

國立交通大學

資訊科學與工程研究所

碩士論文



NAT 行為感知的 TCP 穿越機制

NBA – NAT Behavior Aware TCP Traversal Scheme

研究生：劉坤穎

指導教授：曾建超 教授

中華民國 九十九年 七月

N A T 行 為 感 知 的 T C P 穿 越 機 制
NBA – NAT Behavior Aware TCP Traversal Scheme

研 究 生：劉坤穎
指 導 教 授：曾建超

Student : Kun-Ying Liu
Advisor : Chien-Chao Tseng

國 立 交 通 大 學
資 訊 科 學 與 工 程 研 究 所



Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2010

Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 九 年 七 月

N A T 行 為 感 知 的 T C P 穿 越 機 制

學生：劉坤穎

指導教授：曾建超 教授

國立交通大學資訊科學與工程研究所

摘 要

本論文提出一套 Network Address Translation (NAT) 行為感知 (NAT Behavior-Aware, NBA) 的 TCP 穿越機制，當兩端點分別位於不同 NAT 底下時，若兩者想嘗試建立一條 TCP 的直接連線，NBA 機制會利用兩端的 NAT 資訊，從候選的 TCP NAT 穿越方法中，找尋一個最恰當的穿越技術，進行直連測試。因為 NBA 洞悉連線兩端 TCP NAT 穿越的支援能力，可以避免使用不可能成功的穿越技術來執行直連測試，減少直連測試所花掉的時間與資源。

許多研究已經提出解決 TCP NAT 穿越問題的方法，然而這些方法並沒有將 NAT 的 TCP 狀態追蹤特性列入考量，對於不同 NAT 組合下的適用性亦一無所知，如果盲目嘗試這些 TCP NAT 穿越方法進行直連測試，試圖找到一條直連路徑，會導致連線測試時間冗長與不必要的訊息交換等問題，進而影響到 NAT 穿越的效率與成功率。

為了縮短連線測試延遲與降低訊息的交換量，以及提高直接連線的比率，我們提出一套 NAT 行為感知 (NBA) 的 TCP 穿越機制。NBA 的主要構想是本機端 (host) 的使用者代理人 (user agent, UA) 先收集當地的 NAT 資訊，包含 NAT 的 mapping 行為、filtering 行為與 TCP 狀態追蹤特性，當位於不同 NAT 底下的兩 UA

想要嘗試穿越 NAT 時，NBA 可以利用此兩 UA 所收集到的 NAT 資訊，選擇出一個最恰當的 NAT 穿越技術，並通知這兩 UA 使用。如此一來，這兩 UA 就可以省去執行不可能成功的 NAT 穿越技術的測試時間與系統資源。

我們已經完成 NBA 機制的實作，並針對直連率、測試時間與資源使用量等效能指標，進行 NBA 與循序直連測試(Sequential Connectivity Check, SCC)以及平行直連測試(Parallel Connectivity Check, PCC)兩種機制的效能比較。實驗結果顯示，在相同組合的 NAT 環境下，這三種機制的直連率完全相同，亦即，NBA 選擇 NAT 穿越技術的方法非常精確，不會發生誤選的情況。其次，當進行直接連線測試時，NBA 與 SCC 相較之下具有更短的測試時間延遲，且 NBA 比 PCC 使用更少的資源，故 NBA 的整體效能表現較 SCC 和 PCC 更傑出。

關鍵字: Network Address Translation、NAT、NAT Traversal、TCP、TCP NAT Traversal



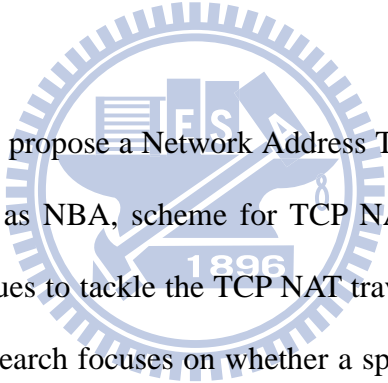
NBA – NAT Behavior Aware TCP Traversal Scheme

Student : Kun-Ying Liu

Advisor : Dr. Chien-Chao Tseng

Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University

ABSTRACT



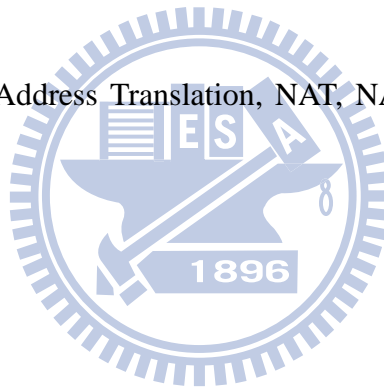
In this thesis, we propose a Network Address Translation (NAT) Behavior Aware, henceforth referred to as NBA, scheme for TCP NAT Traversals. Many researchers have proposed techniques to tackle the TCP NAT traversal problem. However, previous TCP NAT traversal research focuses on whether a specific TCP signaling sequence can establish a direct connection successfully between two peers behind NATs. Because each signaling sequence has its own applicable NAT types, brute force connectivity check may induce a long delay or excessive message exchanges for setting up a connection.

Therefore, NBA utilize TCP state tracking behaviors of NATs as *a priori* knowledge to select the most appropriate Traversal method for the connectivity check between two communicating peers behind. As a consequence, it can eliminate unnecessary checks, shorten the connectivity check delay, reduce the number of message exchanges, and sometimes help to avoid failure in connectivity check that ought to succeed. With NBA, user agents (UAs) collect the NAT information such as mapping rules, filtering

rules and TCP state tracking behaviors, and report the collected information to an NBA server. When two UAs intend to establish a communication session, the server consults the information reported by the two UAs, determines the best traversal method, if exists, and informs the two UAs to check connectivity with the selected method.

We have implemented NBA and compared the performance of NBA with both sequential connectivity check (SCC) scheme and parallel connectivity check (PCC) schemes. The experimental results show that NBA achieves the same direct connection ratio as SCC and PCC do. Furthermore, NBA outperforms SCC in terms of latencies and PCC in system resources utilizations for connectivity checks.

Keywords: Network Address Translation, NAT, NAT Traversal, TCP, TCP NAT Traversal



致 謝

碩士班兩年的生活，隨著此論文的完成，即將畫上句點。從下定決心轉戰資工所的那一刻起，我就不曾後悔過，不管遇到多大的難關與障礙，相信自己都能想盡各種方法克服它。幸虧我遇到了一位好教授，一群好夥伴和好同事，也與我的女朋友相遇在碩士班的時光裡。真的不虛此行，感謝這一路上所有協助過我的人，因為有你們，我的生活充滿鮮豔的色彩；因為有你們，我的成長過程不再孤單。

我要特別感謝我最敬重的指導老師曾建超教授，在你身上我學到了很多包括學術上，生活上或是做人處事的道理。感謝您協助探索我的極限，多少個孤獨的夜晚，埋首看 Paper、Debug。因為有你，我體會到做研究應有的嚴謹度，了解寫程式的奧義，也知道做人要謙虛要對自己負責，並學到原來製作投影片也是一門大學問，很高興您給了我這個學習的機會，祝福您身體健康，謝謝您的指導。其次，也要感謝我的口試委員王讚彬教授，曹孝櫟教授以及張弘鑫教授，因為有你們的指導與建議，讓本論文多添加了幾分有趣的元素。

很高興自己有這個福氣跟 WINLab 的所有學長姐、學弟妹們一起共同學習與成長，包括博士班最有活力的 RT 學長、一起騎腳踏車的 Gunter 學長、姿欣學姊，約好一起去征服六十石山與再一次環島、一起打壘球的 Gary 學長，你傳球真的很準、最認真的承運學長以及瘦身冠軍大樑哥學長。感謝俊羽學長、宗義學長幫我投履歷，在我的求職過程中，一直幫忙我。感謝上一屆人小志氣高的冠銘學長，人非常好的統一獅投手俊良學長，與台南南霸天劉阿舍俊延學長。感謝我最要好的好同學，國科會台柱愛將蛋捲，你每天晚上都會想到的那個人，跟他說話時卻不知道要說什麼，相信我，你真的愛上他了。未來的同事也是環島的好夥伴竣晨，以後在桃園的生活還要多請教你，破釜沉舟那招也記得順便傳授給我。也要感謝一起共同在 D-Link T1 打拼的好 Partner 小毛，最正的昱樺與最認真的銘祥，謝謝你們這一路來的陪伴，風風雨雨都一起走過，祝福大家鵬程萬里，在未來的人生路上一路順風。感謝 D-Link 的學長承遠，你是我的第二個指導教授，不管是工作、論文或是學業上，你都不吝嗇給我最大的協助，謝謝 Vincent 教我這麼多 Linux 的技巧，感謝一起研究程式的文堯，永遠記得一起留在中心打世紀到半夜的時光。也要感謝最幽默的東震，在寫論文無趣的時光裡，給了我這麼多歡笑，一起在精神時光屋修練的日子，謝謝你的陪伴。也謝謝學弟愛將明辰、家明、阿彪、沅春與學妹貞慧、瑜茜。

當然，我要感謝從小到大陪我成長的家人，媽媽、爸爸，謝謝你們讓我無後顧之憂的做我想做的學問，真不知道要如何才能報答你們的恩情，接下來的人生，輪到我來孝順你們。也要感謝我的哥哥與大嫂 Joyce，永遠支持我，在我最無助的時候，給我溫暖，有你們真好。也要感謝我的女朋友宥均，謝謝你這一路來的陪伴，共同分享生活的所有喜怒哀樂。

僅以此論文獻給所有曾經幫助過我的朋友以及我最親愛的家人😊。

Contents

Abstract in Chinese	i
Abstract in English	iii
Acknowledgements	v
Contents	vi
List of Figures	viii
List of Tables	x
Chapter 1 Introduction	1
1.1 Motivation	1
1.2 Objective	5
1.3 Outline of the Thesis	6
Chapter 2 Background	7
2.1 Problems of TCP NAT Traversal	7
2.2 NAT Operation	7
2.3 NAT Mapping Rule	9
2.4 NAT Filtering Rule	11
2.5 NAT Variations	13
2.6 NAT TCP State Tracking	14
Chapter 3 Related Work	16
3.1 UDP NAT Traversal	16
3.2 TCP NAT Traversal	19
3.3 Issue in TCP NAT Traversal	23

Chapter 4 Analyze TCP NAT Traversal Methods	25
4.1 SNT – SYN with Normal TTL	25
4.2 SLT – SYN with Low TTL	27
4.3 ESi – Establish then SYN-in	28
4.4 Summary.....	29
Chapter 5 NBA TCP Traversal Scheme	30
5.1 NBA Overview.....	30
5.2 NBA Operation Procedure	31
5.3 Step 1 of NBA – NAT Information Collection	33
5.4 Step 2 of NBA – Traversal Method Determination.....	39
5.5 Step 3 of NBA – Connectivity Check.....	45
Chapter 6 System Implementation	46
6.1 System Overview.....	47
6.2 NBA Implementation.....	47
6.3 Discuss Ways to Implement NBA Scheme	50
Chapter 7 Experiment	51
7.1 Overview of Experiment	52
7.2 Result of Step 1 in NBA	53
7.3 Result of Step 2 in NBA	56
7.4 Comparison of Direct Connection Rate in Different Schemes	57
7.5 Comparison of System Resource Utilization in Different Schemes...	59
Chapter 8 Conclusions and Future Works	63
Bibliography	65

List of Figures

Figure 1-1 NAT device, private and public networks	1
Figure 1-2 Mapping table	2
Figure 2-1 NAT Operation.....	8
Figure 2-2 Independent mapping	10
Figure 2-3 Address dependent mapping.....	10
Figure 2-4 Address and port dependent mapping.....	10
Figure 2-5 Independent filtering.....	12
Figure 2-6 Address dependent filtering.....	12
Figure 2-7 Address and port dependent filtering	12
Figure 2-8 TCP three-way handshake.....	14
Figure 2-9 TCP state tracking	15
Figure 3-1 STUN Architecture	17
Figure 3-2 TURN Architecture	18
Figure 3-3 ICE Architecture.....	19
Figure 3-4 STUNT #1	20
Figure 3-5 STUNT #2.....	20
Figure 3-6 NATBlaster	22
Figure 3-7 P2PNAT	23
Figure 4-1 SNT	27

Figure 4-2 SLT	28
Figure 4-3 ESi	29
Figure 5-1 NBA system architecture.....	31
Figure 5-2 NBA operated procedure	32
Figure 5-3 Mapping test.....	34
Figure 5-4 ESi test & Si test	35
Figure 5-5 SoSi test & SoRiSi test.....	36
Figure 5-6 SoUiSi test & SoTiSi test.....	37
Figure 5-7 Procedure of Step 1 in NBA	38
Figure 5-8 Initiator of connectivity check.....	40
Figure 5-9 A possible packet sequence of SNT	42
Figure 6-1 System modules of NBA	46
Figure 6-2 Module interaction flow of Step 1	49
Figure 6-3 Module interaction flow of Step 2	49
Figure 6-4 Module interaction flow of Step 3	50
Figure 7-1 Experiment environment	51

List of Tables

Table 5-1 List of NAT type tests	38
Table 5-2 Traversal Method Selection Table.....	41
Table 7-1 Brands of NATs.....	52
Table 7-2 Naming rule of mapping detection.....	54
Table 7-3 Naming rule of filtering detection	54
Table 7-4 Naming rule of TCP state tracking detection	54
Table 7-5 Classify of NATs in group A.....	54
Table 7-6 Classify of NATs in group B.....	55
Table 7-7 Traversal method selected by NBA to group A.....	56
Table 7-8 Traversal method selected by NBA to group B.....	57
Table 7-9 Direct connection of group A made by NBA	58
Table 7-10 Direct connection of group B made by NBA.....	58
Table 7-11 Direct connection of group A made by SCC or PCC.....	59
Table 7-12 Direct connection of group B made by SCC or PCC	59
Table 7-12 connectivity check time	60
Table 7-13 Traversal time of each scheme.....	60
Table 7-14 Numbers of message exchanges of each traversal method	61
Table 7-15 Numbers of message exchanges of each scheme	61

Chapter 1

Introduction

1.1 Motivation

The dramatic development of the internet industry in recent years has led to the depletion of the remaining IPv4 address space. Limited IPv4 addresses could not satisfy a large number of devices on internet nowadays. To alleviate the IPv4 address space exhaustion, network address translation (NAT) [1] appeared and became a popular tool in the mid-1990s. NAT allows several hosts to share one public IPv4 address because NAT divides the network into public and private network as shown in Figure 1-1. Multiple hosts on a private network have their own private IP addresses which are meaningful only within the scope of the private network and can't be used to get into the internet directly. Therefore, the same private IP address can be reused on different private network blocks as long as those private networks cannot communicate with each other directly. Hosts on private networks can take the NAT as a gateway and share a single public IP via address/port translating on the NAT to connect to each other or access to the internet through NAT boxes.

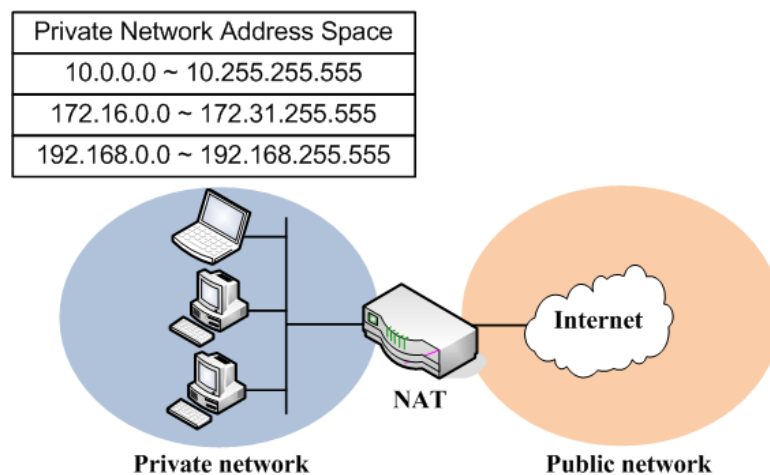


Figure 1-1 NAT device, private and public networks

In order to achieve the capacity to share a public of NAT to multiple hosts, NAT must have some effort on inbound and outbound packets. Hosts who are located on private network are named internal hosts. Once they want to communicate with external hosts via sending/receiving packets, the NAT device would involve rewriting the source and/or destination IP addresses and also the TCP/UDP port numbers of IP packets as they pass through the NAT. Checksums of IP and TCP/UDP headers must also be recalculated to make sure the NAT device transmits packets correctly between internal and external hosts. Once the internal hosts send packets to the external hosts, NAT will record the translating rule between private and public IP address/port on the mapping table. NAT then transmits inbound packets to the correct host according to the mapping table such as Figure 1-2. If the record on the mapping table hasn't been created, NAT would not understand where to route the packets. In the other word, external host cannot initiate unsolicited session to internal host until the internal host sends out packets to the external host, and NAT creates a recode on mapping table.

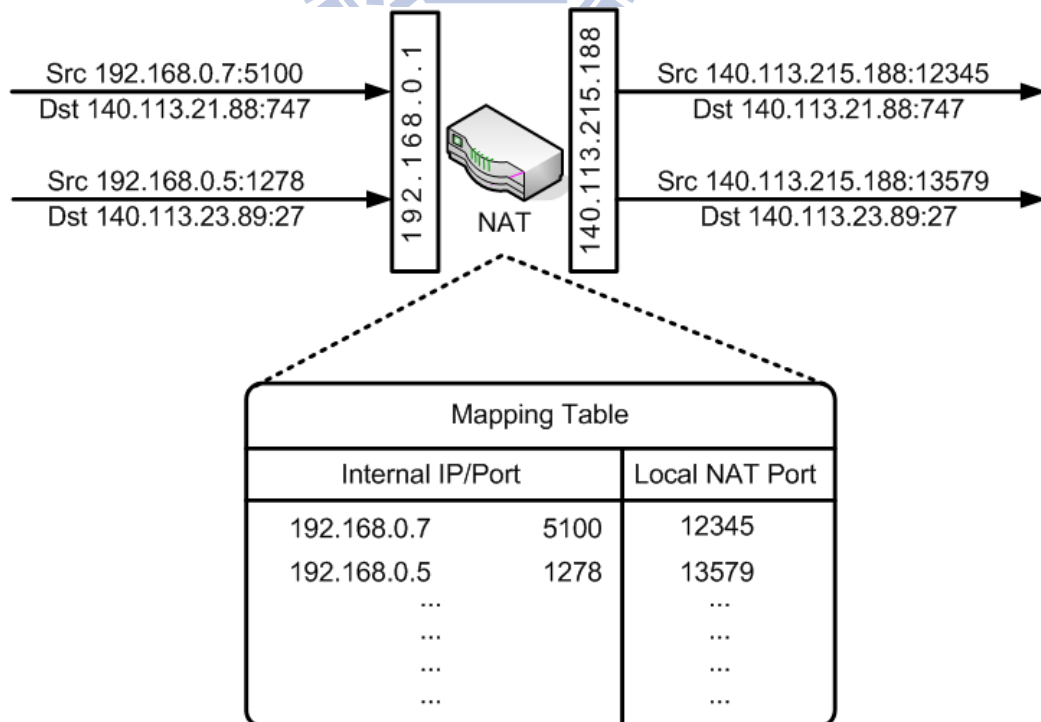


Figure 1-2 Mapping table

Although NAT provides a lot of benefits to the internet nowadays, however, it also incurs some drawbacks. NAT is a barrier for peer-to-peer applications because one peer cannot know the address of the other peer behind an NAT, and NAT may block unsolicited inbound traffics. Many methods have been proposed to solve the issues, and those methods are called NAT traversal. Interactive Connectivity Establishment (ICE) is a well-know protocol for UDP NAT traversal [6]. It uses nine address pairs to perform connectivity checks in order to find out an appropriate connection between two peers behind NATs. In recent years, most peer-to-peer applications use TCP to transmit packets, but establishing a TCP connection is more complex than UDP since two hosts must perform a three-way handshake procedure [17]. Moreover, most NATs implement some sort of TCP state tracking mechanism to trace TCP stages [16]. Different TCP state tracking mechanisms implemented in NATs need different traversal methods to solve them, so many TCP NAT traversal methods was also proposed, and each method is applicable to different NAT combinations.

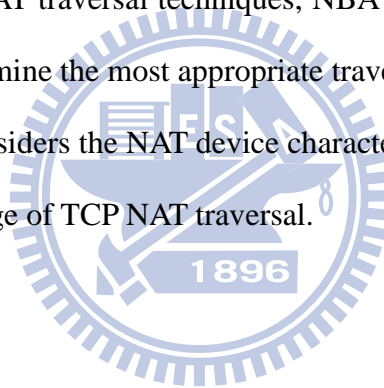
One kind of the most effective NAT traversal methods of establishing peer-to-peer communication between hosts on different private networks is known as "hole punching" [2, 19, 21]. UDP hole-punching was first explored and publicly documented by Dan Kegel [3]. Using the same aspects, techniques was declared such as Simple traversal of User Datagram Protocol (UDP) through NATs (STUN) allows applications to discover the presence, types and IP address of NATs [4]. Traversal Using Relays around NAT (TURN) allows two hosts behind different NATs to exchange packets each other using the relay [5]. Interactive Connectivity Establishment (ICE) makes use of STUN and TURN protocol [6]. TCP hole punching is seem more complexity then UDP, because three-way handshake must be performed to establish TCP connection and most NATs implement TCP state tracking mechanism to track the TCP stages. NatTrav advocate direct TCP connections between peers [7]. Simple Traversal of UDP Through NATs

and TCP (STUNT) is a well-know TCP hole punching method [14]. NATBlaster proposed novel mechanisms to create direct TCP connections between two hosts behind middle-boxes with minimal help from a third-party [15]. However, previous TCP NAT traversal proposals did not consider TCP state tracking of NATs, but they perform brute force connectivity check instead so that this procedure induces a long delay or excessive message exchanges for setting up a connection.



1.2 Objective

In this thesis, we propose a NAT Behavior-Aware (NBA) TCP Traversal Scheme that can eliminate unnecessary connectivity checks from choosing the most appropriate traversal method based on the behaviors of NATs, instead of try-and-error tests [20]. Connectivity check will be more efficient by using NBA and result in shorter connectivity check delay, fewer message exchanged, possible higher Direct Connection Ratios (DCR) and less resource usage or simpler state maintenance. In NBA, we implemented several NAT type detected tests to realize NAT behaviors clearly including behaviors of both UDP and TCP. When two hosts behind different NATs want to establish a TCP connection through NAT traversal techniques, NBA can use the behavior knowledge of the two NATs to determine the most appropriate traversal method for the two hosts. This study successfully considers the NAT device characteristics, and the results may provide an insight into the usage of TCP NAT traversal.



1.3 Outline of the Thesis

This paper is organized as follows. In chapter 2, characteristics and behaviors of NATs are described. Many UDP and TCP NAT traversal methods are surveyed in chapter 3. In chapter 4 we review three practical TCP NAT traversal methods to understand their characters comprehensively. In chapter 5, NAT behavior aware approach is declared in detail. We include system implementations of NBA in chapter 6. In chapter 7, NAT behavior aware approach and previous TCP NAT traversal are compared according to the experiment results. Finally, we draw conclusions and suggest future works.



Chapter 2

Background

In the chapter, NAT will be described carefully including problems of TCP NAT traversal and the basic principle of NAT operation, mapping rules, filtering rules and TCP state tracking. Although the concept of NAT was proposed more than 15 years [1, 10, 11, 12], neither NAT-related standard nor protocol is specified. As a result, current NAT implementations vary among not only vendors but also NAT models.

2.1 Problems of NAT Traversal

Three characteristics of NAT behaviors affect NAT traversal deeply that are NAT mapping rules, filtering rules and TCP state tracking. NAT uses mapping rules to decide which ports for assigning to each connection and filtering rule for determining whether inbound packets can be sent to hosts behind an NAT via existing mappings. The two rules make it difficult to establish direct connection between two peers behind NATs. Besides, NAT implements TCP state tracking to trace TCP stage and the mechanism may block unexpected TCP packet sequences. Different NATs may have different NAT TCP state tracking implementations, so peers are more difficult to establish a TCP connection between two hosts behind NATs. In the following sections, we describe the three characteristics of NAT behaviors in detail.

2.2 NAT Operation

Figure 2-1 illustrates the case that one host located on a private network wants to communicate with the other host located on a public network. Internal host must first send out an outbound packet through the NAT device to the external host, and NAT will

then generate a mapping entry on its mapping table to keep track of the session. For example in Figure 2-1, the internal host sends out an outbound packet to the external host, and NAT must translate the source address of the outbound packet from private IP address (ex: 192.168.1.10) to public IP address (ex: 140.113.215.183) and source port from local port (ex: 5100) to global port (ex: 12500) so that the packet can be routed on the public network. On the other hand, NAT generates a mapping entry to record the mapping between local IP/Port transport address and global IP/Port transport address (ex: 192.168.1.10:5100 → 140.113.215.183:12500). When the NAT device receives the inbound packet from the external host to the internal host, it translates the destination IP/Port transport address (ex: 140.113.215.183:12500) of the inbound packet to the corresponding transport address (ex: 192.168.1.10:5100) according to the mapping table in NAT. The principle of address translation in NAT is based on mapping table. Therefore, any inbound packet from the external host cannot be routed to the internal host by NAT until the internal host has sent out an outbound packet to the external host and a mapping entry was generated on mapping table in NAT.

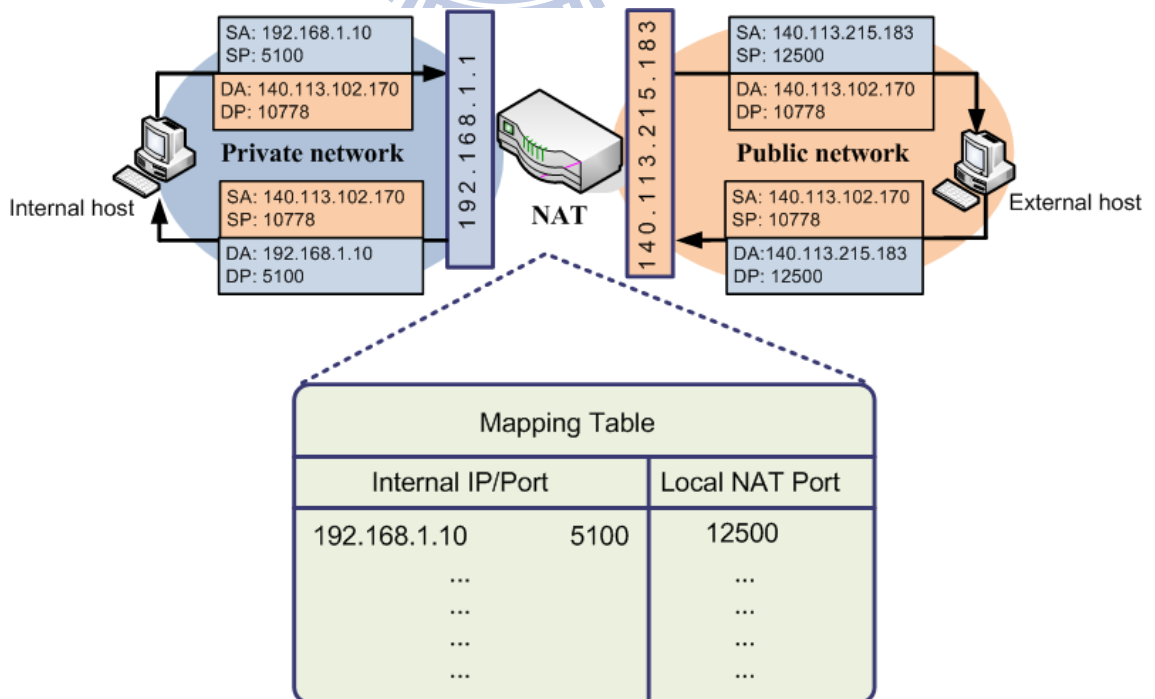


Figure 2-1 NAT Operation

2.3 NAT Mapping Rule

NAT mapping rule defines how NAT assigns public ports to outgoing connections, and it is required for a NAT to maintain a connection between private network and public network. When an internal host initiates an outgoing connection through an NAT, the NAT assigns a global IP address and a global port number to the connection and then create a mapping entry so that subsequent response packets from an external host can be received by the NAT, translated, and forwarded to the internal host according this entry [8]. NAT mapping rule can be classified into three categories:

- **Independent:** NAT reuses the port-mapping for subsequent packets sent from the same internal IP/Port transport address to any external IP/Port transport address. For example, in Figure 2-2, no matter Node A sends packets to different ports P1, P2 on Node B or P3 on Node C, NAT will all reuse the same port-mapping Pa of its external interface.
- **Address dependent:** NAT reuses the port-mapping for subsequent packets sent from the same internal IP/Port transport address only to the same external IP address, regardless of the external port. For instance, Figure 2-3 shows that NAT will use the port-mapping Pa when Node A sends packets to P1 and P2 on Node B. But NAT uses port-mapping Pb when the destination is Node C.
- **Address and port dependent:** NAT reuses the port-mapping for subsequent packets sent from the same internal IP/Port transport address only to the same external IP/Port transport address. As shown in Figure 2-4, if the destination IP address or port is different (Node B with P1 and P2, and Node C with P3), NAT uses different port-mapping (Pa, Pb and Pc).

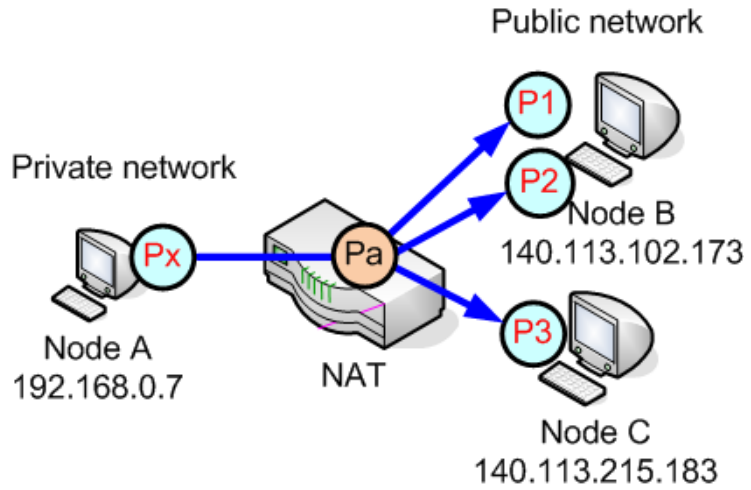


Figure 2-2 Independent mapping

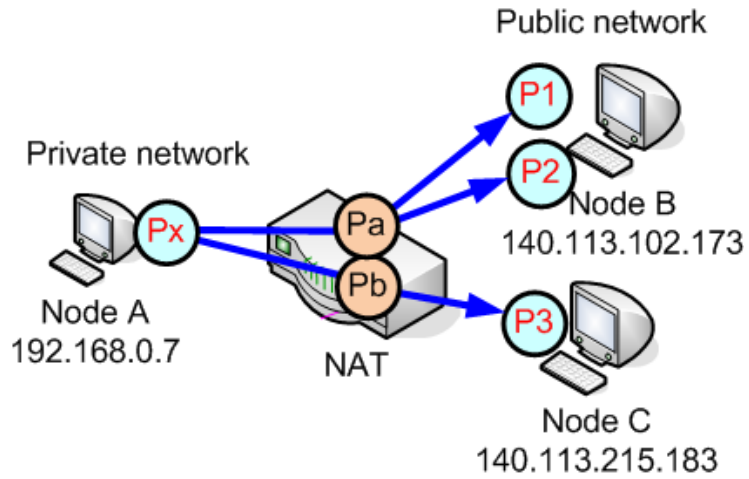


Figure 2-3 Address dependent mapping

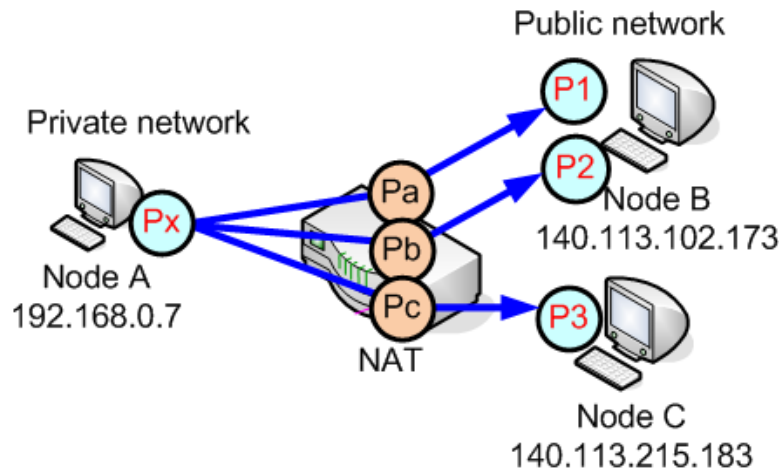


Figure 2-4 Address and port dependent mapping

2.4 NAT Filtering Rule

NAT filtering rule defines which external hosts are allowed to send inbound packets to the corresponding internal hosts via existing mappings [8]. NAT filtering rule can also be classified to three categories:

- **Independent:** Internal hosts send packets to any external IP address is sufficient to allow any packets from external host with any IP address and port back to the internal host. As shown in Figure 2-5, once the session between Node A and Node B has been established by Node A, any inbound packets from external hosts such as Node B and Node C can pass the NAT via this port-mapping.
- **Address dependent:** Internal hosts receiving packets from a specific external host are necessary for internal hosts to send packets first to that specific external host's **IP address**. For instance, Figure 2-6 shows that once the session has been established by Node A between Node A and Node B, only inbound packets from external host Node B with P1 and P2 can pass the NAT via this port-mapping.
- **Address and port dependent:** This behavior is similar to the previous category. Internal hosts receiving packets from a specific external host are necessary for them to send packets first to that specific external host's **IP address and port**. For example, in Figure 2-7, once the connection has been established by Node A between Node A and Node B, only inbound packets from external host Node B's port P1 can pass the NAT via this port-mapping.

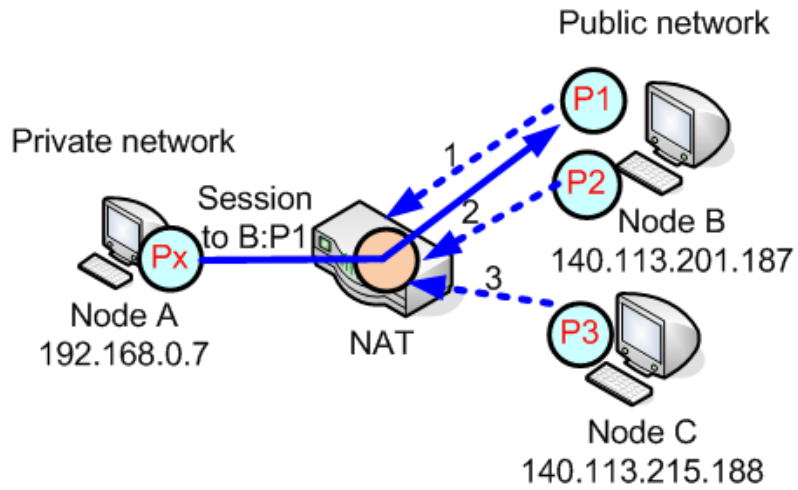


Figure 2-5 Independent filtering

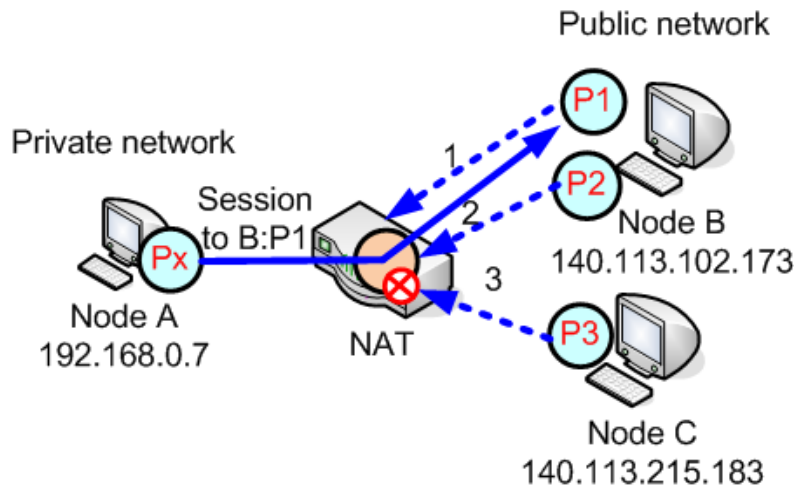


Figure 2-6 Address dependent filtering

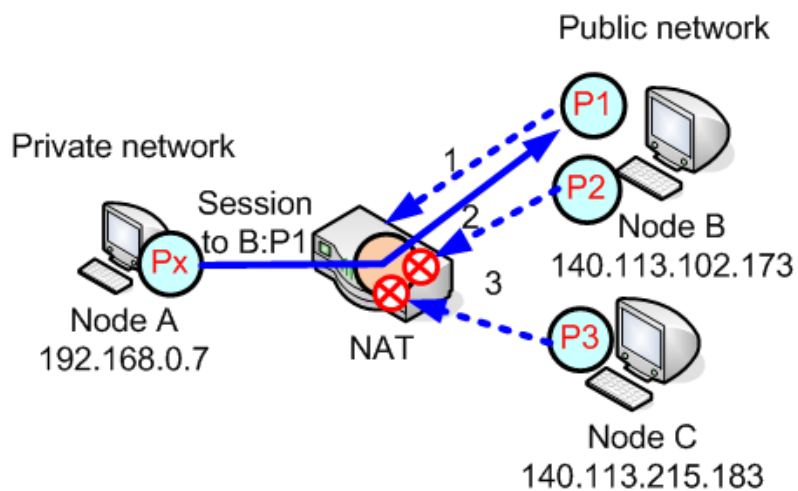


Figure 2-7 Address and port dependent filtering

2.5 NAT Variations

For UDP, NAT's treatment of packets varies among implementations. Typically, NAT devices can be classified into two categories such as Cone NAT and Symmetric NAT based on mapping rule [4]. Cone NAT assigns the same public port for all connections from the same local port; Symmetric NAT assigns a unique public port for different connections.

Cone NAT can be further classified into Full Cone, Restricted Cone and Port Restricted Cone based on filtering rule [4]. In the following, we describe the four categories of NAT carefully.

- **Full Cone:** All outgoing packets from the internal host with the same internal IP/Port transport address are mapped to the same external IP/Port transport address. Moreover, all external hosts can send packets to the internal host by using the mapped external IP/Port transport address.
- **Restricted Cone:** All outgoing packets from the internal host with the same internal IP/Port transport address are mapped to the same external IP/Port transport address. But only internal hosts who ever send packets to the specific external host's IP address can receive packets from the specific external host.
- **Port Restricted Cone:** All outgoing packets from the internal host with the same internal IP/Port transport address are mapped to the same external IP/Port transport address. But only internal hosts who ever send packets to the specific external host's IP/Port transport address can receive packets from the specific external host.
- **Symmetric:** If the same internal host sends a packet with the same source address and port to a different destination IP address or port, and NAT would use a different mapping. Furthermore, only internal hosts who ever send packets to

the specific external host's IP/Port transport address can receive packets from the specific external host.

However, RFC 5389 which evolves from the STUN protocol, RFC 3489, had removed the algorithm for NAT variations detection because NATs nowadays cannot be classified into those four types of behaviors. But this is a familiar classification when we study NAT technology. So, we still describe it in this section.

2.6 NAT TCP State Tracking

The TCP three-way handshake known technically as the SYN, SYN-ACK and ACK is the process for establishing a TCP connection as shown in Figure 2-8. However, two hosts behind different NATs cannot establish a TCP connection through normal TCP three-way handshake process so that special TCP packet sequence flows will appear. The special packet sequence flows are caused not only by current TCP NAT traversal methods but also by NAT response. For example, scenario 1 and scenario 2 in Figure 2-9, there are many vendors use a lightweight state machine within the NAT Session to track the current state of a TCP connection [13] and determine when connection-state can be garbage-collected. NAT may block unexpected packets sequences according to its implementation of TCP state tracking mechanism.

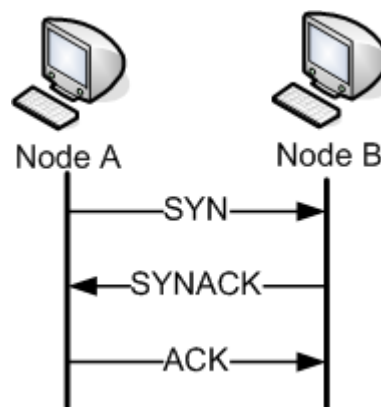


Figure 2-8 TCP three-way handshake

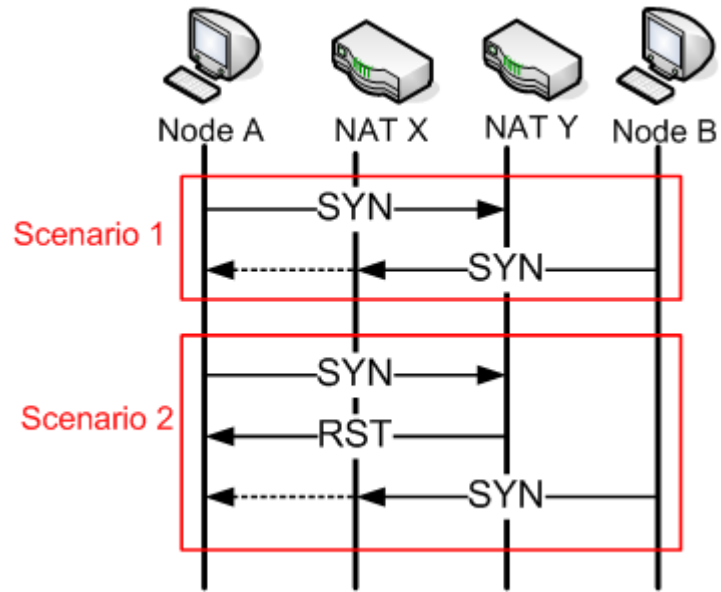
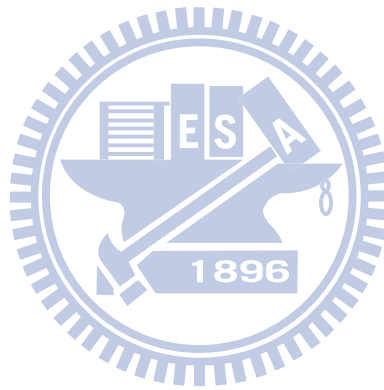


Figure 2-9 TCP state tracking



Chapter 3

Related Work

In this chapter, a number of current researches and techniques related to UDP and TCP NAT traversal problem will be described. Then, a major issue which affects TCP NAT traversal problem will also be depicted.

3.1 UDP NAT Traversal

3.1.1 STUN

Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs) (STUN) is a lightweight protocol that allows hosts to discover not only the presence and types of NATs and firewalls between them but also the public IP/Port transport addresses [4]. STUN Server is a third-party network server with two IP addresses and two ports as shown in Figure 3-1. STUN is a commonly used technique to solve UDP NAT traversal problems. Internal host uses STUN server to realize the public IP and port-mapping on its NAT and then other host may use this port-mapping to send inbound packets to the internal host in some types of NATs.

RFC 5389 [9] is a new specification of STUN named “Session Traversal Utilities for NAT”, and it is an evolution from RFC 3489. The original STUN protocol defined a NAT type discovery process flow for applications to discover the type of an NAT. But new STUN protocol had removed this algorithm for NAT type detection and binding lifetime discovery. Because NATs nowadays may not fit into those type classifications which we have described in session 2.5, and the algorithm was found to have some errors [9]. RFC 5389 also defines STUN protocol to have additional capability such as checking connectivity between two peers behind different NATs and a keep-alive pro-

to maintain NAT mappings.

However, since symmetric NAT assigns a unique public port for each connection to a specific IP/Port transport address, we cannot use STUN to traverse this category of NATs.

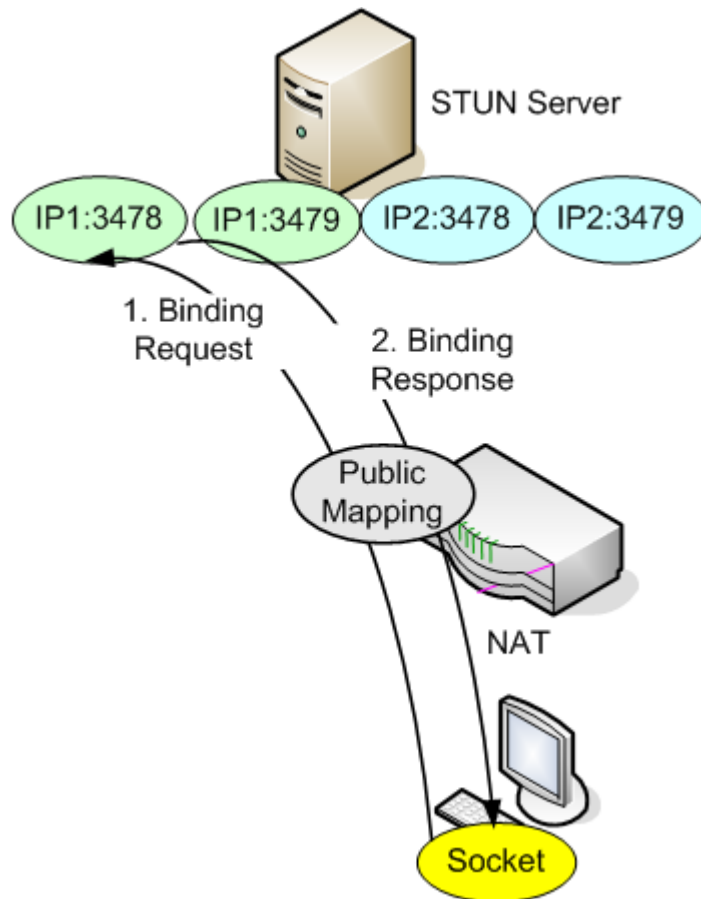


Figure 3-1 STUN Architecture

3.1.2 TURN

Traversal Using Relays around NAT (TURN) is a protocol that allows the host behind a NAT to control the operation of the relay server and to exchange packets with its peers using the relay [5]. Once hosts behind different NATs want to communicate with each other, they can establish their own connection with the third-party network server named TURN server, and TURN server would help them to redirect data to the other hosts. Figure 3-2 shows the typical operation of TURN. In Figure 3-2, Node A connects to TURN for requesting relay resources X, and then Node A would inform the

relay resource X to host B. Once two hosts want to communicate with each other, they can just send data to the relay resource X, and TURN server will redirect data to the other host. This relay approach is a useful NAT traversal method.

However, when two hosts use TURN server as a relay server to communicate with each other, they must occupy additional network bandwidth. Therefore, although TRUN can traverse all types of NATs, it has the lowest priority in NAT traversal methods.

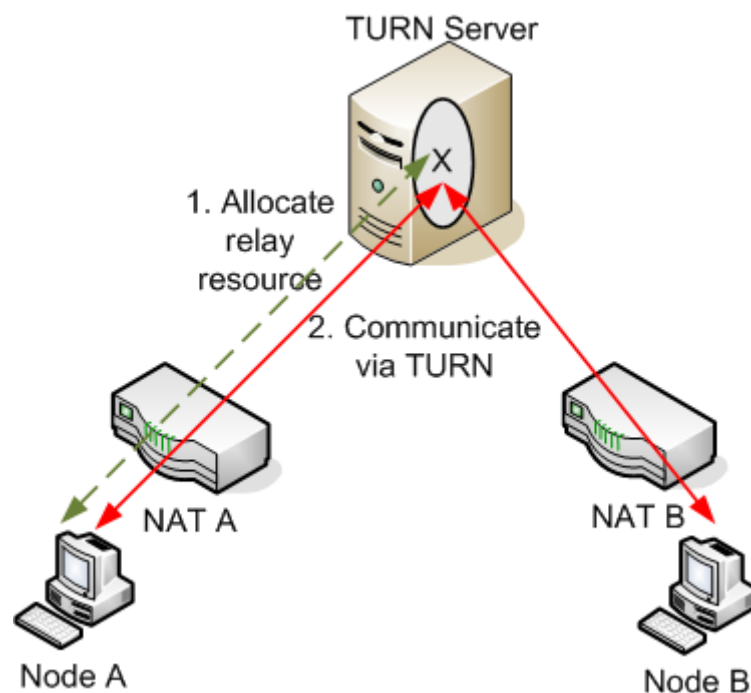


Figure 3-2 TURN Architecture

3.1.3 ICE

Interactive Connectivity Establishment (ICE) makes use of integrating some NAT traversal techniques such as STUN and TURN, and it is a protocol for UDP (ICE-UDP) and TCP (ICE-TCP) NAT traversal. Once two hosts behind NATs want to check connectivity, they would collect their own possible candidates which are a multiplicity of IP addresses and ports from STUN and TURN. Then two hosts use ICE to exchange those candidates, ICE would make pairs of two hosts' candidates and try systematically all

possible pairs until it finds one or more direct connectivity path.

Figure 3-3 is an example for ICE-UDP. Both two hosts must collect its own candidates which are local address, server reflexive address from STUN server and relay address from TURN server. ICE makes pairs of two hosts' candidates to nine possible pairs and checks connectivity of each pair.

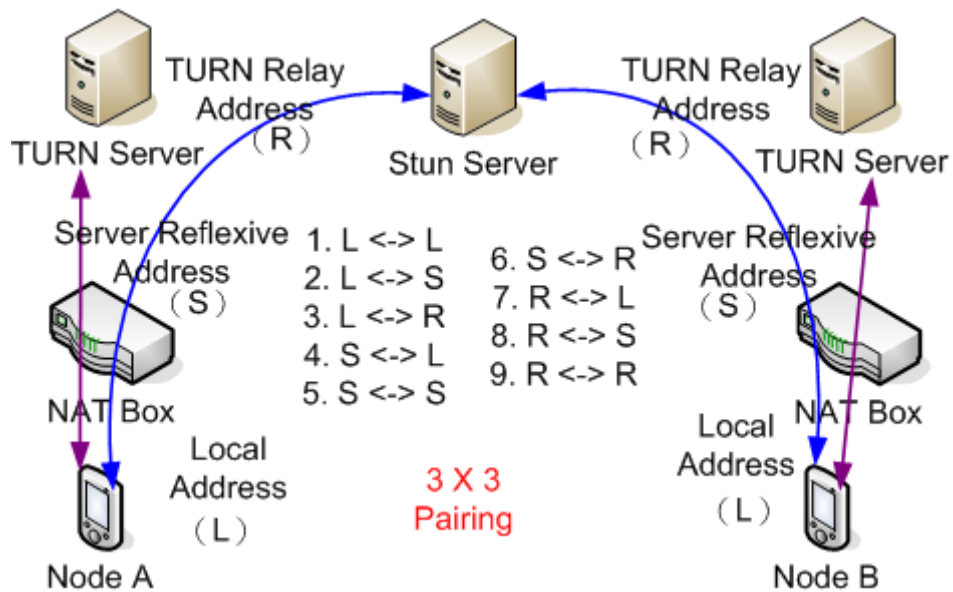


Figure 3-3 ICE Architecture

However, ICE brings about a large delay during connection setup since these collection steps and check procedure have to be performed for any new connectivity request. Moreover ICE only establishes connections between applications on two hosts which have the knowledge of ICE protocol. As a result ICE cannot be seen as a solution for arbitrary applications.

3.2 TCP NAT Traversal

TCP NAT traversal is more complex than UDP because most NATs implement TCP state tracking mechanism to track the current state of a TCP connection. In this section, three implement TCP NAT traversal methods which are proposed in recent literatures will be described.

3.2.1 STUNT

Simple Traversal of UDP Through NATs and TCP too (STUNT), which extends STUN to include TCP functionality, is a lightweight protocol that assists hosts behind NATs to determine external IP address and port number. It also helps hosts traverse NAT to establish TCP connections. STUNT was proposed in [14] including two TCP NAT traversal methods named STUNT #1 and STUNT #2, illustrated in Figure 3-4 and Figure 3-5 respectively.

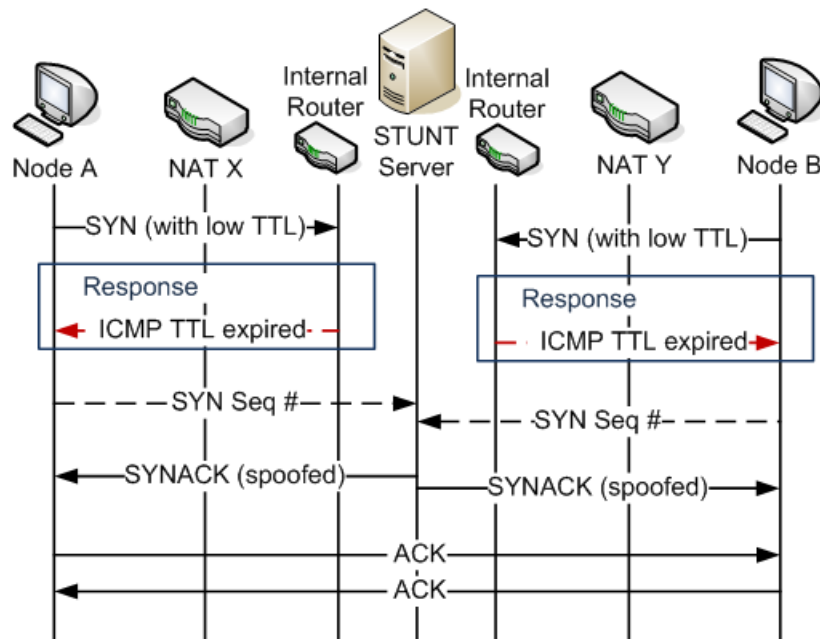


Figure 3-4 STUNT #1

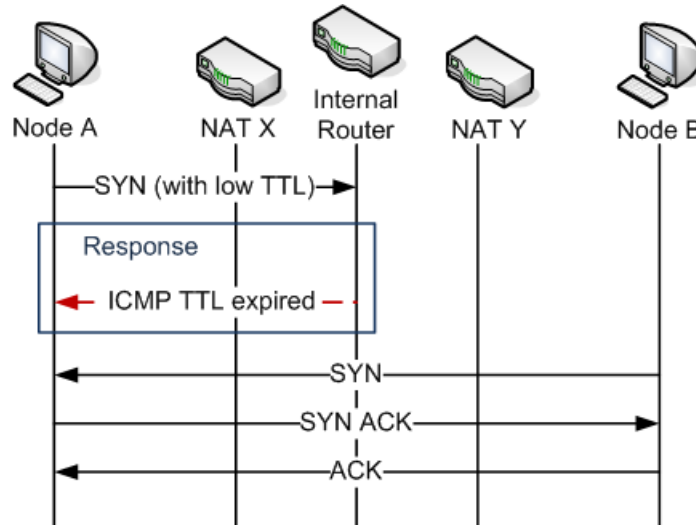


Figure 3-5 STUNT #2

In STUNT #1, when hosts behind different NATs want to establish a TCP connection, both hosts must first send an initial SYN packet to each other. However, unsolicited packets may cause side effects of NAT while establishing two hosts' connections. Therefore, the initial SYN packets are set with low time to live (TTL) values to cross their NATs, but the SYN packets must not reach other host's NAT and will be dropped in the network (once the TTL expires). Second, both hosts learn the TCP sequence number of the initial SYN packets over PCAP or a RAW socket and send their respective TCP sequence number to a globally reachable third-party named STUNT server. The STUNT server resides in public network and spoofs a relative SYNACK to each host with the appropriately sequence numbers as the packet comes from the other host. Finally, host will respond an ACK packet to the other host for completing TCP 3-way handshake.

STUNT #2 is similar to the STUNT #1, but only one host sends out a low -TTL SYN packet. Then sender aborts this connection attempt and creates a passive TCP socket on the same IP address and port number. The other host then initiates a 3-way handshake procedure to establish a TCP connection. However, there are some issues existing on STUNT such as NAT characteristics, host requirements and the spoofing requirement for the STUNT Server.

3.2.2 NATBlaster

In [15], the authors propose a novel mechanism named NATBlaster to create direct TCP connections between two hosts behind middle-boxes with minimal help from a third-party. NATBlaster as shown in Figure 3-6 is similar to STUNT #1 except that instead of spoofing SYNACK packets by STUNT server, the two hosts exchange the sequence numbers and each crafts a SYNACK packet the other expects to receive.

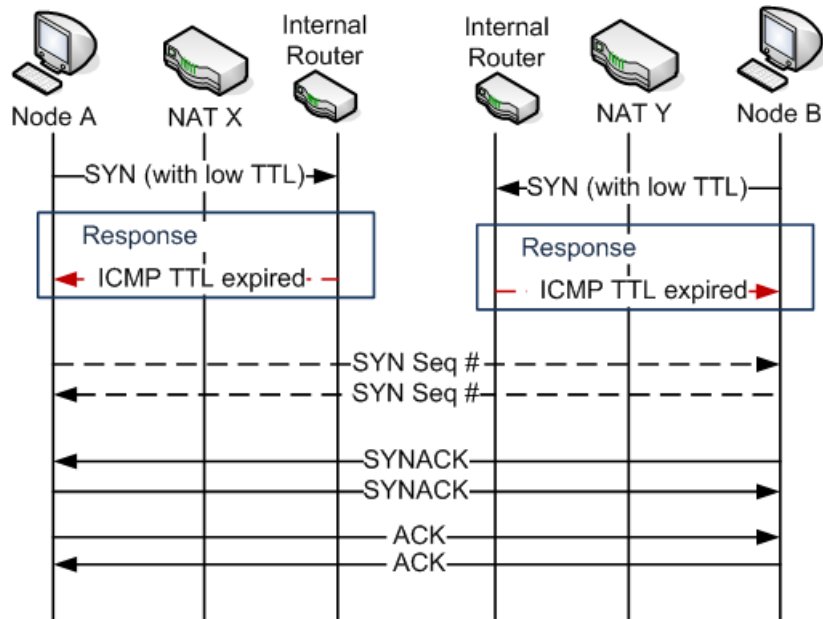


Figure 3-6 NATBlaster

3.2.3 P2PNAT

In [2], the authors take advantage of the simultaneous open scenario defined in the TCP specifications [2]. As illustrated in Figure 3-7, each host initiates a TCP 3-way handshake procedure to establish TCP connections. First, both hosts establish TCP connections with a well-known rendezvous server which records each registered client's public and private IP addresses. Second, client A uses its active TCP session with server to ask server for help connecting to client B. Third, server replies B's public and private address to A, and at the same time sends A's public and private address to B. Finally, client A and B each asynchronously make outgoing connection attempts to the other's public and private address as replied by server, while simultaneously listening for incoming connections on their respective local TCP ports. If one of the outgoing connection attempts fails due to a network error such as "connection reset" or "host unreachable", the host simply retries that connection attempt after a short delay (e.g., one second), up to an application-defined maximum timeout period.

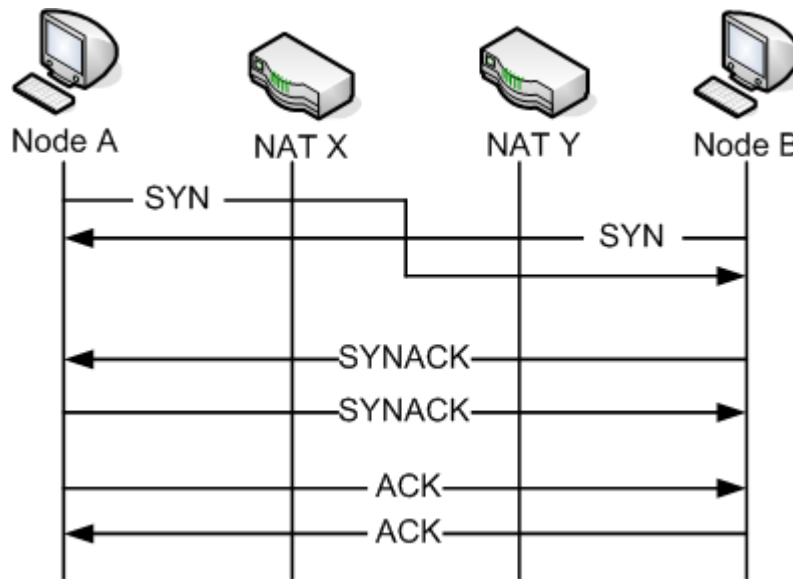
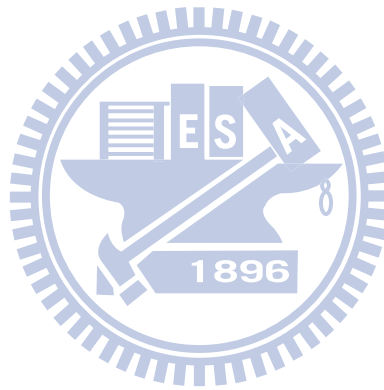


Figure 3-7 P2PNAT

3.3 Issue in TCP NAT Traversal

NAT mapping rule, filtering rule and TCP state tracking are the characteristics that affect TCP NAT traversal. NATs handle outbound packets differently by using different mapping rules and inbound packets by filtering rules so that NAT traversal may fail in establishing a connection between two hosts because of the two rules. NAT may block unexpected TCP packet sequence according to its TCP state tracking mechanism. Many TCP NAT traversal methods have been proposed for TCP NAT traversal problem, and different methods may work for different NAT combinations. Therefore, we need to apply several traversal methods to increase direct connection rate while traversing NATs. There are two general schemes for applying traversal methods to do connectivity check such as sequential connectivity check (SCC) and parallel connectivity check (PCC). The former scheme applies NAT traversal methods sequentially while the latter scheme performs in parallel. However, the two schemes have no idea about applicability of NAT Traversal methods to types of NATs, and apply traversal methods in a try-and-error fashion. So, they consume large amounts of system resources and time delay to traverse NATs.

In this paper, an effective NAT Behavior-Aware (NBA) TCP traversal scheme was proposed. It takes NAT information about mapping rules, filtering rules and TCP state tracking into consideration to select the most appropriate NAT traversal method.



Chapter 4

Analyze TCP NAT Traversal Methods

We have introduced many TCP NAT traversal methods in above chapter, actually there are three practical TCP NAT traversal methods in recent years, and each method may generate a specific packet sequence and thus only works well for the NATs that allow such packet sequences. Therefore, as long as we realize the packet sequences appear while applying each method, we will know what kinds of TCP NAT traversal methods can work well for the combination of two NATs. In this chapter, we review three practical TCP NAT traversal methods to realize kinds of packet sequences which appear while applying these methods. The results are important for NBA to determine NAT type examinations. The three traversal methods are listed as follows:

1. SYN with Normal TTL (SNT)
2. SYN with Low TTL (SLT)
3. Establish then SYN-in (ESi)

4.1 SNT – SYN with Normal TTL

The first TCP NAT traversal method is SNT which is the same as “STUNT #2 with no-TTL” [16]. In SNT, the first unsolicited SYN packet which is sent from caller to callee would be treated as an unsolicited income packet and be filtered by the callee’s NAT. Therefore, the callee’s NAT will choose to either drop the packet silently or notify the client and then different response may cause different packet sequences on the caller’s NAT. Different NATs on the caller side may behave dissimilarly towards those packet sequences. For example, in Figure 4-1, when Node A would like to establish a TCP connection with Node B, it performs SNT to traverse NATs. The first SYN packet

which is sent by Node A will reach the Node B's NAT (NAT Y), and NAT Y filters this unsolicited packet. The responses are important for us to understand SNT deeply. Therefore, the first knowledge we want to obtain is how NAT responds to the unsolicited inbound packet. We named this behavior examination as "*unsolicited inbound SYN (Si) test*" which means we want to test how a NAT behaves when it confronts with an unsolicited inbound packet.

In addition, according to our experiments, there are three kinds of responses which NAT responds to the unsolicited inbound packets such as drop the packet silently, sending back a reset packet and sending back an ICMP host unreachable packet. Therefore, three corresponding packet sequences appear on the caller's NAT and we also want to obtain the knowledge about how NAT behaves towards these packet sequences. For instance, in Figure 4-1, NAT X confronts three different kinds of packet sequences that are an outbound SYN followed by an inbound SYN, an outbound SYN followed by an inbound fatal TCP RST and then inbound SYN, and an outbound SYN followed by an ICMP host unreachable message and then inbound SYN. In NBA, the examinations which exam whether NAT allows the packet sequences are "*outbound SYN and then inbound SYN (SoSi) test*", "*outbound SYN followed by inbound RST and then inbound SYN (SoRiSi) test*", and "*outbound SYN followed by inbound ICMP host unreachable and then inbound SYN (SoUiSi) test*".

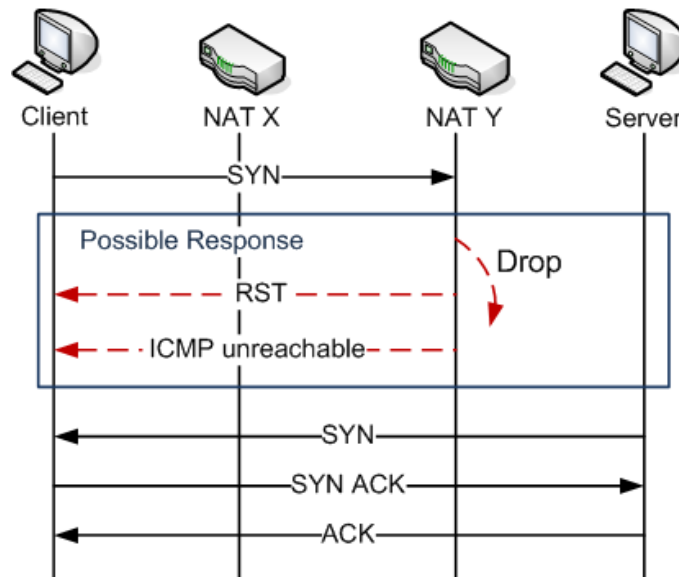


Figure 4-1 SNT

4.2 SLT – SYN with Low TTL

The second TCP NAT traversal method is SLT which is the same as “STUNT #2 with low-TTL” [16]. In SLT, caller sets the time to live (TTL) value of the first SYN packet for a low value and sends to callee. The packet will only pass its NAT and be dropped by the internal router between the two hosts because of TTL-expired. Then, the router would signal an error by sending back an ICMP TTL expired to the caller. Since the first SYN packet of caller does not reach the callee’s NAT instead dropped by the internal router. Packet sequence which appears on SLT is different with them which appear on SNT. For example, in Figure 4-2, when Node A would like to establish a TCP connection with Node B, it performs SLT to traverse NATs. The first SYN packet with low-TTL value is sent by Node A, and it will be dropped by the internal router (Router Z) and not reach the Node B’s NAT (NAT Y). Moreover, Router Z sends back an ICMP TTL expired to Node A, and then callee initiates another three-way handshake procedure. Therefore, NAT X confronts one kind of packet sequences that is an outbound SYN followed by an inbound ICMP TTL-exceeded error and then inbound SYN. In NBA, the examination which exams whether NAT allows the packet sequences is “out-

bound SYN followed by inbound ICMP TTL-expired and then inbound SYN (SoRiSi) test

“.

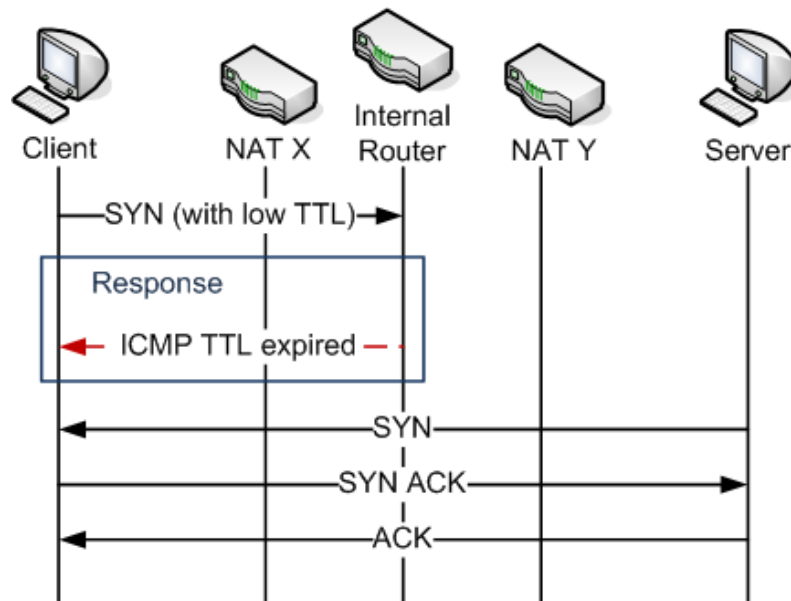


Figure 4-2 SLT

4.3 ESi – Establish then SYN-in

The third TCP NAT traversal method, ESi, is first declared and implemented in NBA. This method uses characteristic of NAT filtering rule to traversal NAT. As we mentioned in chapter 2, if the filtering rule of NAT is classified to be *Independent*, internal hosts send packets to any external IP address is sufficient to allow any packets from external host with any IP address and port back to the internal host. So, if one of the two hosts' NATs is classified to be *Independent*, they can reuse the port-mapping existing on the NAT to traverse NATs. In NBA, UA normally has a TCP connection with NBA Server. If the filtering rule of UA's NAT is *Independent*, other UAs can use the port-mapping of the TCP connection to initiate a three-way handshake procedure, and traverse NATs. For instance, in Figure 4-3, Node A has a TCP connection with NBA Server, and the port-mapping of this TCP connection on NAT X is Port A. In consequence, Node B can use Port A to initiate three-way handshake procedure and establish a TCP connection with Node A.

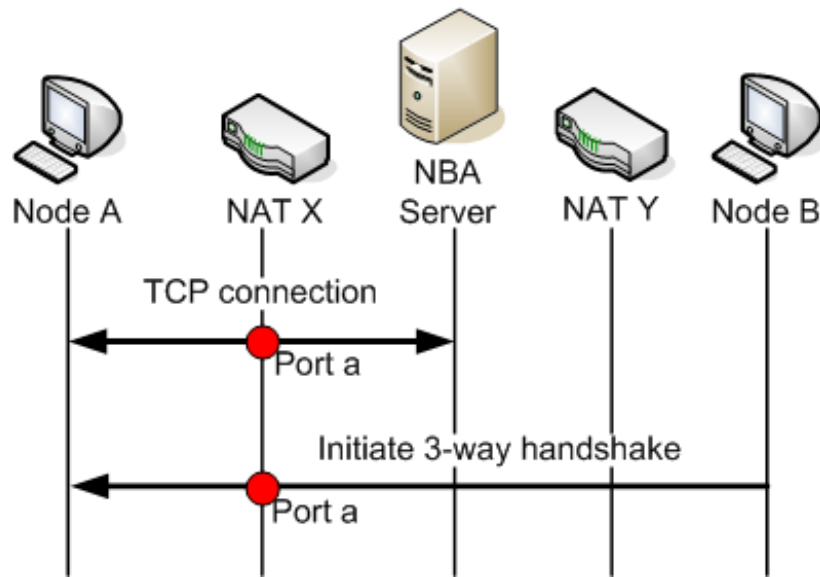
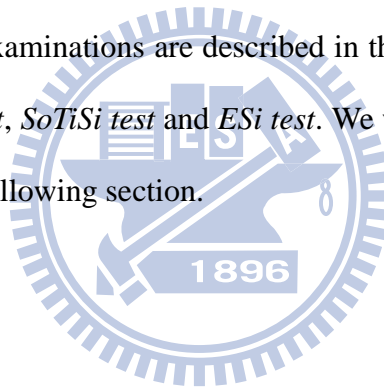


Figure 4-3 ESi

4.4 Summary

Six NAT type examinations are described in this section such as *Si test*, *SoSi test*, *SoRiSi test*, *SoUiSi test*, *SoTiSi test* and *ESi test*. We will explain the six examinations of NBA in detail in the following section.



Chapter 5

NBA TCP Traversal Scheme

Whereas the drawbacks of the other NAT traversal schemes such as SCC and PCC described in chapter 3, we propose a NAT Behavior-Aware TCP traversal scheme, henceforth referred as NBA, to eliminate unnecessary connectivity checks.

5.1 NBA Overview

NBA assists hosts to collect knowledge of their NAT behaviors in advance. Once hosts behind NATs want to establish a direct TCP connection through connectivity check with TCP NAT traversal techniques, the most appropriate method among existent methods is selected based on the knowledge of the two communicating NATs.

Figure 5-1 shows the system architecture and major functional components of NBA. In NBA, NBA Server being a third-party is a global reachable server which resides in public network, and it can assist hosts to traverse NATs effectively. For example in Figure 5-1, hosts (i.e. Node A and Node B) behind NATs (i.e. NAT X and NAT Y) would like to establish a direct TCP connection via connectivity check with TCP NAT traversal methods, NBA Server will assist them to collect their NAT behaviors information comprehensively. This information includes NAT mapping rule, filtering rule and TCP state tracking, and it is collected by hosts after performing several NAT type tests in NBA.

NBA scheme is similar to SCC or PCC scheme since they all apply several traversal methods in order to increase the direct connection rate of connectivity check. However, issues still exist in the two schemes such as long check latencies in SCC and large system resources utilizations in PCC. In NBA, the third-party, NBA Server, be-

comes more intelligent and powerful of assisting hosts to traverse NATs. Of course, hosts also have to pay more effort in order to get more thorough understanding of NAT behaviors.

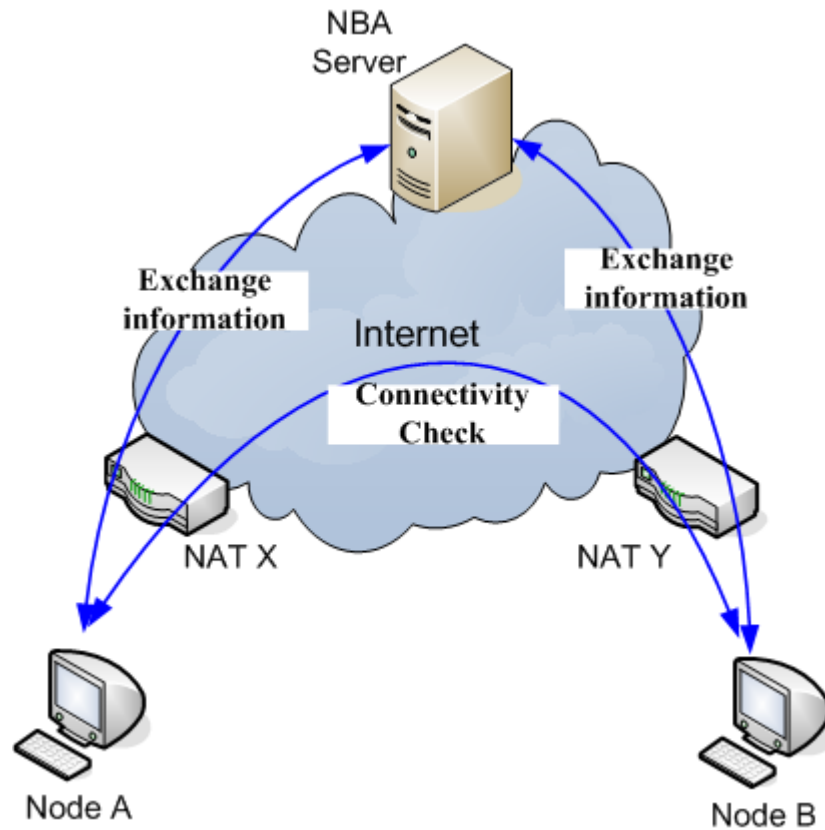


Figure 5-1 NBA system architecture

5.2 NBA Operation Procedure

In this section, we will describe the NBA procedure to introduce how NBA operates by taking Figure 5-2 as an example. There are three steps in NBA that are

1. **Step 1:** NAT Information Collection
2. **Step 2:** Traversal Method Determination
3. **Step 3:** Connectivity Check

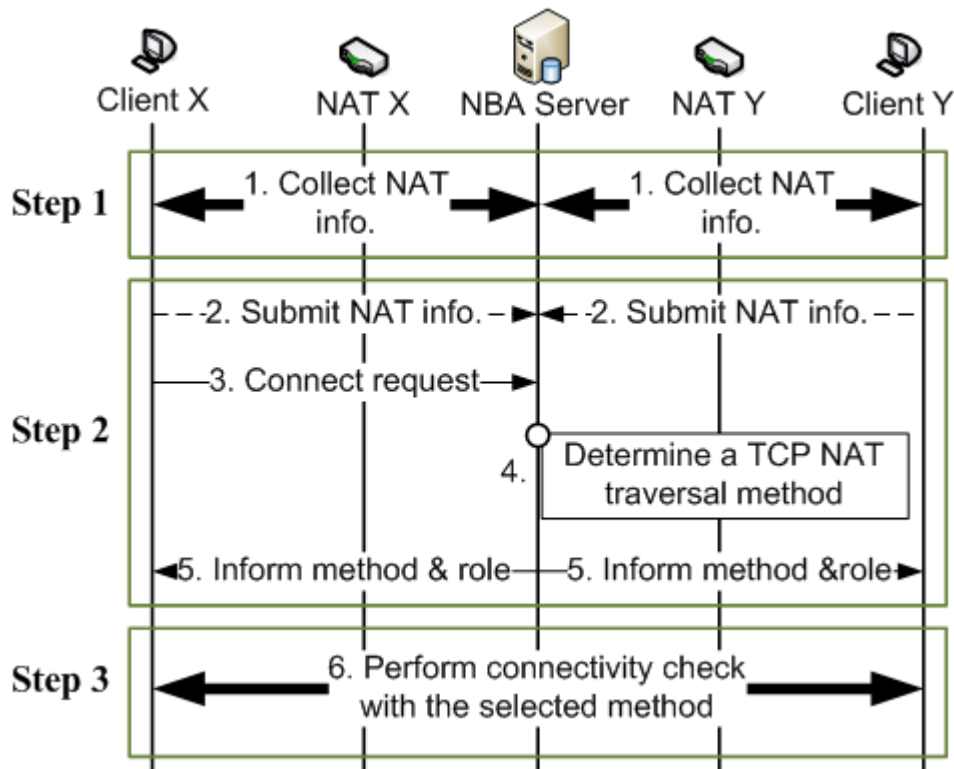


Figure 5-2 NBA operated procedure

In Figure 5-2, Node A and Node B are two hosts behind different NATs (i.e. NAT X and NAT Y), and they want to establish a direct TCP connection. First, *Step 1* of NBA, user agent (UA) of Node A or Node B collects behavior information of its NAT via performing several NAT type tests when it boots up. Second, *Step 2* of NBA, UAs submit NAT information to a NBA Server respectively when it intends to perform connectivity check with other UAs. And then, the NBA Server will determine the most appropriate traversal method according to the behavior information of the two communicating NATs, and it informs both UAs the selected method. Third, *Step 3* of NBA, UAs perform connectivity check with the selected method.

In the fact, there are several ways to implement *Step 1* and *Step 2* of NBA such that NBA Server could maintain behavior information of all NATs instead of submitting by hosts every time while connectivity checks. Besides, as long as host can discover that its NAT is still the serving one or not, it can reuse the NAT information from the NAT type examinations until replacing the NAT. We will discuss in detail many possible

variations of implementation for NBA in chapter 5.

5.3 Step 1 of NBA – NAT Information Collection

In *Step 1* of NBA, UA collects three kinds of information that are mapping rule, filtering rule and TCP state tracking to understand its NAT comprehensively. First, because NAT uses mapping rule to decide port-mapping of each connection, and the port-mapping plays an important role of NAT traversal. Therefore, mapping rule of NAT is critical information in NBA. Second, since filtering rule is used by NAT to determine how to treat inbound packets via an existent port or respond to unsolicited inbound packets via a non-existent port. NBA should also need this kind of information. Finally, because NAT uses TCP state tracking mechanism to decide whether allow the following packet via the port-mapping with special packet sequences and TCP state tracking is important information of TCP NAT traversal. In summary, NAT type tests in *Step 1* of NBA can be classified into three kinds of detections

- Mapping Rule Detection
- Filtering Rule Detection
- TCP State Tracking Mechanism Detection

By the way, we follow a simple rule to name type tests in NBA. The notation ‘S’ means a SYN packet and ‘R’ means a RST packet and ‘T’ means a ICMP TTL Expired packet, and ‘U’ means a ICMP host unreachable packet. Moreover, in order to define the direction of packets. We use the notation ‘i’ means an inbound packet while ‘o’ means an outbound packet. For example, the notation “*SoRiSi*” means that an outbound SYN packet followed by an inbound RST packet and then an inbound SYN packet.

5.3.1 Mapping Rule Detection

The first NAT type test is *Mapping test*. As shown in Figure 5-3, UA sends binding

request messages to different IP address and port number on NBA Server. According to the binding response messages which are send from different combinations of IP address and port number on NBA Server, UA can determine mapping rule of its NAT. As we described in section 2.3, NAT mapping rule can be classified into three categories: Independent, Address dependent and Port & Address dependent. However, NAT may assign different port numbers to connections when its mapping rule is classified to the last two categories. Moreover, different port number assignment can be classify further to linearly dependent (assign different port numbers linearly for connections) and randomly dependent (assign different port numbers randomly for connections). In NBA, we would like use *Mapping test* to realize that mapping rule of an NAT is independent or not. If mapping rule is not independent, what kinds of port assignments do an NAT present? Therefore, *Mapping test* in NBA has three kinds of results that are independent, linearly dependent or randomly dependent mapping.

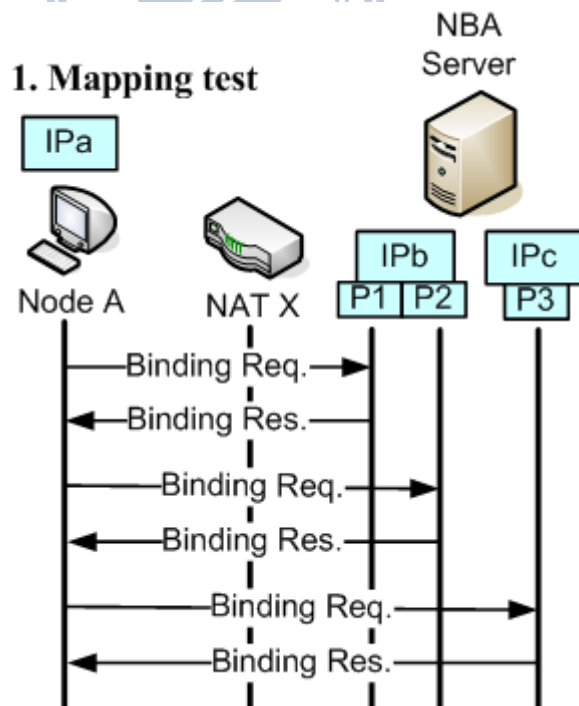


Figure 5-3 Mapping test

5.3.2 Filtering Rule Detection

UA can use *ESi test* in NBA to exam how NAT treats packets inbounded to an existing port. As shown in Figure 5-4 (A), UA on Node A performs a three-way handshake procedure to establish a TCP connection with the first IP address of NBA Server (i.e. IPB). Then, NBA Server initiates other three-way handshake procedure to the UA through the port-mapping on UA's NAT. There are two kinds of result with *ESi test*. If the last inbound SYN packet can pass UA's NAT and be routed successfully to the corresponding UA, we decide the target NAT allows the packet sequence of Establishment then inbound SYN, otherwise it doesn't.

Besides, because of NAT filtering character, NAT is certain of filtering unsolicited inbound packets. However, NBA wants to know the way how NAT filters the packets. Figure 5-4 (B) presents the second NAT type test named *Si test*. NBA Server sends an unsolicited SYN packet to the NAT of UA, and it waits a minute to observe the responses of the NAT. *Si test* has three kinds of result such that NAT drops the unsolicited inbound SYN packet silently, sends back a RST message or an ICMP Host Unreachable error.

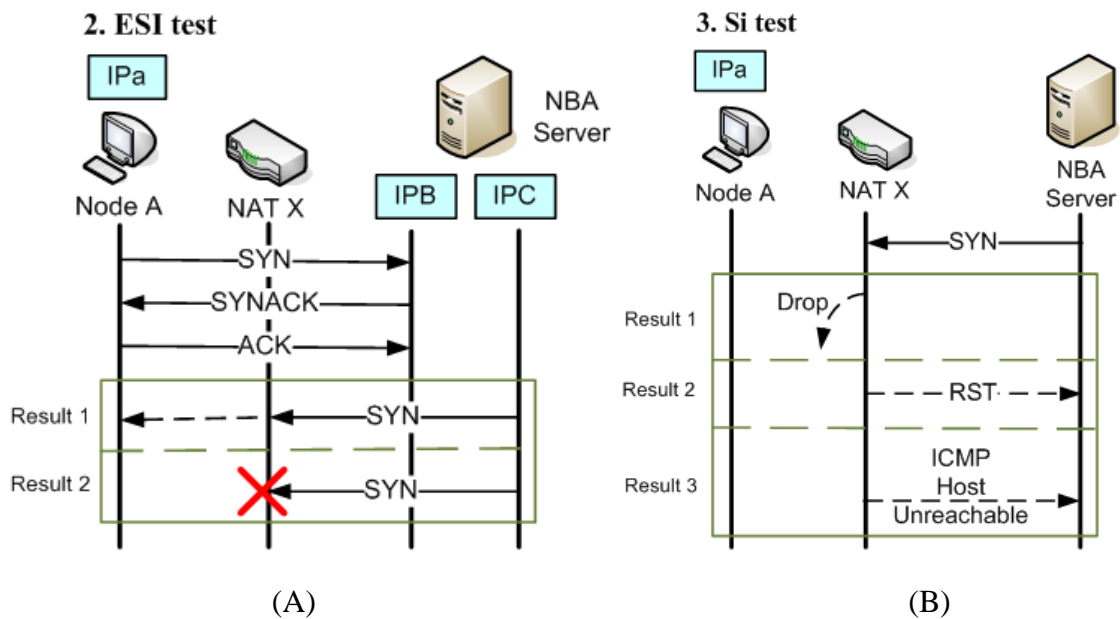


Figure 5-4 ESI test & Si test

5.3.3 TCP State Tracking Detection

As we described in chapter 4, there are many possible packet sequences when UAs perform connectivity check with traversal methods in NBA. In this section, we introduce these tests about TCP state tracking detection to realize how NAT treats those packet sequences.

SoSi test is the fourth NAT type test in NBA. As shown in Figure 5-5 (A), UA sends an outbound SYN packet in the first to NBA Server, and then NBA Server sends back a SYN packet to the UA via the same port-mapping on NAT X. This procedure can exam whether target NAT allows the packet sequence of outbound SYN followed by inbound SYN. Figure 5-5 (B), Figure 5-6 (A) and Figure 5-6 (B) show the remaining three NAT type tests in NBA to detect NAT state tracking that are *SoRiSi test*, *SoUiSi test* and *SoTiSi test*. Each test of the three tests is similar to *SoSi test*, but NBA Server sends a specific message to target NAT before the last inbound SYN. The specific message is a RST packet or an ICMP Host Unreachable packet or an ICMP TTL-Expired packet.

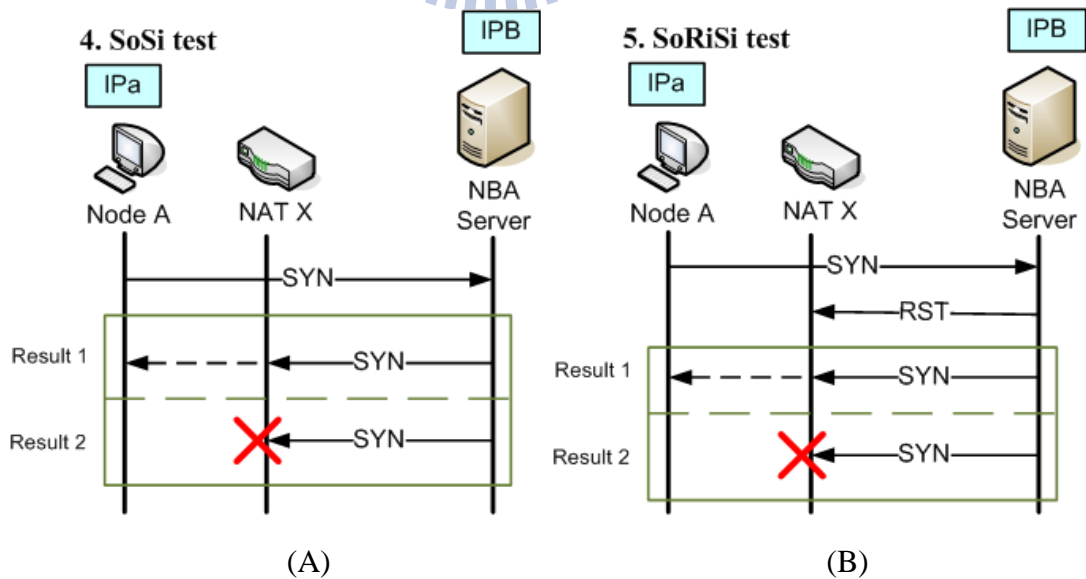


Figure 5-5 SoSi test & SoRiSi test

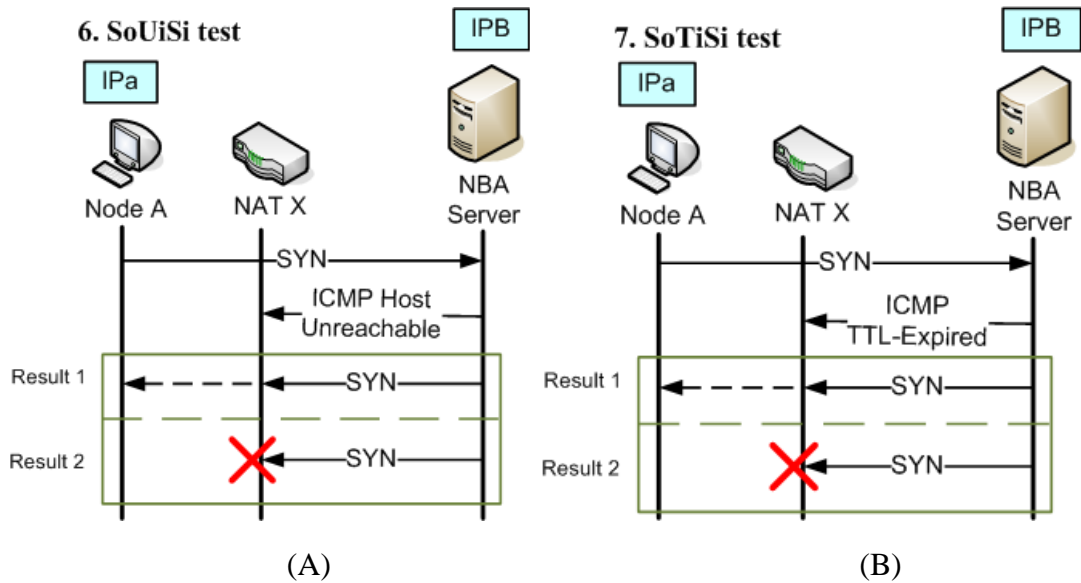


Figure 5-6 SoUiSi test & SoTiSi test

5.3.4 Summary of Type Tests in NBA

In summary, seven tests can be classified into three kinds of detection

- Mapping Rule Detection
 1. *Mapping test*
- Filtering Rule Detection
 2. *ESi test*
 3. *Si test*
- TCP State Tracking Mechanism Detection
 4. *SoSi test*
 5. *SoRiSi test*
 6. *SoUiSi test*
 7. *SoTiSi test*

NBA assists hosts behind a NAT to recognize NAT's behaviors about various sequences of packets. NBA simulates the same packet sequences via NBA Server as those sequences which appear on current TCP NAT traversal methods to examine how NAT behaves to them. UAs perform those examinations with NBA Server respectively, and

they submit the results to NBA Server. The results are a useful knowledge for NBA Server to determine whether NAT traversal methods can work well or not. Figure 5-7 shows the procedure of NAT behaviors examination in NBA.

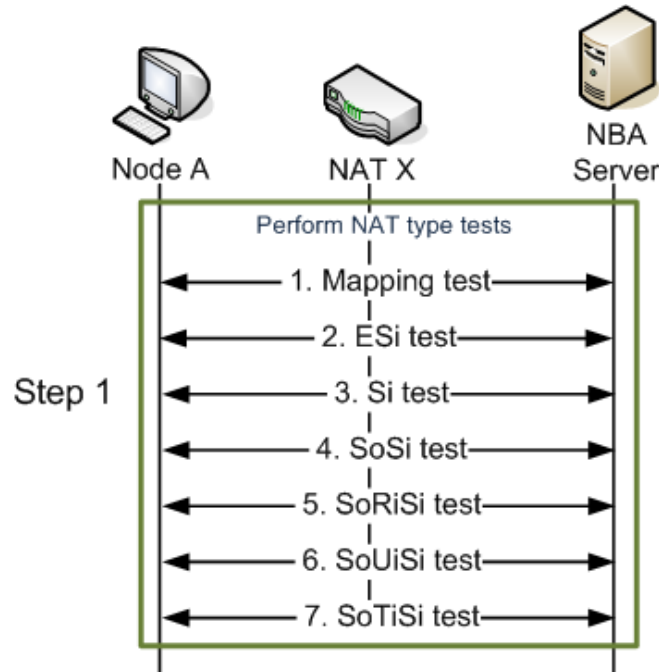


Figure 5-7 Procedure of Step 1 in NBA

After UA performs these NAT type tests, it will have comprehensive knowledge about its NAT's behaviors. Table 5-1 shows possible results of each test in *Step 1*. Then, UA must send the information to NBA Server when it wants to traverse NATs.

Table 5-1 List of NAT type tests

Categories	No.	Test Item	Result
Mapping Rule Detection	1	Mapping	Independent/ Linearly dependent/ Randomly dependent
	2	ESi	Yes/ No
Filtering Rule Detection	3	Si	Drop/ RST/ ICMP Host Unreachable
	4	SoSi	Yes/ No
TCP State Tracking Detection	5	SoRiSi	Yes/ No
	6	SoUiSi	Yes/ No
	7	SoTiSi	Yes/ No

5.4 Step 2 of NBA – Traversal Method Determination

In this section, we describe the selection algorithm of NBA Server to select the most appropriate traversal method based on NATs' behavior knowledge which is collected by UAs via performing those NAT type tests.

5.4.1 Main Idea

Once two UAs behind NATs want to traverse their NATs through NAT connectivity check with some kinds of NAT traversal methods, their NATs must support corresponding special packet sequences so that the UAs can use the method to traverse NATs successfully. So, as long as we take look at the two NATs' behaviors, we can realize which TCP NAT traversal methods can be or cannot be used for NATs to traverse their NATs. In NBA, NBA Server is the role of determination about the traversal method, and it has *a priori* knowledge about NAT connectivity checks. However, UA could also be the role of determination in other concerns about system resource utilizations and so on. We will discuss possible implementation approaches to *Step 2* in chapter 6.

5.4.2 Priority of Traversal Methods

Because we determine the traversal method based on NAT information, several traversal methods may work at the same time for a combination of NATs. In order to achieve an objective about less resource utilization while connectivity checks in NBA, we must prioritize these traversal methods. First, since relay method needs use additional bandwidth to assist UAs to redirect packets, it has the lowest priority. In the other hand, if the combination of NATs is traversable, NBA is as much as possible to find out a traversal method with the highest priority. ESi method can reuse a port-mapping which is created by existent connection on a NAT with NBA Server to traverse NATs. It does not need to create additional port-mapping. Therefore, ESi method has the highest

priority. As for SNT and SLT method, SLT has a potential problem. It requires the UA to determine a TTL large enough to cross its own NAT and low enough to not reach the other UA's NAT. So, SNT method has higher priority than SLT method. In summary, priority of traversal methods is listed as follows:

1. ESi
2. SNT
3. SLT
4. Relay

In fact, the priority is not unique, and it could be dynamically adjusted base on different assumptions while implementations. The adjustment will not influence DCR.

Besides, since which peer to initiate NAT connectivity check may affect the result of the check [18], NBA also determines the most appropriate role between the two UAs while connectivity check. For example in Figure 5-8, Node A and Node B use the TCP NAT traversal method, SLT, to traverse NATs. Node A's NAT (i.e. NAT X) does not allow the packet sequence of an outbound SYN followed by an ICMP TTL-Expired message and then an inbound SYN, but Node B's NAT (i.e. NAT Y) does. When Node B initiates NAT connectivity check with SLT, the NATs can be traversed successfully.

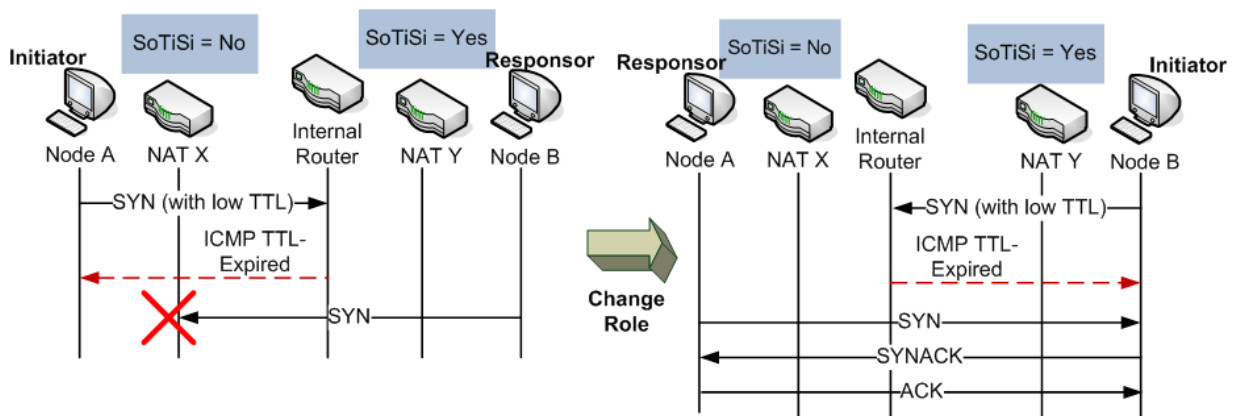


Figure 5-8 Initiator of connectivity check

Moreover, NBA defines *initiator* is the one who initiates NAT connectivity check while *responser* is another peer in the check.

5.4.3 Selection Algorithm

NBA Server uses a selection algorithm to select the most appropriate traversal method based on NAT behaviors. We present this algorithm as a tabular form, and the table is named *Traversal Method Selection Table* (TMST) in NBA.

Table 5-2 Traversal Method Selection Table

Priority	Caller		Callee		Method	Initiator
C1	ESi = Yes		---		ESi	Caller
C2	---		ESi = Yes		ESi	Callee
C3	Mapping = Randomly dep.		---		Relay	Both
	---		Mapping = Randomly dep.		Relay	Both
C4	SoSi = Yes		Si =	Drop	SNT	Caller
	SoUiSi = Yes			UNR	SNT	Caller
	SoRiSi = Yes			RST	SNT	Caller
C5	Si =	Drop	SoSi = Yes		SNT	Callee
		UNR	SoUiSi = Yes		SNT	Callee
		RST	SoRiSi = Yes		SNT	Callee
C6	SoTiSi		---		SLT	Caller
C7	---		SoTiSi		SLT	Callee
C8	---		---		Relay	Both

TMST is shown in Table 5-2. Because ESi method has the highest priority among traversal methods in NBA, if NAT of caller or callee allows *ESi test*, then NBA selection algorithm will select “**ESi**” as the most appropriate traversal method according to the combination of NATs. Moreover, the initiator of NAT connectivity check would be the UA whom NAT allows *ESi test*. Next, if two NATs don’t allow *ESi test*, we should exam whether the results of NATs’ *Mapping test* is randomly dependent or not. It is impossible to use the remaining traversal methods to traverse NATs successfully in NBA when one of the NATs have randomly dependent mapping. As a result, traversal method

which is selected by NBA Server for the UAs with this kind of NAT behaviors is “**Relay**”.

Then, selection algorithm will choose the most appropriate traversal method to UAs continuously. Traversal method with second priority in NBA is SNT, so the selection algorithm uses information of two communicating NATs to decide whether UAs behind the two NATs can use SNT to perform connectivity check. The procedure of SNT causes responder’s NAT to respond three kinds of responses to the unsolicited inbound SYN. Therefore, selection algorithm should use the *Si test* result of responder’s NAT to decide which test result of initiator’s NAT to take look. For example, if the responder’s NAT responds a RST packet to a unsolicited inbound packet, we should take look at the *SoRiSi* test result of initiator’s NAT to determine whether SNL can be applied successfully by UAs with the combination of NATs, as shown in Figure 5-9.

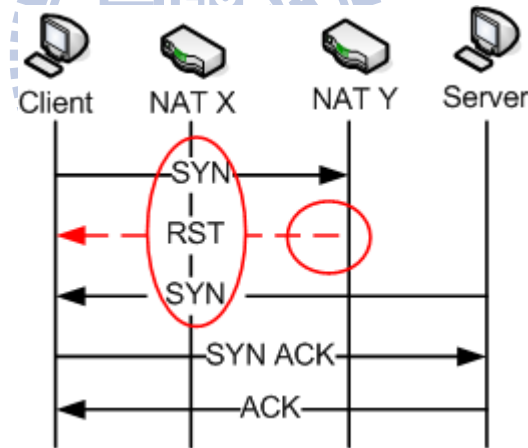


Figure 5-9 A possible packet sequence of SNT

Next, if selection algorithm decides that a combination of NATs is possible to establish a direct TCP connect with a traversal method in NBA, but the traversal method is not ESi or SNT. So, the algorithm then decides whether SLT method can be use to do connectivity check in the combination of NATs. If NAT of caller or callee allows *SoTiSi test*, then the algorithm selects “**SLT**” as the most appropriate traversal method, and the initiator of NAT connectivity check is the UA whom NAT allows *SoTiSi test*. Pseudo

code of selection algorithm is presented in the follow.

In summary, NBA selection algorithm can determine two kinds of results that are the appropriate traversal method and an initiator of connectivity check. NBA informs the results to UAs.



Traversal Method Selection Algorithm

1. *CallerNAT* \leftarrow *NAT Info. of Caller*
2. *CalleeNAT* \leftarrow *NAT Info. of Callee*
3. **if** *CallerNAT.ESi* = *Yes* **then**
4. *Selected traversal method is ESi*
5. *Initiator of connectivity check is Caller*
6. **else if** *CalleeNAT.ESi* = *Yes* **then**
7. *Selected traversal method is ESi*
8. *Initiator of connectivity check is Callee*
9. **else if** *CallerNAT.Mapping* = *Randomly dependent* || *CalleeNAT.Mapping* = *Randomly dependent* **then**
10. *Selected traversal method is Relay*
11. *Both peers initiate connectivity check*
12. **else if** *CalleeNAT.Si* = *Drop* && *CallerNAT.SoSi* = *Yes* **then**
13. *Selected traversal method is SNT*
14. *Initiator of connectivity check is Caller*
15. **else if** *CalleeNAT.Si* = *RST* && *CallerNAT.SoRiSi* = *Yes* **then**
16. *Selected traversal method is SNT*
17. *Initiator of connectivity check is Caller*
18. **else if** *CalleeNAT.Si* = *UNR* && *CallerNAT.SoUiSi* = *Yes* **then**
19. *Selected traversal method is SNT*
20. *Initiator of connectivity check is Caller*
21. **else if** *CallerNAT.Si* = *Drop* && *CalleeNAT.SoSi* = *Yes* **then**
22. *Selected traversal method is SNT*
23. *Initiator of connectivity check is Callee*
24. **else if** *CallerNAT.Si* = *RST* && *CalleeNAT.SoRiSi* = *Yes* **then**
25. *Selected traversal method is SNT*
26. *Initiator of connectivity check is Callee*
27. **else if** *CallerNAT.Si* = *UNR* && *CalleeNAT.SoUiSi* = *Yes* **then**
28. *Selected traversal method is SNT*
29. *Initiator of connectivity check is Callee*
30. **else if** *CallerNAT.SoTiSi* = *Yes* **then**
31. *Selected traversal method is SLT*
32. *Initiator of connectivity check is Caller*
33. **else if** *CalleeNAT.SoTiSi* = *Yes* **then**
34. *Selected traversal method is SLT*
35. *Initiator of connectivity check is Callee*
36. **else**
37. *Selected traversal method is Relay*
38. *Both peers initiate connectivity check*
39. **endif**

5.5 Step 3 of NBA – Connectivity Check

An initiator of two UAs then initiates connectivity check with the notify methods which is selection by NBA Server based on information of their NATs.



Chapter 6

System Implementation

This chapter describes system implementations of NBA scheme. Section 6.1 presents the system overview. Section 6.2 shows our current implementation. Section 6.3 discusses ways to determine the most appropriate traversal method in different considerations.

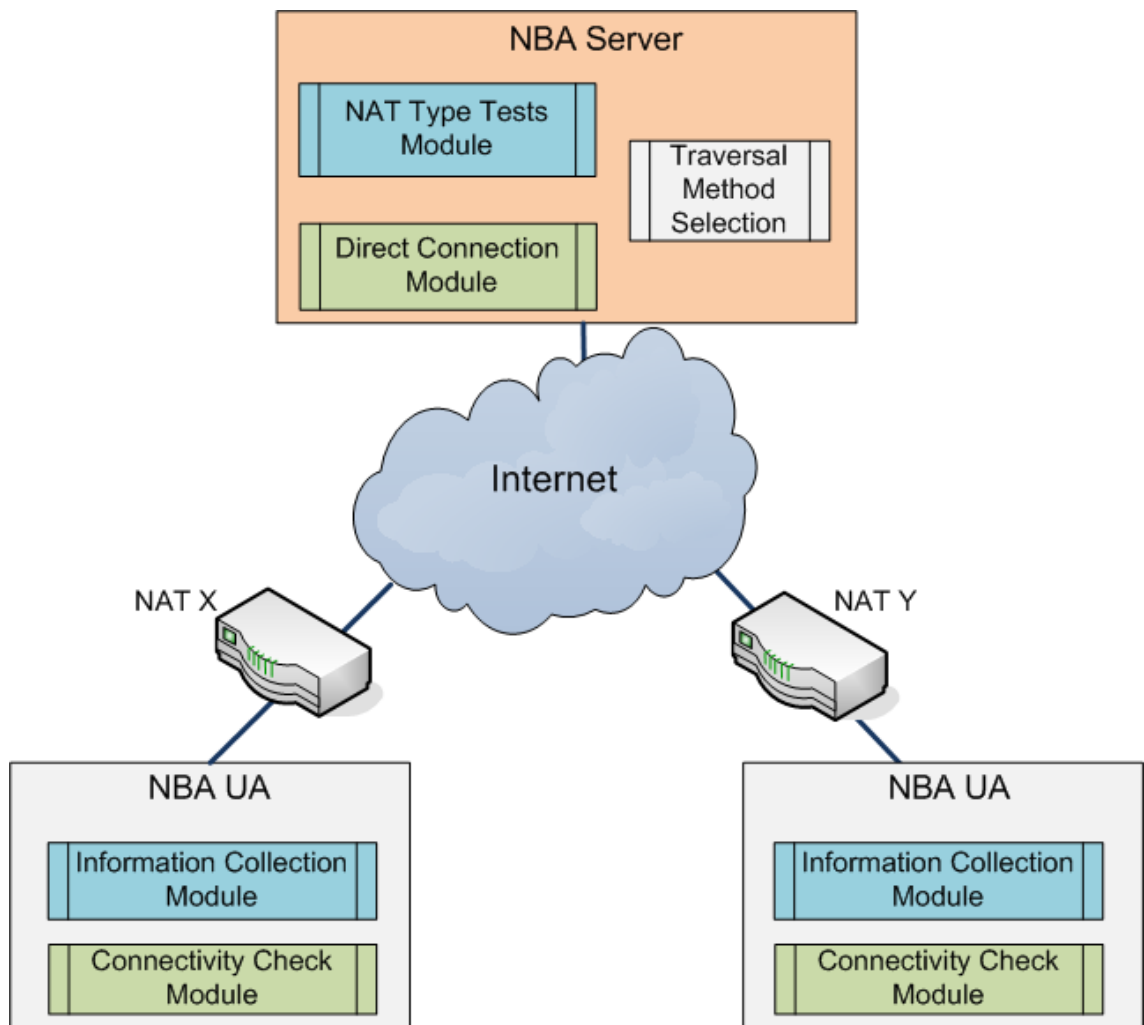


Figure 6-1 System modules of NBA

6.1 System Overview

Figure 6-1 illustrates the system topology and modules of NBA scheme. NBA topology which is similar to traditional NAT traversal topologies consists of two UAs behind NATs and a third-party server named NBA Server. The NBA Server located on the internet provides three modules that are *NAT Type Tests Module*, *Traversal Method Selection Module* and *Direct Connection Module*. As for UA, it has two modules that are *Information Collection Module* and *Connectivity Check Module*.

6.2 NBA Implementation

In NBA, all implementations are based on Linux and use C language. As shown in Figure 6-1, NBA provides the following functional components:

- NBA UA
- 1. **Information Collection Module:** NBA Server provides seven type tests such as *Mapping test*, *ESi test*, *Si test*, *SoSi test*, *SoRiSi test*, *SoUiSi test* and *SoTiSi test*, and each test was described in detail in chapter 5. *Information Collection Module* provides an interface for NBA UA to perform these NAT type tests to collect knowledge of its NAT. This module is integrated from the client side of STUNT. STUNT was implemented by Cornell University at 2005 and it extends STUN to include TCP functionality [22]. STUNT is a lightweight protocol that allows UAs running behind a NAT to determine external IP and port-mapping properties, packet filtering rules and various timeout associated with TCP connections through the NAT.
- 2. **Connectivity Check Module:** *Connectivity Check Module* implemented three kinds of TCP NAT traversal methods that are SNT, SLT and ESi. This module can assist NBA UAs to perform connection check with current traversal methods. It is integrated from the client side of XSTUNT which is a C/C++ li-

brary and can assist two hosts behind NATs to establish a direct TCP connection via traversal methods [23].

- NBA Server
 1. **NAT Type Tests Module:** *NAT Type Tests Module* which corresponds to *Information Collection Module* in NBA UA provides seven NAT type tests, and it assists UAs to collect NAT information. This module is integrated form the server side of STUNT. Similarly, it has the functionality for UAs to determine its NAT's external IP and port-mapping properties, and it can generate several special packet sequences to test TCP state tracking of NAT by using row socket.
 2. **Traversal Method Selection:** *Traversal Method Selection* implements a selection algorithm which is introduced in chapter 5 to select the most appropriate traversal method according to information of two communicating NATs. In NBA, this method selection is implemented in NBA Server. However, there are different approaches to implement this method selection and put it in different location, and we will discuss in detail in the next section.
 3. **Direct Connection Module:** *Direct Connection Attempting Module* provides the functionalities of connectivity check for UAs, and it is integrated form the server side of XSTUNT. This module which corresponds to *Connectivity Check Module* in NBA UA implements the three TCP NAT traversal methods, too. It assists UAs to perform connectivity check.

6.2.1 Interaction of Modules

In this section, we describe the interactions of modules in NBA. Figure 6-2 shows the module interaction of *Step 1* in NBA. UA uses *Information Collection Module* to

interact with *NAT Type Tests Module* in NBA Server for collecting NAT information which are mapping rule, filtering rule and TCP state tracking.

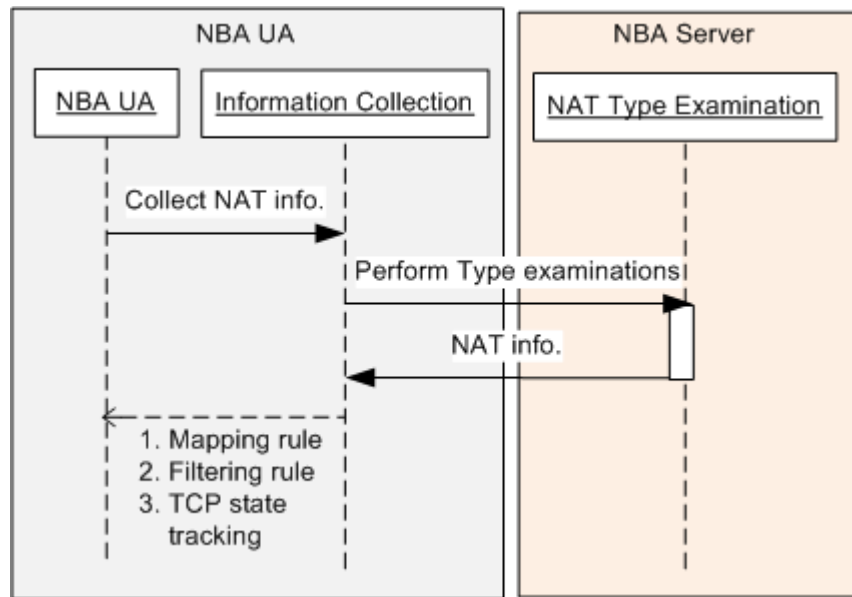


Figure 6-2 Module interaction flow of Step 1

Figure 6-3 shows the module interaction of *Step 2* in NBA. UA interacts with *Traversal Method Selection* in NBA Server by submitting its NAT information, and *Traversal Method Selection* uses behavior information of the two NATs to determine the most appropriate traversal method, and then NBA Server informs UAs the selected traversal method.

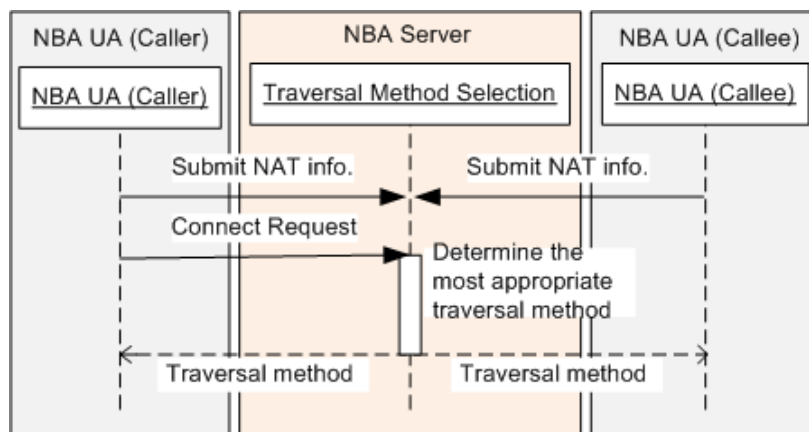


Figure 6-3 Module interaction flow of Step 2

Figure 6-4 shows the module interaction of *Step 3* in NBA. UAs use *Connectivity Check Module* to interact with *Direct Connectivity Module* in NBA Server for perform-

ing connectivity check with the selected traversal method.

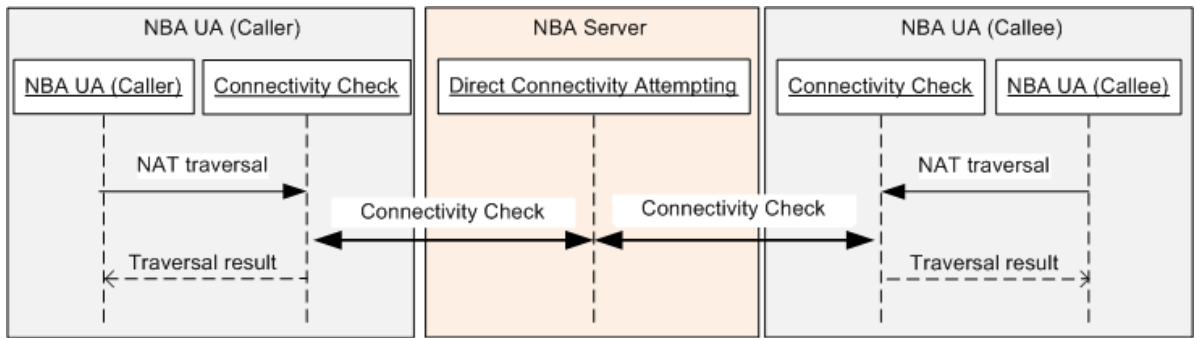
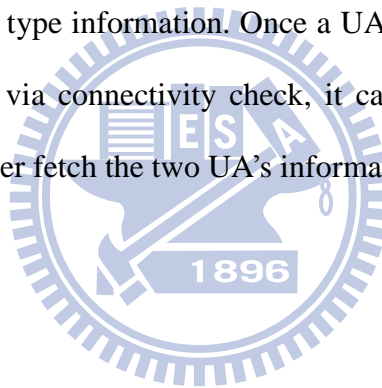


Figure 6-4 Module interaction flow of Step 3

6.3 Discuss Ways to Implement NBA Scheme

We could implement a Signal Server in NBA to maintain all UA's information including ID and NAT type information. Once a UA wants to establish a direct connection with another UA via connectivity check, it can send a request message to NBA Server. Then NBA Server fetch the two UA's information from Signal Server.



Chapter 7

Experiment

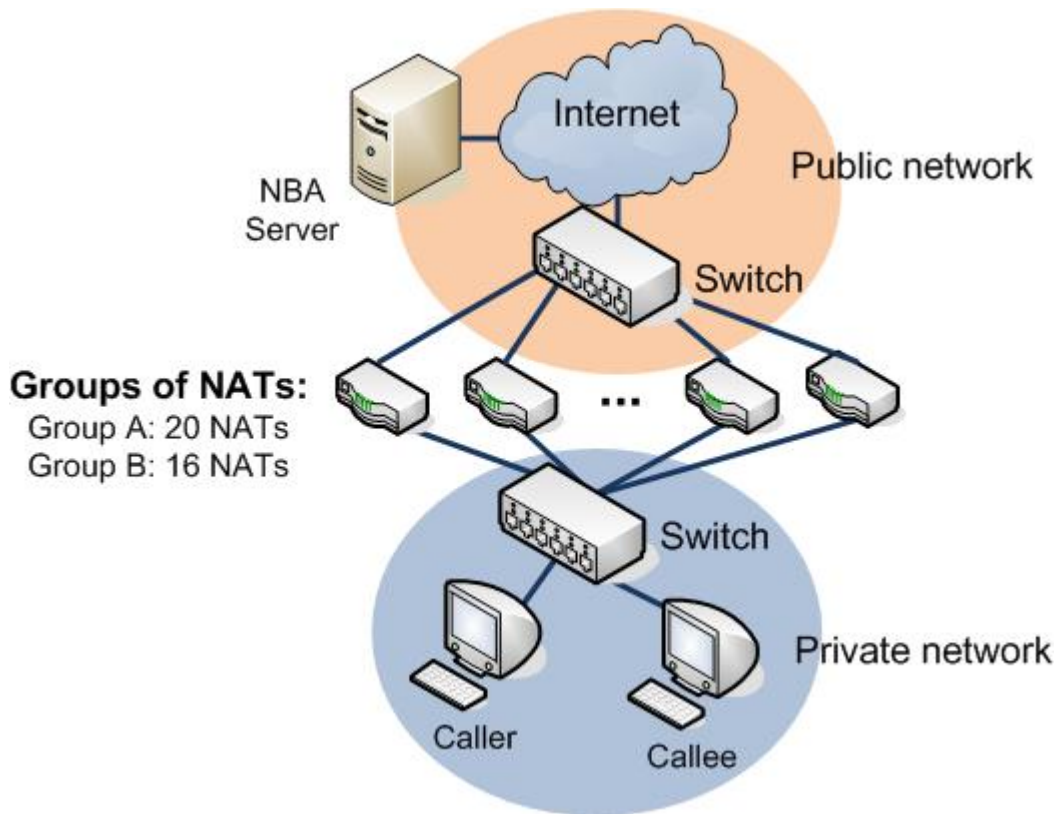


Figure 7-1 Experiment environment

NBA scheme developed several NAT type tests and the selection algorithm of selecting the most appropriate traversal method. This scheme also integrated three kinds of TCP NAT Traversal methods on Linux. In this chapter, we present our experiment environment and describe the process of our experiments and then analyze results of experiments. We use two groups of NATs to construct a fully mesh architecture as shown in Figure 7-1. In our environment, we have a NBA Server and two peers, caller and callee, and execute NBA scheme under two group of NAT which were bought by us at 2008 and 2010. Table 7-1 shows brands of the 36 NATs.

Table 7-1 Brands of NATs

Group A				Group B			
No.	Brand	No.	Brand	No.	Brand	No.	Brand
A1	3Com	A11	Lemel	B1	D-Link 635	B11	PCI WNH
A2	AboCom	A12	Netgear	B2	D-Link 628	B12	ASUS
A3	Asus	A13	Planex	B3	D-Link 615	B13	Abocom
A4	Buffalo	A14	Smc	B4	D-Link 825	B14	Belkin
A5	Belkin	A15	Zyxel	B5	BUFFALO	B15	Aximcom
A6	Corega	A16	Windows	B6	PCI W300	B16	Levelone
A7	Draytek	A17	FreeBSD	B7	Smc		
A8	D-link	A18	Linux	B8	Zyxel		
A9	Edimax	A19	Smc Wireless	B9	Edimax		
A10	Linksys	A20	Linksys N	B10	Corega		

7.1 Overview of Experiment

In our experiment, we analyze performances of NBA under various NAT combinations to establish a direct TCP connection through NAT connectivity check. As we described in chapter 5, there are three steps in NBA. In the following sections, we present the results of these three steps in NBA respectively, and we compare NBA scheme with the following two schemes:

- Sequential Connectivity Check with Initiator Changes (IC):

This scheme tries each traversal method one-by-one and change initiator to tray each method in opposite direction, so the executive order of traversal methods about connectivity check in this scheme is SNT → SNT-IC → SLT → SLT-IC → ESi → ESi-IC

- Parallel Connectivity Check (PCC)

This scheme tries all traversal methods at the same time that are SNT & SNT-IC & SLT & SLT-IC & ESi & ESi-IC

We compare NBA with the two schemes in three metrics that are

1. Direct Connection Rate (DCR)
2. Connectivity Check Time
3. System Resource Utilizations

7.2 Result of Step 1 in NBA

Step 1 of NBA is for UA to collect its NAT information about mapping rule, filtering rule and TCP state tracking. After UA perform NAT type tests, it has a comprehensively knowledge of its NAT. For the purpose of convenience, NATs are marked by us with symbols that mean NATs has corresponding types. The naming rule is listed on Table 7-2, Table 7-3 and Table 7-4. In Table 7-2, if a NAT's test results of *Mapping test* is randomly dependent, we mark the NAT a notation 'I'. That means the NAT is impossible to be traversed. Next, there are three kinds of test results about *Si test*, and each result has a unique mark such as 'D', 'R' or 'U' as shown in Table 7-3. A NAT only has one kind of result, because it chooses one kind of response to respond unsolicited inbound packets. Then, if a NAT allows *ESi test*, we mark the NAT a notation 'E'. Finally, *SoSi*, *SoRiSi*, *SoTiSi* and *SoUiSi test* are four type tests about TCP state tracking detection, if a NAT allows some kinds of these tests, we mark a corresponding symbol that are 'd', 'r', 't' and 'u' as shown in Table 7-4. For example, if a NAT drops unsolicited packets, and it allows "SoSi" and "SoTiSi" tests but does not allow "SoRiSi" test, we mark the NAT as "D-dt".

Table 7-2 Naming rule of mapping detection

Mapping Detection	
Result Test	Randomly dependent
Mapping	I (Impossible to be traversed)

Table 7-3 Naming rule of filtering detection

Filtering Detection			
Result Test	Drop	RST	Host Unreachable
Si	D	R	U
Result Test	Yes		
ESi	E		

Table 7-4 Naming rule of TCP state tracking detection

Filtering Detection				
Result Test	SoSi	SoRiSi	SoTiSi	SoUiSi
Mark	d	r	t	U

We can then mark NATs in group A and group B according to this naming rule.

Table 7-5 presents marking result of NATs in group A while Table 7-6 shows NATs in group B.

Table 7-5 Classify of NATs in group A

No. of NAT	Test Results						Mark
	Mapping	ESi	Si	SoSi	SoRiSi	SoTiSi	
A8	Independent	Yes	Drop	No	Yes	No	ED-r
A7, A11, A13, A15, A19	Independent	No	Drop	Yes	Yes	Yes	D-drt

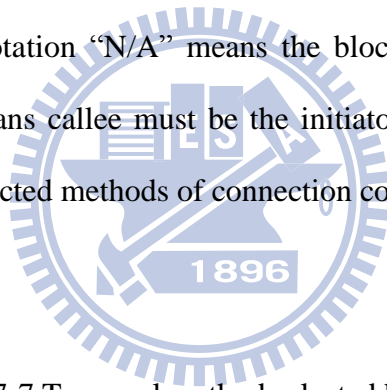
A14, A16, A17	Linearly dependent	No	RST	Yes	Yes	Yes	R-drt
A1, A20	Independent	No	Drop	Yes	No	Yes	D-dt
A4, A9	Independent	No	RST	Yes	No	Yes	R-dt
A6	Independent	No	RST	No	Yes	No	R-r
A5	Independent	No	Drop	No	No	No	D-^
A18	Independent	No	RST	No	No	No	R-^
A2, A3, A10, A12	Randomly dependent	No	Drop	No	No	No	I-D-^

Table 7-6 Classify of NATs in group B

No. of NAT	Test Results						Mark
	Mapping	ESi	Si	SoSi	SoRiSi	SoTiSi	
B16	Independent	Yes	Drop	Yes	Yes	Yes	ED-drt
B1, B2	Independent	No	Drop	Yes	Yes	Yes	D-drt
B5, B10	Independent	No	Drop	Yes	No	Yes	D-dt
B3, B4, B6, B7, B8, B12, B14	Independent	No	Drop	No	No	No	D-^
B9, B11, B13, B15	Independent	No	RST	No	No	No	R-^

7.3 Result of Step 2 in NBA

Step 2 of NBA is for NBA Server to determine the most appropriate traversal method base on information of two communicating NATs. Table 7-7 shows the results of Step 2 in NBA about traversal method chosen by NBA Server to each connection combination of NATs in group A. Behaviors of NATs in the same class are the same, so NBA Server chooses the same traversal method to connection combinations in a kind of class combination. Since we consider two peers are under different NATs only, there have no combination of NATs in some cases. In Table 7-7, numerals mean three kinds of three TCP traversal methods implemented in NBA, and notation ‘R’ means the NATs of combinations cannot be traversed to establish a direct connection and must use relay method. Moreover, notation “N/A” means the block has no connection combination, and notation “IC” means callee must be the initiator of connectivity check. Similarly, Table 7-8 presents selected methods of connection combinations of NATs in group B.



1: SNT; 2: SLT; 3: ESi; R: Relay
N/A: Non-Existent,
IC: Initiator Change

Table 7-7 Traversal method selected by NBA to group A

		Responsor									
		Class	ED-r	D-drt	R-drt	D-dt	R-dt	R-r	D-^	R-^	I-D-^
Initiator	ED-r	N/A	3	3	3	3	3	3	3	3	3
	D-drt	3, IC	1	1	1	1	1	1	1	1	R
	R-drt	3, IC	1	1	1	1	1	1	1	1	R
	D-dt	3, IC	1	1, IC	1	1, IC	2	1, IC	2	2	R
	R-dt	3, IC	1	1, IC	1	2	1, IC	1	2	2	R
	R-r	3, IC	1, IC	1	2, IC	1	N/A	R	1	1	R
	D-^	3, IC	1, IC	1, IC	1, IC	1, IC	1, IC	R	N/A	R	R
	R-^	3, IC	1, IC	1, IC	2, IC	2, IC	1, IC	R	N/A	R	R
	I-D-^	3, IC	R	R	R	R	R	R	R	R	R

Table 7-8 Traversal method selected by NBA to group B

		Responsor				
		Class	ED-drt	D-drt	D-dt	D-^
Initiator	ED-drt	N/A	3	3	3	3
	D-drt	3, IC	1	1	1	1
	D-dt	3, IC	1	1	1	2
	D-^	3, IC	1, IC	1, IC	R	R
	R-^	3, IC	1, IC	2, IC	R	R

7.4 Comparison of Direct Connection Rate in Different Schemes

In this section, we compare direct connection rate of NBA scheme with SCC and PCC schemes.

7.4.1 Result of Step 3 in NBA

Step 3 of NBA is for UAs to perform TCP NAT traversal via the method which is chosen by NBA Server in *Step 2*. Table 7-9 and Table 7-10 present the connectivity check results of group A and group B after performing *Step 3* in NBA. We put an asterisk (★) to a block when each connection combination can establish a direct TCP connection in the block. The DCR is 56.84% in group A and 54.17% in group B by using NBA scheme.

★ : Direct Connection

N/A: Non-Existent

Table 7-9 Direct connection of group A made by NBA

		Responzor								
Class		ED-r	D-drt	R-drt	D-dt	R-dt	R-r	D-^	R-^	I-D-^
Initiator	ED-r	N/A	★	★	★	★	★	★	★	★
	D-drt	★	★	★	★	★	★	★	★	
	R-drt	★	★	★	★	★	★	★	★	
	D-dt	★	★	★	★	★	★	★	★	
	R-dt	★	★	★	★	★	★	★	★	
	R-r	★	★	★	★	★	N/A		★	
	D-^	★	★	★	★	★		N/A		
	R-^	★	★	★	★	★	★		N/A	
	I-D-^	★								

Table 7-10 Direct connection of group B made by NBA

		Responzor				
Class		ED-drt	D-drt	D-dt	D	R
Initiator	ED-drt	N/A	★	★	★	★
	D-drt	★	★	★	★	★
	D-dt	★	★	★	★	★
	D	★	★	★		
	R	★	★	★		

In summary, according to experimental data of section 7.3 and section 7.4.1, we could conclude that it does not have any error selection existing in NBA selection algorithm. Therefore, we could claim that traversal method chosen from NBA Server can succeed in traversal target NATs exactly. NBA has no miscarriage of justice about traversal methods while connectivity checks.

7.4.2 DCR of Other Traversal Scheme

- SCC with change role or PCC

1. Group A

Table 7-11 Direct connection of group A made by SCC or PCC

		Responzor								
Class		ED-r	D-drt	R-drt	D-dt	R-dt	R-r	D-^	R-^	I-D-^
Initiator	ED-r	N/A	★	★	★	★	★	★	★	★
	D-drt	★	★	★	★	★	★	★	★	
	R-drt	★	★	★	★	★	★	★	★	
	D-dt	★	★	★	★	★	★	★	★	
	R-dt	★	★	★	★	★	★	★	★	
	R-r	★	★	★	★	★	N/A		★	
	D-^	★	★	★	★	★		N/A		
	R-^	★	★	★	★	★	★		N/A	
	I-D-^	★								

2. Group B

Table 7-12 Direct connection of group B made by SCC or PCC

		Responzor				
Class		ED-drt	D-drt	D-dt	D-^	R-^
Initiator	ED-drt	N/A	★	★	★	★
	D-drt	★	★	★	★	★
	D-dt	★	★	★	★	★
	D-^	★	★	★		
	R-^	★	★	★		

Direct connection rates of NBA, SCC and PCC are the same in the same group.

DCR of group A is 56.84%; while group B is 54.17%.

7.5 Comparison of System Resource Utilization in Different Schemes

The bottom layer of implementations for the three kinds of TCP NAT traversal methods are the same in SCR, PCC and NBA scheme. All schemes use the prototypes implemented by D-Link NCTU Joint Research Center (DNJRC). Therefore, the

schemes have the same procedures and waiting time in each traversal method.

7.5.1 Connectivity Check Time

In this section, we make a comparison of the total connectivity check time in SCC, PCC and NBA scheme. Table 7-12 presents average connectivity check time of different traversal methods; we experiment ten times of each data. Due to each traversal method only exchange a few message to traverse NATs, it takes a very short time which connectivity checks. However, when connectivity check fails, we must spend several seconds to wait for results. Besides, connectivity check time of Relay is set to be zero because our experiment focuses on direct connection. We assume it is the time that UAs start to establish relay connection.

Table 7-12 connectivity check time

Method \ Result	SNT	SLT	ESi	Relay
Success	1.1602 s	0.1605 s	0.0917 s	0 s
Failure	9.2514 s	8.1507 s	8.0739 s	---

Table 7-13 Traversal time of each scheme

Scheme \ Method	Initiator	SCC	PCC with Relay	NBA
SNT	Caller	1.1602 s	1.1602 s	1.1602 s
	Callee	10.4116 s		
SLT	Caller	18.6633 s	0.1605 s	0.1605 s
	Callee	26.8140 s		
ESi	Caller	34.8959 s	0.0917 s	0.0917 s
	Callee	42.9698 s		
Relay	---	50.952 s	9.2514 s (Apply Relay after all traversal methods) or 0 s (Apply Relay in parallel)	0 s

Table 7-13 shows traversal time of each scheme. Because SCC applies each traversal method one-by-one, it has to accumulate failure time of performed traversal methods. For example, if two UAs use SCC scheme to perform connectivity check, finally callee initiates this check and use ESi traversal method to successfully traverse NATs. This procedure has to accumulate failure time of SNT, SNT-IC, SLT, SLT-IC and ESi. And also add successful time of ESi-IC. It is a long period of time.

The concept of PCC is using all traversal methods at the same time, but there are two approaches to apply Relay method. Someone applies Relay in parallel with other traversal methods; while the other applies it after fail in direct connectivity check. Therefore, while applying Relay in after fail in direct connectivity check, the connectivity check time is the longest one of failure time between traversal methods except Relay.

NBA could eliminate unnecessary connectivity checks, so it does not have to take the time of fail in applying traversal methods.

7.5.2 Resource Utilizations

Table 7-14 shows numbers of message exchanges about each traversal method.

Table 7-14 Numbers of message exchanges of each traversal method

Method \ Result	SNT	SLT	ESi	Relay
Numbers	6	6	3	6

Table 7-15 Numbers of message exchanges of each scheme

Scheme \ Method	Initiator	SCC	PCC with Relay	NBA
SNT	Caller	6	36	6
	Callee	12		
SLT	Caller	18	36	6
	Callee	24		
ESi	Caller	27	36	3
	Callee	30		
Relay	---	36	36	6

Table 7-15 shows numbers of message exchanges about schemes. Because PCC performs all traversal methods in the same time, it must accumulate all numbers of message exchanges in traversal methods. Therefore, it should use a large of system resources within connectivity check procedure.

In summary, according to experimental results of this chapter, NBA has shorter connectivity check delay than SCC and less resource usages or simpler state maintenance than PCC. Besides, NBA always knows whether we can use the existing traversal methods to traverse successfully with specific combination of NATs. But SCC has no idea about traversal methods to NAT combinations. Therefore, PCC sometimes could have higher DCR than SCC.



Chapter 8

Conclusions and Future Works

In this thesis, we have demonstrated that NBA is a powerful TCP traversal scheme. It utilizes NAT information comprehensively to select the most appropriate traversal method to traverse NATs. NBA has a priori knowledge on connectivity of each combination of NAT types. Moreover, NBA could eliminate unnecessary connectivity checks, because it knows what combinations of NAT types are traversable. Therefore, we need not perform NAT traversal when direct connection is impossible. By performing NBA, connectivity check of TCP NAT traversal will be more efficient with shorter check delay, fewer message exchanged, possible higher DCR and less resource usages or simpler state maintenance compared to other schemes.

In NBA, many NAT type examinations are declared to understand a comprehensive set of NAT characteristics as they pertain to TCP, and we develop an algorithm of traversal method determination. We have shown that this algorithm has good judgment in selecting traversal method.

In the future works, there are still many research issues of the proposed scheme. For example, we only implemented three kinds of practical TCP NAT traversal methods in this thesis. Maybe we can implement other current TCP NAT traversal methods such as STUNT #1 and NATBlaster. The direct connection rate will thus increase because of including new methods. In the other hand, perhaps we can develop new TCP traversal methods based on our comprehensively knowledge about TCP NAT behaviors.

NBA uses NAT information to select the most appropriate traversal method, but everyone has his own definitions of appropriateness. Maybe we could give an objective function to our selection algorithm, and using different input parameters will obtain dif-

ferent selection results. Therefore, the procedure of method selection in NBA will be more general and more universal. Besides, we would also try to make our scheme more robust for fault tolerance and shorter time delay.



Bibliography

- [1] K. Egevang and P. Francis, “The IP Network Address Translator (NAT),” *IETF RFC 1631*, May 1994.
- [2] B. Ford, P. Srisuresh and D. Kegel, “Peer-to-Peer Communication Across Network Address Translators,” in USENIX Annual Technical Conference, pp. 179-192, April 2005.
- [3] D. kegel, “NAT and Peer-to-peer Network,” <http://www.kegel.com>, July 1999.
- [4] J. Rosenberg, J. Weinberger, C. Huitema and R. Mahy, “STUN – Simple Traversal of User Datagram Protocol (UDP) through Network Address Translators (NATs),” *IETF RFC 3489*, March 2003.
- [5] J. Rosenberg, R. Mahy and P. Matthews, “Traversal Using Relay around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN),” *IETF RFC 5766*, April 2010.
- [6] J. Rosenberg, “Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols,” *IETF RFC 5245*, April 2010.
- [7] Jeffrey L. Eppinger, “TCP Connections for P2P Apps: A Software Approach to Solving the NAT Problem,” Technical Report CMU-ISRI-05-104, Carnegie Mellon University, January 2005.
- [8] F. Audet and C. Jennings, “Network Address Translation (NAT) Behavioral Requirements for Unicast UDP,” *IETF RFC 4787*, January 2007.
- [9] J. Rosenberg, R. Mahy, P. Matthews and D. Wing, “Session Traversal Utilities for NAT (STUN),” *IETF RFC 5389*, October 2008.
- [10] D.Clark, L. Chapin and V. Cerf, “Towards the Future Internet Architecture,” *IETF*

RFC 1287, December 1991.

- [11] Z. Wang and J. Crowcroft, "A Two-Tier Address Structure for the Internet: A solution to the problem of Address Space Exhaustion," *IETF RFC 1335*, May 1992.
- [12] Y. Rekhter, B. Moskowitz and D. Karrenberg, "Address Allocation for Private Internets," *IETF RFC 1918*, February 1996.
- [13] S. Guha, K. Biswas, B. Ford, S. Sivakumar and P. Srisuresh, "NAT Behavioral Requirements for TCP," *IETF RFC 5382*, October 2008.
- [14] S. Guha, Yutaka Takeday and P. Francis, "NUTSS: A SIP-based approach to UDP and TCP network connectivity," in ACM SIGCOMM Asia Workshops, August 2004.
- [15] A. Biggadike, D. Ferullo, G. Wilson and A. Perrig, "NATBLASTER: Establishing TCP connections between hosts behind NATs," in ACM SIGCOMM Asia Workshop, April 2005.
- [16] S. Guha and P. Frances, "Characterization and Measurement of TCP Traversal through NATs and Firewalls," in 5th ACM SIGCOMM conference on Internet Measurement, pp. 18-18, October 2005.
- [17] S. Guha, K. Biswas, B. Ford, S. Sivakumar and P. Srisuresh, "NAT Behavioral Requirements for TCP," *IETF RFC 5382*, October 2008.
- [18] Cheng-Yuan Ho, Fu-Yu Wang and Chien-Chao Tseng, "To Call or to Be Called under NATs is Sensitive for Solving Direct Connection Problem," Submitted to IEEE Communications Letters, 2010.
- [19] P. Srisuresh, B. Ford and D. Kegel, "State of Peer-to-Peer (P2P) Communication across Network Address Translators," *IETF RFC 5128*, March 2008.
- [20] R. Roverso, S. El-Ansary and S. Haridi, "NATCracker: NAT Combinations Matter," in 18th International Conference on Computer Communications and Network 2009, pp. 1-7, August 2009.

- [21] Zhou Hu, “NAT Traversal Techniques and Peer-to-Peer Applications,” HUT T-110.551 Seminar on Internetworking, 2005.
- [22] STUNT, <http://nutss.gforge.cis.cornell.edu/stunt.php>
- [23] XSTUNT, <http://www.cis.nctu.edu.tw/~gis87577/xDreaming/XSTUNT/index.html>
- [24] R. Denis-Courmont, “Test Vectors for Session Traversal Utilities for NAT (STUN),” *IETF RFC 5769*, April 2010.
- [25] D. MacDonald and B. Lowekamp, “NAT Behavior Discovery Using Session Traversal Utilities for NAT (STUN),” *IETF RFC 5780*, May 2010.

