TP data session is for an individual user to tunnel a PPP session. It is an unreliable transport. This research focuses on the establishment of the control connection.
- The investigation of the simulation environment (*ns*). The simulation is performed on an *ns* platform. *ns* is a network simulator that provides an

environment in which users can construct networks, implement network protocols and perform various simulations.

- The design of the simulation model. The model design mainly focuses on the concept and methodology design. It includes: problem formulation, model conceptualization, the concept and method of model translation, verification and validation, production run analysis etc. After the system is modeled, the basic building blocks of the model can be constructed.
- The development of L2TP agents. A pair L2TP agents is developed. One is named L2tp_lac. It is attached to the LAC. The other is named L2tp_lns, which resides on the LNS. These two agents cooperate to perform the control message exchange to establish the L2TP tunnel between the LAC node and the LNS node. In these two L2TP agents, a CPU competition mechanism is played. A semaphore is used to model the mutually exclusive operations on the CPU resource. When the main "CPU" of a node (represents a router) is occupied by an agent, other agents who want to use the CPU resource have to wait until the CPU is released.
- The construction of network topologies. Four network topologies are constructed. Two of them consist of 54 nodes. The other two are double in size, i.e., 108 nodes. The two networks (for each size) have different node degree: one is 2.19, and the other is around 3.6.
- The generation of background traffic. Background traffic is brought into the networks. For each network, three levels of traffic-sending rate (100 Kbps, 500 Kbps, and 1000 Kbps) are used to generate background traffic.
- The design and generation of simulation scenarios. The simulation building blocks are integrated into a system through a simulation scenario. Twelve scenarios are generated, each corresponding to a combination of one of the four networks and one of the three traffic rates.
- The implementation of simulation in 12 scenarios and output data analysis. The simulation is divided into runs, and each run is divided into batches. The first few batches may be discarded for removing the initial condition influence. Usually, many runs are required in each scenario, depending on the accuracy of the output

data. A 95% confidence interval with 5% - 6.5% half-length width of error is set to be the precision for the data analysis.

- **Performance analysis.** The system behavior in different situations is investigated. The performance analysis shows that the packet retransmission algorithm of L2TP seriously lowers the system performance.
- **The improvement of the retransmission mechanism in L2TP protocol.** A new adaptive algorithm for the packet retransmission is designed. The new algorithm uses a adaptation mechanism to obtain the RTT of a tunnel path. Instead using a set of fixed retransmission intervals for all tunnels, each tunnel uses its own RTT to make the retransmission interval in its establishment.

Through performance analysis, the following conclusions can be derived:

1. The average tunneling time in a network is determined by a number of factors. These factors include: the average length of tunnels, the size and the node degree of the network, the average traffic rate, the number of users, the average retransmission times and the retransmission algorithm etc. Some of these factors are independent, and some are correlated with each other.
2. The connectivity level of the network influences the average length of tunnels. For two networks of the same size (comprise the same number of nodes), the tunnels in the network that has a higher node degree will be shorter (on average) than that in the network which has a lower node degree.
3. The capacity of the network is related to the connectivity level of the network. When two networks are in the same size (same number of nodes), the network with a high node degree can accommodate more traffic. Therefore, the average tunneling time in this network is shorter than that in the network with a low node degree.
4. If two networks have a very high node degree (such as at node degree 3.60), the large network (comprises more nodes) will give a better performance than the small one when there is a heavy traffic (such as the traffic-sending rate of each node is 1000 Kbps) in both networks, even though the average tunnel length in the large network is greater than that in the small network.

5. The packet retransmission in L2TP employs an exponential back-off interval. This algorithm has drawbacks that negatively influence the system performance. A new mechanism which allows the timeout interval for each individual tunnel to be created can notably improve the performance of the system.

As mentioned previously, the simulation of a network protocol is a complex task. Many variances can be brought into the simulation to produce new simulation scenarios. Though 12 scenarios are created and used in this research, they are still limited, since the real networks in the world are so complicated that no one individual simulation can cover all cases of them. Based on this research, some interesting topics may be further investigated. Those topics include:

- i. Identifying the local communication and the inter-stub communication. In a Transit-Stub network, a tunnel may fall in a stub (local tunnel) or go across different stubs (inter-stub tunnel) either through a transit area or via a stub-stub link. In this research, the local tunnels are not distinguished from the inter-stub tunnels since this needs a complicated node identification mechanism. The distinction of the local communication from the inter-stub communication may help us to investigate the influence of the network structure on the system performance.
- ii. Multicast mechanism. Multicast is an efficient mechanism in the one-to-multiple communication. In the Internet, the multicast operation is only supported in UDP. The TCP does not support multicast operation because it is difficult to handle the acknowledgement in a reliable delivery. L2TP protocol does not define multicast issue. In L2TP, the control connection is reliable, and the data sessions are unreliable. Exploring the possibility that introducing the multicast mechanism into L2TP is significant, such as the possibility for generating the tunnel individually, but delivering the data by multicast.

REFERENCES

- [Ahm99] Ahmed, Mahad M., 1999, *Performance Analysis and Improvement of the Open Shortest Path First Routing protocol*, Carleton University, Ottawa, ON, (Master thesis).
- [Ake00] Akesson, Leif, 2000, "Security Beyond Firewalls"
http://idm.internet.com/articles/200003/se_03_29_00a.html
- [AZ00] Aboba, B. and G. Zorn, "Implementation of L2TP Compulsory Tunneling via RADIUS", *RFC 2809*, Apr. 2000.
- [Bal93] Balenson, D., "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers", *RFC 1423*, Feb. 1993.
- [BCN96] Banks, Jerry, John S. Carson and Barry L. Nelson, *Discrete-Event System Simulation (Second Edition)*, Prentice Hall, Upper Saddle River, New Jersey, 1996.
- [Bla00] Black, Uyless, 2000, *PPP and L2TP: Remote Access Communications*, Prentice Hall PTR, New Jersey, 07458.
- [BV98]] Blunk, L. and J. Vollbrecht, "PPP Extensible Authentication Protocol (EAP)", *RFC 2284*, Mar. 1998.
- [CDZ97] Calvert, L. Kenneth, Matt Doar and Ellen W. Zegura. Modeling Internet Topology, *IEEE Communications Magazine*, June 1997.
- [CohR00] Cohen, Reuven, 2000, On the Cost of Virtual Private Networks, *IEEE/ACM Transactions on Networking*, vol. 8, No. 6, December 2000.
- [CohS00] Cohn, Stephen, 2000, "Issues in Intranet Security, A Primer on Keeping the Keys to the Enterprise Safe"
<http://www.intranetjournal.com/features/isecurity.shtml>
- [Ent02] Enterasys Networks Inc., 2002, "Virtual Private Networks A Technology Overview"
<http://www.enterasys.com/products/whitepapers/vpn/9011742.html>
- [FH98] Ferguson, Paul and Geoff Huston, 1998, "What is a VPN?"
<http://www.clark.net/pub/timw/vpn/Tech/vpn.pdf>

- [FV00] Fall, Kevin and Kannan Varadhan, 2000, "ns Notes and Documentation"
<http://www.isi.edu/nsnam/ns/doc/index.html>
- [GLHAM00] Gleeson, B., A. Lin, J. Heinanen, G. Armitage, and A. Malis, "A Framework for IP-Based Virtual Private Network", *RFC 2764*, Feb. 2000.
- [Ham99] Hamzeh, K. *et. al*, "Point-to-Point Tunneling Protocol (PPTP)", *RFC 2637*, July 1999.
- [ISI81] Information Science Institute, University of Southern California, "Transmission Control Protocol" *RFC 793*, Sept. 1981.
- [Jai99] Jain, Raj, 1999, "Virtual Private Networks"
<http://www.cis.ohio-state.edu/~jain>
- [KA98] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", *RFC 2401*, Nov. 1998.
- [Kel86] Kelton, W. D., 1986, Replication Splitting and Variance for Simulating Discrete-Parameter Stochastic Processes, *Operations Research Letters*, Vol. 4,
- [Kos98] Kosiur, Dave, 1998, *Building and Managing Virtual Private Networks*, John Wiley & Sons, Inc. USA
- [L2TPcisco] "Layer 2 Tunneling Protocol, A Feature in Cisco IOS Software, A Key Building Block for an Access Virtual Private Network", May 2000.
<http://www.cisco.com/warp/public/cc/cisco/mkt/ios/archive/prodlit/l2tunds.htm>
- [Law91] Law, Averill M., 1991, *Simulation Modeling and Analysis (Second Edition)*, McGraw-Hill, New York.
- [LK82] Law, Averill M. and W. D. Kelton, 1982, Confidence Intervals for Steady-State Simulation, II: A Survey of Sequential Procedures, *Management Science*, Vol. 28, No. 5.
- [LucTech99] Lucent Technologies, 1999, "Configuring L2TP"
<http://www.livingston.com/tech/docs/pm4-config/l2tp.html>
- [McD00] McDonald, Christopher, "Feature Virtual Private Networks, An Overview"
<http://idm.internet.com/foundation/vpn-2.shtml>
- [Opp98] Oppliger, Rolf, 1998, *Internet and Intranet Security*, Artech House Inc. Norwood, MA.

- [PH01] Pun, Konghong, Mounir Hamdi, 2001, Provision of Quality of Service for Virtual Private Networks, *Proceedings of SPIE*, Vol. 4211 (2001).
- [Pha00] Phaltankar, M. Kaustubh, 2000, *Practical Guide for Implementing Secure Intranets and Extranets*, Artech House Inc. Norwood.
- [Pos81] Postel, J., "Internet Protocol", STD 5, *RFC 791*, Sept. 1981.
- [Pri98] "Private Use of Public Networks for Service Provider"
<http://www.3com.com/>
- [RP94] Reynolds, J. and J. Postel "Assigned Numbers", *RFC 1700*, Oct. 1994
- [RRSW97] Rigney, C., A. Rubens, W. Simpson and S. Willens, "Remote Authentication Dial In User Service (RADIUS)", *RFC 2138*, Apr. 1997
- [SG98] Silberschatz, Abraham, Peter Baer Galvin, 1998, *Operating System Concepts, Fifth Edition*, Addison Wesley Longman, Inc.
- [Sha75] Shannon, Robert E., 1975, *System Simulation: The Art and Science*, Prentice-Hall, Englewood Cliffs, NJ.
- [She00] Shea, Richard, 2000, *L2TP Implementation and Operation*, Addison Wesley Longman, Inc. Massachusetts, 01867.
- [Sim96] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", *RFC 1994*, Aug. 1996.
- [Sim94] Simpson, W., Editor, "The Point-to-Point Protocol (PPP)", STD 51, *RFC 1661*, July 1994.
- [Smi93] Smith, Peter, 1993, LUC Public-Key Encryption, A Secure Alternative to RSA, *Dr. Dobbs's Journal*, January 1993, Vol. 18, Issue 1.
Also at: <http://www.luc.co.nz/public1.html>
- [Sri00] Srisuresh, P., "Secure Remote Access with L2TP", *RFC 2888*, Aug. 2000
- [SSoftware] Shamus Software Ltd., "Multiprecision Integer and Rational Arithmetic C/C++ Library"
<http://indigo.ie/~mscott/>
- [Stev94] Stevens, Richard, 1994, *TCP/IP Illustrated, Volume 1, The Protocol* Addison-Wesley.
- [SWE99] Scott, Charlie, Paul Wolfe and Mike Erwin, 1999, *Virtual Private Networks (second edition)*, O'REILLY, Sebastopol, CA

- [Tab95] Tabor, Daniel Z., "Transmission Control Protocol (Part II)"
<http://ccsweb.njit.edu/~cis456/protected/lesson16/single16.html>
- [TB99] Taylor, Laura and Bradley L., 1999, "VPN Are Hot, but What Are They?"
http://idm.internet.com/articles/199911/ft_11_16_99a.html
- [Tow99] Townsley, W. *et. al*, "Layer Two Tunneling Protocol "L2TP"", *RFC 2661*, Aug. 1999.
- [UCB] "UCB/LBNL Network Simulator: Topology generation"
<http://www.isi.edu/nsnam/ns/ns-topogen.html>
- [VLK98] Valencia, A., M. Littlewood and T. Kolar, "Cisco Layer Two Forwarding (Protocol) L2F", *RFC 2341*, May 1998.
- [VPN98] VPNet Technologies Inc., 1998, "What's a VPN Anyway? A Virtual Private Networking Primer"
<http://www.vpnet.com/>
- [ZCB96] Zegura, W. Ellen, Kenneth L. Calvert and Samrat Bhattacharjee, 1996, How to Model an Internetwork, *Proceedings of IEEE Infocom' 96*, San Francisco, CA.