國立交通大學

資 訊 工 程 學 系

碩 士 論 文

一個網際網路端對端服務仲介之設計與雛型製作

Design and Prototyping of an Internet P2P Service Broker for NAT Traversal

研　究　生：劉榮智

指導教授：陳耀宗　教授

中 華 民 國 九 十 四 年 六 月

一個網際網路端對端服務仲介之設計與雛型製作

Design and Prototyping of an Internet P2P Service Broker for NAT Traversal

研 究 生：劉榮智　　　　Student：Rung-Jr Liu

指導教授：陳耀宗　　　　Advisor：Yaw-Chung Chen

國 立 交 通 大 學
資 訊 科 學 系
碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

In partial Fulfillment of the Requirements

for the Degree of Master

in

Computer Science and Information Engineering

June 2005
Hsinchu, Taiwan, Republic of China

中華民國九十四年六

Design and Implementation of an Internet P2P Service Broker for NAT Traversal

Student：Rung-Jr Liu

Advisor：Yaw-Chung Chen

A Thesis

Submitted to Institute of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

In partial Fulfillment of the Requirements

for the Degree of Master

in

Computer Science and Information Engineering

Hsinchu, Taiwan, Republic of China

June 2005

一個網際網路端對端服務仲介之設計與雛型製作

學生：劉榮智　　　　　　　　　　　　　　指導教授：陳耀宗 博士

國立交通大學資訊工程研究所

摘　　　要

網際網路協定第四版本（Internet Protocol version 4)早在九零年代初期就被發表出來，並且是目前在網際網路上被廣泛使用的通訊協定。但是隨著網際網路使用者的迅速增加，IPv4 的位址數量早已不敷使用。NAT (Network Address Translator) 的使用讓大量的網際網路使用者利用少量的 IPv4 address 來連上 Internet 可以達到減緩 IPv4 位址不夠用的問題。然而，NAT 雖然提供了連結 Internet 的便利性，它卻同時破壞了原有的 Internet 架構，讓 NAT 外部的使用者無法透過 IPv4 address 來定位 NAT 內部的使用者，並且導致 NAT 內部的使用者無法對外提供網際網路服務（Internet Services)。

網際網路協定第六版本（Internet Protocol version 6）在九零年代中晚期被發表出來。它把位址的長度從 32 bits 擴展到 128 bits，同時徹底解決了位址不敷使用的問題。然而，IPv6 的規格雖然目前被廣泛的討論，它的使用普及性卻是進展緩慢。在 IPv6 真正普及之前，IPv4 網路裡的 NAT 問題將持續存在。

本篇論文提出一個完整的系統架構來解決 NAT 內部的使用者無法對外提供 Internet Service 的問題。有鑑於現有的相關研究都因使用的不便利性而無法普及，本論文的系統設計是針對使用的便利性做最佳化，並且引用點對點（Peer to Peer）網路架構來增進效能。另外，本篇提出了 Passive TCP Splice 以及 Pure TCP Splice 兩個 kernel layer 的實做方法來增進系統效能，在本文中將探討完整的實做方法。

本篇的系統架構亦可套用到解決 IPv4/IPv6 Translation 以及 Firewall Traversal 問題。另外，亦可支援連上家用網路的電腦周邊設備的使用，讓使用者在家用網路的 NAT 外部亦能使用 NAT 內部的電腦周邊設備。

Design and Prototyping of an Internet P2P Service Broker for NAT Traversal

Student: Rung-Jr Liu Advisors: Dr. Yaw-Chung Chen

Institute of Computer Science and Information Engineering
National Chiao Tung University

Abstract

Internet Protocol version 4 (IPv4) was devised in early 90's and is broadly used in Internet nowadays. However, the rapidly increasing number of Internet users leads to the insufficiency of IPv4 addresses. NAT (Network Address Translator) was later devised to allow many Internet users to access Internet simultaneously using one single IPv4 address. However, while providing the convenience of accessing the Internet, NAT also breaks the original Internet structure. Users outside NAT can not use IPv4 addresses to locate the users inside NAT, and users inside NAT can not provide Internet services to the users outside NAT.

Internet Protocol version 6 (IPv6) was devised in mid 90's. IPv6 extends the IP address from 32 bits to 128 bits and completely solves the insufficiency of IP addresses. However, the transition to 128-bit IPv6 addresses has been proceeding slowly. The problem of NAT would remain until IPv6 become popular.

This thesis proposes a system to solve the problem of NAT. To avoid the inconvenience of the related work, the main guideline of this thesis is to make the system easier to be deployed. Furthermore, peer-to-peer network structure is introduced to this system to improve performance. In addition, this thesis proposes two new techniques: Passive TCP Splice and Pure TCP Splice to improve system performance.

This system can be applied to IPv4/IPv6 Translation and Firewall Traversal. Furthermore, it can support computer peripherals which connect to home network. Users can use this system to control their computer peripherals in their home network while travelling outside.

Acknowledgement

I would like to express my sincere thanks to my advisor, Prof. Yaw-Chung Chen for his enthusiastic guidance and perspicacious advice throughout the research. In addition, I am grateful to my good friends and lab-mates, especially Dr. Yi-Cheng Chan and Cheng-Yuan Ho, for their useful suggestions and valuable comments.

Finally, I would express my indebtedness to my dear parents, and my lovely girl friend, I-Hsuan Lu, for their love, inspiration and continuous support.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Internet Protocol (IP) was designed few decades ago. The dramatic growth on the population of the Internet in recent years showed that 32-bit IPv4 addresses have been rapidly depleted. However, the transition to 128-bit IPv6 addresses is proceeding slowly. A number of innovative solutions have been devised to work around this problem. In this chapter, we give a brief description and problem statement. Then, we introduce our system infrastructure shortly after the discussion of our motivation and contribution.

1.1 Overview and Problem Statement

Classless Inter-Domain Routing (CIDR) [1] is one of the initial solutions that alleviated the inefficiency in address space allocation to a certain extent. However, the alleviation from CIDR can only slow down the insufficiency tendency. Subsequently, a so-called Network Address Translator (NAT) [2] was devised to solve the problem. NAT allows a large number of hosts using private IP addresses to access the Internet via a single public IP address.

NAT does *IP Masquerading*, in which packets from a local IP-port pair are mapped to a global IP-port pair. This mapping is sometimes referred to as *source NAT*. Since all local hosts are mapped to a single IP address (or one from a small pool of addresses), there is no way for a foreign host to directly locate an individual host which is behind a NAT box.

Another form of NAT called *destination NAT* may be used to forward traffic from a port on the global IP domain to a specific host on the local network. Destination NAT is often used for load-balancing servers, where only a single IP-port pair is visible to the Internet, but incoming requests are locally passed to different machines in a round-robin scheme. This allocation of server ports is typically static and is done by the administrator.

While NAT provides many benefits, it also comes with many drawbacks. The most troublesome of those drawbacks is the fact that it breaks many existing rules of IP applications and makes it difficult to deploy new ones. With the rapidly increasing usage of peer-to-peer (P2P) applications such as File Sharing (i.e., Napster, eDonkey, Kazaa…) or VoIP (i.e., SIP Phone, Skype…), the Internet users establish network connections between two Internet nodes more and more frequently. Unfortunately, we can't directly establish the connections between nodes which connect to Internet via the NATs. This is because the initiator of the connection doesn't know how (where) to create this connection due to the lack of the receiver's IP address. Nodes behind NATs use private IP addresses to access Internet, but nodes outside the NATs can't use these private IP address to locate the nodes behind NATs.

In this thesis, we call the above problem the *NAT traversal* problem, and the techniques which resolve the NAT traversal problem are *NAT traversal techniques*. These techniques feature *NAT traversal capability*, which is the ability to traverse NAT. The hosts that connect to Internet via a NAT box are called *NATed hosts*.

1.2 Motivation and Contribution

Many techniques have been proposed to resolve the NAT traversal problem. Session Initiation Protocol (SIP) [3] users use Application Layer Gateway (ALG), STUN, UPnP, etc. to enable voice traffic to traverse NAT. Some techniques statically assign NAT's mapping table to enable incoming connection from outside to inside of NAT. Another technique requires the modification to TCP/IP protocol stack of OS (Operating System) kernel code. These existing works requires either special hardware or special middle server or administrator privilege or modification to existing kernel code, and furthermore they are not suitable for various existing network applications. Due to such inconvenience, the usage of these techniques is suppressed. Consequently, there is no unique method that can provide a generic solution and well accepted by common network applications.

In this thesis we propose an infrastructure to solve the NAT traversal problem. The main design guideline is to build an infrastructure which can be deployed as well as used easily. Besides, our design follows the following five principles: no configuration or modification to NAT, no modification to TCP/IP protocol stack of user host, no modification to existing network client applications (i.e., IE, outlook,…), no modification to existing network server applications (i.e., apache, sendmail,…) and must be compatible with existing network applications. In other words, this work tries to propose a general purpose solution for NAT traversal which allows the existing network applications still workable. To distinguish our infrastructure from traditional client-server model, we name the client of our system as *service subscriber* and the server as *service provider*.

The basic concept we propose here is that a service provider runs an application layer daemon to forward client's service request to a server process (i.e., apache, sendmail…). A service subscriber runs an application layer daemon to forward server's service response to a client process (i.e., IE, outlook…). In the globally addressed Internet, we setup an *Internet Service Broker* (termed as *ISB* here after) to help the negotiation of service subscriber and service provider. In addition, we setup an *Internet Service Translator* (termed as *IST* later) to improve the throughput of user traffic between service subscriber and service provider.

1.3 Organization of the Thesis

The rest of this thesis is organized as follows. In Chapter 2, we review the related works and address the advantages and disadvantages of them. The proposed infrastructure and improved scheme are detailed in Chapter 3 and 4. In Chapter 5, we present the implementation and the evaluation of the system. Finally, Chapter 6 concludes the thesis and points out some future works.

Chapter 2

Related Work

In this chapter, we describe the related techniques which are devised to solve the NAT traversal problem, also we discuss techniques similar to our infrastructure in detail. These techniques feature two main characteristics: *Middle Node Translation* (MNT) and *Local Side Alteration* (LSA). In Section 2.1, we describe the techniques with LSA attribute. Techniques requiring modifications within the same *Autonomous System* (AS) are classified into this section. In Section 2.2, we present the techniques which demand MNT. Finally, we give a brief comparison of these techniques in Section 2.3. We use "service provider" and "service subscriber" to replace "server host" and "client host", respectively.

2.1 Local Side Alteration Scheme

The Local Side Alteration scheme here is used in the techniques that a service provider or a service subscriber has to modify the operating system kernel, or require some specific network devices beyond NAT, or need the administrator privilege of NAT. In this section, we describe the following four techniques: SOCKS [4], statically assign [5], call interception [6] and UPnP [7].

2.1.1 SOCKS

SOCKS [4] is an application-layer protocol-independent proxy. It requires specific application support to work for both inbound and outbound connections. Existing "SOCKSifiter" programs and libraries make SOCKS-unaware programs work in the way of library/DLL replacement which alters the behavior of original network API (i.e., bind ( ), listen ( ), accept ( ), connect ( )…). Unfortunately, most commodity hardware routers such as those built into wireless access points or the so called "broadband routers" do not support SOCKS protocol. Therefore, a solution that works within the NAT framework may be preferable.

2.1.2 Statically Assign

The statically assign mechanism was first devised in [5]. It is the most obvious and simple way to solve the NAT traversal problem. A service provider only needs to modify the NAT address-port mapping table, while a service subscriber just has to connect to the pre-allocated address and port. It doesn't need any special application layer program or hardware, or any modification to TCP/IP protocol stack. However, the most troublesome drawback is that service providers are required to have the administrator privilege of the NAT, and every time when the service provider wants to provide service, he/she has to modify the NAT mapping table.

2.1.3 Call Interception

Pai's [6] proposes a transparent and dynamic system which has three components, the call interception module, the client daemon and the gateway daemon, to enable incoming connections to NATed hosts. When the service provider has an application trying to open a socket for incoming connections, the system call is intercepted and a message is sent to the client daemon. The client daemon then attempts to setup a forwarding rule on NAT gateway with the assistance of the gateway daemon. This is an automatic mechanism that the service provider does not need any extra operation, and therefore, it is very convenient for users to use. However, there are still two drawbacks. First, the service provider needs to modify his/her OS kernel implementation to enable the intercept module, which means none of current OS can support this functionality without the modification. And the second one, an even more troublesome drawback, is that it has to implement a special functionality into NAT box to execute the gateway daemon. Unfortunately, almost all commodity hardware NATs do not have this dynamic adding on feature.

2.1.4 UPnP

UPnP [7] is the abbreviation of "*Universal Plug and Play*", which is developed by Microsoft Corporation and some other vendors. Although the original motivation of UPnP project is not mainly focus on enabling incoming connection to hosts behind NATs, UPnP provides a set of APIs, called "NAT traversal", to enable NAT traversal. UPnP works like the call interception technique as mentioned above. Gateways which implement UPnP are called *IGD (Internet Gateway Device)*. A service provider runs a program through the API provided by UPnP to negotiate with IGD, which provides the functionality for NAT, to modify the NAT mapping table. After modifying the NAT mapping table, the service provider can run the original server program to accept incoming connection.

The main difference between call interception technique and UPnP is that UPnP standardizes the messages exchange between user host and IGD, and formalizes the behavior of IGD. Furthermore, a service provider needs to execute an additional program before execute the server program. The disadvantage of UPnP is similar to the call interception technique.

2.2 Middle Node Translation Scheme

The "Middle Node Translation" scheme here is used by the techniques that require administrators to setup one or multiple globally-addressable nodes to translate network traffic. In this section, we describe the following five category of techniques: IPNL [8], SIP Related techniques, Skype [9], VPN and Connection Splicer [10]. We give more detailed description of SIP related techniques, Skype and connection splicer due to some of their characteristics are used in our system infrastructure.

2.2.1 IPNL

Francis' [8] presents and analyzes IPNL (abbreviation for IP Next Layer), an NAT-extended Internet protocol architecture designed to solve the NAT traversal problem. In the NAT-extended architecture, hosts and NAT boxes are modified, and the IPv4 routers and supported protocols are kept untouched.

The IPNL architecture is performed by each privately addressed realm associating with one or more DNS zones. An additional packet header, which contains the *Fully Qualified Domain Name* (FQDN) of the destination host, is placed above IP header. IPNL users use FQDNs instead of IP address to locate the server host. Furthermore, NAT boxes are modified to route the packet using the information of the new header and FQDN.

The drawbacks of IPNL are obvious. Users of IPNL should modify their OS kernel code to place the additional header above the IP header, and NAT boxes should be modified to enable the routing functionality. These two drawbacks are complicated so that the deployment of IPNL is limited.

2.2.2 SIP Related: ALG, Session Controller, STUN, and TURN

The Session Initiation Protocol (SIP) [3] is a middleware signaling protocol that allows Internet endpoints to negotiate how to communicate with each other. SIP negotiates various parameters including port numbers, IP addresses, whether to use unicast or multicast, IPv4 or

IPv6 (or some other network protocol), and details of the media stream such as encoding methods. When SIP sessions are established through an NAT, the IP address in IP header will be modified by NAT and thus this IP address differs from that in SIP message. Figure 2.1 shows the NAT traversal problem of SIP.

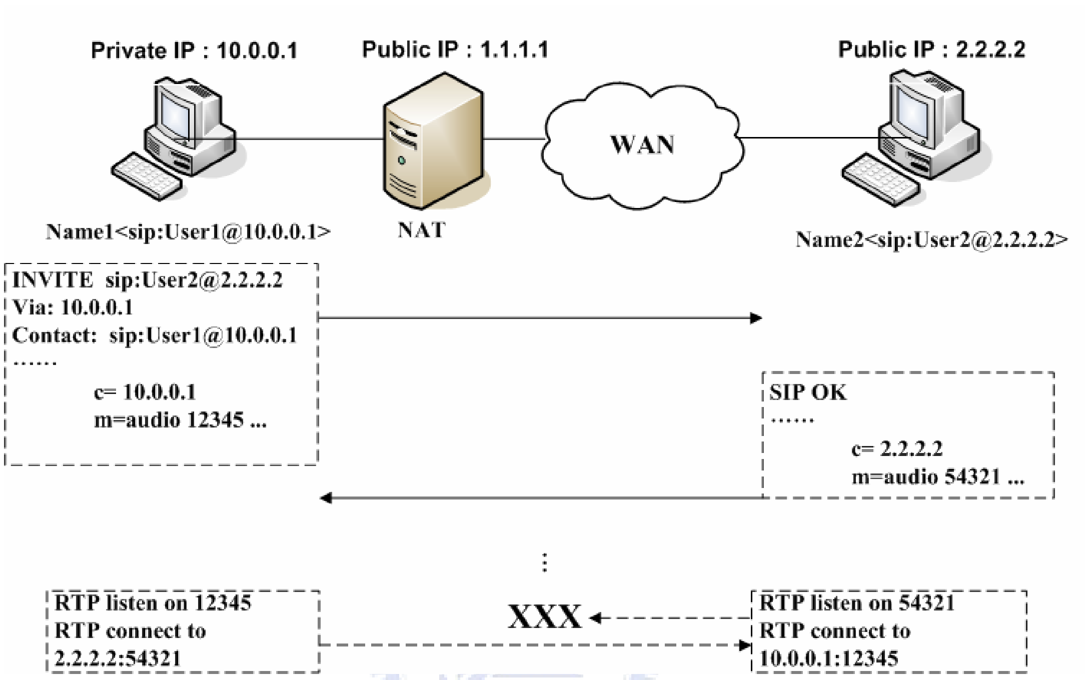


Figure 2.1: NAT Traversal Problem of SIP.

There are several solutions for NAT traversal problem of SIP: Application Layer Gateway (ALG), Session Controller, STUN [14] and TURN [13]. Each of these techniques requires the assistance of a third party server.

ALG detects the public IP address and port mapped by the NAT and modifies the SIP messages as shown in Figure 2.2. ALG modifies the “INVITE”, “Via”, “Contact”, “c”, and “m” fields in SIP and SDP message. Any field in SIP and SDP message which contains the information of private IP address are translated into public IP address, and port number in transport layer is also modified to an appropriate one.

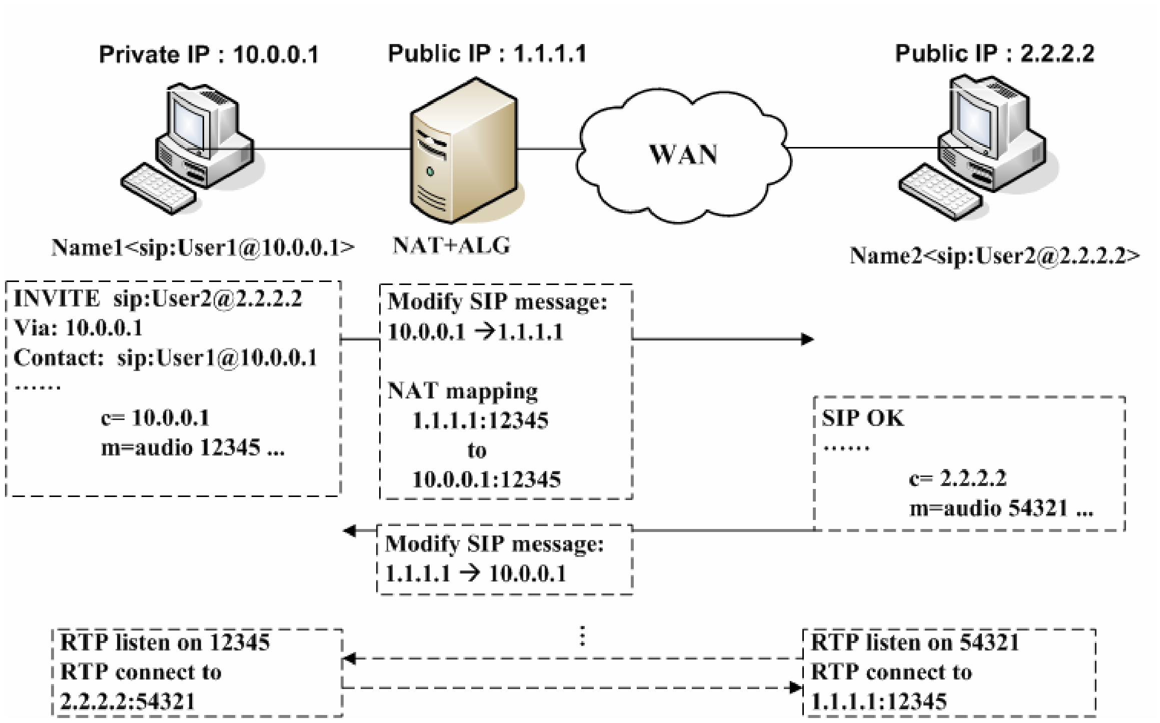


Figure 2.2: Solve NAT Traversal Problem of SIP using ALG.

Session controller is the combination of SIP proxy and RTP proxy. SIP proxy handles SIP signaling message, and RTP proxy is responsible for forwarding RTP voice packets. When SIP messages are sent to a SIP proxy, the message is modified by adding a “Record-Route” header. Therefore, all the following SIP messages will pass through this SIP proxy, and the SIP proxy will be able to control the behavior of each SIP client. When SIP clients are behind NATs, SIP proxy will modify the SIP message to change the destination IP addresses of RTP connections to RTP proxy. RTP proxy forwards the RTP voice packets for both the caller and callee. Figure 2.3 depicts the behavior of a session controller.

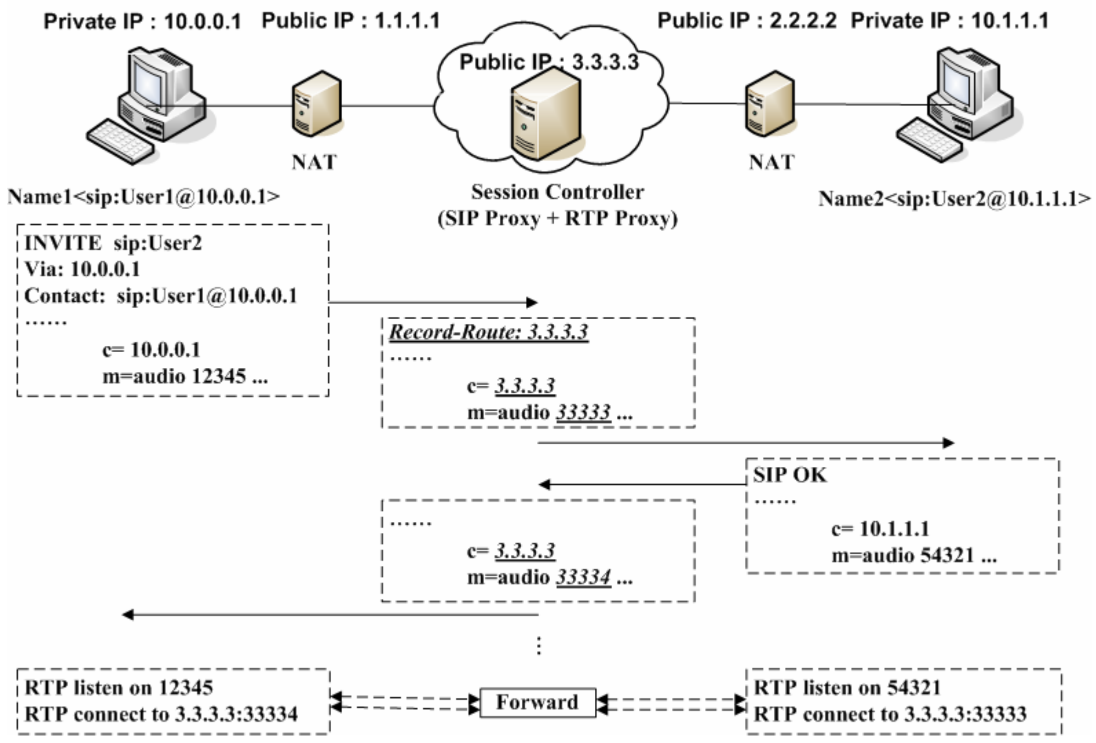


Figure 2.3: Solves the NAT Traversal Problem of SIP by using Session Controller.

STUN [14] allows a host to learn the global IP address and UDP port assigned by its outermost NAT box. This address can be subsequently conveyed by SIP to allow direct UDP connectivity between hosts. TURN [13] allows a host to select a globally-addressable TCP relay, which can subsequently be used to bridge a TCP connection between two NATed hosts. Unlike STUN, TURN does not allow direct connectivity between NATed hosts.

These techniques can solve the NAT traversal problem of SIP. STUN, TURN and ALG can be used by many other applications for NAT traversal and for many other purposes. However, the session controller technique can only be used for SIP application. To fully prove these techniques, we will cite some characteristics of these techniques to design our system infrastructure.

2.2.3 Skype

Skype [9] is a peer-to-peer VoIP application developed by KaZaa [11]. Skype claims that it can work almost seamlessly across NATs and firewalls. It encrypts calls end-to-end, and stores user information in a decentralized fashion. Although the detailed mechanism of Skype was not made public, Baset's [12] has finished the analyses of the Skype functions by carefully study the Skype network traffic.

From the analyses of [12], there are two types of nodes in this overlay network: ordinary hosts and super nodes. Any node using a public IP address and having sufficient CPU, memory, and network bandwidth is a candidate to become a super node. An ordinary host must connect to a super node and must register itself with the centralized Skype login server for a successful login. It is conjectured that Skype client uses a variation of STUN and TURN protocols to determine the existence of NAT. Once the Skype Client discovers that it is behind a NAT, it creates a TCP connection to one super node. Afterward all voice traffic sent to this NATed Skype client was forwarded by the super node via this TCP connection.

The Skype way for NAT traversal utilizes the nature of P2P system and provides excellent performance for Internet telephony. It is now popularly accepted to use third party relay to solve the NAT traversal problem. The main difference between Skype and session controller technique is that Skype utilizes the P2P infrastructure and disperses the bottleneck to super nodes which spread out in the whole Internet. The only shortcoming of the Skype way for NAT traversal is that it is suitable for the third party node to forward only the small-scale voice traffic. Since super nodes are also system users, they will not expect to exhaust their bandwidth to forward other people's traffic. When applying to general solution for NAT traversal, some other techniques for performance improvement are required.

2.2.4 VPN

VPN (abbreviation for Virtual Private Network) is a way to provide remote access to an organization's network via the Internet. VPN clients send data over the public Internet through secure “tunnels.” In addition to providing secure tunnels, VPN can also be viewed as another approach for NAT traversal. The VPN administrators can prepare a set of public IP addresses to the VPN server to allocate one public address to each VPN client, and the NATed VPN client connecting to this VPN server is considered as a host with public IP address. Since most current OSs have supported VPN, which may be the easiest way to solve the NAT traversal problem. The only drawback is that the NAT administrator should arrange a public IP address for each VPN client, and this drawback transfers the situation back to the insufficiency of IP address.

2.2.5 Connection Splicer

NUTSS [10] proposes a SIP-based approach to both UDP and TCP network connectivity. Furthermore, NTUSS constructs a mechanism, called *Connection Splicer,* to avoid the third party forwarder. It establishes a protocol, called *STUNT*, to extend STUN to include TCP functionality.

At the initial stage, hosts A and B establish their intent to communicate with each other via a proxy (as depicted in Figure 2.4). The proxy is the combination of connection splicer and STUNT server. Host A and B predicts their NAT’s behavior to get the mapping of global IP address and the transport port mapping on their NATs by performing *Port Prediction* with the assistance of the STUNT server.

Afterward host A sends it’s global address and global port to B via the connection splicer and likewise receives B’s global mappings. Upon receiving B’s global mapping, A initiates a TCP three-way handshake with it. Each host must also open a RAW socket so that it can see a copy of the first TCP SYN packet (packet 3) of the three-way handshake. This SYN packet must have a sufficiently low TTL that the packet is dropped between host A and host B. Host A then encapsulates the content of this TCP SYN (heard via the RAW socket) and sends it to host

B via the connection splicer (packet 4). Likewise, host A receives the contents of host B's TCP SYN via the connection splicer. Host A then constructs a message containing both its and host B's SYNs, and sends the message (packet 5) to STUNT server. From these two SYNs, the STUNT server is able to construct the SYNACKs (packet 6) that host A, host B and their NATs expect to see. This SYNACK is transmitted by the STUNT server to host A's NAT, spoofing the source address so that it appears to have come from host B via host B's NAT. Table 1 shows the source and destination IP address of these packets.

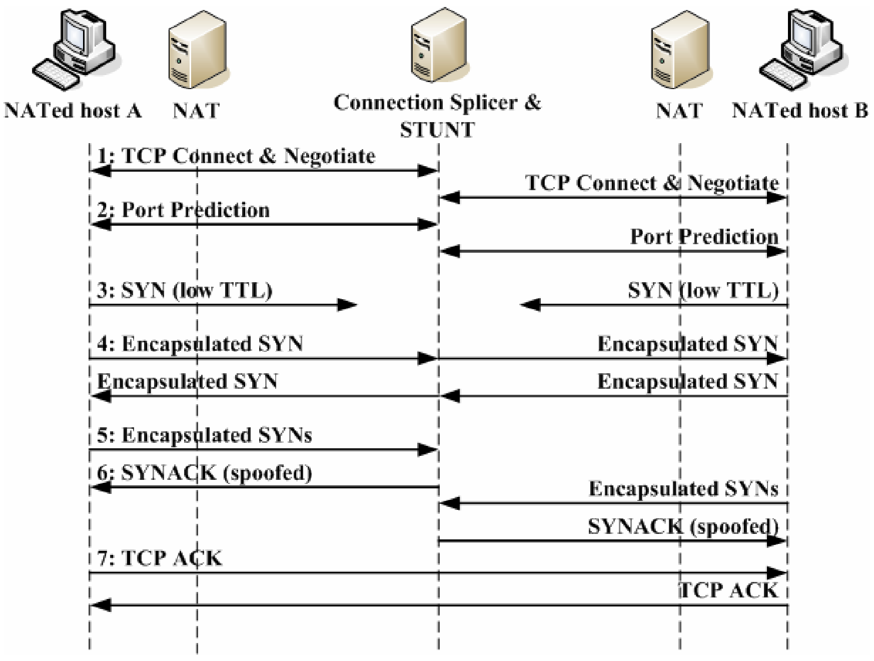
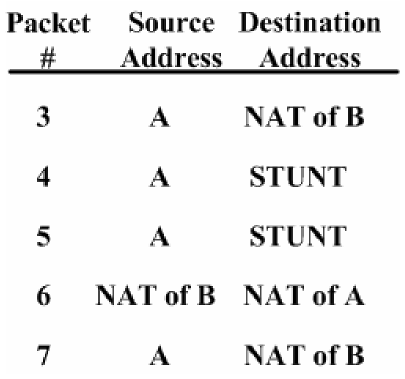


Figure 2.4: TCP connection setup of connection splicer.

Host A's NAT finds the SYNACK it is expecting to see and translates the destination address to A's private address and routes it accordingly. The packet is then received by host A, and therefore the three-way handshake is completed.



Packet #	Source Address	Destination Address
3	A	NAT of B
4	A	STUNT
5	A	STUNT
6	NAT of B	NAT of A
7	A	NAT of B

Table 2.1: Connection Setup for Connection Splicer.

The performance of connection splicer mechanism is excellent. Since it doesn't need a third party forwarder, the performance of this mechanism is equal to the best throughput which is based on the IP routing protocol. However, it still has some drawbacks. First, it needs to perform the Port Prediction which features some uncertainty. Second, the connection splicer needs to perform IP address spoofing which would not work when the spoofing packet is routed across the routers which perform ingress filter. Third, although RAW socket is implemented in almost all current OSs, some OSs require super-user privilege to grant access to RAW socket.

2.3 Comparison

In this section, we give a brief comparison of the NAT traversal techniques discussed above. Table 2.2 shows the additional requirements of these NAT traversal techniques, which can be categorized into "Modification of Application", "Modification of NAT", "Modification of Kernel" and "Third Party Server".

The modification of application mentioned here means that users need to either modify their computer programs or adjust some information in their computer programs to enable the NAT traversal capability. "Connection Splicer" belongs to the former type because computer programs should be modified to set the TTL of IP header to a small value. "SOCKS" and "SIP

Related" belong to the latter type because users should give the information of either SOCKS server or STUN server to their computer programs. "Statically Assign" and "UPnP" do not require the modification of applications because users perform some operations to setup NAT box or Internet Gateway Device (IGD) before they execute their computer programs. "IPNL" and "Call Interception" don't need the modification of applications because they transfer this requirement to the modification of kernel. "Skype" uses P2P infrastructure and is a specific application, therefore it doesn't need the modification of applications.

Requirements / Techniques	Modification of Application	Modification of Kernel	Modification of NAT	Third Party Server
SOCKS	Y	N	N	Y
Statically Assign	N	N	N	N
Call Interception	N	Y	Y	N
UPnP	N	N	Y	N
IPNL	N	Y	Y	Y
SIP Related	Y	N	N	Y
Skype	N	N	N	Y
VPN	N	N	N	Y
Connection Splicer	Y	N	N	Y

Table 2.2: Comparison of NAT traversal techniques: Additional Requirements.

"Call Interception", "UPnP" and "IPNL" require the modification of NAT box. "Call Interception" modifies NAT box such that it can automatically alter NAT mapping table. "UPnP" modifies NAT box to perform UPnP protocol. "IPNL" modifies NAT boxes to route packets according to the IP next layer header. The ALG technique we describe in "SIP Related" doesn't need modification of NAT because it is not restricted to associate with NAT and can be deployed in globally-addressable Internet.

Most techniques we discussed in this thesis require the assistance of third party server. "SOCKS" need a SOCKS server. "IPNL" needs other modified NATs to route users' packets. "SIP Related" requires ALG, Session Controller, STUN and TURN. "Skype" demands third party super nodes. "VPN" needs VPN server. "Connection Splicer" needs STUNT and connection splicer. The common characteristic of these techniques is that all the third party servers use public IP addresses. Advantages and disadvantages of the aforementioned NAT traversal techniques are listed in Table 2.3.

	Advantages	Disadvantages
	Local Side Alternation	
SOCKS	Has been carried out for many years, and many existing applications support SOCKS.	Applications should be modified to perform SOCKS protocol.
Statically Assign	Doesn't need any modification of application or kernel code or any assistance of third party node.	Requires administrator privilege of NAT box.
Call Interception	Provides an automatic mechanism which is very easy to use.	Needs modification of OS kernel code and NAT boxes.
UPnP	Supported by Microsoft, and therefore, is much easier to be popularized.	NAT boxes should be modified to carry out UPnP protocol, and is not well accepted by other UNIX-like OSs.
	Middle Side Translation	
IPNL	The performance is better than that traffic forwarded by global third party node.	The functionality requires users to modify their OS kernel, which means it would not work on some OSs.
SIP Related	Most popularly accepted, and many of them are standards.	Some of them are only suitable for SIP, and there is no well designed solution for general purpose.
Skype	Utilizes the P2P structure and doesn't have the bottleneck of global third party node.	When it is applied to general purpose, it consumes super nodes', which are also system users, bandwidth to forward other users' traffic.
VPN	Easy to use. All the user has to do is just connecting to a VPN server.	Requires a public IP address for each VPN client.
Connection Splicer	Doesn't need third party node to forward traffic, which provides the best point to point throughput.	Performs IP address spoofing which would be dropped by some routers. Some OSs require administrator privilege of client host to perform this mechanism.

Table 2.3: Advantages and Disadvantages of NAT traversal techniques.

Each of the NAT traversal techniques described above has some disadvantages and inconveniences. In Chapter 3, we propose a general purpose solution for NAT traversal to alleviate these disadvantages and inconveniences. Although it is also inevitable that our solution still has some drawbacks, our approach provides a much simpler infrastructure to be utilized and popularized for NAT traversal and hence can be applied to any kind of existing network applications.

Chapter 3

Proposed System Architecture

In this chapter, we propose an infrastructure for NAT traversal. The main objective of our design is to provide a solution which is easier to be popularized. In Section 3.1, we describe the design guidelines of our solution and based on these guidelines we make our solution much easier to be deployed. From Section 3.2 to 3.5, we discuss our infrastructure and mechanisms for NAT traversal. Section 3.7 discusses the deployment issue of our infrastructure. And finally in Section 3.8, we address the performance issues and additional capabilities of our solution.

3.1 Design Guidelines

The prerequisites of the existing techniques discussed in Chapter 2 are listed as follows.

- NAT boxes need to be revised.
- The existing client applications should be modified.
- The existing server applications should be modified.
- The original TCP/IP protocol stacks need to be revised.
- The OS kernel implementation or network API or libraries need to be revised.
- Does not provide a general purpose solution which can be applied to any kind of existing network applications.
- The demand of third party forwarder reduces the throughput between service

subscribers and service providers.

These annoying prerequisites of the existing techniques lead to the fact that none of them are popularly accepted for NAT traversal. To overcome these inconvenience, we setup five design guidelines for our solution as follows.

1. Neither re-configuration nor administrator privilege required for NAT boxes.
2. No re-configuration to "TCP/IP protocol stack" and "kernel implementation" of user host.
3. No modification to existing client applications (i.e., IE browser, putty, outlook…).
4. No modification to existing server applications (i.e., httpd, sshd, pop3d…).
5. Must provide general purpose solution and can be applied to all sorts of existing network applications.

Guideline 1 and 2 keep the NAT boxes and the TCP/IP protocol stack of user hosts untouched. Without such requirements related to NAT boxes, users can perform NAT traversal transparently through NAT boxes. Furthermore, guideline 2 allows users to have NAT traversal capability without the administrator privilege of localhost, since it achieves the independency of operating systems.

Guideline 3 and 4 ensure the compatibility to the existing applications. As a consequence, our proposed solution is feasible for commercialization.

Finally, guideline 5 aims to provide a general purpose solution, which means any kind of existing network applications can be supported by our solution. We use the term "any kind" here due to the fact that some network applications' behavior differs from the normal client-server model, in which client creates one TCP or UDP connection to the server to obtain the service. Some network applications (i.e., FTP, Netperf [15] …) require consequent connections after the first service connection from subscriber (client) to provider (server). Some network applications which run SIP protocol would transmit their private IP address in SIP messages and finally result in failure (discussed in Section 2.2.2). Our solution is designed for supporting these kinds of network applications and, furthermore, we expect to support all sorts of network services in the future.

3.2 System Components

There are three main components in our NAT traversal infrastructure.

- *Internet Service Broker (ISB).*
- *Service Provider*.
- *Service Subscriber*.

Service Provider and Service Subscriber reside in each user's host. Each user can be a Service Provider or a Service Subscriber or both. When users want to provide network services, they become Service Providers. On the other hand, when users want to subscribe services, they become Service Subscriber. User hosts can be either NATed or not NATed. The connectivity between NATed Service Provider and NATed Service Subscriber was carried out with the assistance of ISB and IST. Figure 3.1 shows the basic components in deployment.

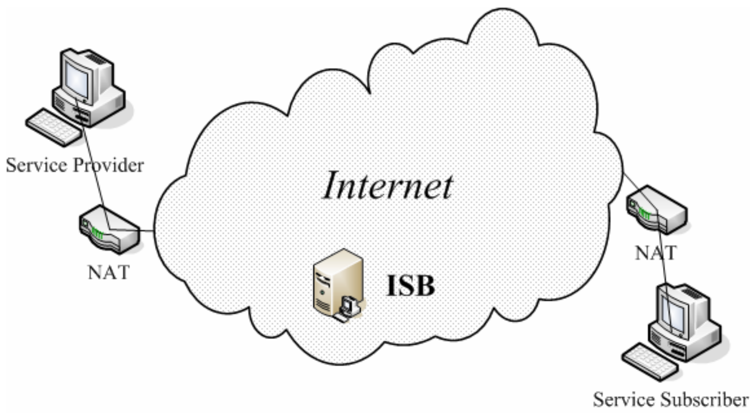


Figure 3.1: Basic components in deployment.

ISB is built in advance in globally-addressable Internet. The main functionality of ISB is to help the negotiation between the service subscriber and service provider. Each user wants to use this infrastructure needs to login to ISB. When a user tries either to provide or subscribe services, they should negotiate with other users via ISB, which maintains the profile information of users. Another functionality of ISB is to forward the traffic between a service provider and a service subscriber. To improve the forwarding performance, we can setup an additional host, *Internet Service Translator (IST),* to perform the forwarding task. IST can be deployed anywhere in globally-addressable Internet, and multiple ISTs can be deployed to improve the system performance. In the rest of this thesis, we discuss the scenarios which the forwarding job is performed by IST, and the functionality of ISB is only to help the negotiation between the service subscriber and service provider. ISB and IST exist in individual hosts.

3.3 Mechanism Outside the NAT

Both Internet Service Broker (ISB) and Internet Service Translator (IST) are deployed in globally-addressable Internet in advance. ISB maintains system users' profiles and those who want to use the system have to login to ISB. The login process can be accomplished through a TCP connection. After the completion of login, users keep the TCP connection persistent for the future negotiation. Figure 3.2 shows the connection mechanism.

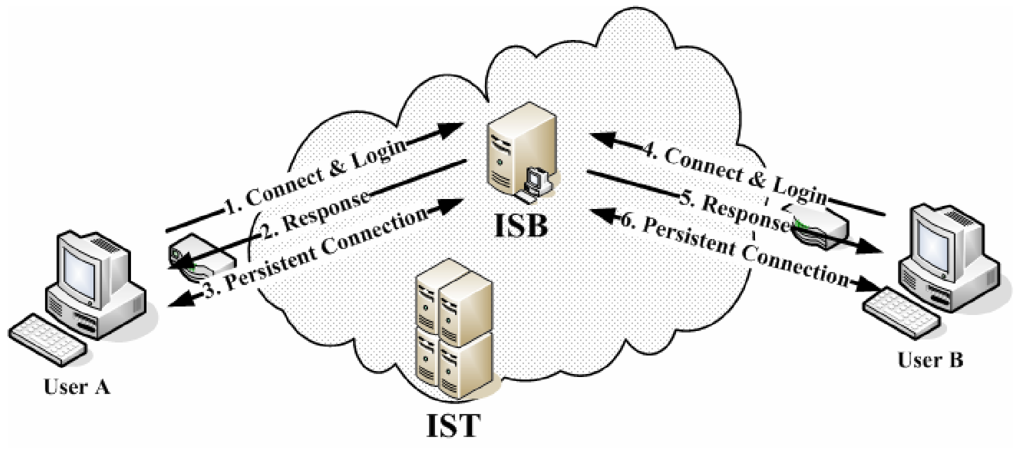


Figure 3.2: Persistant connections between the users and the ISB.

To keep the TCP connection persistent, User A and User B can turn on the “keepalive” option of Berkley Socket or WinSock implementation, or send hello messages to ISB periodically.

After system users login to ISB, they can check all available on-line services provided by other users. Furthermore, they can submit a request to ISB for subscribing services. ISB then checks this request and forwards it to the corresponding service providers via the persistent TCP connections between ISB and each system user host. Upon reception of the request, the service provider negotiates with the subscriber via ISB. We call this process as *Service Negotiation*.

If both service provider and service subscriber are NATed hosts, after the service negotiation finishes, ISB tries to find an appropriate IST which is close to either the service subscriber or the service provider. Furthermore, ISB sends a *splice request* to IST and instructs IST to create an UDP or TCP socket to listen to the port and be ready to splice connections from service provider and service subscriber. Finally, ISB tells both service provider and service subscriber the configuration of IST. The service provider and service subscriber then create a connection to IST and these two connections will be spliced into one single connection. Figure 3.3 depicts the scenario which assumes that the service subscriber creates the connection before the service provider does it.

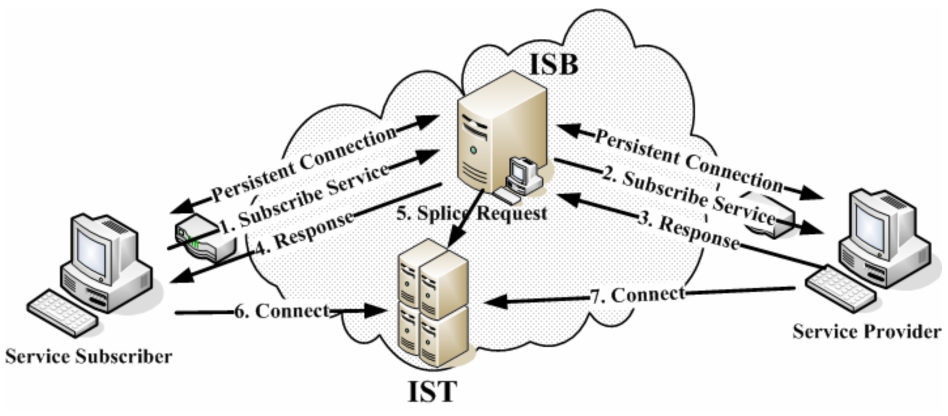


Figure 3.3: The service connection process via ISB and IST.

If only the service subscriber is NATed, our system can be viewed as a service discovery system. Service providers use our system to announce services they provide, and service subscribers perform service negotiation with the service provider via ISB and directly connect to the service provider to access the service. Therefore, under this situation, we don't need an IST to forward network traffic.

On the other hand, Figure 3.4 depicts the situation while only the service provider is NATed. The service negotiation is also accomplished via ISB and it again doesn't need an IST to forward network traffic, however, the service connection is created by the service provider. Each service subscriber or service provider needs to run an application layer daemon to execute this mechanism, and we discuss this daemon in Section 3.4.

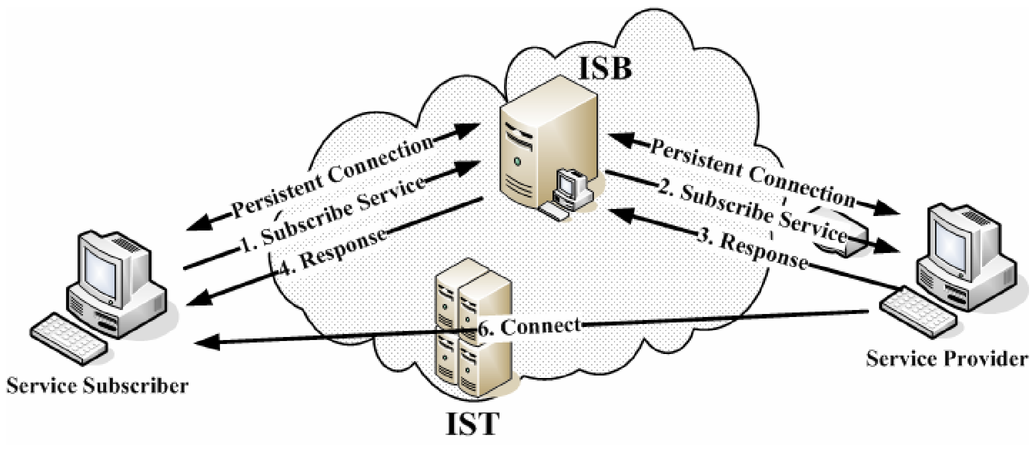


Figure 3.4: The service connection setup process via ISB only

After accomplish the connection setup, the service subscriber and the service provider are considered to have an end-to-end connection, and the service subscriber can send service request through it, while the service provider can reply through this connection also. In the next section, we describe the mechanism performed on those user hosts behind the NAT boxes.

3.4 Mechanism Inside the NAT

We describe the basic scheme in Section 3.4.1 and then the enhanced scheme for SIP NAT Traversal and other special applications in Section 3.4.2.

3.4.1 Basic Scheme

The login process and connecting to IST are performed by an application layer daemon running on the user host. Each user runs an application layer daemon to login and maintain the persistent connection to ISB, and this daemon maintains the behavior of subscribing service and providing service. Figure 3.5 shows the action of this application layer daemon when users on NATed hosts try to subscribe or provide services.

The actions in Figure 3.5 are performed after login and service negotiation. The user host A wants to subscribe service and the user host B likes to provide service. The daemons at both sides create a connection to IST. Besides connecting to IST, the daemon on host B also creates a connection to localhost port to which server process is listening. The main task of the daemon now is to forward messages between these two connections.

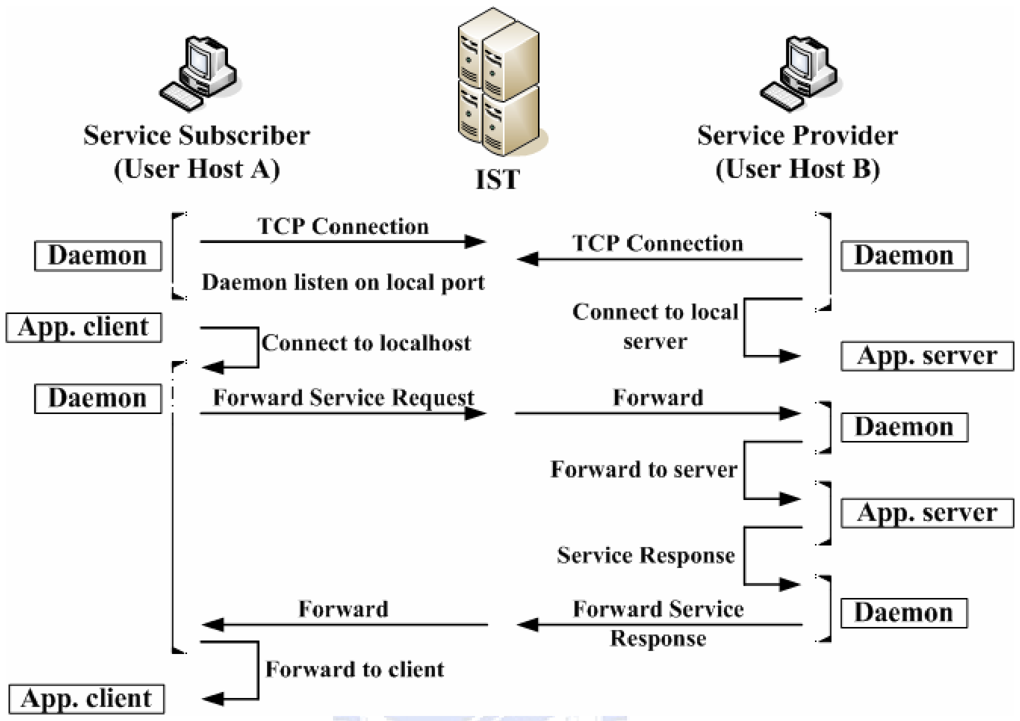


Figure 3.5: Actions of the application layer daemon on user hosts.

Besides connecting to IST, the daemon on host A also listens to a TCP or UDP port. When users on host A intend to access the internet service provided by host B user, they have to go through their client application (i.e., IE browser, FTP browser …) to setup connecting and send request to localhost port to which the daemon is listening. Upon receiving the request, the daemon on host A forwards it to the service provider via IST.

Take the web service as an example, the host B user provides a web service and the host A user subscribes this service. After the service negotiation is done and connecting to IST is established, the host A user can open the browser and key in the localhost IP address (127.0.0.1) and a TCP port number (get from the daemon). The browser will send an http service request (i.e., “GET /”) to a TCP port of localhost. Then the daemon forwards this service request to service provider via IST. The daemon on host B will then receive the service request from IST and forward it to the httpd on the localhost. The httpd then replies a service response to the daemon and the daemon forwards it to service subscriber via IST. Finally, the daemon on host A receives the service response and then forwards it to browser and the whole scenario is finished.

Instead of connecting to IST, if only one of the service provider and service subscriber is NATed and the other one is globally-addressable host, the application layer daemon on the NATed host can connect directly to the globally-addressable host. Therefore, the performance can be improved by reducing the latency and throughput limitation of the third party forwarder. In addition, our system can also utilize the benefit of P2P structure to improve system scalability. Section 3.6.3 describes the utilization of P2P structure.

3.4.2 Enhanced Scheme

As indicated in Section 3.1, our solution should be suitable for any kind of network applications. Some network service may not be accessed by a single TCP or UDP connection, some may requires multiple connections; therefore, we propose the enhanced scheme to make our solution suitable for various applications. The main ideas are to combine the Application Layer Gateway (ALG) with our daemon or to pre-define the behavior for special applications.

Take Netperf [15] as an example, it is a benchmark that can be used to measure various aspects of networking performance. When executing Netperf client, the first thing that will happen is the establishment of a control connection to the remote Netperf server. This connection will be used to pass test configuration information and results to and from the remote Netperf server. Once a control connection is setup and configuration information has been passed, a separate connection will be opened for the measurement using the appropriate APIs and protocols for the test.

If we like to make our system architecture suitable for Netperf, our daemon needs to know in advance that the service is Netperf. Furthermore, the daemon should negotiate with ISB to create two splices on IST. The use scenario is that a service provider and a subscriber accomplish the negotiation and create two splices on IST, and the daemons on both sides will fork two processes to handle the connection control and test of Netperf. When a service subscriber execute Netperf client to connect to localhost, Netperf client will automatically create another TCP connection to a specific port number on the localhost. The daemon on service subscriber host should forward the traffic of these two connections to the service provider.

Another type of special network service is SIP phone. A SIP phone application creates a RTP connection to transmit voice data after the SIP negotiation. As discussed in Section 2.2.2, the information of how to create this RTP connection is encapsulated in SIP message. If we like to make our system architecture suitable for SIP phone application, we should combine our daemon with an Application Layer Gateway (ALG).

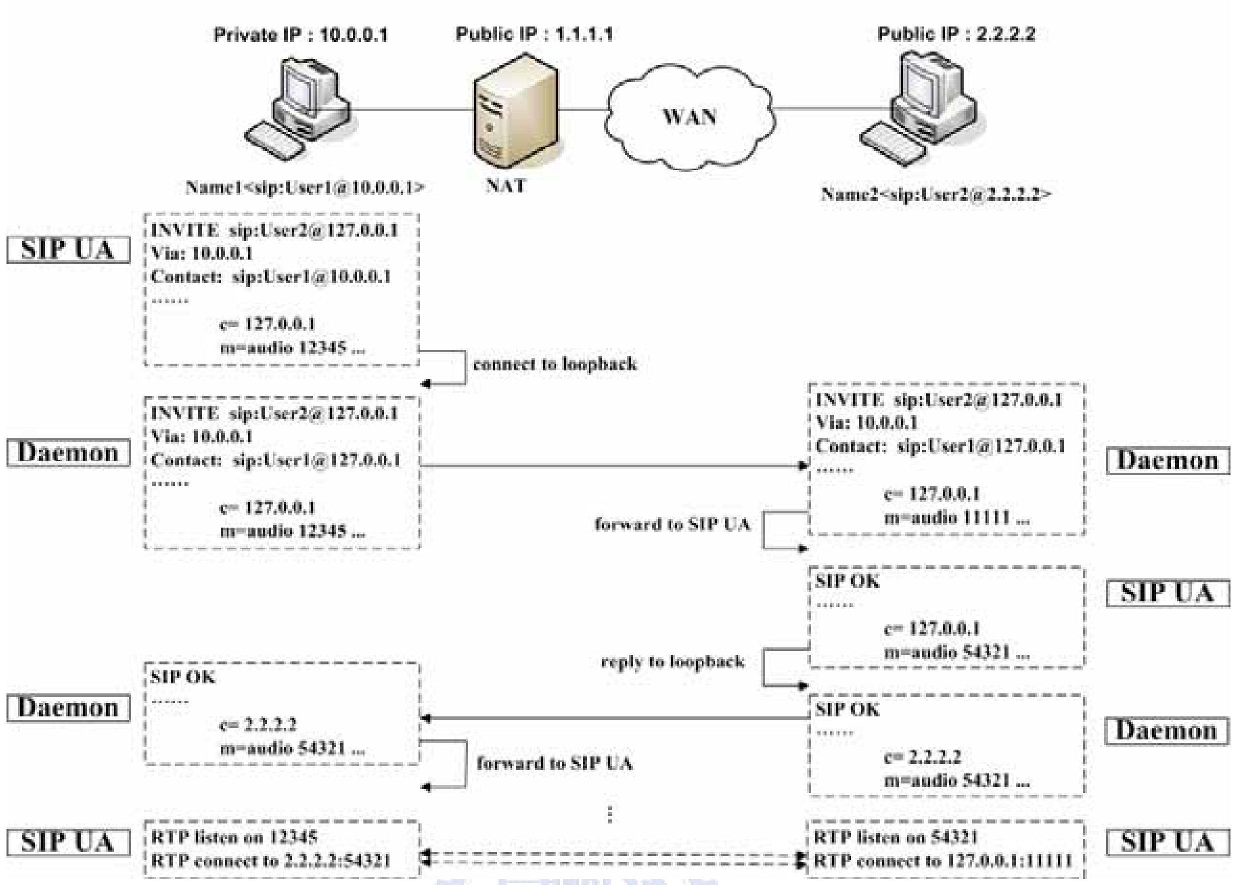


Figure 3.6: ALG is combined with daemon when traversing one NAT.

As discussed in Section 2.2.2, User1 in Figure 3.6 (left side) can create a RTP connection to User2 (right side) but User2 can not connect to User1 because User1 uses private IP address. Our solution is to combine the application layer daemons with ALGs. The ALG can be a new application layer program or just a function call in the daemon program. After the service negotiation of daemons, User1 send the SIP invite message to localhost which be forwarded to User2. The ALG on User2 host modifies the SIP message to change the field “c” of IP address and RTP port to localhost and the port which the daemon is listening to. Therefore, the RTP connection of User2 connects to localhost daemon and voice data is forwarded to the service subscriber via IST. In this scenario, we need to setup two splices on IST, one is for SIP message, and the other one is for RTP connection from User2 to User1.

Figure 3.7 is another example of SIP phone problem which is a little more complex than that in Figure 3.6. In Figure 3.7, both User1 and User2 are behind NAT. The main difference of ALG actions in Figure 3.6 and Figure 3.7 is that we need to setup three splices on IST, one is for SIP message, and the other two are for RTP connections in both directions.

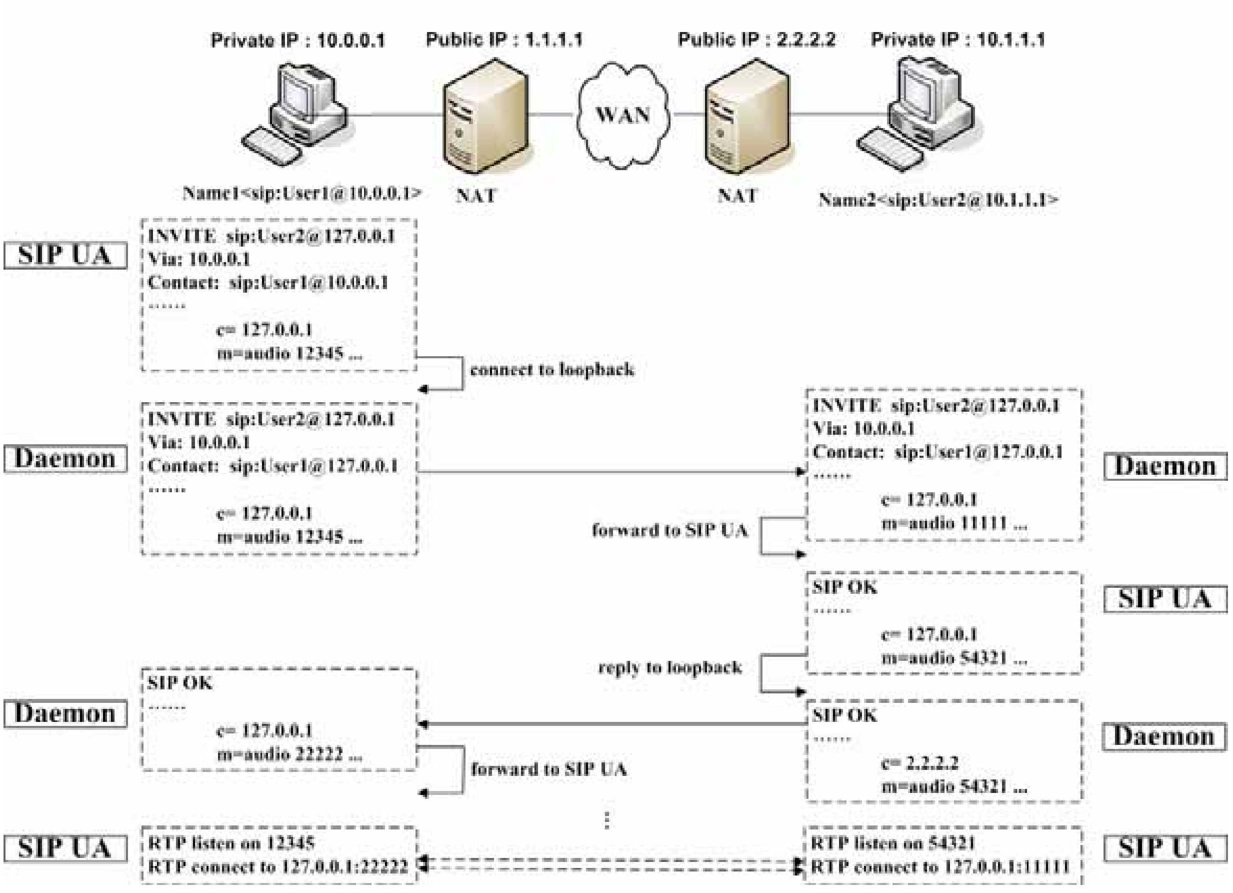


Figure 3.7: ALG is combined with daemon when traversing two NATs.

Based on the method of combining with ALG and the method of pre-defining the behavior, we believe that our solution could be applied to all sorts of network applications. The main disadvantage is that we should write an ALG or pre-define the behavior for each special application. The ALG is either an individual application layer program or just a function call, and we need to update our system software when a new application is to be deployed. However, we consider this as an inevitable overhead for solutions which provides an automatic mechanism. Without this drawback, our solution is very simple to use. All the system users have to do is executing an application layer daemon and the daemon will automatically perform the service negotiation.

3.5 Message Exchange

The messages exchanged between ISB, IST and Daemons are shown in Figure 3.8. The ISB and IST can be integrated into a single host. The forked daemon process and the original daemon process are combined into ISB Daemon in Figure 3.8.

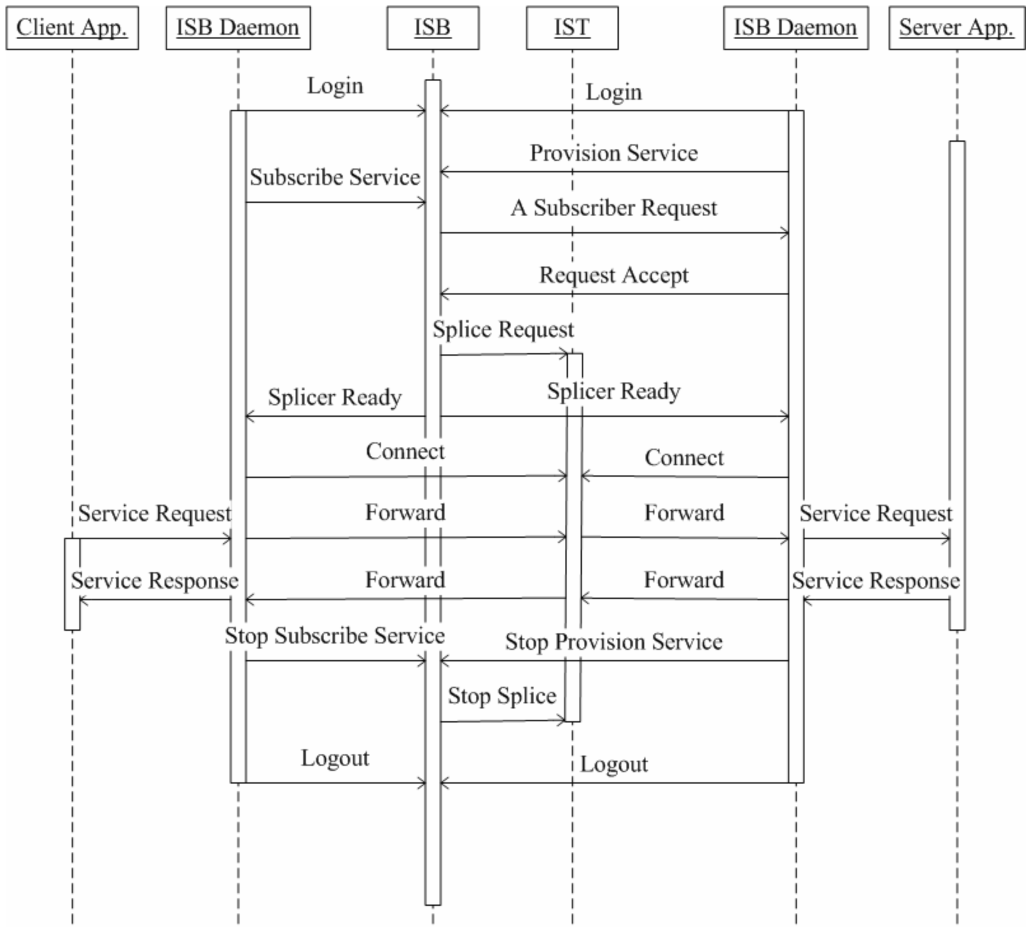


Figure 3.8: Message Exchanged between ISB, IST and Daemons.

3.6 Deployment Issues of ISB and IST

In this section, we discuss the deployment issues of ISB and IST. Multiple ISTs can be deployed on the edge routers of ISP (Internet Service Provider) to achieve a better system performance. In Section 3.6.1, the deployment of IST is addressed. Moreover, we can deploy multiple ISBs to build up a P2P structure in the globally-addressable Internet to improve performance of service negotiation. In Section 3.6.2, we discuss the deployment issue of ISB. Finally, in Section 3.6.3, we address the method which allows NATed users to act as an IST. Combining all techniques discussed in this chapter, globally-addressable users are able to perform the functionalities of both ISB and IST to form a P2P infrastructure, and our proposed system would feature good scalability.

3.6.1 Deployment of IST

The main functionality of IST is to splice connections from service providers and service subscribers. When both service subscriber and service provider are NATed, all service request messages and service response messages will pass through IST. It is obvious that IST will be the bottleneck of system performance. In this thesis, we propose two approaches to minimize the bottleneck effect. One is to enhance the performance of IST which we discuss in Chapter 4. The other is to consider the deployment issues of IST and ISB, and we discuss this part here.

We can deploy multiple ISTs in globally-addressable Internet. Multiple ISTs can be considered as load sharing of the packet forwarding task, therefore, the more the ISTs we deploy the better performance we can achieve. However, when we deploy multiple ISTs in the Internet, a new problem may arise. After service providers and service subscribers finish their service negotiation, ISB has to choose an appropriate IST to forward their service traffic. The term “appropriate” here is that the IST should be located on the shortest path or best routing path between the service provider and service subscriber. The selection method of IST is out of this thesis’s scope, and we regard this topic as our future work.

An obvious way of IST deployment is to install it along with the edge routers of ISP (Internet Service Provider). Since the user traffic of an ISP would go through the ISP edge router, the hop count between a service provider and a service subscriber in our system is the hop count of the best routing path plus one, so the extra overhead of routing a packet through our proposed system could be minimized.

3.6.2 Deployment of ISB

The main functionality of Internet Service Broker (ISB) is to maintain the profile information of system users and help them accomplish the service negotiation. If we deploy multiple ISTs in the Internet, the ISB is also required to find out the most appropriate IST for a service subscriber and a service provider.

Ws can deploy multiple ISBs in globally-addressable Internet to improve the performance of user login and service negotiation. Although the overhead of ISB tasks is not heavy, we can still deploy multiple ISBs for scalability. Once we have a large number of users who frequently provide and subscribe services, the deployment of multiple ISBs would become necessary.

Another interesting idea is that we can deploy multiple ISBs in Internet to form a P2P network. Each ISB provides the functionality of IST. The profile information database can be divided into several smaller parts and spread into the ISB P2P system. Furthermore, users who are globally-addressable also can provide the functionalities of ISB and IST. Then our system would become well scalable because the task of original ISB and IST are decentralized and distributed to P2P system users. Figure 3.9 shows the configuration. There are several existing works devised to form P2P network structure [24, 25 and 17], therefore we don’t discuss the details of how to form the P2P network structure in this thesis.

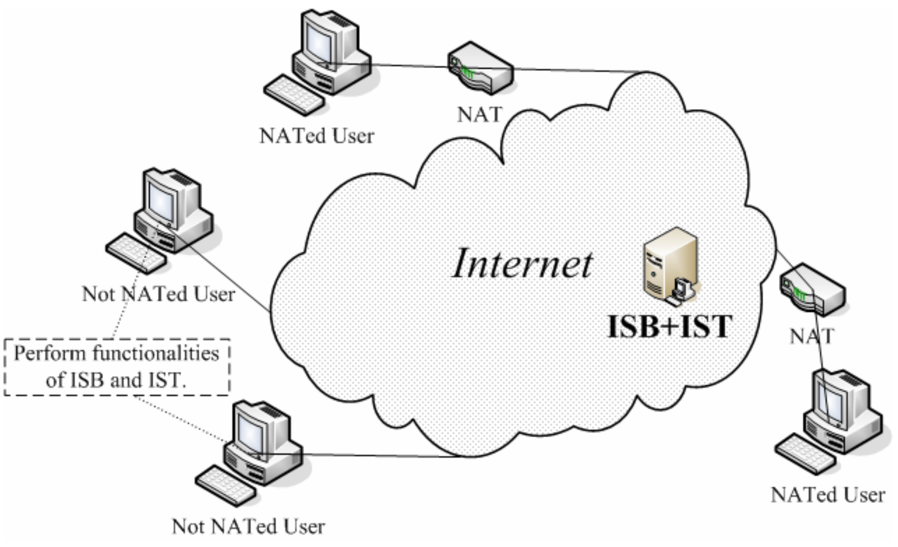


Figure 3.9: Decentralized operation of ISB and IST.

Some may raise the question that why nodes with public IP addresses want to use our system because our system is originally designed for NAT traversal. The reason is that nodes which like to subscribe services from NATed host are required to participate in our system. They can access the services of NATed host only through participating the system and perform the service negotiation via ISB. When more and more globally-addressable nodes participate in our system and form a large P2P structure, the overall system performance would be improved.

3.6.3 Providing IST Functionality on NATed Host

In this section, we describe the approach to perform the IST function on NATed hosts. The RFC 3489 defines four types of NAT behavior, and NATed host can perform the functionality of IST if they are using specific kind of NATs. We redefine these four types of NAT behavior as follows.

- **Full Cone**: The NAT mapping table records NATed host's source IP address (internal address), source port (internal port), NAT's external source IP address (external address) and external source port (external port). No two entries have the same pair of internal address and internal port.

- **Restricted Cone**: The NAT mapping table records internal address, internal port, external address, external port and destination host's IP address (destination address). No two entries have the same pair of internal address and internal port.
- **Port Restricted Cone**: The NAT mapping table records internal address, internal port, external address, external port, destination address and destination host's port (destination port). No two entries have the same pair of internal address and internal port.
- **Symmetric**: The NAT mapping table records internal address, internal port, external address, external port, destination address and destination port. There will be some entries which have the same pair of internal address and internal port.

As long as the system users' NAT type is "Full Cone", it is able to perform the function of IST. This is done by that ISB learns the source IP address and source port then check the type of NAT, as shown in Figure 3.10. When ISB receive the first login packet from a NATed host, it records the source IP address and source TCP port. Then, ISB uses another Network Interface Card or another IP address to send a test packet to the source IP address and source TCP port of NATed host. If the NATed host receives the test packet, he can make sure that his NAT is "Full Cone". The NATed host then replies a message to inform ISB his NAT is "Full Cone". Then, ISB and the NATed host negotiate regarding whether to perform the IST function on the NATed host..

If the NAT type is not "Full Cone", we can use "Port Prediction" technique in [10] to perform the IST function on the user host. However, port prediction features some kind of uncertainty.

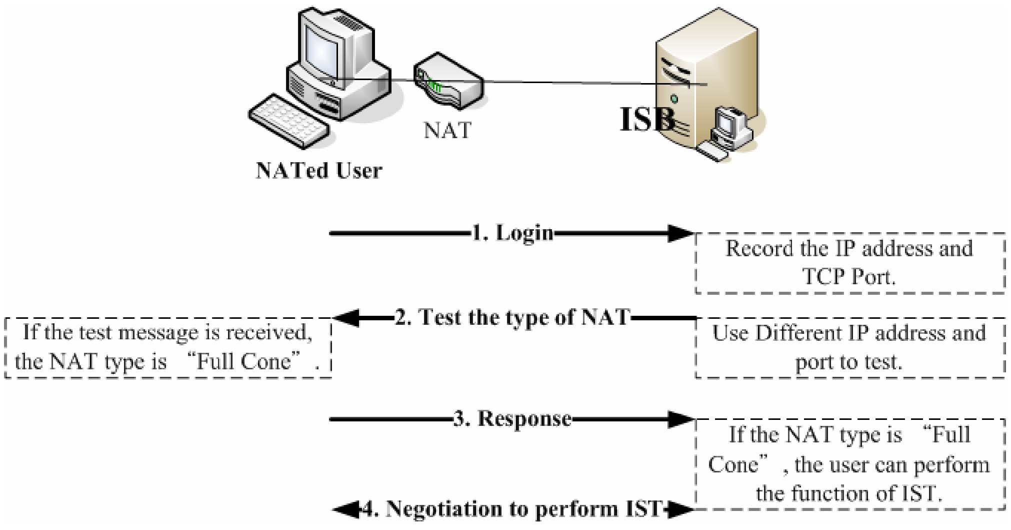

NATed User
NAT
ISB
1. Login
Record the IP address and TCP Port.
2. Test the type of NAT
If the test message is received, the NAT type is "Full Cone".
Use Different IP address and port to test.
3. Response
If the NAT type is "Full Cone", the user can perform the function of IST.
4. Negotiation to perform IST

Figure 3.10: ISB test the type of NAT.

If we can perform the IST function on each user's host, the infrastructure would become well scalable. Furthermore, we can combine the functionality of the user daemon into a home gateway (in fact, this violates our system design guideline in Section 3.1), which is a network device running as a gateway of home network. The advantages of combining the IST and user daemon into home gateway are two folds: good system scalability and persistent connection to ISB. Since the nature of a home gateway is always on-line, persistent login to ISB allows the home network users access to the resource of his/her home network while travel outside. In addition, these advantages are great help for the deployment of our proposed infrastructure.

3.7 Discussion

In this section, we discuss four additional issues: firewall traversal, IPv4/IPv6 translation, multicast and computer peripheral applications.

3.7.1 Firewall Traversal

Our solution can be also applied to firewall traversal. As long as the system users have the capability to connect to an ISB and an IST, they can use our system and the firewall can be viewed as a special type of NAT box. The firewall here is a filter which is deployed at the edge of Autonomous System (AS) of Internet. A firewall filters packets based on the information in packet headers. If the connections to an ISB and an IST are not filtered by the firewall, users behind the firewall would be able to use our system for firewall traversal.

3.7.2 IPv4 / IPv6 Translation

Our system can also be applied to IPv4/IPv6 translation. Either an ISB or an IST has at least two network interfaces, one connects to IPv4 network and the other one connects to IPv6 network. The functionalities of ISB and IST keep unchanged. The only required modifications are to assign the ISB to handle login and negotiation from both IPv4 and IPv6 networks, and to assign the IST to forward the traffic from both IPv4 and IPv6 network, as depicted in Figure 3.11 and Figure 3.12.

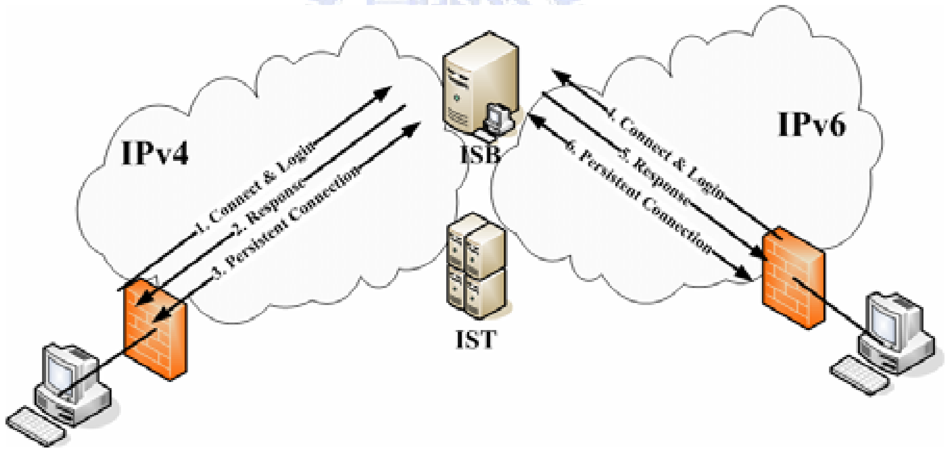


Figure 3.11: Login process of IPv4/IPv6 translation using ISB and IST.

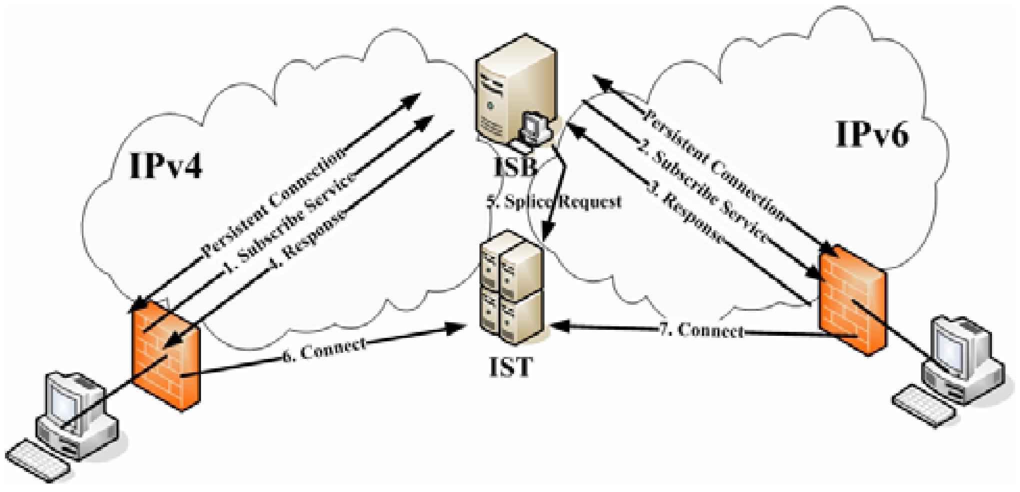


Figure 3.12: Service negotiation of IPv4/IPv6 translation using ISB and IST.

3.7.3 Multicast

Our system can also support multicast as shown in Figure 3.13. NATed user could use this system to act as a source node of multicast.

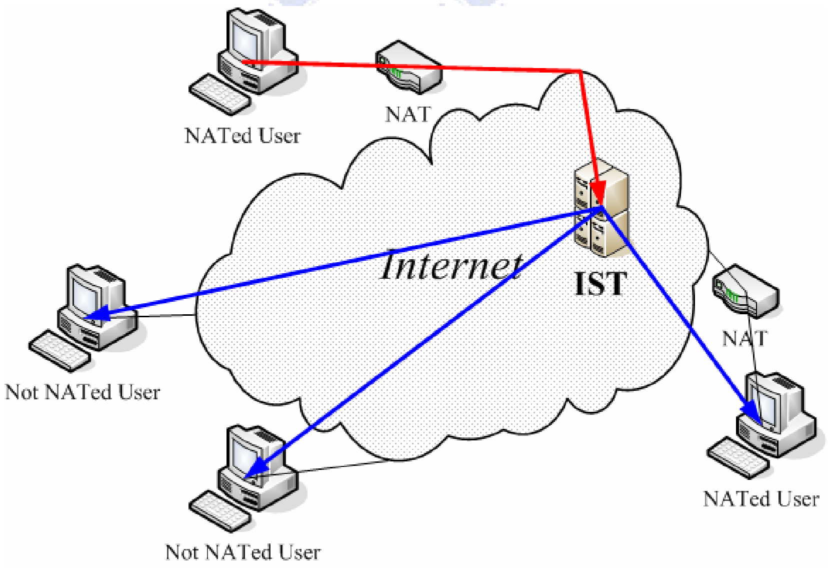


Figure 3.13: IST supports multicast.

3.7.4 Computer Peripheral Applications

In addition to network services, the service providers can also provide the control of their computer peripherals. Figure 3.14 depicts the situation that there are many computer peripherals connecting with PC through home networks, which can be Ethernet or 802.11 series or some alternatives. Once the daemon on a service provider host receives the service request from a subscriber via IST, the daemon can forward the service request to a computer peripheral instead of forward to server process on localhost. As long as the computer peripheral can recognize the service request and reply effectively, the service subscriber is considered to have the control of this computer peripheral.

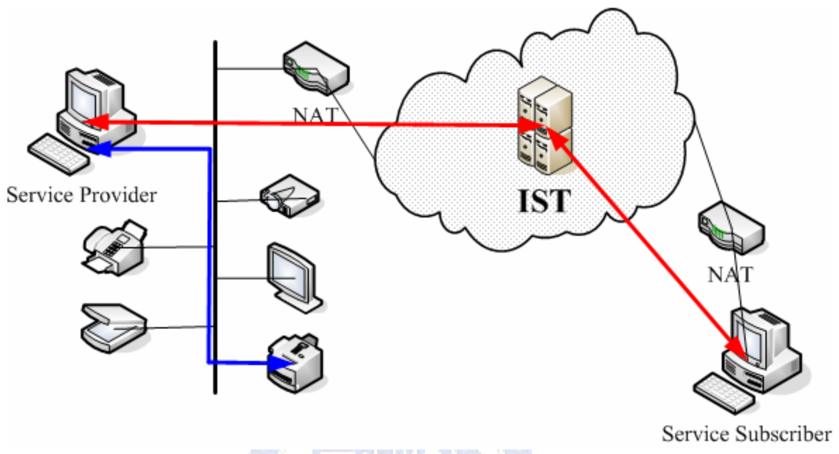


Figure 3.14: Applied to computer peripherals.

Chapter 4

Passive and Pure TCP Splice

Since the IST is the potential bottleneck of our system, we propose methods to improve the performance of IST in this chapter. Due to the rapid advance in computing power of PC, the overhead of forwarding packets by daemon is considered as small, we assume that nowadays users' PCs have sufficient capability to provide the functionalities of the user daemon. Therefore, we focus on the forwarding performance in globally-addressable Internet. In Section 4.1, we depict the easiest way to implement the forwarding function on IST. Section 4.2 describes an existing work "TCP Splice" which is commonly deployed by proxies for performance improvement. In Section 4.3 and 4.4, we introduce our proposed method to improve the forwarding performance on IST.

4.1 Application Layer Proxy

The easiest way to implement the forwarding function on IST is to build it as an application layer proxy program. It is often used by the simple proxy which is responsible for light traffic.

This application program creates a socket and listens to the incoming connection request. When a connection request arrives, it creates a section connection to another destination host. All the actions of this application program are accomplished by calling the APIs provided by

OS, and afterward these APIs make certain system calls, which results in frequently context switching between kernel process and application process, and hence produces a very large overhead. Furthermore, the overhead of memory copy is also huge. When an incoming packet arrives at the proxy's NIC (Network Interface Card), the NIC generates a hardware interrupt and then the kernel process copies the packet from NIC's buffer to the kernel buffer. When the application layer proxy program invokes the receiving system call, the packet payload was copied again into the application buffer. Then the application layer proxy program sends the packet to the destination host, this requires another two memory copies; from application buffer to kernel buffer and from kernel buffer to NIC buffer. Therefore, it needs four memory copies for each byte of packet payload when the application layer proxy wants to forward the packet.

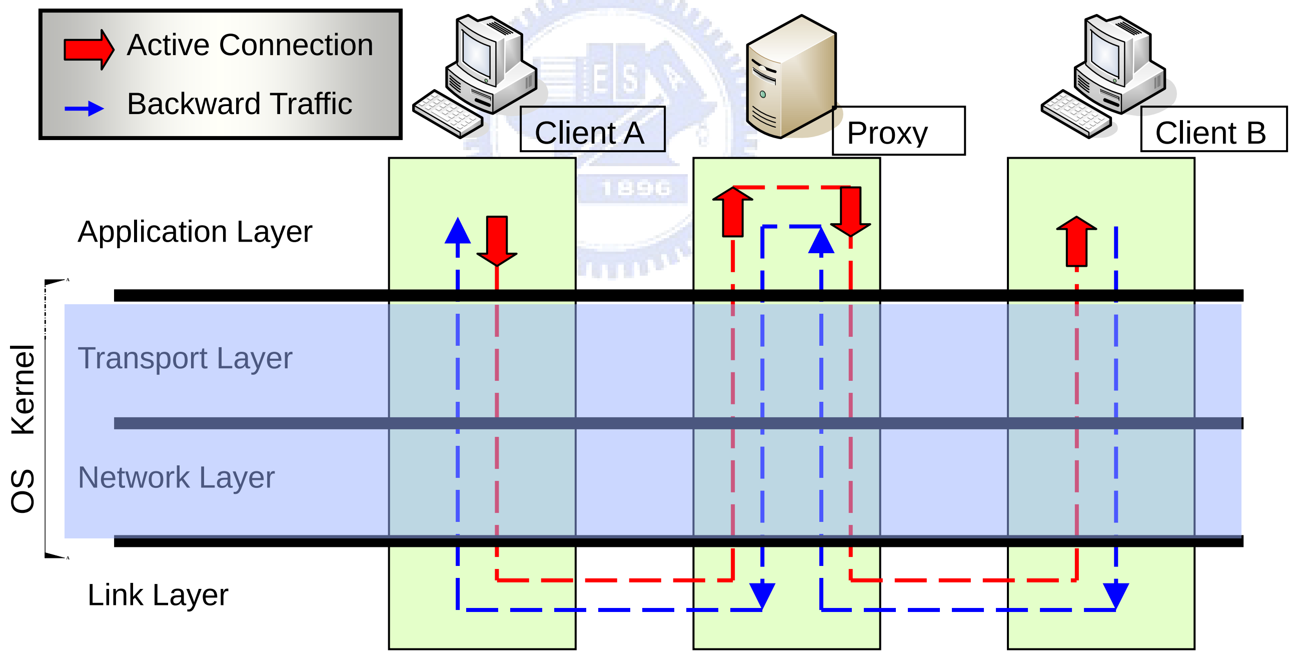


Figure 4.1: The path of packet traversing an application layer proxy.

As Figure 4.1 shows, client A creates a connection to the proxy and then the proxy creates another connection to client B. Each packet traverses the proxy across four layers. When applying the proxy to IST, the IST would listen to two connections and both client A and client B create a connection to IST.

4.2 TCP Splice

The idea of *TCP splicing* was developed by researchers at IBM [16]. The main concept of TCP splice is to forward the packets between two connections without letting the packets traverse across application layer. As Figure 4.2 shows, packets don't touch the application layer and only reach the transport layer. It needs only two memory copies for a TCP splicer to forward a packet; from NIC buffer to kernel buffer and kernel buffer to NIC buffer.

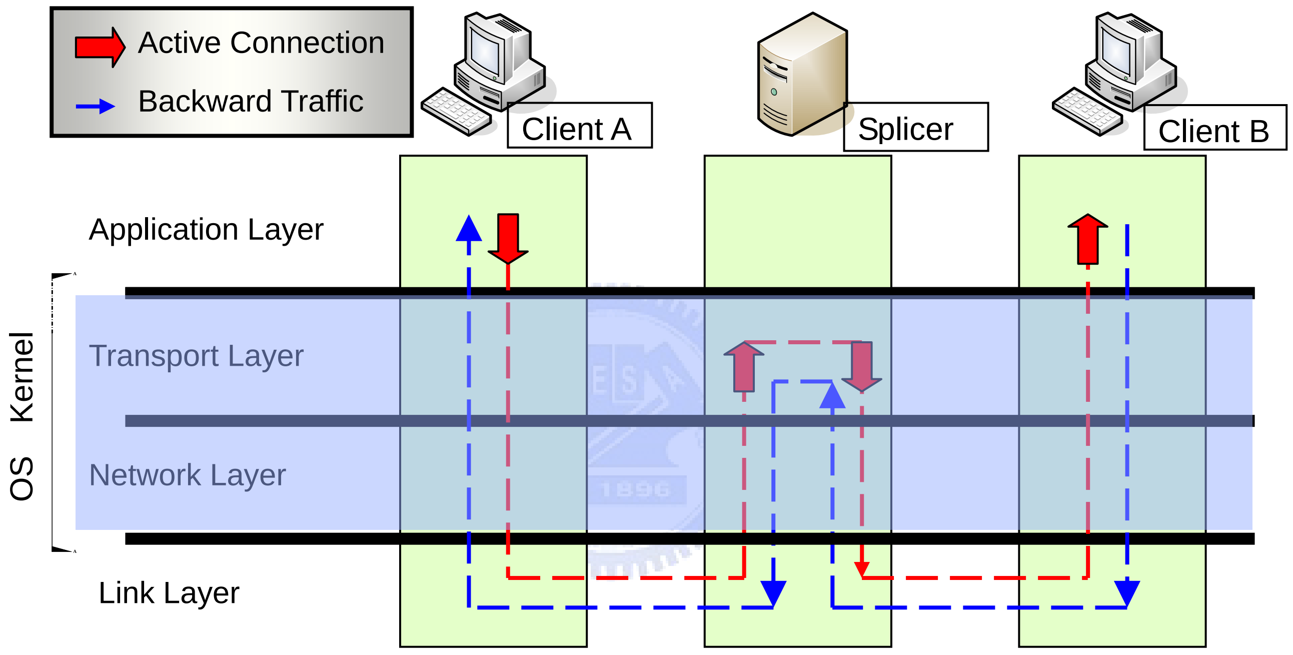
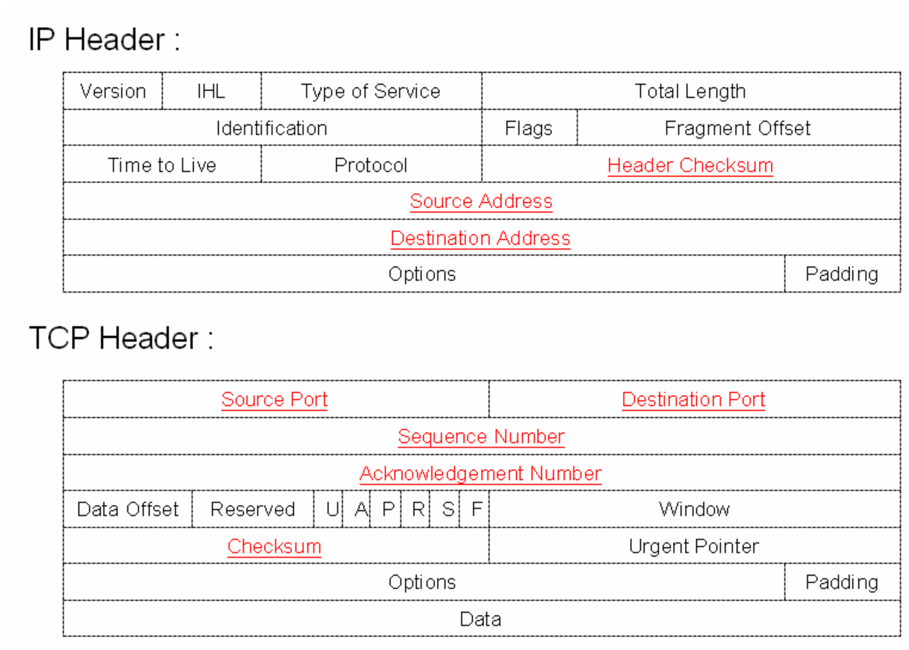


Figure 4.2: The path of packet traversing a TCP splicer.

To implement TCP splicer we have to modify the TCP/IP protocol stack in kernel. Whenever the TCP splicer receives an incoming TCP connection request and finishes the three-way handshake, it immediately creates a second TCP connection to another host and maintains a sequence number table to record the offset of the sequence numbers of these two connections. When a packet comes from one of these two connections, the TCP splicer modifies the TCP sequence number and acknowledgement number based on the sequence number table and updates TCP checksum. Finally, the packet is transmitted via the other TCP connection. Table 4.1 shows the fields of packet header which need to be modified by TCP splicer. The underlined fields need modification.



IP Header :

Version	IHL	Type of Service	Total Length		
Identification			Flags	Fragment Offset	
Time to Live		Protocol	Header Checksum		
Source Address					
Destination Address					
Options					Padding

TCP Header :

Source Port								Destination Port	
Sequence Number									
Acknowledgement Number									
Data Offset	Reserved	U	A	P	R	S	F	Window	
Checksum								Urgent Pointer	
Options									Padding
Data									

Table 4.1: The fields need to be modified by TCP splicer.

In addition to both sequence number and acknowledgement number, TCP splicer also modifies source port, destination port, source address and destination address. These four fields are used to identify a unique connection. Finally, TCP splicer updates the checksum of both IP header and TCP header, and transmits this new packet to the other connection.

The main advantage of using TCP Splice is that copying packets would stay in OS without touching the application layer, and therefore, it needs only two memory copies to forward a packet; from NIC buffer to kernel buffer and vice versa. Furthermore, comparing with application layer proxy, TCP Splice reduces the overhead of performing TCP congestion control and flow control of two connections between client hosts and the proxy.

4.3 Passive TCP Splice

We propose a new mechanism for performance improvement of our IST, which is very like TCP Splice, and we call it *Passive TCP Splice*. Figure 4.3 shows the packet traversal path of a passive TCP splicer. Similar to TCP splicer, the path of packets traversing a passive TCP splicer would not touch the application layer, and all the activities are performed by OS kernel processes.

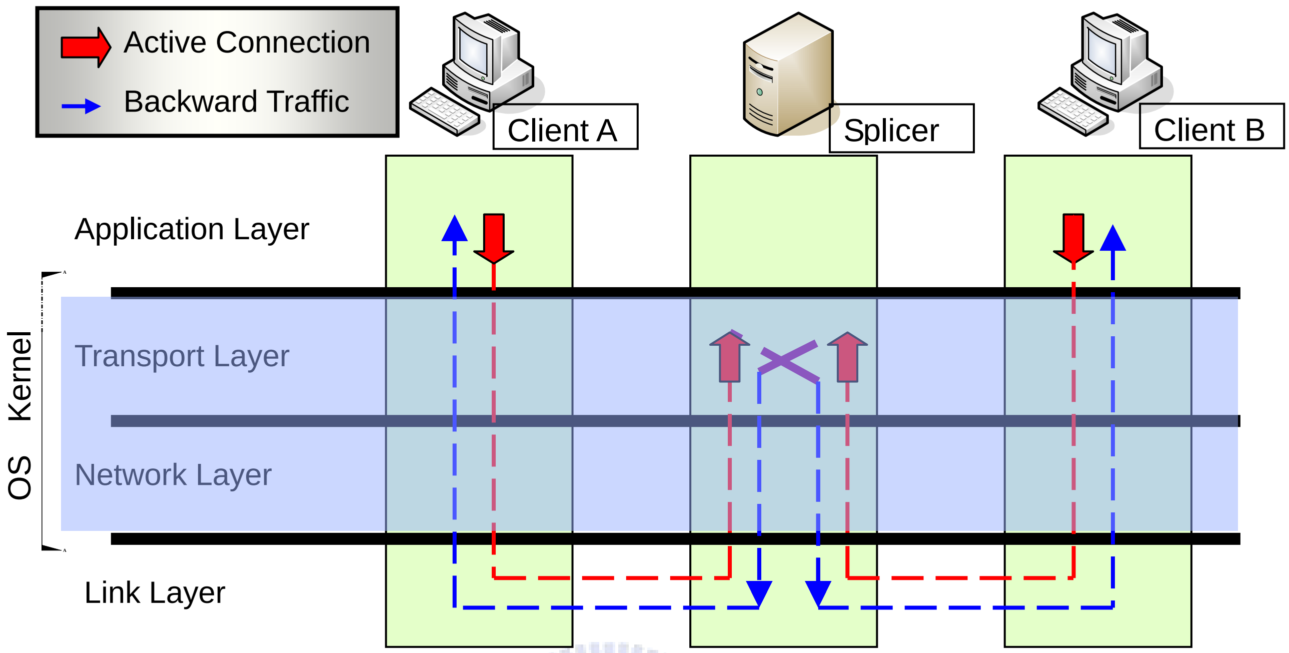


Figure 4.3: The path of packets traversing a passive TCP splicer.

Instead of modifying the sequence number of each packet based on the sequence number table, passive TCP splicer neither modify the sequence number nor change acknowledgement number of TCP header. It splices two three-way handshakes. As Figure 4.4 shows, the passive TCP splicer sends SYNACK, which is the second packet of TCP three-way handshake, to client A using client B's sequence number and sends SYNCACK to client B using client A's sequence number. Thus, when forwarding packets, passive TCP splicer doesn't have to modify the sequence number and acknowledgement number of TCP header.

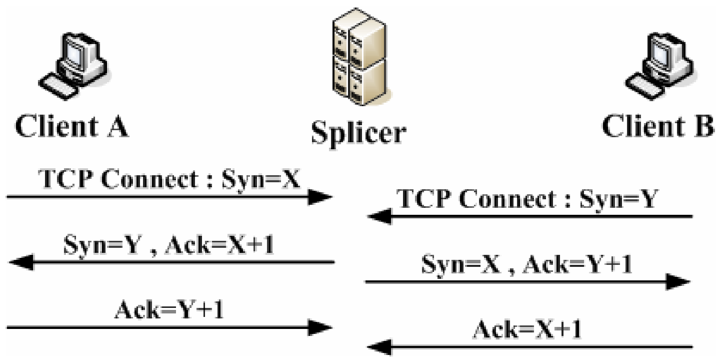
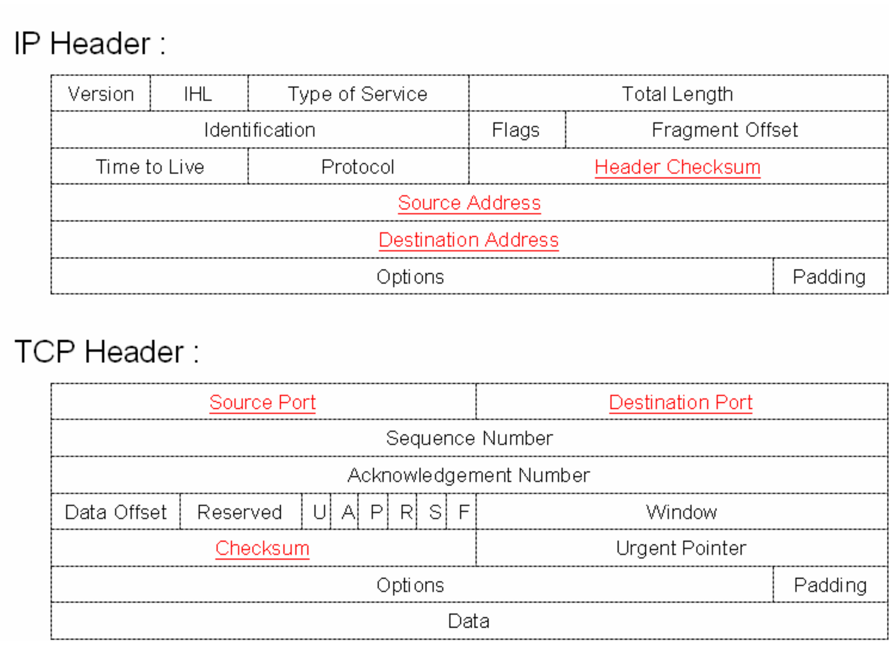


Figure 4.4: Splice TCP three-way handshake.



IP Header :

Version	IHL	Type of Service	Total Length	
Identification			Flags	Fragment Offset
Time to Live		Protocol	Header Checksum	
Source Address				
Destination Address				
Options				Padding

TCP Header :

Source Port								Destination Port	
Sequence Number									
Acknowledgement Number									
Data Offset	Reserved	U	A	P	R	S	F	Window	
Checksum								Urgent Pointer	
Options									Padding
Data									

Table 4.2: The fields need to be modified by passive TCP splicer.

The main difference between TCP Splice and Passive TCP Splice is that Passive TCP Splice listens to two active TCP connections from two client hosts, while TCP Splice creates another active connection after an incoming connection request. In addition, Passive TCP Splice does not modify the sequence and acknowledgement numbers in TCP header. The underlined fields in Table 4.2 need to be modified by passive TCP splicer.

Figure 4.5 depicts the sequence of message exchange when an IST implements Passive TCP Splice. Comparing with Figure 3.5, the main difference is that IST uses passive TCP splicer instead of application layer forwarding between service subscriber and service provider.

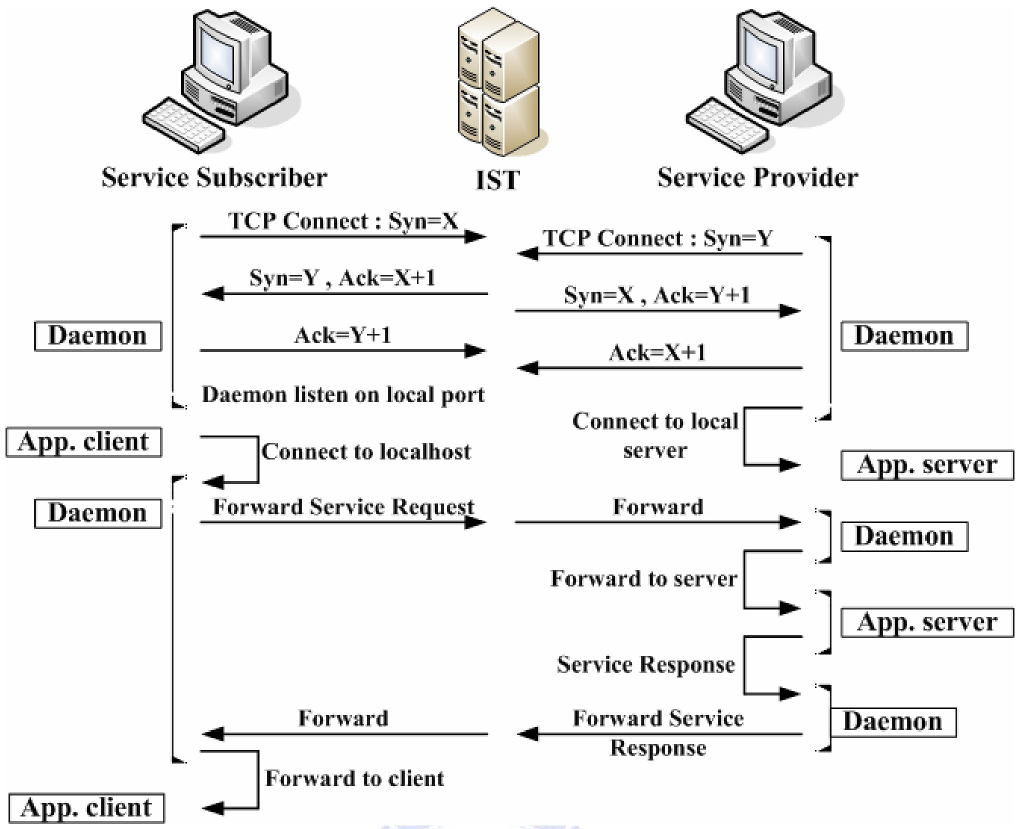


Figure 4.5: Implementation of passive TCP splice on IST.

4.4 Pure TCP Splice

We propose another performance improvement scheme for IST which is called *Pure TCP Splice*. Refined from Passive TCP Splice, Pure TCP Splice does not modify the TCP header except TCP checksum. Since TCP header fields except checksum are kept changeless, we consider that the Pure TCP Splice operates below transport layer. As Figure 4.6 shows, packet traversing a pure TCP splicer only touches network layer and link layer.

The pure TCP splicer is realized with a prerequisite. Only when one of the clients can assign the source port of its TCP connection and the other user knows this source port, then the splicer can perform Pure TCP Splice. Take Figure 4.5 as an example, client A uses source port X and destination port Y, and client B uses source port Y and destination port X to connect to the pure TCP splicer. Afterward, the pure TCP splicer can splice these two connections without modifying TCP port numbers.

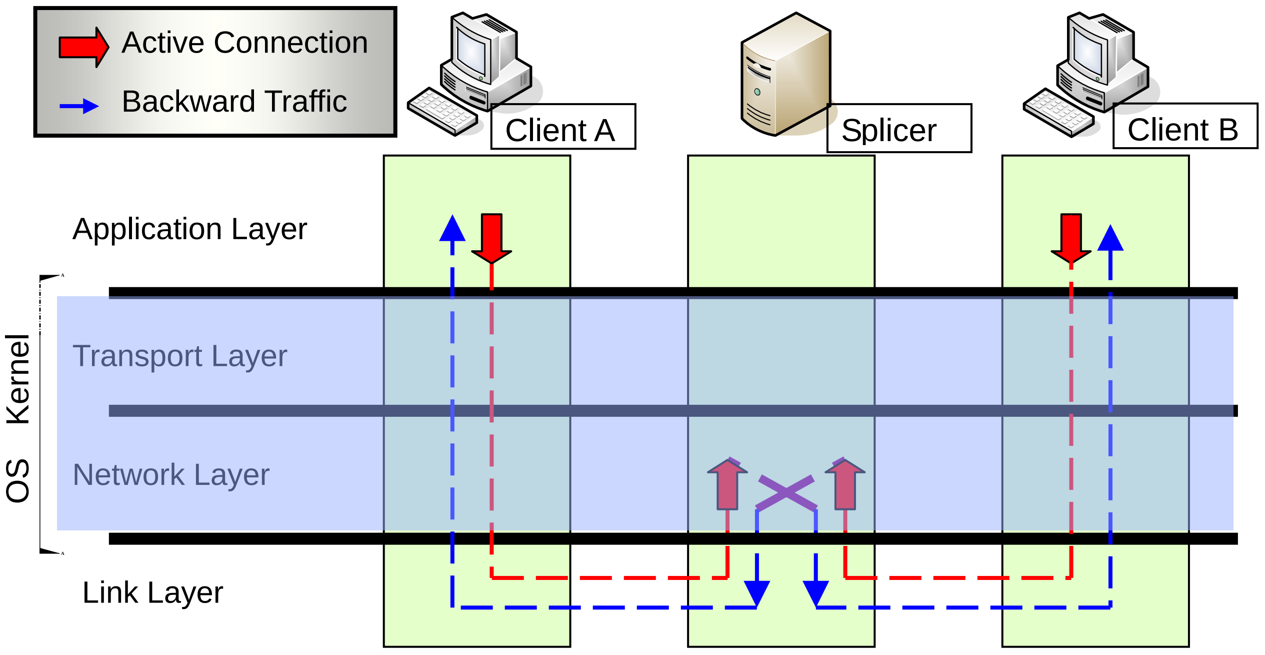


Figure 4.6: The path of packet traversing a pure TCP splicer.

As Figure 4.7 shows, client B can assign the source port of his TCP connection but client A cannot. Before creating the connections to IST, client B tells client A in advance that it would use P1 as his source port. However, client A responses to client B that it does not has the ability to assign the source port of its connection. Then, client A connects to IST port P1, and client B waits for pure TCP splicer to tell it the source port number of client A. When client B knows that the source port of client A is P2, it connects to port P2 of IST. Finally, pure TCP splicer obtains sufficient information of the sequence number of each user and is able to splice the three-way handshakes.

Table 4.3 depicts the fields of packet header which needs to be modified. When performing Pure TCP Splice, the only field needs to be modified in TCP header is the checksum. Since the source and destination addresses in IP header would be modified by pure TCP splicer and the TCP checksum contains a IP pseudo header, therefore, the splicer also modifies the TCP checksum.

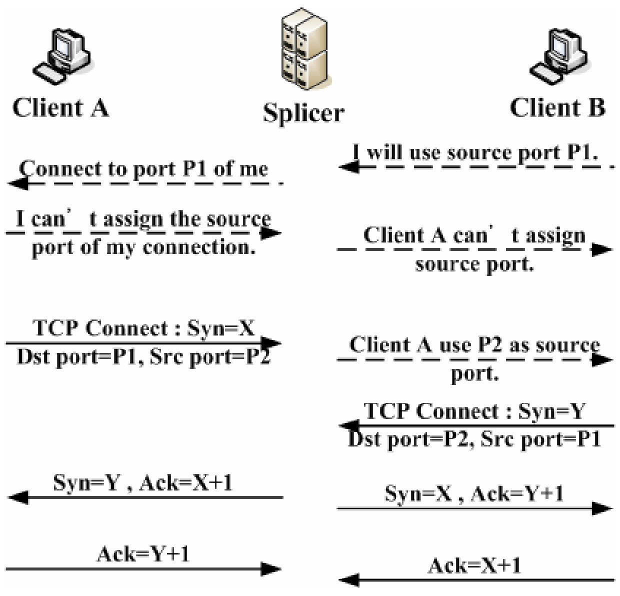
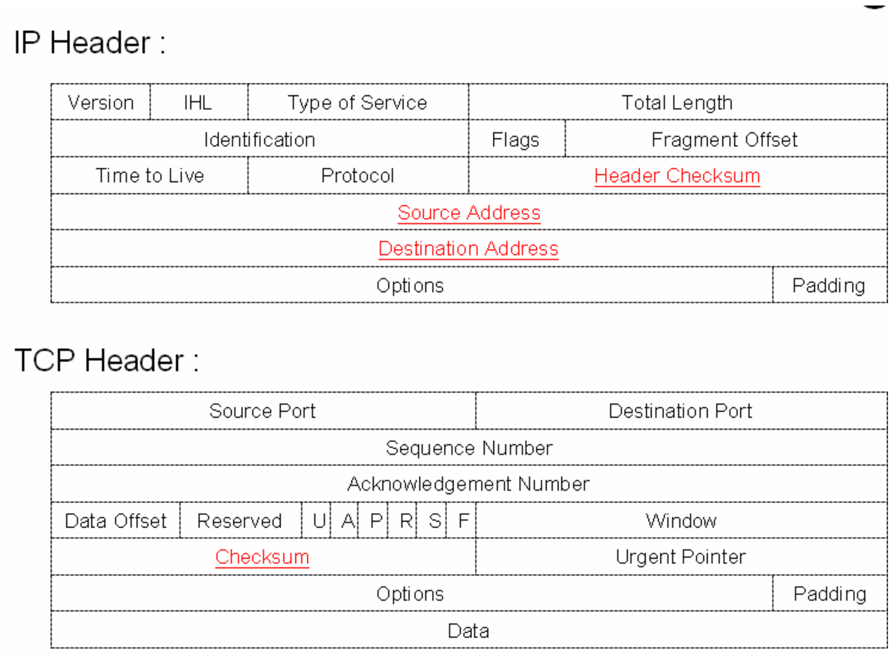

Client A
Splicer
Client B
Connect to port P1 of me
I will use source port P1.
I can't assign the source port of my connection.
Client A can't assign source port.
TCP Connect : Syn=X Dst port=P1, Src port=P2
Client A use P2 as source port.
TCP Connect : Syn=Y Dst port=P2, Src port=P1
Syn=Y , Ack=X+1
Syn=X , Ack=Y+1
Ack=Y+1
Ack=X+1

Figure 4.7: Port negotiation of Pure TCP Splice.



IP Header :

Version	IHL	Type of Service	Total Length
Identification		Flags	Fragment Offset
Time to Live		Protocol	Header Checksum
Source Address			
Destination Address			
Options			Padding

TCP Header :

Source Port								Destination Port
Sequence Number								
Acknowledgement Number								
Data Offset	Reserved	U	A	P	R	S	F	Window
Checksum								Urgent Pointer
Options								Padding
Data								

Table 4.3: The fields need to be modified by pure TCP splicer.

Chapter 5

Implementation and Emulation Results

In this chapter, we describe the implementation of our system components and emulation results. The system components include the user daemon, Internet Service Broker in application layer and the Internet Service Translator in OS kernel. It is not difficult to implement application layer ISB and user daemon. Section 5.1 gives a brief description of them. In Section 5.2, we present the implementation of Passive TCP Splice and Pure TCP Splice in OS kernel. The implementation in the kernel involves in the modification to TCP state transition. Our implementation is done on Intel IXP425 network processor running MontaVista Linux. Section 5.3 and Section 5.4 shows the emulation environment and results.

5.1 Implementation of Application Layer User Daemon and Internet Service Broker

The main functionality of ISB is to maintain the profile information of system users and help the negotiation between service providers and service subscribers. Since ISB runs at application layer, all functions are implemented by invoking system APIs and libraries. The ISB listens to a well known TCP port for client login messages. Clients create a TCP connection to login into ISB and keep this connection alive. Therefore, if there are n on-line users, ISB should maintain n persistent TCP connections. Then, ISB uses these connections to communicate with each client. ISB helps the service provider and service subscriber negotiate for accessing the services. We implement ISB on Linux host using Berkley socket APIs and ordinary system libraries.

The main functionality of the user daemon is to provide a user interface in our system. Users use the daemon to login into ISB and communicate with it for provisioning service to subscribers. The user daemon can intercept each message between system users and ISB. Therefore, when a user wants to either provide service or subscribe service, the user daemon can automatically creates a socket and listens on localhost for connection splicing. The user daemon of our implementation runs on Linux host using Berkley socket APIs and ordinary system libraries.

5.2 Implementation of Internet Service Translator in OS Kernel

We have described Passive TCP Splice and Pure TCP Splice in Section 4.3 and Section 4.4 respectively. In this section we describe the implementation of them in OS kernel. Although packets traversing a passive TCP splicer would touch transport layer, from the view point of kernel implementation we can implement it in network layer, and only when splicing the three-way handshake the packets would touch transport layer. Likewise, packets traversing a pure TCP splicer would only touch the network layer, and in kernel implementation the packets would touch transport layer when splicing three-way handshake.

Figure 5.1 from Steven's [18] shows the original TCP state transition diagram. The start state is "CLOSED." When a host starts to listen to an incoming connection request, the state transits from "CLOSED" to "LISTEN". Once the host receives the first packet (SYN) of three-way handshake, it should immediately send back the second packet (SYNACK) of three-way handshake and the state transits from "LISTEN" to "SYN_RCVD". If the remote host sends an ACK packet later, the three-way handshake is completed and the state transits from “SYN_RCVD” to “ESTABLISHED”.

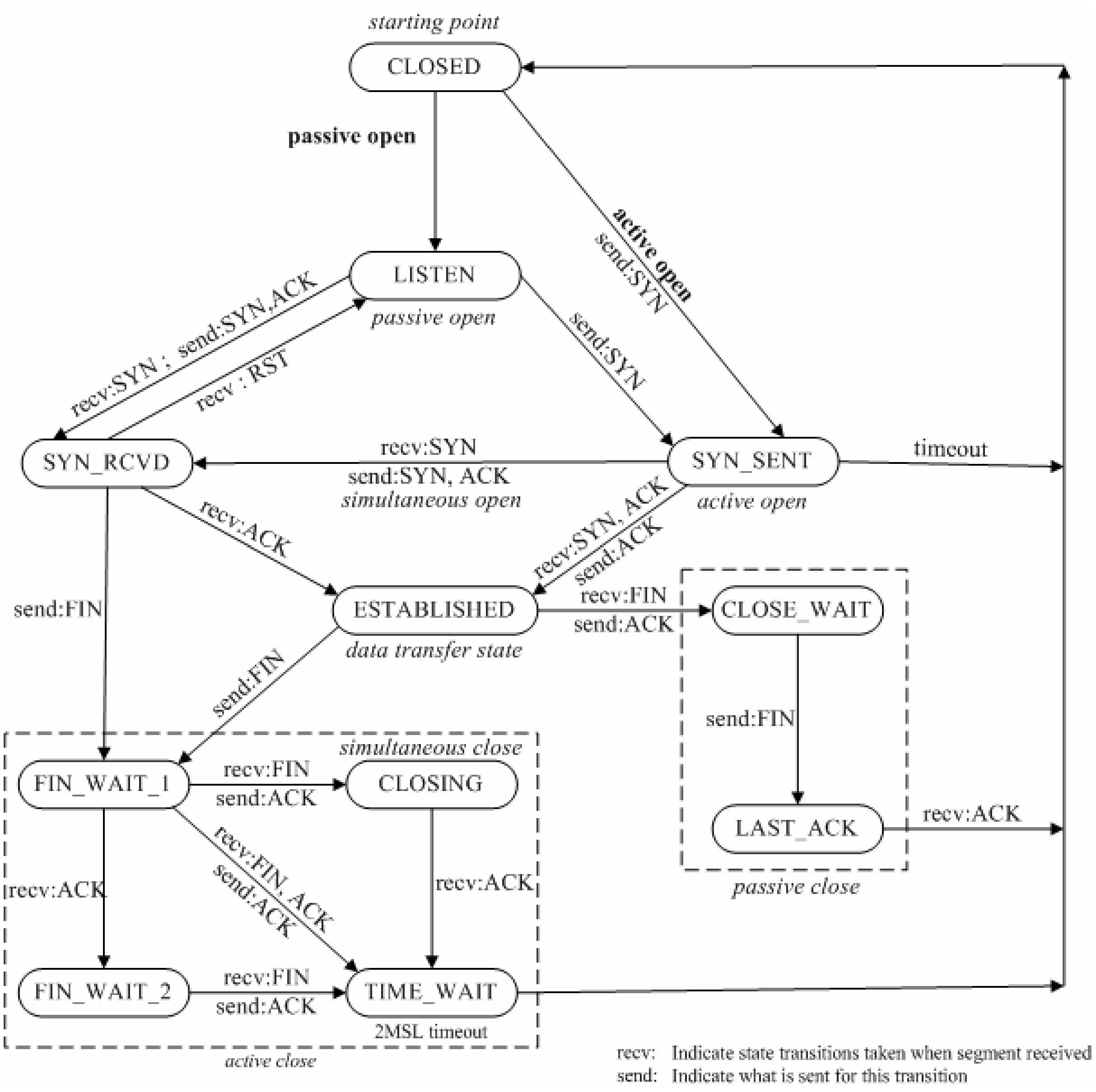


Figure 5.1: Original TCP state transition diagram.

The termination of a TCP connection can be bidirectional. Ether localhost or remote host sends a FIN packet to terminate the connection in one direction. The other host can continually send data, and the TCP connection is terminated when this host sends a FIN packet later. When a localhost wants to stop sending message first, a FIN packet is sent to the remote host and the state transits from “ESTABLISHED” to “FIN_WAIT_1” to keep receiving message from the remote host. Otherwise, if the remote host wants to terminate first, localhost would receive a

FIN and the state changes from “ESTABLISHED” to “CLOSE_WAIT”. Finally, after accomplishing the termination mechanisms, the state goes back to “CLOSED”. The detail state transition mechanism is depicted in Steven’s [18].

Figure 5.2 shows the modified TCP state transition diagram of IST for splicing TCP connections. We add four new states which are in gray color. When the TCP state is “LISTEN” and a SYN is received, the kernel checks the port number to determine whether this packet is for the IST. If it is for IST, the TCP state changes from “LISTEN” to “IST_SYN_LOOKUP”, and the IP addresses, TCP ports and sequence number are kept in an IST table. Subsequently, when another SYN arrives and it is also for IST, its IP address, TCP port and sequence number will be kept in the IST table. Then two SYNACK are built based on the information in the IST table and sent back to the remote hosts, and the TCP state transits from “IST_SYN_LOOKUP” to “IST_SYN_RCVD”. If IST receives two ACKs later, the connection splicing is successful and the TCP state changes from “IST_SYN_RCVD” to “ESTABLISHED”. To support this mechanism, we need to modify the tcp_rcv_state_process ( ), tcp_conn_request ( ) and tcp_v4_do_rcv ( ) functions of Linux kernel to change the behavior of connection establishment; all these functions are performed in transport layer.

The “ESTABLISHED” state is modified to forward message for spliced connections. In kernel implementation, we need only to modify the ip_local_deliver_finish ( ) function which is the final function of ip_input procedure. When ip_local_deliver_finish ( ) receives a packet, it check the IST table and updates both IP header and TCP header, then calculates the checksum and sends the packet out. All mechanisms are performed in the network layer, therefore, the packet would not touch the transport layer.

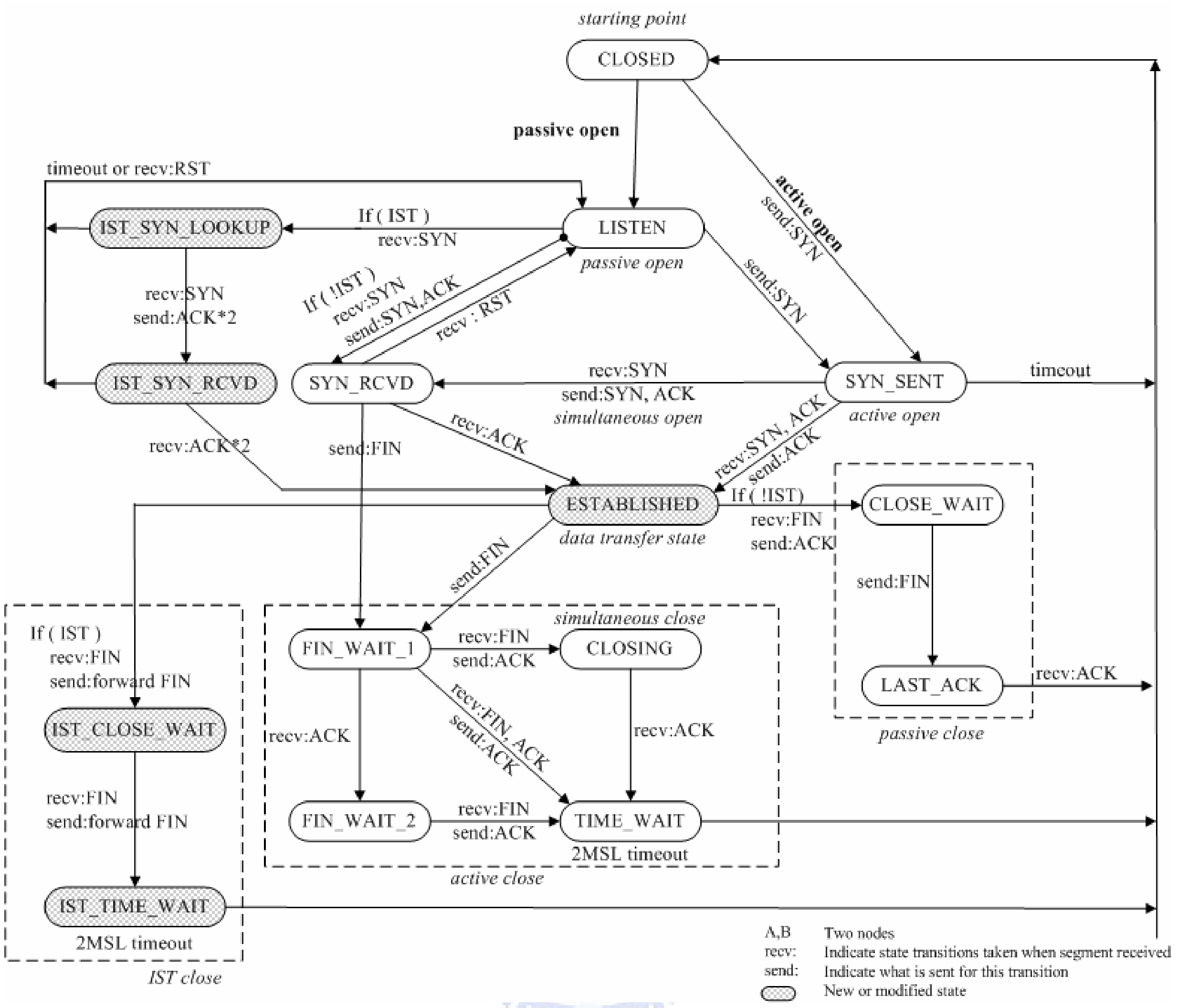


Figure 5.2: Modified TCP state transition diagram for splicing TCP connection.

When IST receives a FIN packet, it changes the TCP state from “ESTABLISHED” to “IST_CLOSE_WAIT” and stops forwarding message from the remote host. When the other host also sends a FIN packet to the IST, the IST changes the TCP state from “IST_CLOSE_WAIT” to “IST_TIME_WAIT” and stops forwarding message except the final FINACK. After waiting for 2MSL, the state goes back to “CLOSED”.

The kernel implementation for Passive TCP Splice and Pure TCP Splice is the same except that Pure TCP Splice does not modify source and destination ports in the TCP header. The information of addresses and ports is obtained from IST table, and we implement this IST table using hash. The checksum of both IP header and TCP header should be recalculated. It is a large overhead to go through the whole IP header and payload to recalculate IP checksum and TCP checksum. The theorem of checksum and checksum modification method and pseudo code are described in [19, 20, 21].

The RFC recommends the following equation for computing the updated checksum *C'* from the original checksum *C*, and *m*, *m'* are the old and new field values respectively. The values of *m* in our system are old IP address and old TCP port, while the values of *m'* are new IP address and new TCP port.

$$C' = C + (m' - m) \tag{5.1}$$

Although the equation above is correct, it is not very useful for incremental updates since the equation above updates the checksum C, rather than the 1's complement of the checksum ~C, which is the value stored in the checksum field. The useful calculation for checksum modification is:

$$\sim C' = \sim(C + (-m) + m') = \sim C + (m - m') = \sim C + m + \sim m' \tag{5.2}$$

Equation 5.2 means that the new value of checksum field is the old checksum plus the old field value plus one's complement of the new field value. We implement this equation in our IST kernel code to update IP and TCP checksum. Since the IP addresses and TCP ports are fix values in IST table, we can pre-calculate the value of $m + \sim m'$ to speedup checksum modification.

There is another implementation issue of our IST: the TCP timeout period is 21 seconds (in Microsoft Windows XP implementation). TCP timeout means that the first user sends a SYN to IST and waits for response, however, it waits longer than the TCP timeout period and the other user does not send a SYN to IST. Therefore, IST does not send back SYNACKs and the first remote host will encounter TCP timeout. The TCP timeout period is dependent of TCP implementation in OS and maybe longer or shorter than 21 seconds. Since less than few seconds a packet can traverse the whole Internet and the overhead of IST splicing three-way handshake is not heavy, we believe that it is sufficient for most TCP implementations in various OSs to use our mechanism.

The implementation of UDP forwarding in OS kernel is almost the same as TCP forwarding except that the three-way handshake can be ignored. We modify the udp_rcv ( ) function to keep the information of IP addresses and UDP ports in an IST table. The whole UDP forwarding functionalities are implemented in ip_input procedure in network layer, it is like TCP forwarding discussed above.

5.3 Emulation Platform

We use Intel IXP425 network processor [22] for emulation and the software is running on MontaVista Linux [23]. There are two reasons for choosing IXP425 for our emulation. One is that IXP425 is an embedded processor commonly used for small network device. Section 3.6.3 describes that we can get great benefit from combining home gateway box with the functionalities of user daemon and IST. The other reason is that IXP425 has limited computing resource, and the performance improvement of our Passive TCP Splice and Pure TCP Splice would be more obvious on IXP425.

Intel IXP425 network processor is a highly integrated, versatile single-chip processor that can be used in variety of products that need network connectivity and high performance. It has an XScale core processor operating at 533 MHz, two integrated 10/100 Base-T Ethernet MACs, 33/66 MHz PCI v2.2 bus, SDRAM controller supports from 8 to 256 Mbytes of SDRAM memory and many other functionalities as shown in Figure 5.3. We use IXP425 develop platform (IXDP425) to implement our system which has 256 M**b**ytes memory, PCI slots and two on-board Ethernet PHYs.

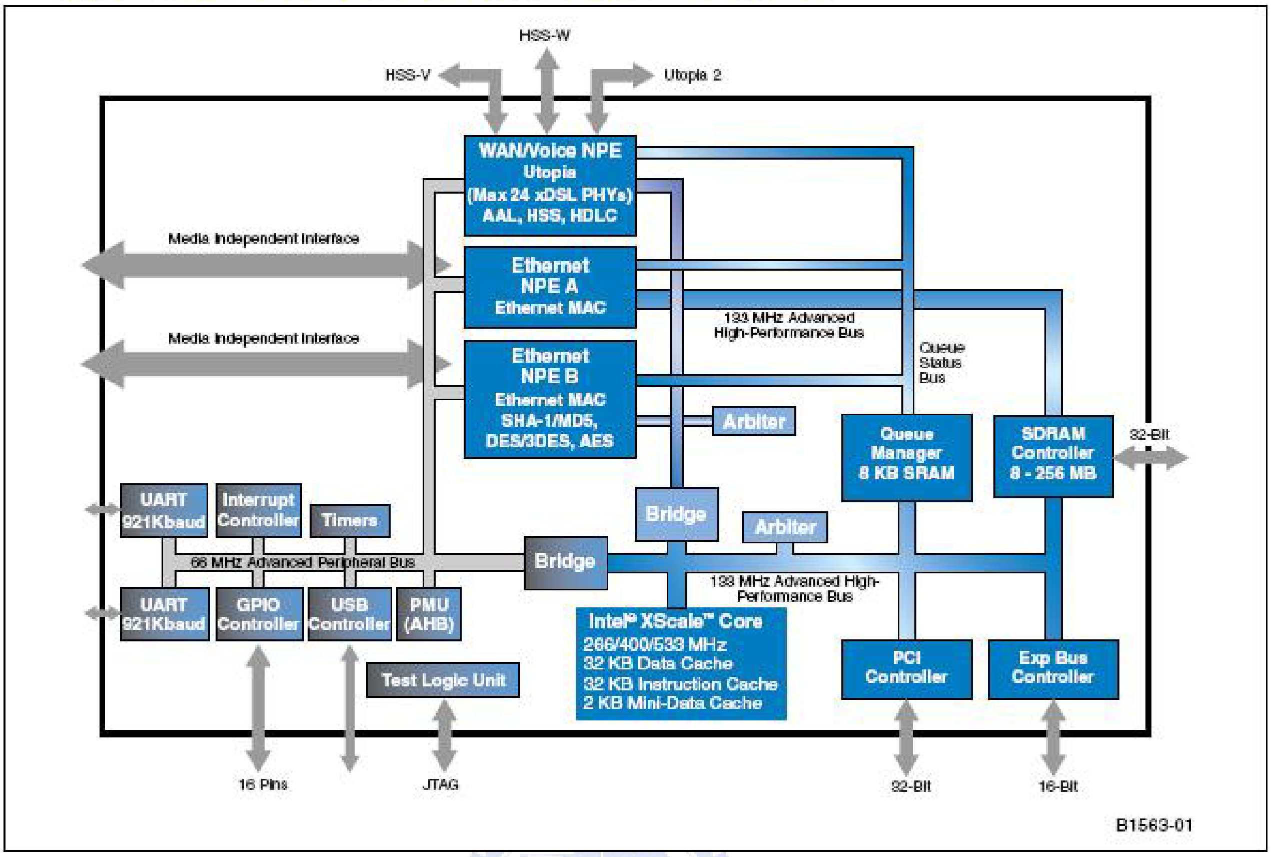

HSS-W
HSS-V
Utopia 2
WAN/Voice NPE Utopia (Max 24 xDSL PHYs) AAL, HSS, HDLC
Media Independent Interface
Ethernet NPE A Ethernet MAC
Media Independent Interface
Ethernet NPE B Ethernet MAC SHA-1/MD5, DES/3DES, AES
133 MHz Advanced High-Performance Bus
Queue Status Bus
Arbiter
Queue Manager 8 KB SRAM
SDRAM Controller 8 - 256 MB
32-Bit
UART 921Kbaud
Interrupt Controller
Timers
Bridge
Arbiter
66 MHz Advanced Peripheral Bus
Bridge
133 MHz Advanced High-Performance Bus
UART 921Kbaud
GPIO Controller
USB Controller
PMU (AHB)
Intel® XScale™ Core 266/400/533 MHz 32 KB Data Cache 32 KB Instruction Cache 2 KB Mini-Data Cache
Test Logic Unit
PCI Controller
Exp Bus Controller
16 Pins
JTAG
32-Bit
16-Bit
B1563-01

Figure 5.3: Block diagram of Intel IXP425 network processor.

As shown in Figure 5.3, there are two on-chip Ethernet MAC on IXP425. These two Ethernet MACs connect to XScale processor via 133 MHz Advanced High-Performance Bus and a Queue Manager. A PCI Controller also connects to this High-Performance Bus; we can plug in additional PCI NICs to extend the number of ports of IXDP425. From the emulation experience, we observe that if the IST is implemented in kernel layer and forwards the traffic between two on-chip Ethernet MACs, the throughput would be kept as high as the full line speed which is about 98Mbps. This result means that the XScale processor and the High-Performance Bus are capable of handling all traffic without discarding packets. If the IST forwards traffic between one on-chip Ethernet MAC and a PCI Ethernet NIC, some packets would be discarded. The main reason for these two different results may be due to the on-chip Ethernet MAC which has faster response time and higher interrupt priority than PCI NIC. When traffic comes from one on-chip Ethernet MAC and goes to one PCI Ethernet NIC, the smaller response time and higher interrupt priority of on-chip Ethernet MAC and the CPU processing latency would lead to packet loss in kernel buffer because packets come in faster than they go out. If the traffic is in the opposite direction, packets would be dropped on PCI NIC buffer due to the lower interrupt priority and longer response time and the CPU processing latency. In this thesis, the emulation results are all performed by forwarding traffic between one on-chip Ethernet MAC and a PCI Ethernet NIC. This is to emphasize our implementation which reduces CPU processing latency and features better performance than the normal implementation. And the emulation of forwarding traffic between two on-chip Ethernet MACs is not conducted because that throughput is equal to the maximal line speed.

5.4 Emulation Result

We compare the performance of IST implemented in application layer as well as in kernel layer. Since application layer forwarding requires at least four memory copies to forward a packet and TCP Splice in OS kernel only needs two memory copies, the latter would obviously feature better performance than the former. The main purpose of this emulation is to show how much improvement our method can achieve on a network device with limited computing power.

We estimate the improvement of our method by performing the emulation in five phases: Direct Connect, Forwarder, NAT, IST in Application Layer and IST in OS Kernel. The topologies are shown in Figure 5.4. Host A and host B are normal PCs which have 256 Mbytes memory and operate at 568 MHz. We test Direct Connect performance by measuring the maximal throughput between host A and host B. The forwarder here is the forwarding functionality in Linux kernel. Once the functionality is turned on, Linux will forward packets from each NIC to all NICs. We test the performance of Linux forwarder to get the maximal throughput that IXP425 can achieve. The third phase is NAT. We turn on the NAT ability of Linux kernel to test its maximal throughput. Since the Linux kernel is finely tuned, we consider the throughput of Linux NAT as the best throughput a forwarder can achieve when it needs to check some information in a table while forwarding a packet. The fourth and fifth phases are implementation of IST in application layer and kernel layer respectively. We compare the results of these two phases to show the improvement of our method. Since the implementation of IST in application layer is similar to Skype [9], we can consider the emulation result of phase four as the performance of Skype when applying to general purpose solution.

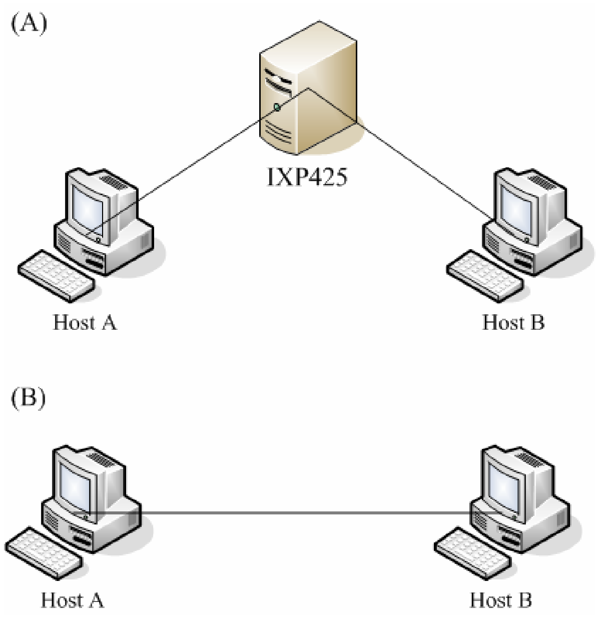


Figure 5.4: Emulation Topologies. (A): Forwarder, NAT and IST. (B): Direct Connect

Figure 5.5 shows the comparison of round trip time of five phases. The results are measured by a UDP Ping-Pong program. The Direct Connect phase has the lowest RTT. The RTT of forwarder and NAT phases is more than one hundred nano-seconds longer than that of Direct Connect. This result shows that the forwarder and NAT implementation of Linux Kernel is well tuned and the IXP425 processor is capable of doing this job, because it needs only about one hundred nano-seconds to duplicate and transmit a packet. The RTT of our IST implementations in application layer and kernel layer are about 498 and 423 nano-seconds. The long latency of IST in application layer is obvious since it requires at least four memory copies.

However, the RTT of IST in kernel layer is still up to 425 nano-seconds which may be considered as too long. We believe the long RTT of IST in kernel layer is due to our kernel code that has not been finely tuned. If we carefully tune our kernel code, the best RTT of IST in kernel maybe reduced to equal to the RTT of NAT.

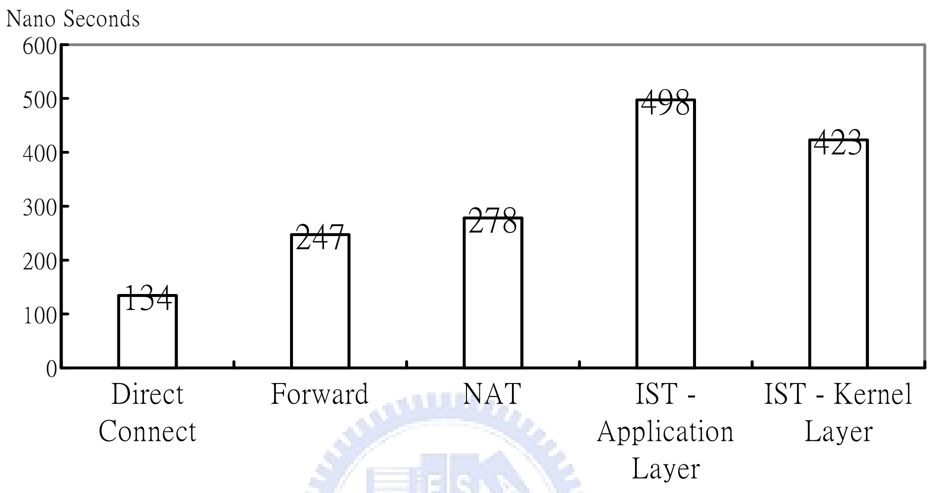


Figure 5.5: Comparison of Round Trip Time.

We test the TCP throughput of five phases above. One host creates one TCP connection, ten TCP connections, twenty TCP connections … one hundred TCP connections to the other host via either a direct link or a forwarder and sends huge volume of data. Furthermore, the sender estimates the total number of data it sends in a time period as transmission throughput; and the receiver estimates the total number of data it receives in a time period as receiving throughput. Finally we average these two throughputs as the final result. Figure 5.6 shows the results of TCP throughput. It is obvious that only IST in application layer phase has poor performance, while IST in kernel layer phase performs as well as first three phases. The main reason of IST in application layer phase has such poor performance may be due to the fact that it requires additional two memory copies (kernel layer to application layer and application layer to kernel layer). This emulation result also shows that if we use Skype to transmit normal application data instead of voice data, we would get very poor performance.

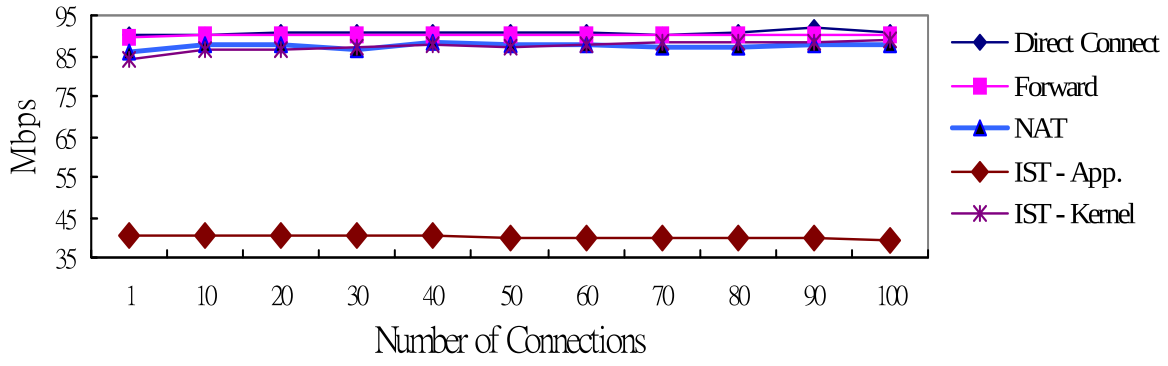


Figure 5.6: Comparison of TCP throughput.

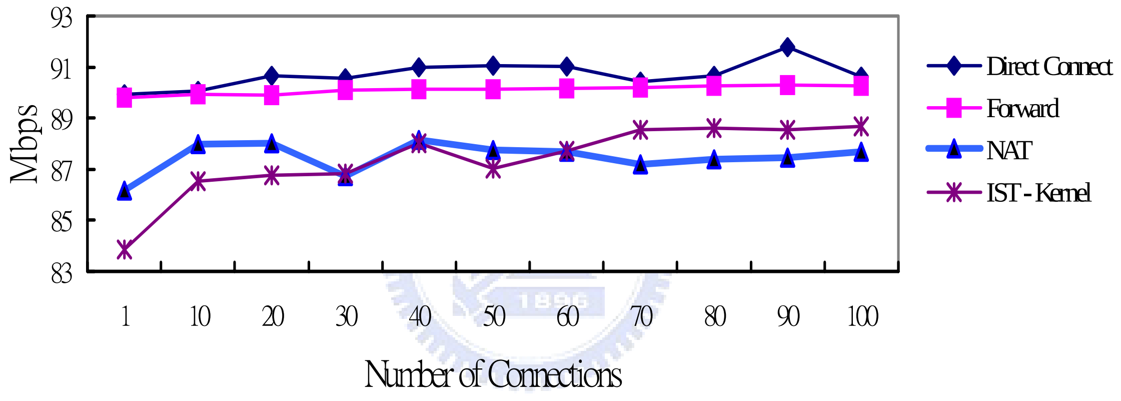


Figure 5.7: Magnification of TCP throughput comparison.

Figure 5.7 magnifies Figure 5.6 and shows that Direct Connect phase has the best throughput. There is a trend that IST in kernel layer would perform better than NAT when the number of connections increases. This maybe due to that NAT needs to lookup its NAT mapping table to forward a packet. When the mapping table becomes bigger, it may require a little more table lookup time to lookup this table. Our IST kernel implementation uses hash table to maintain the connection splicing information. When the number of connections becomes larger, the table lookup time would not increase. With the trend that TCP throughput increases with the number of connections, IST kernel implementation performs better than

NAT when the number of connections is bigger than 70.

We test the UDP performance by using one host sending variable data rate to another host and measure the packet loss rate. The emulation experience shows an interesting phenomenon, the packet size plays an important role when measuring the UDP loss rate. When the packet size is bigger than a certain threshold, the UDP loss rate is kept persistent no matter how light the sending rate is.

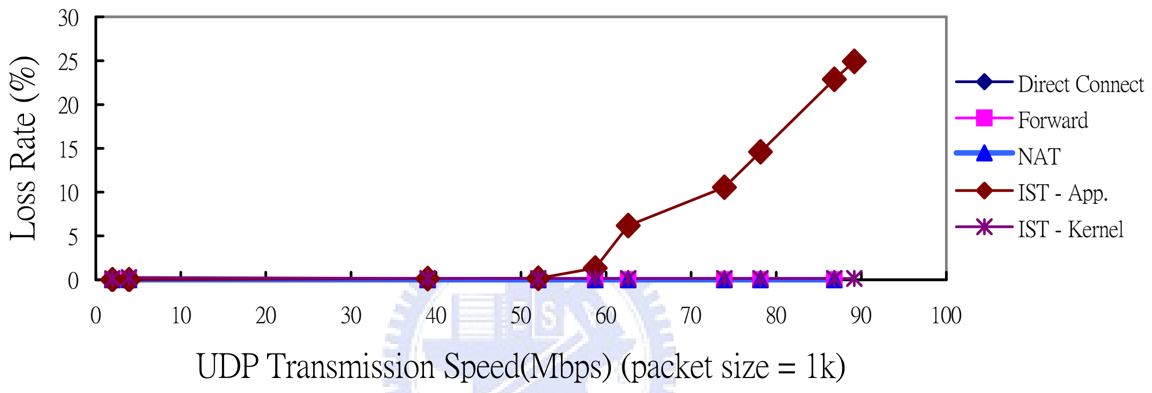


Figure 5.8: Comparison of UDP loss rate with 1Kbyte packet size.

Figure 5.8 shows the UDP loss rate of variable sending rate when packet size equals 1 Kbytes. Only IST implementation in application layer would drop packets, while IST kernel implementation would not. This result shows that IXP425 is sufficient to perform the functionality of IST when it runs MontaVista Linux kernel. However, when the packet size increases up to 6 Kbytes, NAT phase and IST kernel layer phase drop packet when the sending rate is larger than 63 Mbps. The overhead of memory copy is so enormous and IST application layer phase drops almost all packets no matter what the data rate is.

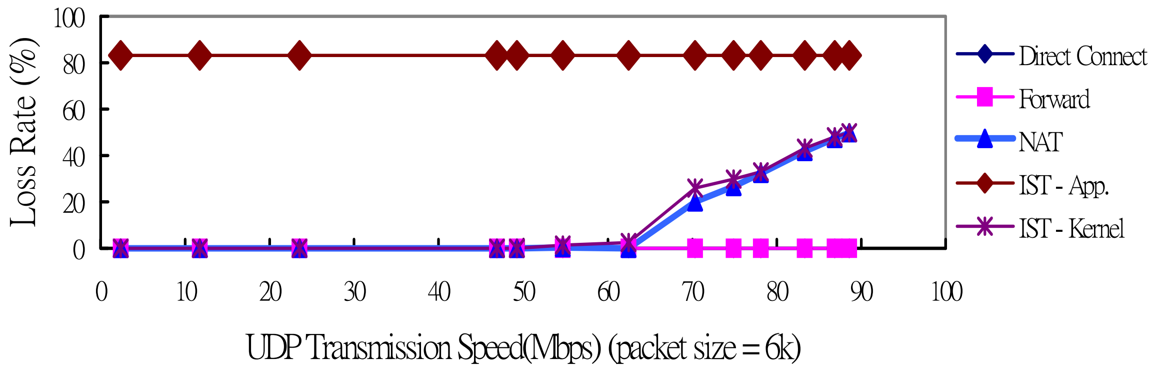


Figure 5.9: Comparison of UDP loss rate with 6Kbyte packet size.

Figure 5.10 shows the UDP loss rate with variable packet sizes and UDP is sending at full line speed. Direct Connect phase and Forward phase perform well and never drop packets. The performance of IST kernel implementation is similar to NAT since they do almost the same job (modification to packet header and recalculate checksum). The performance of IST application layer implementation is the worst due to the overhead of memory copies.

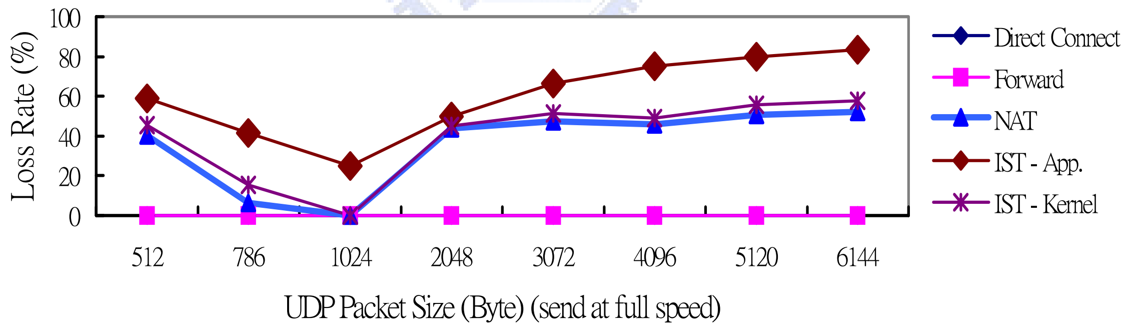


Figure 5.10: Comparison of UDP loss rate with variable packet sizes.

It is interesting that when packet size is less than 1 Kbytes the UDP loss rate increases rapidly. We consider the reason of this phenomenon is similar to the “Live Lock” problem of operating system. Since the emulation is performed by sending UDP packets with maximal rate, the IXP425 needs to handle too many small packets at the same time and finally drops some of them. This figure also shows that Linux kernel is optimized for handling packets with size around 1 Kbytes. Linux kernel adjusts its data structures and default buffer size to optimize it whole system performance.

When packet size is bigger than 1KB, the UDP loss rate also increases. We consider it is due to the overhead of memory copies. The bigger the packet size is, the more memory copy latency would be required.

Since the default Ethernet MTU (Maximum Transmission Unit) of most OSs are 1500 bytes, we can consider that our IST kernel implementation on IXP425 network processor is very capable of performing the functionalities of IST. And we can deploy ISTs to each system users' home network as their home gateway to improve the whole system performance.

Chapter 6

Conclusion and Future Work

In this thesis, we design and implement the infrastructure of Internet Service Broker for NAT traversal. There are many related techniques devised to NAT traversal problem. However, each of them has some drawbacks or inconvenience. We propose an infrastructure of Internet Service Broker to overcome these disadvantages and provide a general purpose solution which can be applied to all sorts of existing network applications. The key design guideline is to make our solution feasible for deployment. Users of our system only need to execute an application layer daemon and login to the Internet Service Broker located in globally-addressable Internet, and existing network applications can get the capability of NAT traversal without any modification. In addition, our system can be also applied to Firewall Traversal and IPv4 / IPv6 translation.

We propose Passive TCP Splice and Pure TCP Splice to improve the forwarding performance of third party node for our system. Furthermore, our system can utilize P2P structure to improve scalability. The emulation result shows that we can apply our TCP Splice mechanisms to the computing power limited network device and achieve satisfactory performance.

Since the emulation result only demonstrates that our Internet Service Translator can be implemented to achieve good performance and achieve better system scalability, it still can not guarantee the whole system performance when there are huge number of system users. Our future work is to implement our system in P2P structure. The main objective is to let every system users to perform the functionalities of centralized servers. In addition, we have to propose an algorithm for Internet Service Broker to help system users find the closest Internet Service Translator. Once these works were accomplished, our system can provide the guaranteed quality of service.

Reference

[1] V. Fuller, T. Li, J. Yu, and K. Varadhan, “Classless Inter Domain Routing (CIDR): an address assignment and aggregation strategy,” *IETF RFC 1519*, Sept. 1993.

[2] P. Srisuresh and K. Egevang, “Traditional IP Network Address Translator (Traditional NAT),” *IETF RFC 3022*, January 2001.

[3] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley and E. Schooler, “SIP: Session Initiation Protocol,” *IETF RFC 3261*, June 2002.

[4] M. Leech, M. Ganis, Y. Lee, R. kuris, D. Koblas, and L. Jones, “SOCKS protocol version 5”, *RFC 1928*, April 1996.

[5] E.S. Lee, H.S. Chae, B.S. Park and M.R. Choi, “An Expanded NAT with Server Connection Ability”, *TECON 99*, Proceedings of the IEEE Region 10 Conference, Volume: 2, pages 1391 - 1394, Sept. 1999.

[6] V. Pai and P. Rana, “A Transparent Framework for Enabling Incoming TCP Connections to Hosts Behind a NAT Gateway,” *IEEE Computer Communications and Networks*, 2003. ICCCN 2003. Proceedings. The 12th International Conference, pages 572 – 575, October 2003.

[7] Microsoft Corporation, UPnP – Universal Plug and Play Internet Gateway Device v1.01,

Nov. 2001. Available online http://www.upnp.org/standardizeddcps/documents/ UPnP_IGD_1.0.zip. 30 April 2004.

[8] P. Francis and R. Gummadi, "IPNL: A NAT-Extended Internet Architecture," *ACM SIGCOMM Computer Communication Review*, Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, Volume 31 Issue 4, August 2001.

[9] Skype, http://www.skype.com.

[10] S. Guha, Y. Takeda and P. Francis, "NUTSS: A SIP-based Approach to UDP and TCP Network Connectivity," *ACM SIGCOMM*, Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture, pages 43 – 48, August 2004.

[11] KaZaa, http://www.kazaa.com.

[12] S.A. Baset and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," Available online http://arxiv.org/ftp/cs/papers/0412/0412017.pdf, Sept. 2004.

[13] J. Rosenberg, R. Mahy, and C. Huitma, "TURN – Traversal Using Relay NAT," *IEEE Internet draft*, Feb. 2004.

[14] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "STUN – Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," *IEEE RFC 3489*, Mar. 2003.

[15] Netperf, http://www.netperf.org/netperf/NetperfPage.html.

[16] D. Maltz and P. Bhagwat, "TCP Splicing for Application Layer Proxy Performance," IBM Research Report, Mar. 1998.

[17] B.Y. Zhao, J. Kubiatowicz and D. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," *Selected Area in Communications*, IEEE Journal on Volume 22, Issue1, Jan. 2004.

[18] Richard Steven, “TCP/IP Illustrated Volume 1,” Addison Wesley.

[19] B. Braden, D. Borman and C. Partridge, “Computing the Internet Checksum,” *IEEE RFC 1071*, Sep. 1988.

[20] T. Mallory and A. Kullberg, “Incremental Updating of the Internet Checksum,” *IEEE RFC 1141*, Jan. 1990.

[21] K. Egevang and P. Francis, “The IP Network Address Translator (NAT),” *IEEE RFC 1631*, May 1994.

[22] Intel IXP425 Network Processor, http://www.intel.com.tw/design/network/products/npfamily/ixp425.htm

[23] MontaVista Linux, http://www.mvista.com/

[24] I. Stoica, R. Morris, D. Karger, M. Frans and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,” *ACM SIGCOMM*, Proceedings of the ACM SIGCOMM, Volume 11, Issue 1, Feb. 2003.

[25] S. Ratnasamy, P. Francis, M. Handley and R. Karp, “A Scalable content-Addressable Network,” *ACM SIGCOMM*, Proceedings of the ACM SIGCOMM, Volume 31, Issue 4, Aug. 2001.