Automatic Network Address Adjustment (ANAA)

by

Anand Dersingh

Faculty of Engineering

November, 2001

Automatic Network Address Adjustment (ANAA)

A thesis

submitted to the Faculty of Engineering

by

Anand Dersingh

in partial fulfillment of the requirements

for the degree of

Master of Engineering in Broadband Telecommunications

Advisor: Assist. Prof. Dr. Kittiphan Techakittiroj

Assumption University

Bangkok, Thailand

November 2001

"Automatic Network Address Adjustment (ANAA)"

by

Mr.Anand Dersingh

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Engineering
Majoring in Broadband Telecommunications

Examination Committee:

1. Asst.Prof.Dr.Kittiphan	Techakittiroj	(Advisor)	[illegible]
2. Dr.Sudhiporn	Patumtaewapibal	(Member)	[illegible]
3. Asst.Prof.Dr.Putchong	Uthayopas	(Member)	[illegible]
4. Dr.Wittawat	Na Nacara	(MUA Representative)	[illegible]

Examined on: November 10, 2001

Approved for Graduation on: December 15,2001

Faculty of Engineering, Assumption University
Bangkok, Thailand

ABSTRACT

This thesis proposes a new technique of address translation called Automatic Network Address Adjustment (ANAA). ANAA allows computer that contains wrong IP configuration to be able to access the Internet without any modification. ANAA translates the packets generated from these computers. The translation starts from capturing the packet from the Ethernet network, verifying the packet information, correcting (if needed) the wrong information and then putting it back to the network.

The proposed method allows the computer from outside network (e.g. mobile computer) or the computer that mis-configures the network configuration, to use network immediately without reconfiguration. This technique also allows the use of private IP to solve the shortage of IP addresses.

The thesis includes the study and implementation of ANAA technique and testing result is given.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Assist. Prof. Dr. Kittiphan Techakittiroj, for all of his advice and motivation throughout the completion of this thesis. I also would like to thank Dr. Sudhiporn Patumtaewapibal, Dean of Faculty of Engineering, for his closed follow up of my thesis. Furthermore, I would like to thank Jerapong Rojanarowan for his suggestion and guidance. I also would like to dedicate a very special thanks to all of my classmates and friends for their great support.

Most of all, I would like to thank my family for their support, encouragement, understanding, and everything they have been giving to me.

TABLE OF CONTENTS

ABSTRACT	i
ACKNOWLEDMENTS	ii
TABLE OF CONTENTS	iii
LIST OF FIGURES	v
LIST OF TABLES	vii
CHAPTER 1 INTRODUCTION	1
1.1 Literature Survey	2
1.2 Problem Formation	4
1.3 Summary	7
CHAPTER 2 BACKGROUND	9
2.1 LAN Addressing	9
2.2 Address Resolution Protocol (ARP)	11
2.3 Internet Protocol (IP)	14
2.4 Transmission Control Protocol (TCP)	19
2.5 User Datagram Protocol (UDP)	23
CHAPTER 3 ARP SPOOFING	25
CHAPTER 4 ADDRESS TRANSLATION	33
4.1 TCP/IP Packet	35
4.1.1 Setting Up Connection	35
4.1.2 Incoming Packets	39
4.1.3 Outgoing Packets	43
4.2 UDP/IP Packet	46
4.2.1 Outgoing Packets	47

4.2.2 Incoming Packets 50

CHAPTER 5 TESTING 54

5.1 Application over ANAA 55

5.2 Performance Comparison 56

CHAPTER 6 CONCLUSION 64

6.1 Future Work 65

Bibliography 67

LIST OF FIGURES

Figure	Page
1.1 Ethernet Network	4
1.2 ANAA over Ethernet Network	6
1.3 Network Configurations Implementing ANAA	7
2.1 Star Network Topology	10
2.2 ARP packet format	12
2.3 IP header format	15
2.4 TCP Header format	20
2.5 UDP Header format	23
3.1 Flow chart of ARP Spoofing	26
3.2 ARP Request for wrong IP computer	31
3.3 Faked ARP Reply Packet	32
4.1 Flow chart of Address Translation in ANAA	34
4.2 Flow Chart of function send_tcp_opt_ip_to_gw ()	39
4.3 Flow Chart of function send_tcp_ip_to_pc ()	41
4.4 Flow chart of function send_tcp_ip_to_gw ()	45
4.5 Flow chart of function send_udp_to_dns ()	48
4.6 Flow chart of function send_udp_to_pc ()	51
5.1 ANAA Network Configurations	54
5.2 Network configuration of correct IP	56
5.3 ANAA Network configurations	57
5.4 Graphs of Correct IP Vs ANAA (Server Connection Rate) 10Mbit/sec	58
5.5 Graphs of Correct IP Vs ANAA (Average Response Time) 10Mbit/sec	59

5.6 Graphs of Correct IP Vs ANAA (Throughput) 10Mbit/sec 60

5.7 Graphs of Correct IP Vs ANAA (Server Connection Rate) 100Mbit/sec 61

5.8 Graphs of Correct IP Vs ANAA (Average Response Time) 100Mbit/sec 62

5.9 Graphs of Correct IP Vs ANAA (Throughput) 100Mbit/sec 62

LIST of TABLES

Table	Page
3.1 Example of MAC table	29
4.1 Example of IP Table	37
5.1 Applications over ANAA	55
5.2 Simulation Result of 10Mbit/sec Network	59
5.3 Simulation Result of 100Mbit/sec Network	61

CHAPTER 1. INTRODUCTION

The increasing numbers of computers connected to the Internet create problems with the IP addresses. Many of these problems have been issued through research works [4], [5], [6]. The problems can be classified into two major categories.

The first problem concerns with the limited amount of IP addresses. IPv4 provides a network address space of 32 bits, allowing approximately 2^{32} or four billion hosts to exist on the Internet. According to the vast growth rate of new hosts get connected to the Internet, the current Internet protocol IPv4 does not provide enough unique addresses for every hosts on the Internet. Two current approaches have been proposed to solve this problem. The first approach is to use new Internet protocol known as IPv6 [5] that is designed to ease potential address shortages by replacing IPv4's 32-bit address space with the new bit scheme. The other approach is to use private IP address [2] for internal network. Although these two approaches are the solutions for the limited amount of IPv4 address, there is a problem in connecting to the Internet by using these two approaches. IPv6 and private IP address currently cannot be routed on the existing Internet. The address translation is necessary for the case of private IP address; packet translation can be easily implemented by using NAT [1] at the edge of the network to allow access from internal network to the Internet. In the case of using IPv6, the migration technique is still in the development phase.

The second problem concerns with mobile computers. When mobile computer changes its physical network, the IP configuration of that mobile computer must be renumbered to suit with the new network setting, i.e. subnet, gateway, machine's IP address. Using Dynamic Host Configuration Protocol (DHCP) [7] can solve this problem. DHCP server provides all necessary IP configurations to the client computer. The problem of using DHCP is that the mobile computer must be configured to be a DHCP client or to obtain an IP address automatically. If the mobile computer use a static IP address in the previous network, when it changes to the new network it must be reconfigured manually which increases work of administrator to provide IP information such as IP address, subnet mask, gateway, DNS. The other problem of using DHCP is when the network size grows up; reconfiguration of all computers in the network is required.

1.1 Literature Survey

There have been many efforts to solve the problem concerning the computer address. Some efforts [3] concentrate at the data link layer. While others [1], [6], [4], [8], [9], [10] solve the problem of the Internet layer.

[3], introduce Proxy ARP that lets a router that connect two physical networks answer ARP requests pretending that they are in the same physical network. Proxy ARP is useful for communicating among sub networks, when a computer was assigned a wrong subnet numbers. The router will answer ARP request to the computer that has the wrong subnet mask and will forward packets between these two subnets.

According to the limited amount of IPv4, [2] proposed a method of assigning private IP addresses to the internal or private network. This method does not allow internal host to access to the Internet. [1] has proposed translation of private IP address to make host with non-routable IP address to be able to access to the Internet. [6], [9] implemented the Network Address Translation technique called IP Masquerade. IP Masquerade translates packets from computers inside the internal network to one real IP address that can be routable. NAT can be configured easily, and many moderns' routers implement this feature. NAT can be implemented together with DHCP [7] by assigning a pool of private IP addresses to the DHCP server and it will provide all information of IP configuration in the network to the computer inside internal network behind NAT router. When accessing to the public network, translation of the packet is required by the NAT router at the edge of the network.

[4], [8] propose another address reusing technique called Realm Specific IP (RSIP). This technique operates under the same assumptions of physical architecture as NAT. RSIP is defined in terms of operations on the flow, and the signaling association between the client host (RSIP client) and gateway router (RSIP server). When RSIP client want to send packet to the public network, it first has to signal to the RSIP server requesting for the appropriate public IP address and port number(s) to utilize, and then prepares a packet using these parameters. The RSIP client tunnels this packet over the private network to the RSIP server, which then only has to strip off the tunnel header, and forward the packet on to the public network. This method requires a small software change to be embedded on both client and server. This requirement is the main disadvantage for the general applied.

[10] proposes a modified version of NAT called DNAT (Distributed NAT) in which the subnet hosts perform the address translation, and tunnel translated packets to the DNAT router. The DNAT router strips off the tunnel (outside IP header) and forwards the inner packets, unaltered, on to the public network. In fact, it would be more accurate to say that each host constructs externally bound packets in such a way that address translation at the router is not required. The main advantage of DNAT is that it avoids having to code application-specific support into the translation

implementation. DNAT also distributes some of the processing requirements from the router to the hosts, which may enable larger subnets to be supported by a single router.

The concept of RSIP and DNAT that use tunneling are similar, but technically the signaling between client and server are totally different. Both techniques require modification to all clients; therefore it requires more works for the system administrator or the owner to reconfigure system before using these address translation system.

1.2 Problem Formulation

A typical Ethernet network connected to the Internet is shown in Figure 1. In this scenario, as an example, there are five PCs connecting together as a small network. In order to access to the Internet, all computers in the network must be assigned the correct IP configuration to send packets to the gateway and gateway will forward these packets to the Internet.

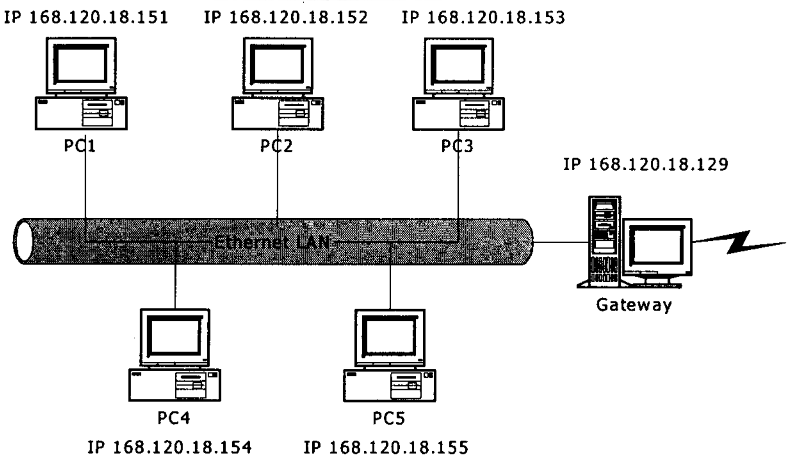

Figure 1.1 Ethernet Network

The problem arises when a computer that uses static IP address from different network try to access to the Internet via this Ethernet network. It is necessary to modify IP configurations every time they change network. This modification increases work of network administrators to take care of renumbering of IP address to supply the gateway and DNS information.

In order to help users and network administrators to solve this problem, the technique called Automatic Network Address Adjustment (ANAA) will be designed, implemented and analyzed. The aim of this technique is to allow relocation of the client computer that moves from one network to another network and client computer that was configured mismatch IP configuration for example, wrong gateway, to access to the Internet without changing IP configuration. The technique is designed in the way that only a few or no modifications to the network computer or administrators are required.

The network configuration is simple as shown in the Figure 1.2. We do not have to change any configurations. One machine will be set as the virtual gateway that will translate the packets that is not routable on the Internet. This virtual gateway can be part of the router, embedded inside any workstation or as a stand-alone terminal. This will be appropriate for the small network where router, DNS, DHCP server and the virtual gateway will be on the same machine.

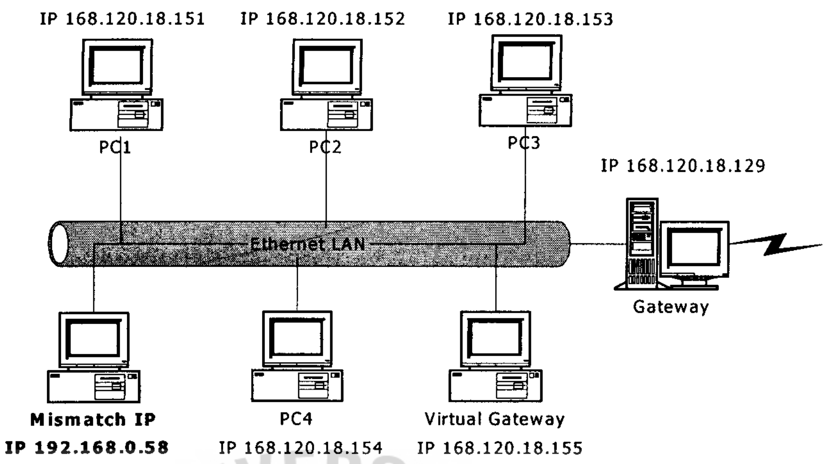

Figure 1.2 ANAA over Ethernet Network

The only requirement is that it must be Ethernet (broadcasting network) in order to see all the packets transmitting in the LAN.

Two main categories of the packet that will be handled by the virtual gateway causes by:

1) Non-routable IP information. Usually this packet happens to the machine that comes from the different network, use the private IP address, or it is mis-configured.

2) The mis-configuration in the DNS portion. This packet is the UDP/IP packet.

The implementation of ANAA in this thesis is to modify the non-routable packet to be able to route on the Internet by dealing with the data link layer, network layer and transport layer. The types of the packets that we consider are only

1) TCP/IP packets that contains problem with wrong IP address, gateway information, and/or subnet mask.

2) UDP/IP packets with application port 53 that is used to resolve name to IP address.

The network configuration in implementing ANAA is shown in Figure 1.3 that is going to be used throughout this work.

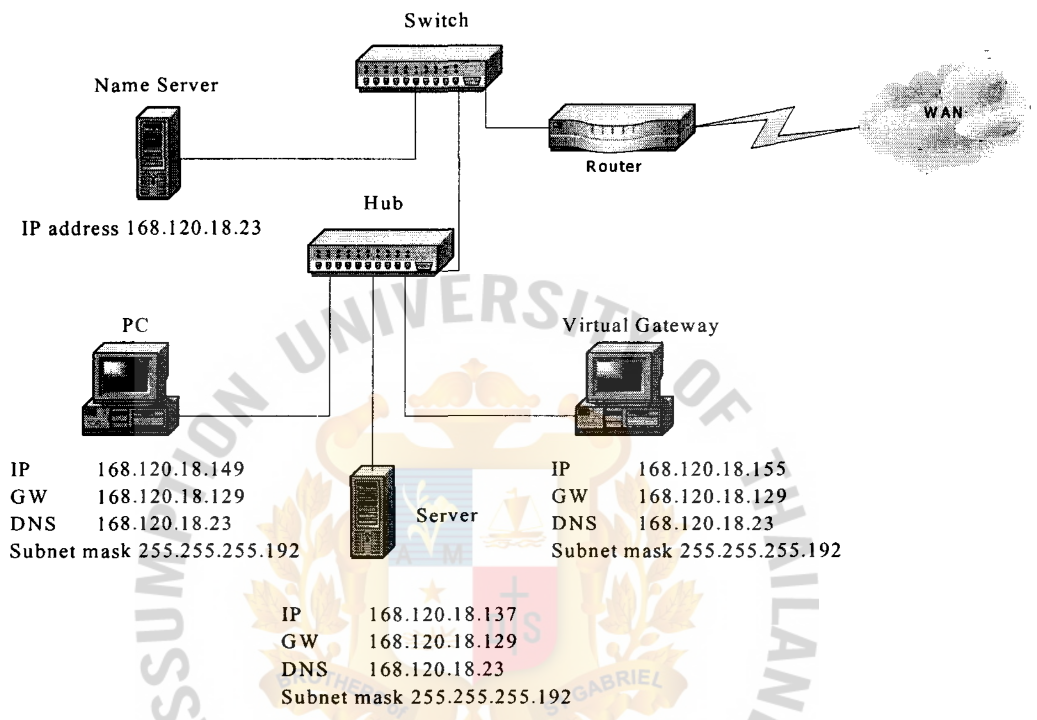

Figure 1.3 Network Configurations Implementing ANAA

1.3 Summary

This thesis proposes a new technique to solve the described problems. This technique is called Automatic Network Address Adjustment (ANAA), ANAA allows IP address to be reused to allow the private IP address to be able to communicate inside the network. It corrects the mismatch IP address to be the correct one and be able to route to the Internet. This technique uses one computer, which may be located inside a network or at the edge, to act as a gateway of address translation. This computer translate packet by modifying data link layer, network layer and transport layer. The modification serve two proposes, i.e. address translation of private IP address and correction of mis-configured address.

Chapter 2 gives the background knowledge of the internetworking. Chapter 3 describes the ARP spoofing technique that is the pre-processing of packets before the translation. The detail of packet translation for TCP/IP and UDP/IP are described in Chapter 4. In chapter 5 shows the testing results in term of supporting applications and performance. Chapter 6 is a conclusion of this work.

CHAPTER 2. BACKGROUND

According to the incompatibilities among network technologies, there is a scheme that provides universal service among heterogeneous networks. It is called internetworking that uses both hardware and software. Additional hardware systems are used to interconnect a set of physical networks. Software on all the attached computers then provides universal service. The resulting system of connected physical networks is known as an internetwork or Internet. There are many methods to connect to the Internet, for example connect through the local area network (LAN) or connect through modem. However, in this work we concentrate on connecting to the Internet via LAN (Ethernet). This chapter provides background knowledge of how computers connected together as a network and protocols used to communicate on the Internet.

2.1 LAN Addresses

LAN is a computer network that is concentrated in a geographical area such as in a building or on a university campus. When a user accesses the Internet from university or corporate campus, the access is almost always by way of a LAN. For this type of Internet access, the user's host is a computer on the LAN, and the LAN provides access to the Internet through router.

Many LAN technologies have been invented; the topology that we are using in this thesis is star topology that all computers attach to a central point as shown in Figure 2.1. The wiring scheme is 10Base-T and 100Base-T.

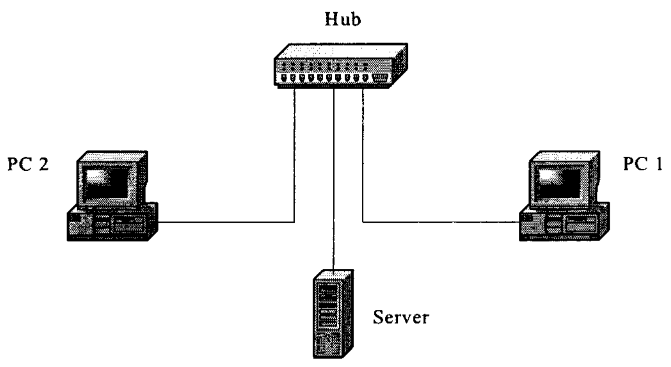

Figure 2.1 Star Network Topology

Computers in LANs send frames to each other over a broadcast channel. This means that when a computer in a LAN transmits a frame, every other computer connected to the LAN receives the frame. But usually, a computer in the LAN doesn't want to send a frame to all of the other LAN computers but instead wants to send to some particular computer in the LAN. In order to send a frame to a particular computer, the computers on the LAN must be able to address each other when sending frames, that is, the computer need LAN addresses and the link-layer frame needs a field to contain such a destination address.

In fact, it is not a computer that has a LAN address but instead a computer's adapter that has a LAN address. A LAN address is also called physical address, Ethernet address, or MAC address. MAC address is six-bytes long, giving 2^{48} possible LAN addresses. These six bytes addresses are expressed as a pair of hexadecimal numbers. An adapter's MAC address is permanent when an adapter is manufactured and each adapter has its unique address.

The LAN interface hardware uses physical addressing to prevent the computer from receiving all packets that travel across the LAN. It is the fact that as a frame travels across the shared medium; a copy of the signal passes to each station. Once it has captured a complete frame, the interface hardware compares the destination address in the frame to the station's physical address. If the destination address matches the station's physical address, the hardware accepts the frame and passes it to the operating system. If not, the hardware discards the frame and waits for the next frame.

As described that when adapter wants to send a frame to some destination adapter on the same LAN, the sending adapter inserts the destination's LAN address into the frame. When the destination adapter receives the frame, it extracts the enclosed datagram and passes the datagram up the protocol stack. All the other adapters on the LAN also receive the frame. This other adapters discard the frame without passing the network-layer datagram up the protocol stack. However, sometimes a sending adapter wants all the other adapters in the same LAN to receive and process the frame. Therefore, the sending adapter inserts a special LAN broadcast address into the destination address field of the frame. The broadcast address for Ethernet is FF-FF-FF-FF-FF-FF.

2.2 Address Resolution Protocol (ARP)

A computer connected to an IP/Ethernet LAN has two addresses. One is the address of the network card, called the MAC address. The MAC, in theory, is a globally unique and unchangeable address that is stored on the network card itself. MAC addresses are necessary for the Ethernet protocol to send data back and forth, independent of whatever application protocols are used on top of it. Ethernet builds "frames" of data, consisting of 1500 byte blocks. Each frame has an Ethernet header, containing the MAC address of the source and the destination computer.

The second address is the IP address. IP is a protocol used by applications, independent of whatever network technology operates underneath it. Each computer on a network must have a unique IP address to communicate. IP addresses are the virtual and are assigned via software.

When an Ethernet frame is constructed, it must be built from an IP packet. However, at the time of construction, Ethernet has no idea what the MAC address of the destination machine is, which it needs to create an Ethernet header. The only information it has available is the destination IP from the packet's header. Address Resolution Protocol (ARP) is the way for the Ethernet protocol to find the MAC address of the destination machine, given a destination IP.

ARP operates by sending out "ARP request" packets. An ARP request asks the question, "Is your IP address 168.120.18.145? If so, send your MAC address back to me." These packets are broadcast to all computers on the LAN. Each computer examines the ARP request, checks if it is currently assigned the specified IP. If IP matches with its own configured IP the network stack sends an ARP reply containing its MAC address to the computer that sends ARP request.

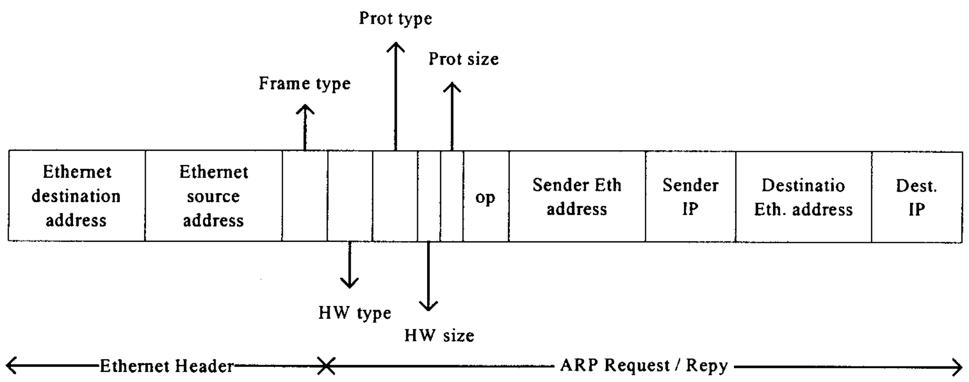

Figure 2.2 ARP packet format

Figure 2.2 shows the ARP packet format. The first two fields in the Ethernet header are the source and destination Ethernet addresses. The special Ethernet destinations address of all one bits (FF-FF-FF-FF-FF-FF) means the broadcast address, which is used in ARP request for mapping IP address and MAC address. The next are

Frame type (2-byte)

Ethernet frame type specifies the type of data that follows. For an ARP request or an ARP reply, this field is 0x0806.

Hardware type (2-byte)

The HW type field specifies the type of hardware address. For Ethernet this value is 0x0001.

Protocol Type (2-byte)

Protocol type specifies the type of protocol address being mapped. For IP address this field is 0x0800.

Hardware Size (1-byte)

HW size specifies the size in bytes of the hardware addresses. For an ARP request or reply of an IP address on an Ethernet it is 0x06.

Protocol Size (1-byte)

Protocol size specifies the size in byte of the protocol address. For an IP address on an Ethernet this field is 0x04.

Operation (2-byte)

The operation field specifies whether the operation is an ARP request (a value of 0x0001), ARP reply (0x0002), RARP request (0x0003), or RARP reply (0x0004). This field is required since the frame type field is the same for an ARP request and an ARP reply.

The last four fields that follow are the sender's hardware address (an Ethernet address in this example), the sender's protocol address (an IP address), the target hardware address, and the target protocol address.

In ARP request, the target hardware address is set to all 0's and the target protocol address is the IP address of the computer you want to communicate inside LAN. When a system of computer that has IP address match to target protocol receives an ARP request directed to it, it fills in its hardware address, swaps the two sender addresses with the two target addresses, sets the operation field to 2, and sends the reply.

Keeping a cache of ARP replies can reduce a number of ARP packets being broadcast in the LAN. When a computer receives an ARP reply, it will update its cache with the new MAC/IP association. As ARP is a stateless protocol, most operating systems will update their cache if a reply is received, regardless of whether they have sent out an actual request.

2.3 Internet Protocol (IP)

IP has made it possible for the Internet to handle heterogeneous networks, dramatic changes in hardware technology, and extreme increases in scale. To handle heterogeneity, IP defines a uniform packet format (the IP datagram) and a packet transfer mechanism. IP datagrams are the fundamental unit of communication in the Internet. IP also defines a set of addresses that allow applications and higher layer protocols to communicate across heterogeneous networks without knowing the differences in hardware addresses used by underlying network systems.

2.3.1 IPv4

In the TCP/IP protocol stack the Internet Protocol version 4 (IPv4) specifies addressing. The IPv4 standard specifies that each host is assigned a unique 32-bit number known as the host's Internet Protocol Address. Each packet sent across an Internet contains the 32-bit IP address of the source as well as the destination. Thus, to transmit information across a TCP/IP Internet, a computer must know the IP address of the remote computer to which the information is being sent.

Figure 2.3 shows the format of an IPv4 datagram. The normal size of the IPv4 header is 20 bytes, unless options are present. The following are the fields in the IP header

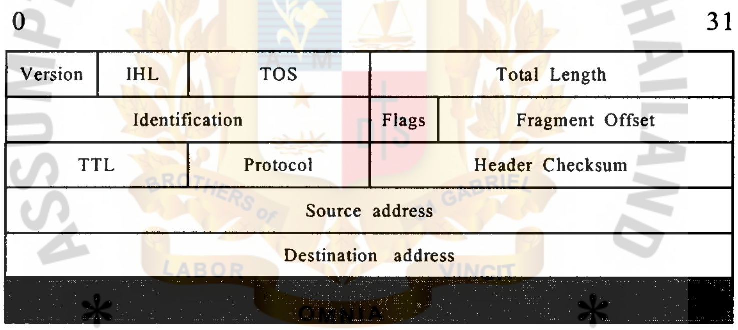

Figure 2.3 IP header format

Version (4 bits)

The Version field indicates the format of the Internet header. In this thesis we consider only version 4.

IHL (4 bits)

Internet Header Length is the length of the Internet header in 32 bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5

TOS (8 bits)

The Type of Service provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. This field is composed of a 3-bit precedence field (which is ignored today), 4 TOS bits, and an unused bit that must be 0. The 4 TOS bits are minimize delay, maximize throughput, maximize reliability, and minimize monetary. Only 1 of these 4 bits can be turned on. If all 4 bits are 0 it implies normal service. In this work, this field is 0x00.

Total Length (16 bits)

Total Length is the length of the datagram, measured in octets, including internet header and data. This field allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments). It is recommended that hosts only send datagrams larger than 576 octets if they have assurance that the destination is prepared to accept the larger datagrams. The number 576 is selected to allow a reasonable sized data block to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximal Internet header is 60 octets, and a typical Internet header is 20 octets, allowing a margin for headers of higher-level protocols.

Identification (16 bits)

An identifying value assigned by the sender to aid in assembling the fragments of a datagram. It normally increment by one each time a datagram is sent.

Flags: 3 bits

There are various control flags

Bit 0	reserved, must be zero	
Bit 1 (DF)	0 = May Fragment,	1 = Don't Fragment.
Bit 2 (MF)	0 = Last Fragment,	1 = More Fragments.

Fragment Offset (13 bits)

This field indicates where in the datagram this fragment belongs. The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

Time to Live (8 bits)

This field indicates the maximum time the datagram is allowed to remain in the Internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in Internet header processing. The time is measured in units of seconds, but since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bind the maximum datagram lifetime.

Protocol (8 bits)

This field indicates the next level protocol used in the data portion of the Internet datagram.

Header Checksum (16 bits)

A checksum is calculated over the IP header only. It does not cover any data that follows the header. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the Internet header is processed.

The checksum field is the 16 bit one's complement of the one's complement sum of all 16-bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

Source Address (4-byte)

This field specifies the IP address of the sender.

Destination Address (4-byte)

This field specifies the IP address of the target host.

Options

The options field is a variable-length. It may appear or not in datagrams. They must be implemented by all IP modules (host and gateways). What is optional is their transmission in any particular datagram, not their implementation. The options field always ends on a 32-bit boundary. Pad bytes with a value of 0 are added if necessary. This assures that the IP header is always a multiple of 32 bits.

2.3.2 IPv6

The current version of IP, IPv4, has been extremely successful. The primary motivation for change arises from the limited address space, larger address space are necessary to accommodate continued growth of the Internet. The secondary motivations for changes in IP have arisen from new Internet applications such as the applications that deliver real-time audio and video.

IPv6 retains many of the design features from the IPv4. Despite retaining the basic concepts from the current version, IPv6 changes all the details. For example, IPv6 uses a series of fixed-length headers to handle header information. Thus, unlike IPv4, that places key information in fixed fields of the header is always variable size.

The new features in IPv6 can be grouped into five broad categories:

- Address Size: instead of 32 bits, each IPv6 address contains 128 bits
- Header Format: the new IPv6 datagram header is completely different that the IPv4 header. Almost every field in the header has been changed.
- Extension Headers: unlike IPv4, that uses a single header format for all datagrams, IPv6 encodes information into separate headers. A datagram consists of the base IPv6 header followed by zero or more extension headers, followed by data.
- Support for audio and video: IPv6 includes a mechanism that allows a sender and receiver to establish a high-quality path through the underlying network and to associate datagrams with that path.
- Extensible Protocol: unlike IPv4, IPv6 does not specify all possible protocol features. Instead, the designers have provided a scheme makes IPv6 more flexible than IPv4, and means that new features can be added to the design as needed.

2.4 Transmission Control Protocol (TCP)

The transmission control protocol (TCP) is one of the main transport protocols in the TCP/IP protocol suite. TCP provides application programs with a reliable, flow controlled, full duplex, stream transport service multiplexing, demultiplexing, and error detection. It is connection-oriented because before one application process can begin to send data to another, the two processes must first "handshake" with each other that is, they must send some preliminary segments to each other to establish the parameters of the ensuing data transfer.

An example of how a TCP connection is established. Suppose an application running in one computer wants to initiate a connection with another application in another computer. The client application process first informs the client TCP that it wants to establish a connection to a process in the server. The transport layer in the client then proceeds to establish a TCP connection with the TCP in the server. The client first sends a special TCP segment; the server responds with a second special TCP segment, and finally the client responds again with a third special segment. The first two segments contain no "payload," that is, no application-layer data; the third of these segments may carry a payload. Because three segments are sent between the two hosts, this connection establishment procedure is often referred to as a three-way handshake.

After requesting TCP to establish a connection, an application program can use the connection to send or receive data, TCP guarantees to deliver the data in sequence order without duplication. Lastly, when both side of computer applications want to terminate the connection, they request the connection to be terminated.

Figure 2.4 shown below is a TCP segment header format.

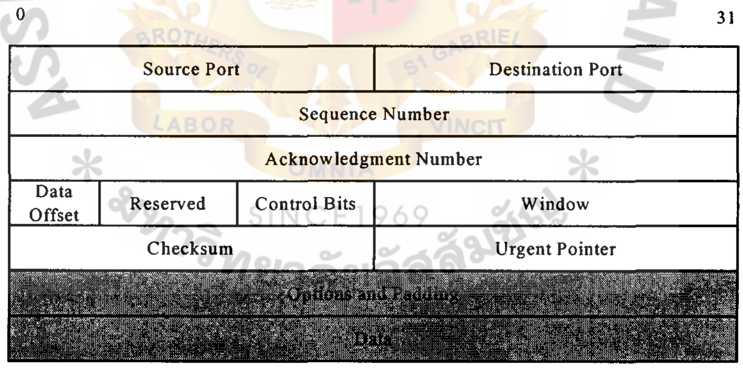

Figure 2.4 TCP Header format

Source Port (16 bits)

Source application port

Destination Port (16 bits)

Destination application port

Sequence Number 32 bits

The sequence number of the first data byte in this segment. If the SYN bit is set, the sequence number is the initial sequence number and the first data byte is initial sequence number + 1.

Acknowledgment Number 32 bits

If the ACK bit is set, this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

Data Offset (4 bits)

The number of 32-bit words in the TCP header. This indicates where the data begins. The length of the TCP header is always a multiple of 32 bits.

Reserved (6 bits)

Must be set to zero

Control Bits (6 bits)

There are six flags in the TCP header. One or more of them can be turned at the same time.

U, URG. 1 bit

Significant urgent pointer field

A, ACK. 1 bit

Significant acknowledgment field

P, PSH. 1 bit

Push Function field, the receiver should pass this data to the application as soon as possible.

R, RST. 1 bit

Reset connection field, reset the connection

S, SYN. 1 bit

Synchronization sequence number field; synchronize sequence numbers to initiate a connection.

F, FIN

End of data field, finish sending data.

Window (16 bits)

This is the number of data bytes beginning with the one indicated in the acknowledgment field that the sender of this segment is willing to accept.

Checksum. 16 bits

This is computed as the 16-bit one's complement of the one's complement sum of a pseudo header [pseudo header] of information from the IP header, the TCP header, and the data, padded as needed with zero bytes at the end to make a multiple of two bytes.

Urgent Pointer (16 bits)

This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set.

Options (variable length)

Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. The option that we are dealing in this work is maximum segment size option, called MSS. It indicates the maximum sized segment that the sender expects to receive.

2.5 User Datagram Protocol (UDP)

UDP is another transport layer protocol. UDP is a connectionless since there is no handshaking between sender and receiver. UDP takes messages from the application process, attaches source and destination port number fields for the multiplexing or demultiplexing service, adds two other small fields (header length and checksum), and passes the resulting segment to the network layer. The network layer encapsulates the segment into an IP datagram and then makes a best-effort attempt to deliver the segment to the receiving host. If the segment arrives at the receiving host, UDP uses the destination port number to deliver the segment's data to the correct application process.

Figure 2.5 shown below is a UDP header format

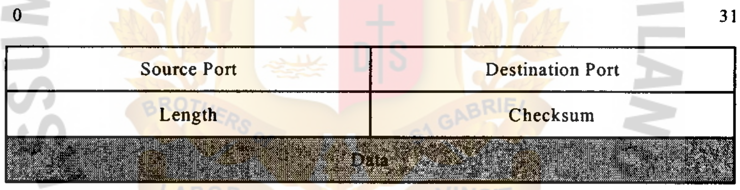

Figure 2.5 UDP Header format

Source port (16 bits)

Indicates the port of the sending process and it is used to be the port to which a reply should be addressed.

Destination port (16 bits)

Indicates the port of the receiving process, it is used to demultiplex incoming data from IP.

Length (16 bits)

Length field indicates the length of the UDP header and the UDP data.

Checksum (16 bits)

Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

DNS is one of the application-layer protocols that use UDP. When the DNS application in a host wants to make a query to find the IP address of the specific name, it constructs a DNS query message and passes the message to a UDP socket. Without performing any handshaking, UDP adds header fields to the message and passes the resulting segment to the network layer. The network layer encapsulates the UDP segment into a datagram and sends the datagram to a name server. The DNS application at the querying host then waits for a reply to its query. If it doesn't receive a reply (possibly because the underlying network lost the query or the reply), it either tries sending the query to another name server, or it informs the invoking application that it can't get a reply. We mention that the DNS specification permits DNS to run over TCP instead of UDP, in practice, however, DNS almost always runs over UDP.

CHAPTER 3. ARP SPOOFING

The information or data that going to send from any computers inside network must be coded in the form of Ethernet frame. ARP is the method of the Ethernet protocol to translate the IP address of the destination computer to the destination MAC address.

ARP operates by broadcasting ARP Request to all computers inside the network asking the question "Is this your IP address? If so, give me your MAC address". After receiving the ARP request, the destination computer will send an ARP Reply containing its MAC address.

To satisfy the objective of this thesis, we focus on the computer that has the wrong or mismatch IP configuration to the physical network. Mismatching of IP configuration can occur by means of assigning wrong IP configuration, using private IP, or relocating computer from different network. This computer wants to access to the Internet without changing IP configuration of the computer. In order to access to the Internet, this computer will first broadcast ARP Request to all computers inside this network asking for the MAC address of the gateway or DNS. Because the IP address of gateway or DNS could not be found in this network, some mechanisms have to map or translate this unusable ARP Request to the usable ARP Request. These mechanisms will be implemented on a computer named "virtual gateway". The first mechanism is to capture all ARP Request packets broadcasting in the LAN. By looking at the source and destination IP address in the captured packets. These packets can be categorized into two groups. The first group is the usable or correct packet containing the correct source and destination IP address. Correct source or destination IP number is the IP number located on the same network as the virtual gateway. Packets in this first group will be discarded by the virtual gateway because it can reach to the destination according to the destination IP address in the packet. The second group is the unusable ARP packet that contains the wrong information of source or destination IP address in the packet. The virtual gateway will send the ARP Reply pretending that the virtual gateway is a gateway or DNS of that computer. This process of pretending to be other machine is called ARP spoofing. Figure 3.1 shown below is the process of ARP spoofing.

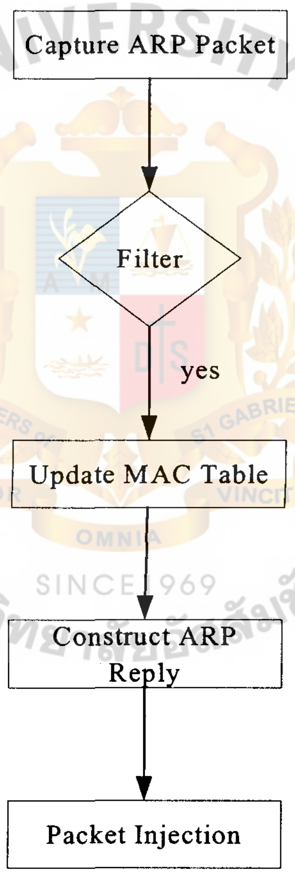

Figure 3.1 Flow chart of ARP Spoofing

This chapter will describe the concept and implementation by following the ARP spoofing process. Firstly is to capture all packets transmitting in the network. Then, is to filter packets by selecting the considered packet. After that is to update the MAC table. And lastly is to construct and inject modified packet back to the network.

To capture all Ethernet packets from the network, the network card has to be in promiscuous mode. Without being in the promiscuous mode, a network card will capture only the packets that match its own MAC address and the broadcast packet. After being in the promiscuous mode, the network card captures all Ethernet packets and forwards them for further processing, explaining later. The implementation has been done by the help of "libpcap".

Libpcap is publicly available as the open-source domain library for packet capturing process. To capture the Ethernet packets in the network, firstly we have to choose the device of the virtual gateway to capture packets. This device will be obtained by using function "pcap_lookupdev ()" and obtain device name "eth0". In order to capture all packets transmitting in the network, this device must be opened for packet capture and must be in promiscuous mode by using function "pcap_open_live ()". The step is to filter out the correct packet and passes the wrong packet to the next step. To distinguish between the correct and wrong packet we need to look at the Ethernet header and the ARP message of the captured packet to filter packet. The rules used to filter packets in ARP spoofing process are

1. Frame type in Ethernet header must be 0x0806, which is corresponding to ARP message.
2. Operation field in ARP message must be 0x01, which is corresponding to ARP Request.
3. Sender or destination protocol address in ARP message not equal to 168.120.18.X. This rule corresponds to the packets that generate from mismatch IP computer.

The following are the example of the packet that passes these rules is shown below.

Ethernet Header

Frame type	→	0x0806

ARP Message

Operation	→	0x01 (ARP Request)
Sender protocol address	→	not equal to 168.120.18
Destination protocol address	→	not equal to 168.120.18

Packets that pass these rules will be considered as an ARP Request packet generating from mismatch IP configuration computer. The above rules will pass the ARP Request packet that contain mismatch sender or destination protocol address to the function "send_arp ()" for processing. This mismatch or wrong configuration packet will be corrected by giving the ARP Reply telling that the virtual gateway is the computer that the ARP Request sender is looking for. The virtual gateway needed to remember that it self become the gateway for the sender, which is a mismatch IP computer. This makes the need of table mapping.

In function send_arp (), before processing this packet the information of this packet must be kept in the MAC table. MAC table is used to store the information of this packet. The information needed to store are sender MAC address, sender IP address and target IP address. This information is necessary in constructing packets and translating of TCP/IP and UDP/IP packets that going to describe in later chapter. Table 3.1 shows the example of MAC table.

Sender MAC address	Sender IP address	Target IP address
00:00:E8:64:C5:C6	192.168.0.5	192.168.0.1
00:10:4B:C7:18:1A	192.169.1.14	192.169.0.50

Table 3.1 Example of MAC table

The contents of the mismatch ARP Request is then reconstructed by changing

1. Destination MAC address in Ethernet header is source MAC address of the mismatch ARP Request.
2. Source MAC address in Ethernet header is the virtual gateway MAC address.
3. Operation code in ARP message is changed to 0x0002 to be ARP Reply.
4. Sender MAC address in ARP message is the virtual gateway MAC address.
5. Sender protocol address in ARP message is the target IP address of the mismatch ARP Request.
6. Target MAC address in ARP message is the sender MAC address of the mismatch ARP Request.
7. Target protocol address in ARP message is the sender IP address of the mismatch ARP Request.

To construct and inject the packet we use another open source library called "libnet". Libnet is a C library used to construct and inject packet to the network. However, libnet and libpacap are not the same, libnet do nothing about capturing process instead its construct the packet using buffer and write the packet that is stored in buffer to the network. The functions used to construct ARP Reply are "libnet_build_Ethernet ()" and "libnet_build_arp ()" functions. The values in the Ethernet header and ARP message in the constructing ARP Reply are as follow

Ethernet header:

Destination MAC address	→	MAC address field in MAC table
Source MAC address	→	Virtual gateway MAC address
Frame type	→	0x0806

ARP message:

Hardware type	→	0x0001
Protocol type	→	0x0800
Hardware size	→	0x06
Protocol size	→	0x04
Operation code	→	0x0002
Sender MAC address	→	Virtual gateway MAC address
Sender Protocol address	→	Target IP address field in MAC table
Target MAC address	→	MAC address field in MAC table
Target Protocol address	→	Sender IP address field in MAC table

Finally the reconstructed packet will be send back to the network by using function "libnet_write_link_layer ()". After successfully injected this packet to the network, wrong or mismatch IP computer will receive this packet and will map MAC address of the virtual gateway with the IP address of the gateway or DNS that is not actually located in this network.

The IP range in this network is 168.120.18.128 to 168.120.18.191. The virtual gateway has IP address of 168.120.18.155 with MAC address of 00:01:02:92:76:42. A wrong IP computer has IP address of 192.168.0.4 with 192.168.0.1 is a gateway IP address and MAC address is 00:00:C0:D5:6A:D1. Figure 3.2 shown below is the ARP Request packet information generates from wrong IP computer. All computers in this network ignore this packet because it is asking for the MAC address of IP address that is not located in this network. This packet is captured by the virtual gateway. Then the virtual gateway notices the sender protocol address and the target protocol address in ARP frame. The virtual gateway take a look at the 29th - 32nd byte and 39th -42nd byte which are sender protocol address and the target protocol address respectively. If the IP is not beginning with 168.120.18 means wrong IP computer sends out this packet. Here in Figure 3.2 sender protocol address is 192.168.0.4 and the target protocol address is 192.168.0.1. Figure 3.3 shows the ARP Reply packet that is injected by the virtual gateway.

```
----- General -----
Item number 1, position in logfile 9%
Timestamp: 16h:55m:10s:928925us
----- Description -----
Item type: Partial frame, 60 bytes available
Frame size is 60 (3C hex) bytes
----- MAC Header ----- [0-13]
Destination = Broadcast FFFFFF-FFFFFF - [0-5]
Source = Computer 0000C0-D56AD1 (Universal; Vendor: ???) - [6-11]
Ethertype = 0806h (???) - [12-13]
----- ARP frame ----- [14-41]
Hardware type 1 Ethernet - [14-15]
Protocol Type = 0800h IP - [16-17]
Length of hardware address = 6 bytes - [18-18]
Length of protocol address = 4 bytes - [19-19]
Operation code 1 Request - [20-21]
Sender's hardware address = 0000C0-D56AD1 (Vendor: ???) - [22-27]
Sender's protocol address = [192.168.0.4] - [28-31]
Target hardware address = 000000-000000 (Vendor: ???) - [32-37]
Target protocol address = [192.168.0.1] - [38-41]
----- Padding ----- [42-42]
Padding 0x0 - [42-59]
=================================================================
* FF FF FF FF | FF FF 00 00 | C0 D5 6A D1 | 08 06 00 01 [..........j.....]
* 08 00 06 04 | 00 01 00 00 | C0 D5 6A D1 | C0 A8 00 04 [..........j.....]
* 00 00 00 00 | 00 00 C0 A8 | 00 01 00 00 | 00 00 50 04 [..............P.]
* 00 00 0A 9C | 00 00 02 04 | 05 B4 01 01 |             [............]
```

Figure 3.2 ARP Request for wrong IP computer

```
----- General -----
Item number 2, position in logfile 17%
Timestamp: 16h:55m:10s:930161us
----- Description -----
Item type: Partial frame, 60 bytes available
Frame size is 60 (3C hex) bytes
----- MAC Header ----- [0-13]
Destination = Computer 0000C0-D56AD1 (Universal; Vendor: ???) - [0-5]
Source = Computer 000102-927642 (Universal; Vendor: ???) - [6-11]
Ethertype = 0806h (???) - [12-13]
----- ARP frame ----- [14-41]
Hardware type 1 Ethernet - [14-15]
Protocol Type = 0800h IP - [16-17]
Length of hardware address = 6 bytes - [18-18]
Length of protocol address = 4 bytes - [19-19]
Operation code 2 Reply - [20-21]
Sender's hardware address = 0001 2-927642 (Vendor: ???) - [22-27]
Sender's protocol address = [192.168.0.1] - [28-31]
Target hardware address = 0000C0-D56AD1 (Vendor: ???) - [32-37]
Target protocol address = [192.168.0.4] - [38-41]
----- Padding ----- [42-42]
Padding 0x0 - [42-59]
=====================================================================
* 00 00 C0 D5 | 6A D1 00 01 | 02 92 76 42 | 08 06 00 01 [....j......vB....]
* 08 00 06 04 | 00 02 00 01 | 02 92 76 42 | C0 A8 00 01 [..........vB....]
* 00 00 C0 D5 | 6A D1 C0 A8 | 00 04 00 00 | 00 00 00 00 [....j...........]
* 00 00 00 00 | 00 00 00 00 | 00 00 00 00 |             [............]
```

Figure 3.3 Faked ARP Reply Packet

After successfully injected ARP Reply the wrong IP computer will map 192.168.0.1 with 00:01:02:92:76:42. The virtual gateway has completed pretending to be the gateway that wrong IP computer is looking for. Therefore, every packet from wrong IP computer will be automatically sent to the virtual gateway.

CHAPTER 4. ADDRESS TRANSLATION

Internet Protocol (IP) is a connectionless protocol that gateways use to identify networks and paths to networks and hosts. IP handles the transmission of data from an originating computer to the computer specified by the IP address. Each time a message arrives at a router, the router decides where to send it next according to the destination IP address of that packet. In this work we consider the packet generated from wrong IP address that cannot be routed directly, because of the mis-configuration of the network parameters. Address translation of this packet is required in order to make this packet routable on the Internet.

The process of address translation is implemented on the virtual gateway. ARP spoofing technique that described in previous chapter have already pretended the virtual gateway as a gateway of the mismatch IP address computer. Therefore, every time when the mismatches IP address computer want to access to the Internet, the TCP/IP and UDP/IP packets generated from this computer will be automatically sent to the virtual gateway. The virtual gateway has to capture, modify and inject all TCP/IP and UDP/IP packets generated by the mismatch IP address computer. The virtual gateway also handles incoming packets from the Internet translate them and send back to that computer.

From the previous chapter, the virtual gateway captures the non-routable packets, modifies the information in the packet header then pushes it back. This chapter extends the translation process to the network layer and transport layer. The modification in the previous chapter redirects all packets from the mis-configured machine to the virtual gateway. The modification in both network layer and transport layer make these packets routable through the Internet.

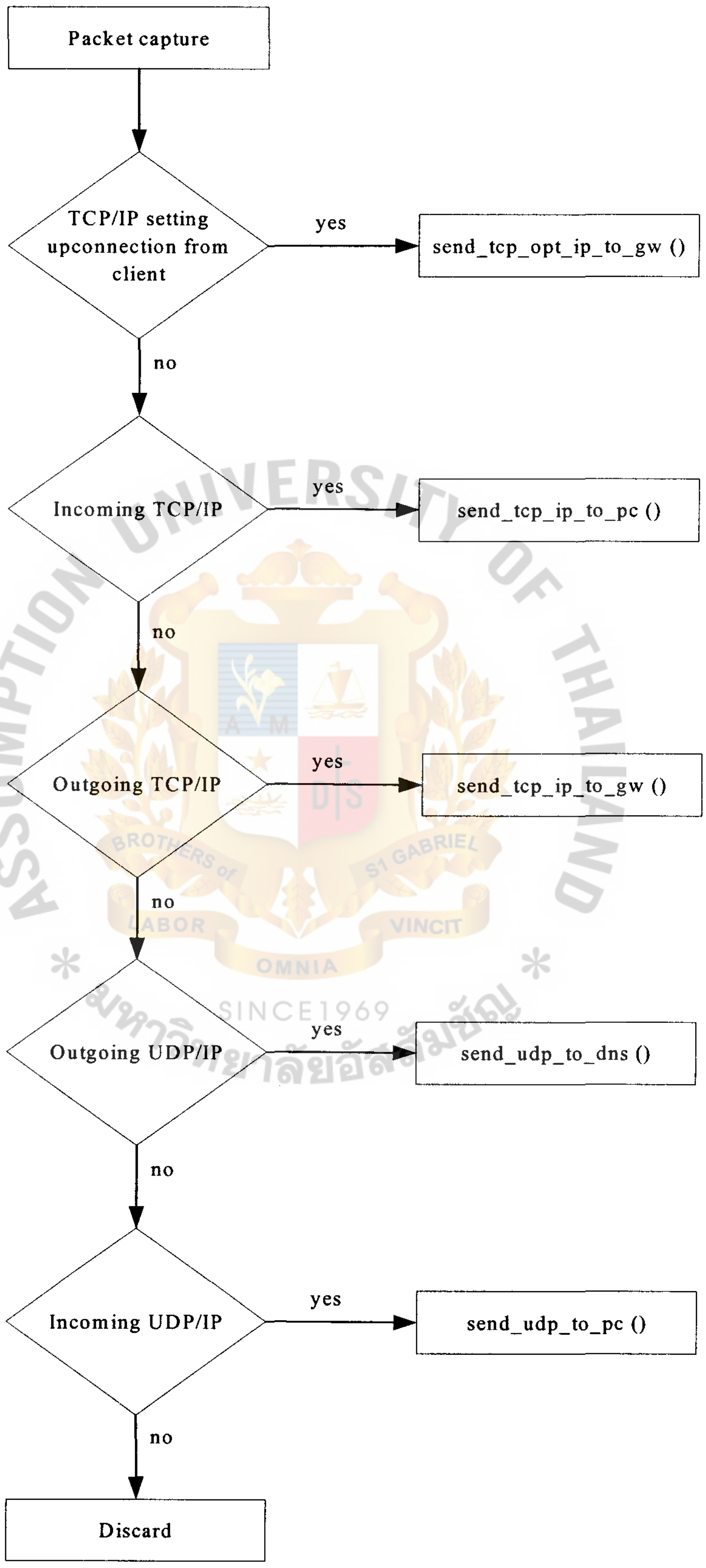

Figure 4.1 Flow chart of Address Translation in ANAA

The procedures of address translation for both type of packets, TCP/IP and UDP/IP, are the same by capturing, filtering, constructing and injecting packet. However both type of packets will be handled separately because TCP is connection oriented that require connection setup, but UDP does not. And the translation in this work mainly considered the TCP/IP connection, means we will translate all TCP/IP packets without considering the application port whereby UDP/IP packets are considered only application port 53 that is a DNS application. Figure 4.1 is a flow chart of packet translation.

4.1 TCP/IP Packet

TCP/IP packet requires one option for setting up connection. This option is a maximum segment size (MSS) option that going to be included in the TCP header of the first packet generating from the client to the server and the first packet that server replies to the client. To translate TCP/IP in this work we classified TCP/IP packet into three cases. The first case is the first packet from the mismatch IP computer to the server that is considered as a setting up connection packet. The second case describes all incoming packets from the server to the mismatch IP computer. The third case represents all the outgoing packets from client to the server except the first packet of that connection.

4.1.1 Setting Up Connection

We consider only the first packet from client, mismatch IP computer, to the server. This packet is considered as a setting up connection packet by looking at the TCP options that is included in the TCP header and acknowledgment. This option is the MSS option, which is used to negotiate the packet size between client and server and it is generated automatically by an operating system. The value in the TCP option that includes MSS option is 0x0204. However, when the TCP header includes option the size of the header will be changed. Therefore, the size of this packet from the IP header will be greater than 40 bytes.

The acknowledgment of this packet must equal to zero because it is the first packet of the connection. The acknowledgement of zero means no previous packet or data has to be synchronized. The contents in the packet that we have to consider for filtering are

1. Frame type in Ethernet header must be equal to 0x0800 for IP packet, and
2. Total length in IP header is greater than 40 bytes, and
3. Protocol type in IP header is 0x06 for TCP, and
4. Acknowledgement in TCP header is zero, and
5. Option is TCP header contains MSS option, 0x0204.

The following are the example of the filter rules

Ethernet header

Frame type	→	0x0800

IP header

Total length	→	greater than 40 bytes (greater than 0x0028)
Protocol	→	0x06

TCP header

Acknowledgement	→	0x00000000
Options	→	MSS option, 0x0204

After successfully filtered packet, this mismatch TCP/IP packet will be translated by the virtual gateway. The virtual gateway translates by reconstructing the packet and sends the modified packet back to the network. The function that used to process this mismatch TCP/IP packet is "send_tcp_opt_ip_to_gw ()", Figure 4.2. In this function, first it check the source MAC address of the packet that pass to the filter with the MAC table to confirm that it come from the computer that we have already spoofed. The sender MAC address field in the MAC table is used as a key. After that, function "search ()" is called for finding the available port and then put it into the source port of the TCP header after translation. This port number is used to demultiplex the incoming packets and then translate the incoming packets and send them back to the original mismatch IP computer, the virtual gateway need a table for mapping. This table is called IP table. The information of the mismatch TCP/IP packet that must be kept in this table are source MAC address, source IP address, destination IP address, source port, destination port and assigned port. Table 4.1 shows the example of the IP table.

Source MAC address	Source IP address	Destination IP address	Source port	Destination port	Assigned port	Timestamp
00:104B:C7:18:1A	192.168.0.4	202.6.101.255	1045	80	10001	1002155102
00:00:E8:64:C5:C6	192.168.0.5	202.6.101.2	1039	23	10002	1002155409

Table 4.1 Example of IP Table

Then, the contents of this TCP/IP packet is reconstructed by doing the following

1. Ethernet header is handled by the kernel of the OS in the virtual gateway.
2. Source IP address in IP header is changed to be the virtual gateway IP address.
3. Checksum in IP header, this field will be calculated by the kernel of the virtual gateway's OS.

4. Source port in TCP header to be available port of the virtual gateway by using function search (), and
5. Checksum in TCP header, calculated by using function "libnet_do_checksum ()".

The following are the example of the modified contents of the mismatch TCP/IP packet header

IP Header

Source IP	→	168.120.18.155
Checksum	→	Kernel will handle this field

TCP Header

Source port	→	search (), search available port
Checksum	→	libnet_do_checksum (), calculate checksum

To construct the TCP/IP packet we use function "libnet_build_ip ()", "libnet_build_tcp ()" and "libnet_insert_tcp ()". These functions will construct the packet in the form of buffer before it is injected to the network. Finally the reconstructing packet is sent back to the network by using function "libnet_write_ip ()". This function will pass this TCP/IP packet to the kernel to encapsulate this packet by the Ethernet header. Therefore, it is the duty of the kernel to add Ethernet header to this TCP/IP packet and also calculate the checksum of the IP header. In this case we do not have to go to the data link layer because the injecting packet is now considered as the correct packet that can be routable to the desire destination.

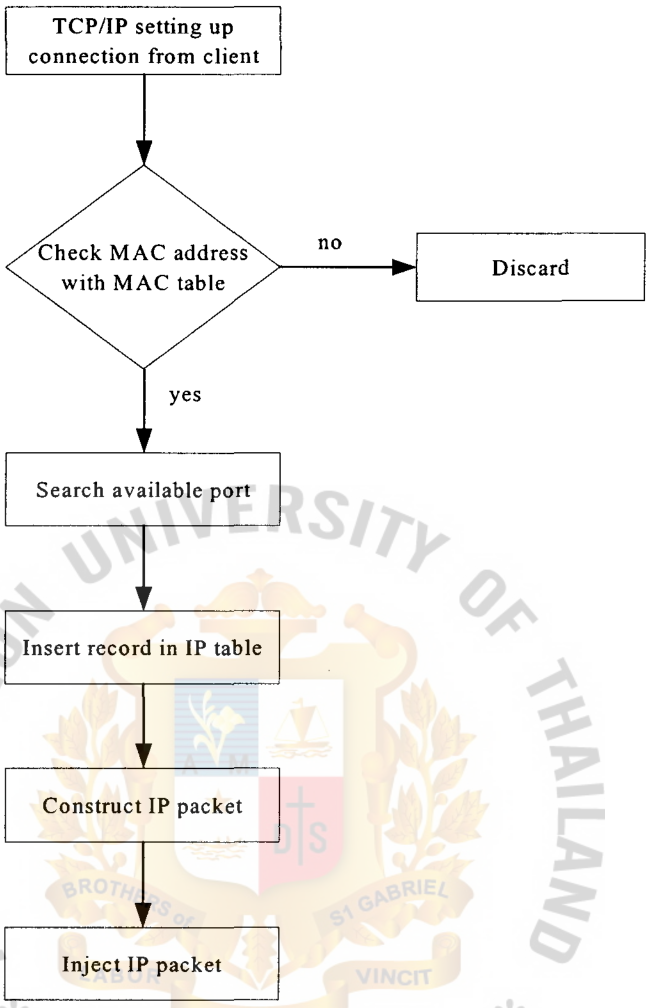
TCP/IP setting up connection from client
Check MAC address with MAC table
no
Discard
yes
Search available port
Insert record in IP table
Construct IP packet
Inject IP packet

Figure 4.2 Flow Chart of function send_tcp_opt_ip_to_gw ()

4.1.2 Incoming Packets

In this case we consider all incoming packets from the server to the virtual gateway. These packets have to be translated and send back the information from the server to the mismatch IP computer. According to the packet that we translated and injected in the first case, the reply from server of this packet will be sent to the virtual gateway automatically by the gateway. However, we have to block the ports of the virtual gateway to ignore this reply packet to go to the TCP/IP stack, otherwise the

TCP/IP stack will send the reset connection to the server. The reset signal will happen when the TCP/IP stack get the packet at the port it haven't setup.

The method to block port range that we are going to assign to setup connection packet in the virtual gateway is to use command "ipchains" [11]. Ipchains is used to set up, maintain, and inspect the firewall rules in the Linux kernel. The rules used to deny these incoming packets to the virtual gateway are

1. Incoming TCP packet with destination IP address is 168.120.18.155 and destination port between 10000 and 50000.
2. Incoming UDP packet with destination IP address is 168.120.18.155 and destination port between 10000 and 50000.

After blocking all incoming packets, these packets are still sent to the virtual gateway, but according to the ipchains rules, these packets will not be handled directly by the TCP/IP stack. Our network adapter is in promiscuous mode, therefore we can capture and translate these packets by first filtering out unwanted packets, modifying and injecting packet to the network.

The filtering rule in this case is to get the incoming packet that is the reply of the packet that we injected in the first case. To get this packet we have to look at the contents in the header of capturing packet. The contents in the packet that we have to consider for filtering are

1. Frame type in Ethernet header must be equal to0x0800 for IP packet, and
2. Protocol type in IP header is 0x06 for TCP, and
3. Destination IP address in IP header is 168.120.18.155, which is an IP address of the virtual gateway.

The following is the example of the filtering rules

Ethernet Header

Frame type	→	0x0800

IP Header

Protocol	→	0x06
Destination IP	→	168.120.18.155 (IP of the virtual gateway)

TCP header

Options	→	First two bytes equal to 0x0204

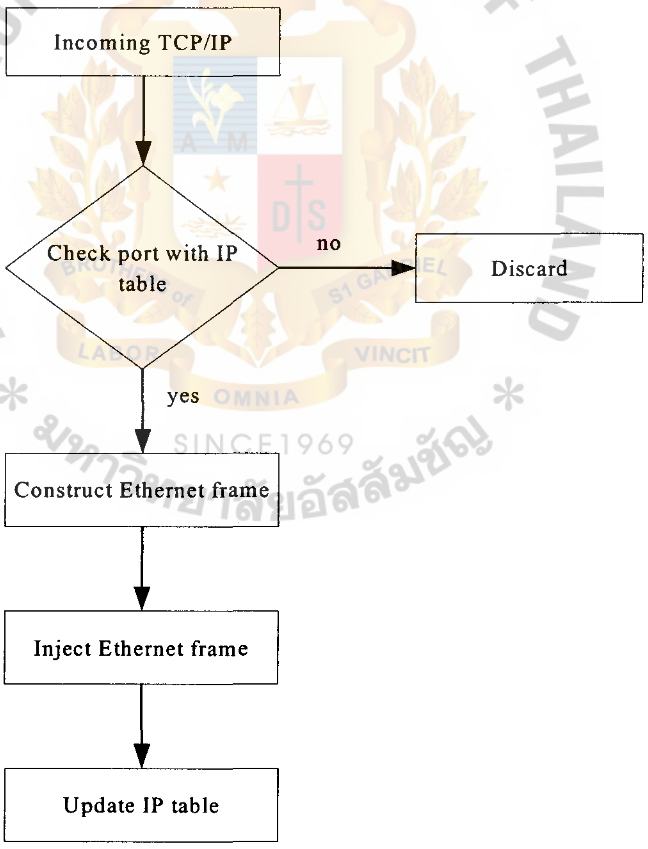

Figure 4.3 Flow Chart of function send_tcp_ip_to_pc ()

Then, this packet is passed to function "send_tcp_ip_to_pc ()". Figure 4.3 shows a flow chart of function "send_tcp_ip_to_pc ()". In this function, the incoming packet has to map with IP table to find the original destination MAC address, destination IP address and destination port. The key used to map with IP table is the destination port of the incoming packet that must be equal to the port assigned by the virtual gateway.

Then, the packet will be constructed according to the information in that record. The contents of the packet that have to modify are as follow:

1. Source MAC address in Ethernet header is changed to be the virtual gateway MAC address.
2. Destination MAC address in Ethernet header is changed to be the MAC address of original mismatch IP computer by taking the information from source MAC address field in IP table.
3. Destination IP address in IP header is changed to be the IP address of original mismatch IP computer by taking the information from source IP address field in IP table.
4. Checksum in IP header is recalculated by using function "libnet_do_checksum ()".
5. Destination port in TCP header is changed to be the port number of the original mismatch IP computer by taking the information from the source port field in IP table.
6. Checksum in TCP header is recalculated by using function "libnet_do_checksum ()".

The following is the example of the modifying contents in the packet

Ethernet Header

Destination MAC address	→	Mismatch IP computer MAC address
Source MAC address	→	Virtual gateway MAC address

IP Header

Destination IP address	→	Mismatch IP computer IP address
Checksum	→	Recalculated by "libnet_do_checksum ()"

TCP header

Destination port	→	Mismatch IP computer port
Checksum	→	Recalculated by "libnet_do_checksum ()"

In this case we have to construct the Ethernet frame of TCP/IP packet. Unlike the first case that we construct only TCP/IP packet and let the kernel handle the link layer because kernel does not recognize mismatch IP computer in the network. Therefore, in the function "send_tcp_opt_ip_to_pc ()" will construct Ethernet frame by using function "libnet_build_Ethernet ()", "libnet_build_ip ()", "libnet_build_tcp ()" and "libnet_insert_tcpo ()". Then, the constructing Ethernet frame is injected to the network using function "libnet_write_link_layer ()". After successfully injected packet to the network, IP table has to be updated in the time field.

4.1.3 Outgoing Packets

In this case we consider outgoing packets that generated from mismatch IP computer. This case does not include the first packet from the mismatch IP computer to server because it is handled by the first case. We can distinguish this kind of packet by looking at the acknowledgement in the TCP header not equal to zero that used to synchronize, check with IP table to make sure that the connection has already been served. The method of translating this packet is similar to the previous case by filtering, modifying and injecting packet.

To filter out unwanted packet we have to look at the contents in the packet header. The contents that we have to consider for filtering are

1. Frame type in Ethernet header must be equal to 0x0800.
2. Protocol in IP header is 0x06 for TCP packet.
3. Acknowledgement in TCP header is not equal to zero

The following is the example of the packet header that used to filter

Ethernet Header

Frame type	→	0x0800

IP Header

Protocol	→	0x06

TCP Header

Acknowledgement	→	not equal to zero

After filtering, this packet will be passed to function "send_tcp_ip_to_gw ()". Figure 4.4 shows procedure of function "send_tcp_ip_to_gw ()". In this function, it first map the source IP address in the IP header and source port number in the TCP header with the source IP address field and source port field in the IP table respectively to make sure that this connection has already been served. If the mapping fail it means that the connection has not been served and the translation of this packet will not be done. If the connection has already been served there must be a record in the IP table and used these two fields as a key to find assigned port number that we use in this connection.

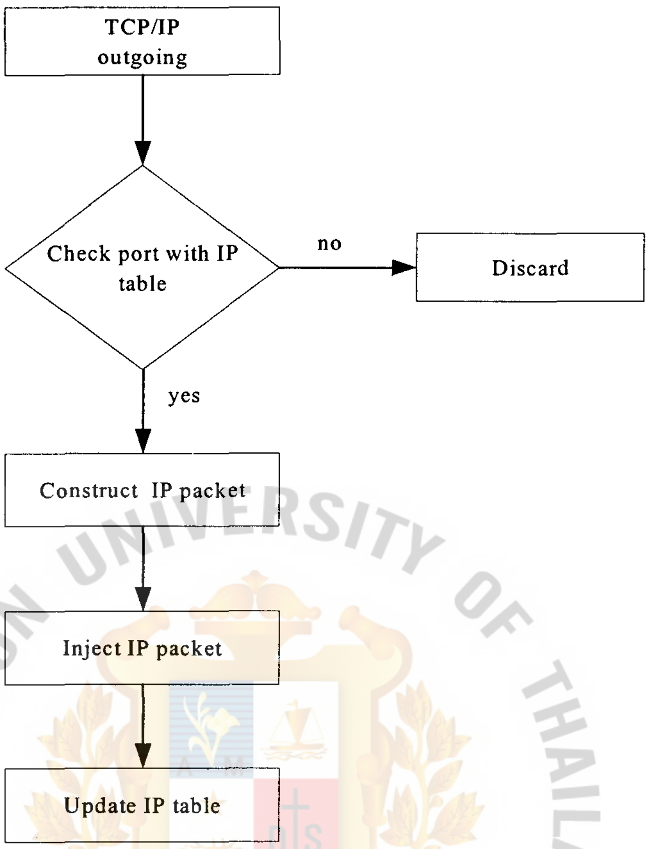

Figure 4.4 Flow chart of function send_tcp_ip_to_gw ()

Then, this packet will be reconstructed by modifying the contents in the packet header. The contents that will be modified are as following:

1. Ethernet header, will be handled by the kernel of the virtual gateway.
2. Source IP address in IP header is changed to be the virtual gateway IP address.
3. Checksum in IP header, this field will be calculated by the kernel of the virtual gateway.
4. Source port in TCP header is changed to be assigned port by using the value in assigned port field in IP table.
5. Checksum in TCP header, calculated by using function "libnet_do_checksum ()".

The following are the example of the modified contents of the mismatch TCP/IP packet header

IP Header

Source IP	→	168.120.18.155
Checksum	→	Kernel will handle this field

TCP Header

Source port	→	Assigned port field in IP table
Checksum	→	Recalculated by using "libnet_do_checksum ()"

Then, the new reconstructing packet will be injected to the network by using "libnet_write_ip ()". Similar to the first case that the kernel can handle data link layer because the packet is now the correct packet that can be routable to the server or destination. Finally the time field in the IP table has to be updated to know the last use of this connection.

4.2 UDP/IP Packet

The TCP/IP protocols within the kernel know nothing about the DNS. DNS is one of the application-layer protocols that use UDP or TCP to convert a hostname to an IP address before opening a connection. The port number used for DNS name server is UDP port 53 or TCP port 53. This implies that the DNS supports both UDP and TCP. However, to guarantee the DNS operation in different platform this work covers the case of UDP/IP with application port 53. To translate the UDP/IP packet we categorize into two sections, outgoing packet and incoming packet. In both section method of translating are the same by first filtering, modifying and injecting packet.

4.2.1 Outgoing Packet

This packet is the packet that generates from the mismatch IP computer to the name server to resolve the name to an IP address. This packet contains the destination IP address that is an IP address of the name server that cannot be founded in this network by putting wrong information or relocating of computer. This packet will be filtered by looking at the contents of the header. The contents in the header that used to filter are the following.

1. Frame type in Ethernet header must be equal to 0x0800 for IP packet.
2. Protocol type in IP header is 0x11 or 17 in decimal that correspond to the UDP type.
3. Destination port in UDP header is 0x0035 or 53 in decimal that correspond to the DNS application.

The following is the example of the packet header that used to filter

Ethernet Header		
Frame type	→	0x0800
IP Header		
Protocol	→	0x11 (17 in decimal)
UDP Header		
Destination port	→	0x0035 (53 in decimal)

After filtering, this packet will be passed to function "send_udp_to_dns ()". Figure 4.5 shows the procedure of function "send_udp_to_dns ()". In this function, first it checks source MAC address of this frame with the source MAC address field in the MAC table to make sure it came from the mismatch IP computer that we have spoofed an ARP reply.

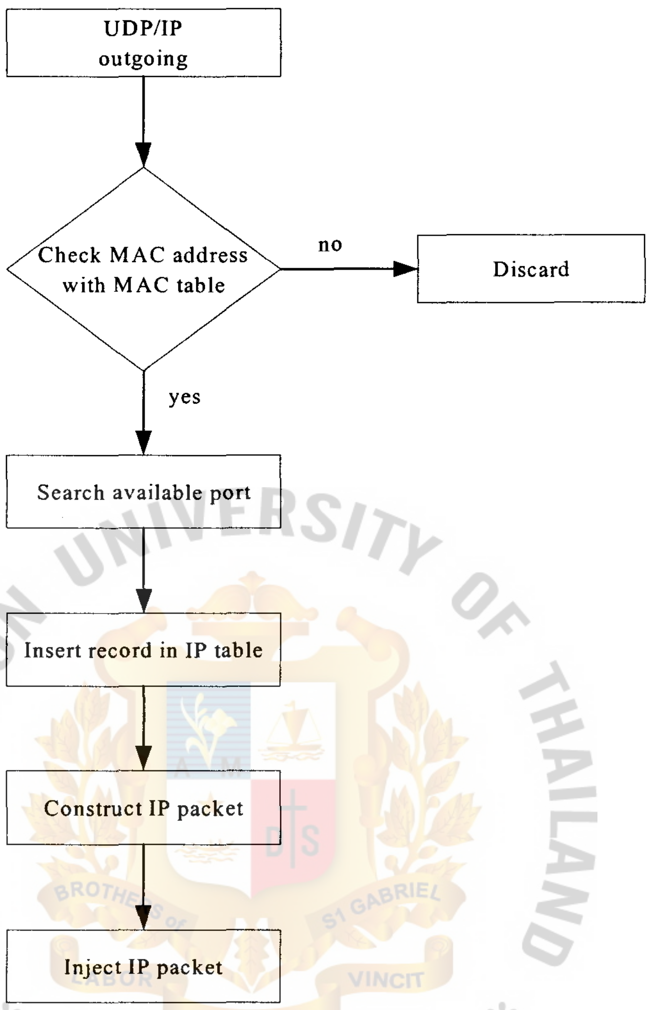

Figure 4.5 Flow chart of function send_udp_to_dns ()

Then, this packet is considered as the new connection therefore we have to search an available port of the virtual gateway to find a new source port of a new constructing packet and insert the information of this packet in IP table as described in previous section. After that the mismatch packet has to be reconstructed by modifying the contents in the header. The contents in the header has to be modified are as follow:

1. Ethernet header, will be handled by the kernel of the virtual gateway.
2. Source IP address in IP header is changed to be the virtual gateway IP address.
3. Destination IP address in IP header is changed to be the DNS IP address of this physical network or the DNS IP address of the virtual gateway.
4. Checksum in IP header, this field will be calculated by the kernel of the virtual gateway.
5. Source port in UDP header is changed to be assigned port by using the value in assigned port field in IP table.
6. Checksum in UDP header is recalculated by using function "libnet_do_checksum ()".

The following are the example of the modified contents of the mismatch UDP/IP packet header

IP Header

Source IP	→	168.120.18.155 (Virtual gateway IP address)
Destination IP	→	168.120.18.23 (DNS IP address)
Checksum	→	Kernel will handle this field

TCP Header

Source port	→	search (), search available port
Checksum	→	libnet_do_checksum (), calculate checksum

To construct the new UDP/IP packet in this case we use function "libnet_build_ip ()", "libnet_build_udp ()" and "libnet_build_dns ()". Then, this reconstructing packet is injected to the network by using function "libnet_write_ip ()". Similar to previous section that let the kernel handles link layer process. After putting this packet back to the network, this packet is now a correct UDP/IP packet and it will be routed to the name server according to the destination IP address that we assigned in IP header.

4.2.2 Incoming Packet

This incoming packet is the reply from the name server that contains the DNS message. This packet is routed to the virtual gateway automatically because this packet contains correct routing information. However, this packet cannot route to the mismatch IP computer without translating. Before translating packet we have to filter to get the incoming packet that is a reply to the mismatch IP computer. The contents in the packet header that used to filter are

1. Frame type in Ethernet header must be equal to0x0800 for IP packet.
2. Protocol type in IP header is 0x11 or 17 in decimal for UDP.
3. Source port is 0x0035 or 53 in decimal that correspond to the reply of the name server.

The following is the example of the filtering rules

Ethernet Header		
Frame type	→	0x0800
IP Header		
Protocol	→	0x11 (17 in decimal)
UDP Header		
Source port	→	0x0035 (53 in decimal)

After filtering this packet will be passed to function "send_udp_to_pc ()". Figure 4.6 shows the flow chart of function "send_udp_to_pc ()". In this function, first map the destination port of this packet with the assigned port field in the MAC table to make sure that this connection has already been served by the virtual gateway. If the mapping fail this packet will be discarded.

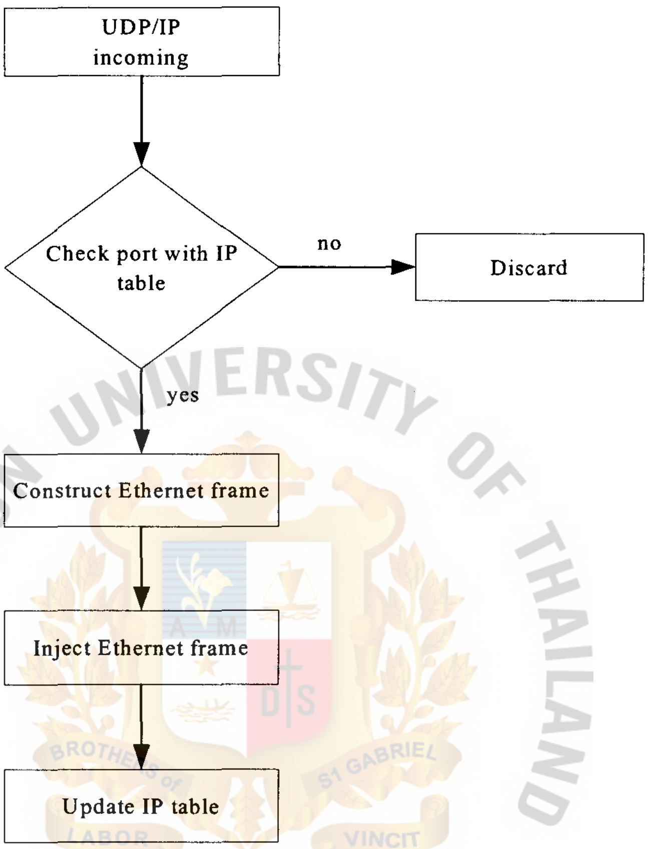

Figure 4.6 Flow chart of function send_udp_to_pc ()

Then, the packet will be reconstructed according to the information of that record in IP table. The contents of the packet that have to modify are

1. Source MAC address in Ethernet header is changed to be the virtual gateway MAC address.
2. Destination MAC address in Ethernet header is changed to be the MAC address of original mismatch IP computer by taking the information from source MAC address field in IP table.

3. Destination IP address in IP header is changed to be the IP address of original mismatch IP computer by taking the information from source IP address field in IP table.
4. Checksum in IP header is recalculated by using function "libnet_do_checksum ()".
5. Destination port in UDP header is changed to be the port number of the original mismatch IP computer by taking the information from the source port field in IP table.
6. Checksum in UDP header is recalculated by using function "libnet_do_checksum ()".

The following is the example of the modifying contents in the packet

Ethernet Header

Destination MAC address	→	Mismatch IP computer MAC address
Source MAC address	→	Virtual gateway MAC address

IP Header

Destination IP address	→	Mismatch IP computer IP address
Checksum	→	Recalculated by "libnet_do_checksum ()"

UDP header

Destination port	→	Mismatch IP computer port
Checksum	→	Recalculated by "libnet_do_checksum ()"

Similar to 4.1.2 of TCP/IP packet that we have to construct entire Ethernet frame because the kernel does not recognize mismatch IP computer. The construction of the Ethernet header and the IP header in this case are also similar by using function "libnet_build_ethernet ()" and "libnet_build_ip ()". But the transport layer header and the contents are not the same; therefore we use function libnet_build_udp ()" to build UDP or transport layer header and "libnet_build_dns ()" to build the DNS message.

Finally, is to inject the new reconstructing packet back to the network to send it to mismatch IP computer. The function that used to inject this packet is "libnet_write_link_layer ()".

CHAPTER 5. TESTING

This chapter provides the testing of the implementation of the ANAA. The purpose of this test is to verify that the computer that contains mismatch IP configuration can access to the Internet without changing or modifying IP configuration and to measure the performance of the network. The network environment of this test is shown in Figure 5.1. In this test, we separate the test into two sections the first section is the test of the applications over ANAA and the last section is the performance testing by comparing mismatch IP configuration computer with correct IP configuration computer.

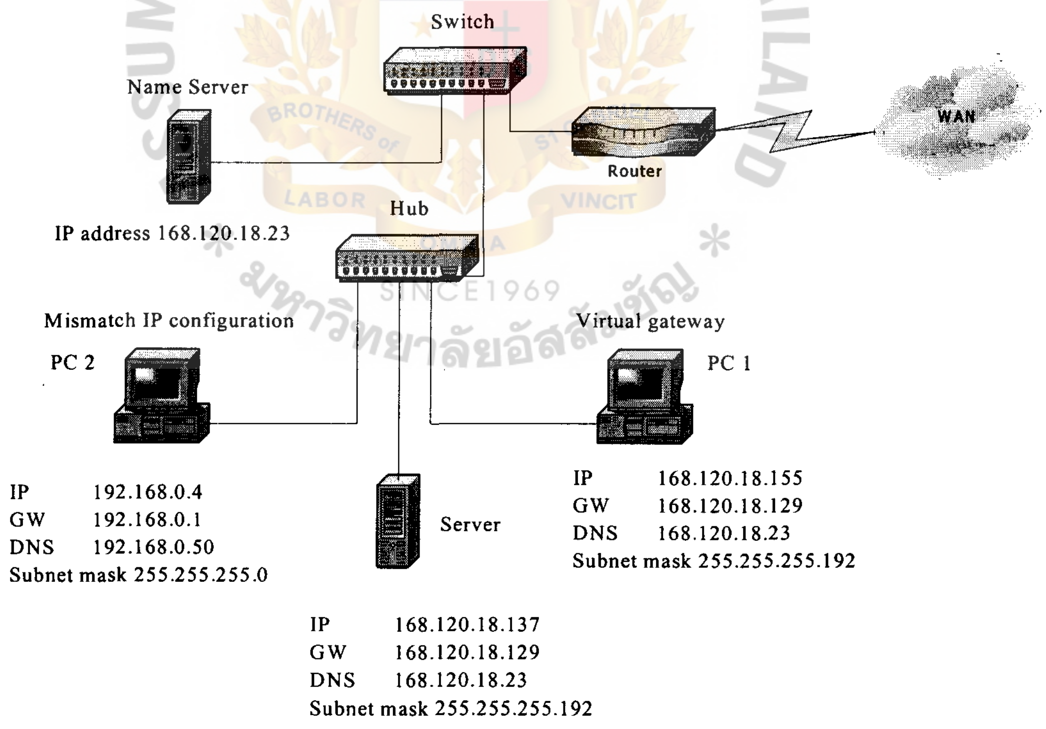

Figure 5.1 ANAA Network Configurations

5.1 Applications over ANAA

There are a lot of applications that used to access or communicate over the Internet. Most of these applications based on TCP/IP or UDP/IP to transfer the information from one computer to another computer. This information is wrapped in the form of packet and this packet will be routed correctly in the Internet depends on the information in the packet header such as source and destination IP address.

This thesis modifies the information in the packet header of mismatch IP computer to make it routable in the Internet. However, in modifying the packet header we do not modify or change the information in the data of the packet. This cause some applications such as File Transfer Protocol (FTP) cannot access to the Internet when using with ANAA because these applications add some header information into the data section of the packet. This is a limitation of this work. Another limitation of ANAA is that it supports only TCP/IP packet and UDP/IP packet with application port 53 therefore application that uses UDP/IP is not possible except DNS application. Table 5.1 shows the supporting applications of ANAA that has been tested.

Application	TCP/IP	UDP/IP	Port
Web browser	Yes	-	80
Telnet	Yes	-	23
SMTP	Yes	-	25
POP3	Yes	-	110
MSN	Yes	-	1863
ICQ	Yes	-	5190
DNS	-	Yes	53

Table 5.1 Applications over ANAA

5.2 Performance Comparison

We measure the performance of ANAA by using Webstone benchmark [12] test. Webstone creates load on a Web server by simulating the activity of multiple clients, which are called Web clients and which can be thought of as users, Web browsers, or other software that retrieves files from a Web server.

The purpose of this benchmark is to compare performance of the computer that has correct IP configuration with mismatch IP configuration computer that implement ANAA in 10Mbits/sec Ethernet network and 100Mbits/sec Fast Ethernet network. Three parameters that we measure are Server connection rate, Average response time and Throughput. Figure 5.2 shows the network configurations of correct IP configuration.

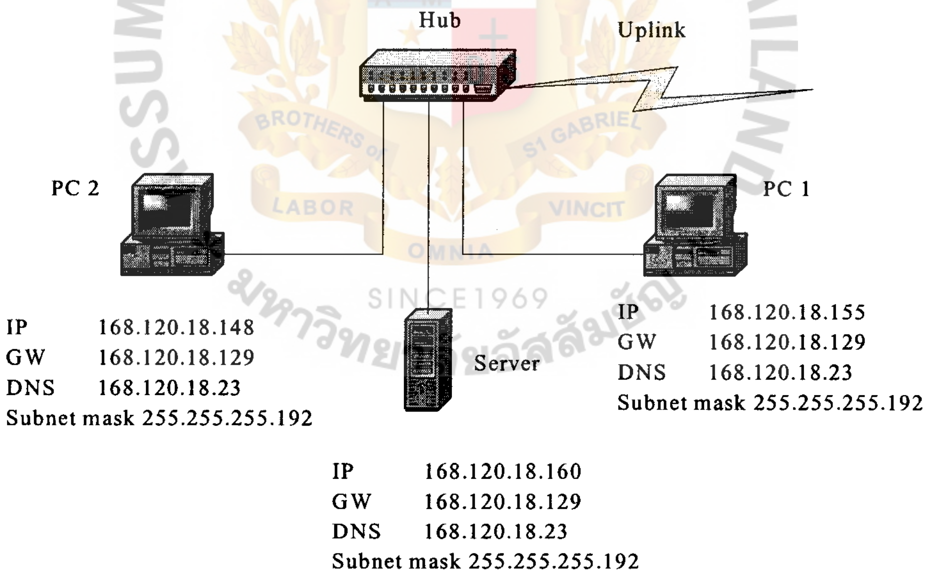

Figure 5.2 Network configuration of correct IP

The specifications of computers in this test are

PC1 CPU Intel Celeron 703 MHz, RAM 64 Mbytes, OS Linux Kernel 2.2-16

PC2 CPU Intel Celeron 703 MHz, RAM 64 Mbytes, OS Linux Kernel 2.2-16

Server CPU Dual Processors Intel P III, RAM 256 Mbytes, OS Linux Kernel 2.4

In this work PC 2 is running webstone trying to retrieve test files from Server 168.120.18.160, running as a Web server, by using HTTP protocol. There are 5 files to retrieve by webstone. Each of them has size of 500, 5K, 50K, 500K and 5M bytes. However, each file has different weight to be hit by the client. File 500 bytes is weight 350 hits out of 1000, 5Kbytes is 500 hits out of 1000, 50Kbytes is 140 hits out of 1000, 500Kbytes is 9 hits out of 1000 and 5Mbytes is 1 hit out of 1000. This simulation is to simulate at the number of 20, 40, 60 and 80 clients. Then we change the IP configuration of PC 2 to be 192.168.0.4 with gateway 192.168.0.1, and PC 1 acts as a Virtual gateway as shown in Figure 5.3. And run webstone in order to retrieve the same five files from the same Server. The output results of simulation under 10Mbits hub is shown in table 5.2

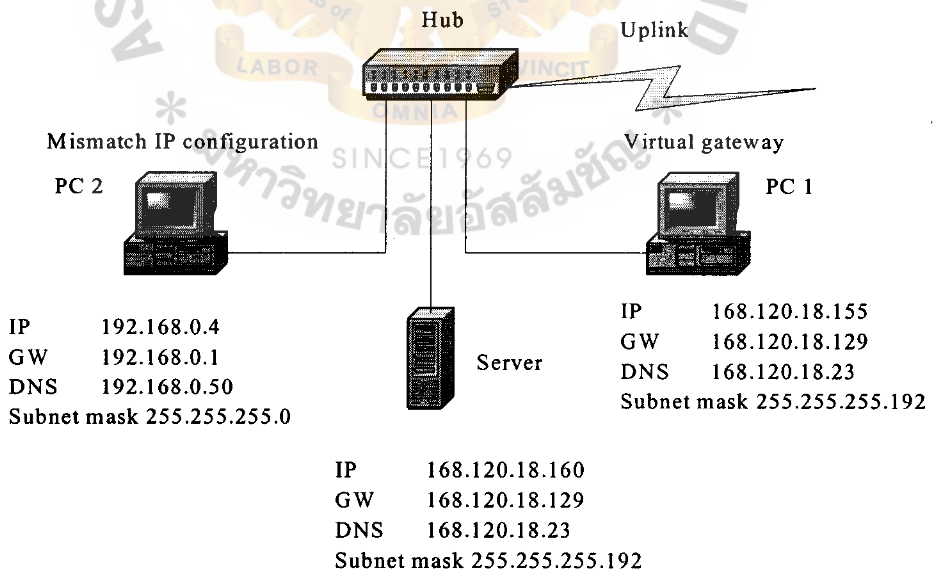

Figure 5.3 ANAA Network configurations

Number of Clients	Server Connection Rate		Average Response Time		Throughput	
	Correct IP	ANAA	Correct IP	ANAA	Correct IP	ANAA
20	50.53	24.92	0.382	0.727	6.08	3.03
40	50.57	23.92	0.708	1.349	6.02	2.59
60	53.77	24.87	0.988	1.788	6.26	2.69
80	58.53	22.68	1.173	2.389	6	2.72

Table 5.2 Simulation Result of 10 Mbit/sec Network

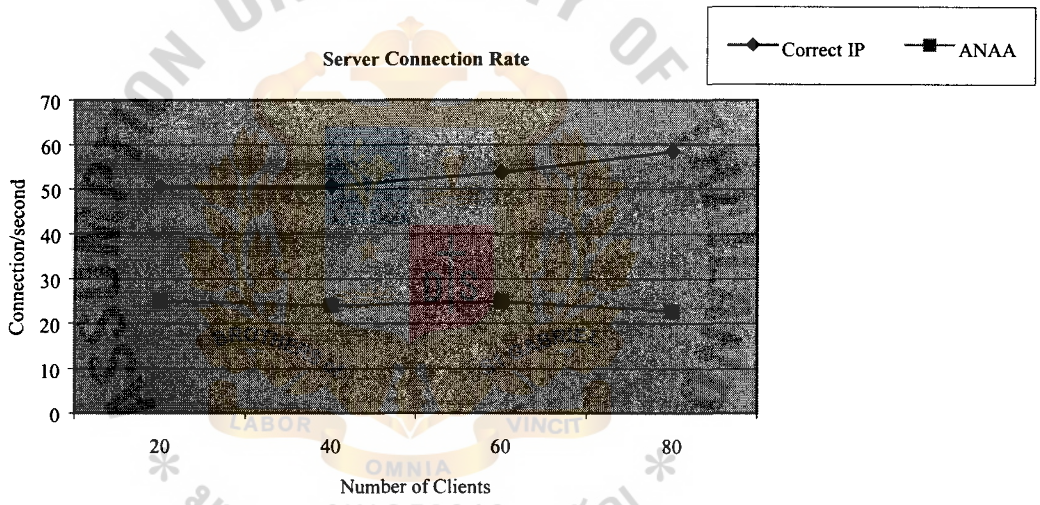

Figure 5.4 Graphs of Correct IP Vs ANAA (Server Connection Rate) 10Mbit/sec

Figure 5.4 shows the result of the rate of the connection that the Server response in both cases. We can notice that the implementation of ANAA reduces the rate of the connection that can be made between client and server.

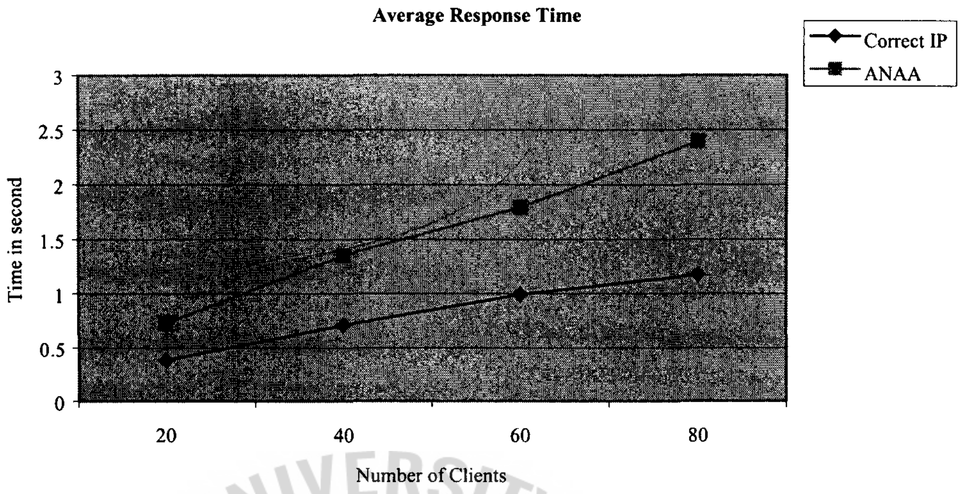

Figure 5.5 Graphs of Correct IP Vs ANAA (Average Response Time) 10Mbit/sec

Figure 5.5 shows average response time. Average response time is the total amount of response time divided by the total number of successful connections. This can also be called as Latency. Total response time includes the time to connect, the time to process the request on the server, and the time to transmit the response back to the client. The time requires to process the request of higher number of clients is increase when the number of clients increases because the server has to process all the requests from every client. In the case of ANAA the average response time is longer because the packets from the clients have to transmit to the virtual gateway first, then virtual gateway processes those packets by modifying information in the header and transmits to the server. After the server processes the requested packets and transmits back to the virtual gateway, virtual gateway takes another processing time in order to modify packet to send to the original client.

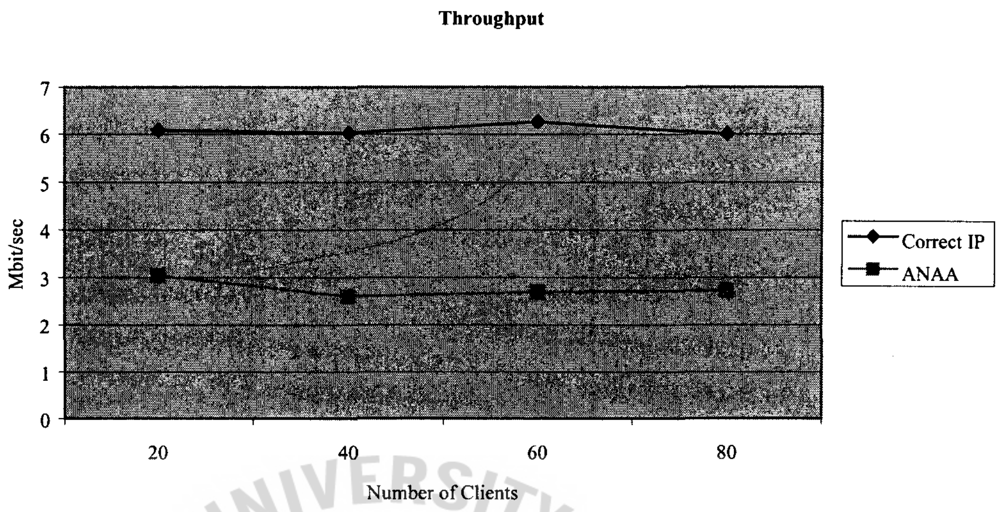

Figure 5.6 Graphs of Correct IP Vs ANAA (Throughput) 10Mbit/sec

Throughput is the total number of bits of data received from the server expressed in megabits per second. This data is considered as the data rate of successful connections. Figure 5.6 shows the throughput of both cases. In the case of correct IP the throughput is 6 Mbits/sec this mean that the network utilization can achieve up to 60%. However, in the case of ANAA the throughput is low because of the collision of the packets, since we implement ANAA over one network card that used to receive and transmit at the same time. Another reason for the slower throughput is that the virtual gateway has a limitation in handling the limited number of connections, as shown in figure 5.4. Therefore if the number of connection is reduced, the utilization of the network is also reduced.

In this work we also compare the result of correct IP and ANAA in the 100Mbit/sec environment by replacing hub in Figure 5.2 and Figure 5.3 to be a 100Mbit/sec hub. The specifications of all computers are the same as in 10Mbit/sec environment. Table 5.3 shows the simulation result of the 100Mbit/sec.

Number of Clients	Server Connection Rate		Average Response Time		Throughput	
	Correct IP	ANAA	Correct IP	ANAA	Correct IP	ANAA
20	427.87	54.6	0.047	0.354	70.45	7.29
40	441.97	48.57	0.089	0.788	70.79	8.1
60	445.48	47.25	0.133	1.115	70.18	7.93
80	461.75	52.42	0.17	1.132	72.6	7.37

Table 5.3 Simulation Result of 100 Mbit/sec Network

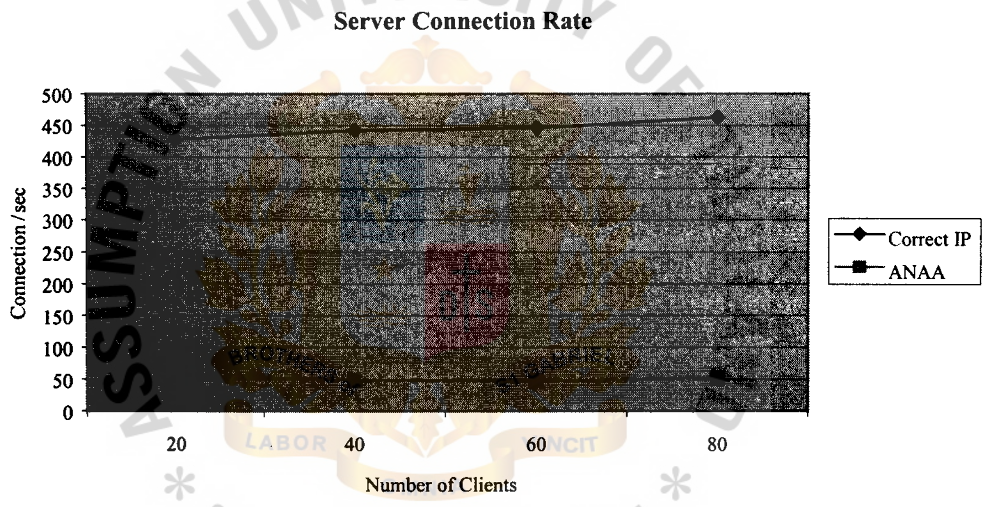

Figure 5.7 Graphs of Correct IP Vs ANAA (Server Connection Rate) 100Mbit/sec

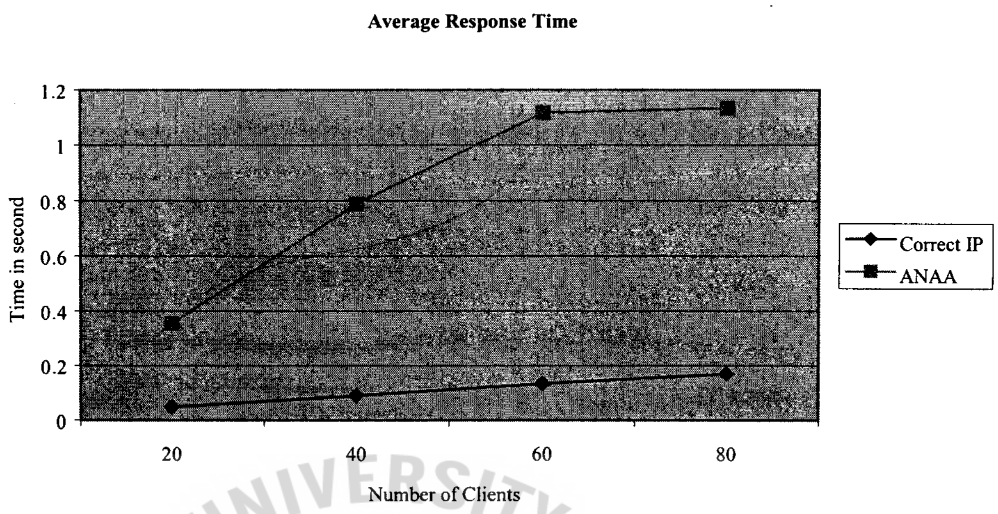

Figure 5.8 Graphs of Correct IP Vs ANAA (Average Response Time) 100Mbit/sec

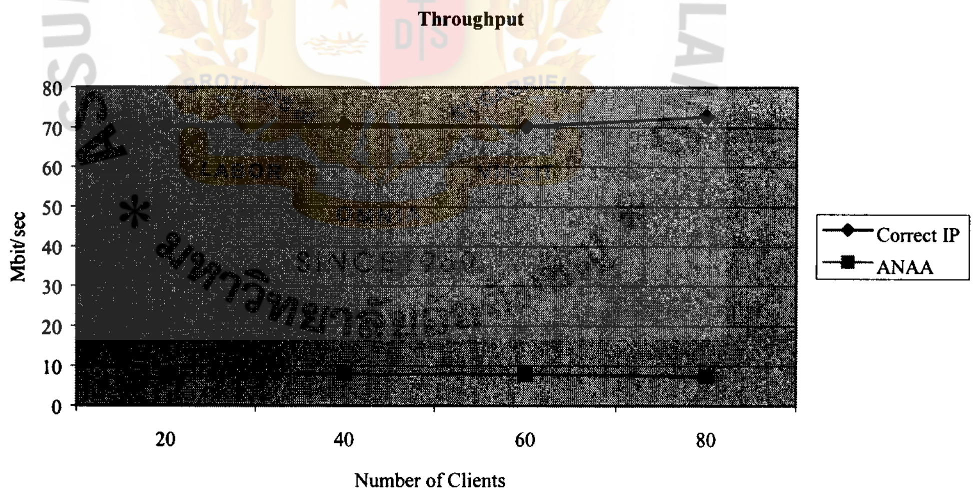

Figure 5.9 Graphs of Correct IP Vs ANAA (Throughput) 100Mbit/sec

According to the results of 100Mbit/sec environment we can see that ANAA can solve the problem of mismatch IP address. However, the performance of the system when implementing ANAA over 100Mbit/sec is very poor. The utilization of the network can achieve up to only 8.1% at 40 clients. In case of correct IP, the utilization of the network can achieve more than 70%. The main factor that effect to overall performance in 100Mbit/sec is that the virtual gateway has a limited capability in handling a large number of packets. By improving the virtual gateway hardware, the performance is expected to be increased.

According to the results we can conclude that the performance of the mismatch IP configuration computer in order to access to the Internet is reduced because the limitation of the virtual gateway. The first limitation is that we used only one network card to capture and inject packets at the same time. There might be the case of packets collision. The second limitation is the limited capability in handling a large number of packets as shown in Figure 5.5 and 5.8 that the virtual gateway takes longer time in translating packet when there are a large number of packets.

CHAPTER 6. CONCLUSIONS

This thesis presents the study and implementation of ANAA technique. ANAA allows computer that contains wrong or mismatch IP configuration to be able to access the Internet without any modification to the network configuration of client computer. This technique can be used for three main purposes. First is to allow the use of non-routable IP address in the internal network to be able to access the Internet. According to the limited amount of IP addresses, non-routable IP addresses are widely used to save cost and to expand the size of the network without requesting new unique IP address.

Second is to automatically take care of the mobile computer when it changes the network location. The relocation of this mobile computer can cause the mismatch of IP configuration in the new network that is not possible to access to the Internet or access to the server in the same network. Mismatch computer occur when the mobile computer use static IP configuration or DHCP server does not have enough IP addresses available to new coming computer. This helps user and network administrator not to reconfigure every time.

Third, this technique helps the poor configured computer still be used on the network. Poor configuration in this work means wrong IP address, wrong gateway address or wrong DNS address. If one of these addresses is configured incorrectly, accessing to the Server in the network or to the Internet may have problem.

The implementation of ANAA in this work is the software implementation done over normal computer connected in the Ethernet network with Linux as an Operating System. This computer is called the virtual gateway. The virtual gateway starts the process by spoofing the ARP packet. ARP spoofing is a method to make the mismatch IP computer think that the virtual gateway is a real gateway of this network. This process is done by sending ARP Reply to the mismatch IP computer.

ANAA translates packet from the mismatch IP computer by capturing the packet from the Ethernet network, verifying the packet information, correcting (if needed) the wrong information and then putting it back to the network. The type of the packet that going to be translated are TCP/IP packet and UDP/IP packet with application port 53.

Additionally, this technique reduces the performance in access to the server or Internet because of three factors. The first factor is it doubles the traffic between mismatch IP computer and the gateway in the local area network. Increasing the traffic in local area network can cause the denial of service in the network. The second factor is the collision of the packets over the Ethernet adapter of the virtual gateway. This causes the retransmission of the packet. The last factor is the processing time required modifying the information of the packet and the time requires transmitting from the client to the virtual gateway and from the virtual gateway to server back and forth.

6.1 Future Work

According to the translation of the packet by modifying the contents in the packet header without considering data in the packet cause some application that embeds addressing information in the packet payload, e.g. FTP cannot access to the Internet under ANAA because this application add information of the packet header to the data section. In order to solve this problem we can extend this work by dealing with data section of the packets. However, modifying the contents in the data of the packet will change the value of the size of the packet in the header and the value of SYN and

ACK of the TCP header also. Therefore, to deal with the data of packets the information of the size of data is important.

Another limitation of ANAA is that it supports only TCP/IP packet and UDP/IP packet with application port 53 therefore application that uses UDP/IP is not possible except DNS application. ANAA can further be developed to support multimedia applications that use other protocols. Types of the protocols that can be further developing over ANAA are UDP, ICMP, and IGMP. However, when dealing with other protocols or applications one important thing that needed to be concerned is time-out of the connection. Time-out indicates the maximum period of time that the connection stays idle. This time has to be vary depend on the protocol and application.

To increase the performance of ANAA we can improve by first increase the number of network adapter of the virtual gateway to reduce the collision of the packet. One network card can use for capturing packet and another one can use for injecting packet to the network. Second is improving the hardware specification of the virtual gateway. And lastly we can modify the algorithm in storing, retrieving and editing the value in the mapping table. One possible solution is to use hash table.

BIBLIOGRAPHY

[1] P. Srisuresh, M. Holdrege, "The IP Network Address Translation (NAT)", Internet RFC 2663, August 1999

[2] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear, "Address Allocation for Private Internet", Internet RFC 1918, February 1996

[3] Stevens, W. R., "TCP/IP Illustrated, Volume I: The Protocols", Addison-Wesley, 1994

[4] C. Zaccone, Y. T'Jones, B. Sales, "Address reuse in the Internet, adjourning or suspending the adoption of IP next generation", Proceeding of IEEE International Conference on Networks (ICON'00), 2000, Page(s): 462 - 468

[5] M.V. Loukola, J.O. Skytta, "New Possibilities Offered by IPv6", Computer Communication and Networks, 1998. Proceeding 7th International Conference on Published: 1998, Page(s): 548 - 552

[6] Dewayne Lamont Herbert, S.S. Devgan, C. Beane, "Application of Network Address Translation in Local Area Network", Southern Symposium on System Theory, 2001. Proceeding of the 33rd Published: 2001, Page(s): 315 - 318

[7] R. Droms, "Dynamic Host Configuration Protocol", Internet RFC 2131, March 1997

[8] Michael S. Borella, Gabriel E. Montenegro, "RSIP: Address Sharing with End-to-End Security" Proceeding of the Special Workshop on Intelligence at the Network Edge, March 2000

[9] Brian T. Kurotsuchi, Eric Collins, "Linux IP Masquerading", 1998

[10] Michael S. Borella, David Grabelsky, Ikhlaq Sidhu, Brian Petry, "Distributed Network Address Translation", citeseer.nj.nec.com

[11] Rusty Russell, "Linux IPCHAINS-HOWTO", v1.0.8, 2000

[12] Mindcraft Inc.,"Webstone", v2.5, www.mindcraft.com/webstone