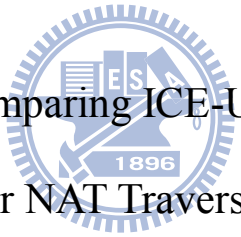
國立交通大學

網路工程研究所

碩士論文

比較與分析 NAT 穿透技術 ICE 在 UDP 和 TCP 上的差異



Analyzing and comparing ICE-UDP and ICE-TCP
for NAT Traversal

研 究 生：吳又賢

指導教授：林盈達　教授

中 華 民 國 九 十 八 年 六 月

比較與分析 NAT 穿透技術 ICE 在 UDP 和 TCP 上的差異

Analyzing and comparing ICE-UDP and ICE-TDP for NAT Traversal

研 究 生：吳又賢　　Student: Yu-Hsien Wu

指導教授：林盈達　　Advisor: Dr. Ying-Dar Lin

國立交通大學

網路工程研究所

碩士論文

A Thesis

Submitted to Institutes of Network Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
In

Computer Science and Engineering

June 2009

HsinChu, Taiwan, Republic of China

中華民國九十八年六月

比較與分析 NAT 穿透技術 ICE 在 UDP 和 TCP 上的差異

學生: 吳又賢 **指導教授: 林盈達**

國立交通大學網路工程研究所

摘要

Network Address Translation (NAT) 的出現破壞了以往 peer-to-peer (P2P) 的連線模式，使得無論 UDP 還是 TCP 在 P2P 連線上出現了許多問題，因此如何穿越 NAT 的方法逐漸被重視。再加上 NAT 處理 UDP 與 TCP 的 connection 行為不同與 UDP 和 TCP 本身的差異(即 UDP 為非連接傳輸模式而 TCP 是非對稱的連線)，因此在 UDP 及 TCP 上各自擁有不同的 solution 來解決穿越 NAT 的問題。雖然目前有 STUN、STUNT、P2PNAT 和 TURN 等穿透 NAT 方法被提出來，但因為這些技術是各別使用而無法一一針對不同的 NAT 行為來解決所有問題，因此 Interactive Connectivity Establishment (ICE)提供一套機制將這些技術整合運用，並透過路徑檢查的機制來探測所有整合技術可穿透 NAT 的路徑。在這裏我們提一個實驗用來測量 ICE 在 UDP/TCP 上的直連率以及分析其實作在 socket layer 以及直連路徑測試程序上的不同以及他們的穿透能力。最後我們發現目前 ICE-UDP 和 ICE-TCP 穿透能力已經達到 DCR 目前所能達成的上限, 並且建議 NAT 廠商們如何改善 NAT 行為來增加目前軟體所及的 DCR 上限.

關鍵字: 穿透 NAT, ICE, 點對點, UDP, TCP

Analyzing and Comparing ICE-UDP and ICE-TDP for NAT Traversal

Student: Yu-Hsien Wu **Advisor: Dr. Ying-Dar Lin**

Institutes of Network Engineering

National Chiao Tung University

Abstract

The appearance of Network Address Translation (NAT) breaks the common peer-to-peer (P2P) communication model and causes difficulties for UDP/TCP P2P communications. For this reason, the NAT traversal problem is important. NAT has the distinct behavior to handle UDP and TCP connections. In addition, the UDP and TCP protocol have their own characteristics. For example, UDP is connectionless while TCP is connection-oriented. Therefore, various solutions on UDP and on TCP are proposed, respectively. Several traversal techniques such as STUN, STUNT, P2PNAT, and TURN are proposed and individually utilized but they cannot traverse all NATs because of various NAT behavior. Interactive Connectivity Establishment (ICE) provides a mechanism to integrate these techniques to traverse a NAT via connectivity check which tests and verifies all traversable paths. Here we design an experiment to measure the direct connection ratio (DCR) and analyze the main differences between ICE-UDP and ICE-TCP such as the implementation of socket layer and the procedure of connectivity check and NAT traversal ability. Finally, we find out the DCR of ICE-UDP and ICE-TCP has reached the upper-bound, and propose that the NAT venders should reform NAT to increase the room of DCR upper-bound for applications

Keywords: NAT traversal, ICE, peer-to-peer, UDP, TCP

Contents

List of Figures

List of Tables

1. Introduction

Recent years Internet tremendously growths and spreads, in position, it gains the complexity and evolves in ways that make life difficult for many applications. Now the Internet's uniform address architecture bases on IPv4, hosts use a fixed 32-bit globally unique address to identify and contact directly with each other. However, limited IP addresses could not satisfy a large number devices on Internet nowadays. To solve the IP shortage problem, Network Address Translation (NAT) [1] technique appears. It is a process that translates packet IP address and transits packets between private and public domains, and allocates a temporary public address for outgoing UDP/TCP connections, enable multiple hosts on a private network to access the Internet using a single public IP address. Because the translation of addresses breaks the end-to-end connectivity model of the IP, newly developed services follow the peer-to-peer (P2P) paradigm such as file sharing, instant messaging, and voice over IP (VoIP) applications suffer from the existence of NAT. Thus, NAT traversal is an important problem today.

NAT traversal problem makes adverse effects on many hosts work behind NATs. There are two directions of possible approaches to the problem. One direction is to cope with existing NAT implementations and establish standards for the detection of the NAT behavior for NAT traversal. On the other hand, the IETF also standardizes behavioral properties for NATs to work in conjunction with IETF protocols. However, making a change on the large scale deployment of residential NATs is huge work. Therefore, we focus on the discussion about the NAT traversal techniques for NAT behavior in this article.

For application programmers to develop application to resist the NAT traversal

problem, it is very important to understand NAT behavior in order to design applications that can work in combination with current NATs. We can identify classifications of NAT behavior derived from simple traversal of User Datagram Protocol (UDP) through NAT(STUN)[2].The NAT behaviors include the mapping, binding , filtering behavior which are defined in RFC 4787[3], and TCP state tracking which is defined in RFC 5389[4]. Mapping, binding, and filtering react on both UDP and TCP connections, and TCP state tracking only influences on TCP.

ICE: Optimal NAT Traversal Solution

Aiming at the above-mentioned NAT behavior, several behavior-based traversal methodologies are proposed to establish UDP [2, 5, 6, 7] and TCP [6, 8, 9, 10, 11] connections; among these, traversal techniques [5, 8] are proposed recently so that NAT traversal problem is still a current issue today. However, not all NATs in the wild react the same way. This will cause these approaches to fail in various cases. Among these traversal techniques, Interactive Connectivity Establishment (ICE) is the integrated and flexible UDP and TCP solution for both, because it makes use of many of the techniques, but uses them in a specific methodology which avoids many of the pitfalls of using any one alone.

ICE provides a mechanism that learns possible connectivity when it performs traversing process on UDP and TCP, for applications applied on UDP and TCP protocol could use ICE-UDP first, and if failed, try ICE-TCP. Therefore, it could raise the chance to traverse the NAT. The big difference between ICE-UDP and ICE-TCP is the connectivity checks of peer-to-peer, since ICE-UDP and ICE-TCP use different traversal techniques based on UDP and TCP, respectively.

Distinct Traversal Process on ICE-UDP and ICE-TCP

ICE performs distinct processes on connectivity checks based on different

protocols, and therefore, the implementation has differences that handle socket layer and ICE transport session. In order to understand details and analyze what factors differentiate the Direct Connect Ratio (DCR) of ICE on UDP and TCP, overall this paper makes three works. First, we implement the ICE library to use ICE-UDP and ICE-TCP separately and compare the characteristics of them. Second, we design an experiment, in order to analyze the NAT behavior emerges on UDP and TCP connections, could speculate the upper-bound of DCR of ICE-UDP and ICE-TCP according to behavioral test results, and measure the DCR of ICE in our experiment environment. Third, based on the measurement results, we observe the other factors affecting the DCR of ICE approach, and suggest the modification to the approach

The rest of this paper is organized as follows. Section 2 introduces NAT traversal techniques, comparisons of advantages and disadvantages, and the overview of ICE. Section 3 details the ICE implementation and discusses the differences of ICE on UDP and TCP, and Section 4 introduces the designed experiments and presents the measurement results. Section 5 summarizes and concludes this paper.

2. Background

Since NAT traversal problems are caused by NAT behaviors, in this chapter, we not only describe how NAT behaviors operate: Mapping and Filtering, and their definitions but also classify the NATs by measuring and analyzing the traversal ability of ICE in our experiments.

NAT behavior

NAT Mapping

A NAT assigns an external mapping for each UDP and TCP connection based on the source and destination IP and port number. The NAT variation are defined in STUN [2], *Cone behavior NAT* reuses the mapping which means an address assigned on NAT , as long as the source address and port number of a new connection matches those of the preceding connection while *Symmetric behavior NAT* assigns the different mapping if the destination of new connection change.

Endpoint Filtering

NAT may filter the inbound packets according to the rule, which is created when an internal endpoint opens an outgoing session through NAT. In this section, we detail the criteria of the filtering rule.

In RFC 4787 [3], authors describe three various filtering behaviors: Endpoint-Independent, Address-Dependent, and Address and Port-Dependent. The classifications of filtering behaviors are the same on UDP and TCP. If a NAT allows inbound UDP/TCP connections independent of the source address and port number as

long as the necessary state exists for routing the request, we classify such filtering behavior as Endpoint-Independent. If a NAT filters inbound packets whose source matches the destination address of the connection creating this mapping, the filtering behavior of such NAT is classified into Address-Dependent. Similarly, if both address and port number are required, this filtering behavior is called Address and Port-Dependent.

TCP State Tracking

NATs implement a state machine to track the connection-state of a TCP stream. Although NATs handle the TCP 3-way handshake to establish a connection, not every NAT handles all possible cases of packet sequences. This leads the NAT to close its mapping prematurely and breaks a TCP connection.

Accordingly, we understand that distinct NAT behavior will break peer-to-peer connections and make NAT hard to traverse, and we induce that both Mapping and Filtering behaviors restrict seriously to TCP connections, causing the TCP NAT traversal problem more complicated.

Various NAT combinations

So far endpoints are located behind the four types of NAT, when two endpoints under four NAT types has 4 x 4 = 16 various combinations, we summarize the 16 combinations into 4 combinations in Table 1 and discus which the scenario under combinations could traverse successful in Figure 1.

We estimate which kind of NAT combinations could be traversed as Table 1 shows if endpoints are behind the combinations and attempt to traverse NAT. The white fields of Table 3 mean the combinations that could be traversed successfully. If two peers are behind Cone NAT (Table 1. Combination 1), peer could traverse NAT

because the peers always send packets to the fixed and correct mapping; if one peer under Full Cone or Address Restricted NAT and another under Symmetric NAT (Table 1. Combination 2), the case SHOULD be success because the Filtering behavior of Full Cone and Address Restricted NAT is loose, so the packet 2 in Figure 1(2), which source port has changed and sent out from Symmetric NAT, could traverse them; if one peer is under Port Restricted and another is under Symmetric NAT (Table 1. Combination 3), Port Restricted NAT will filter inbound packet 2 in Figure 1(3) from Symmetric NAT since the source port of packets from Symmetric NAT will be changed, and the Symmetric NAT also filters inbound packets 1 from Port Restricted NAT since the packets reach the previous mapping created by STUN, this combination could NOT be passed; if two peers are behind Symmetric NATs (Table 1. Combination 4), both peers will send packets 1 and 2 in Figure 1(4) to the other side NAT mapping, which is get by traversal technique, but not a correct mapping created by traversal connection, so the combination fails.

Table 1. Classifications of NAT combinations

Type / Type	Full Cone	Address Restricted	Port Restricted	Symmetric
Full Cone	1			2
Address Restricted				
Port Restricted				3
Symmetric	2		3	4

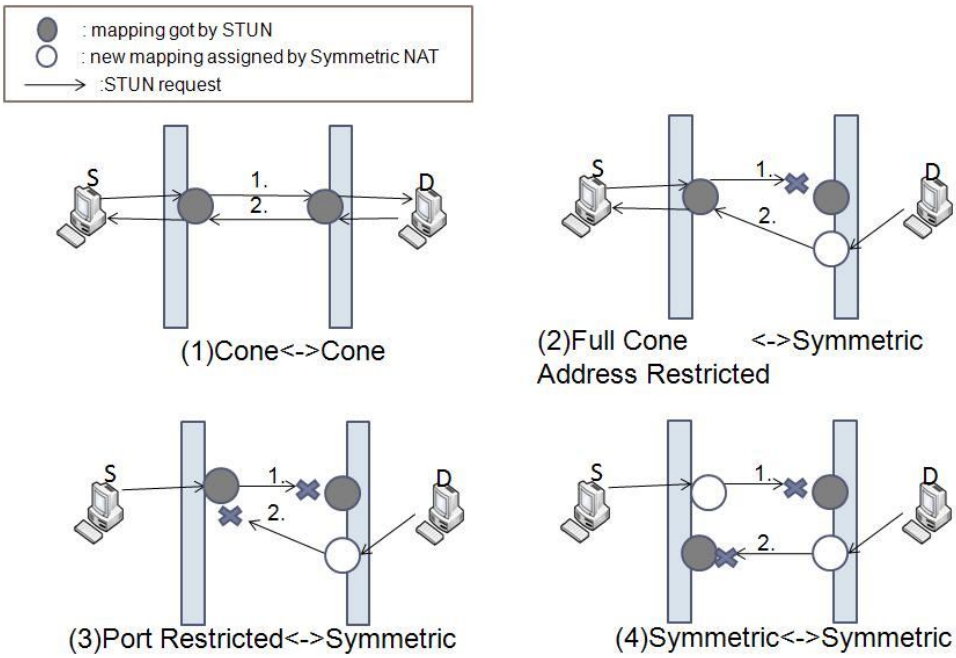


Figure 1. NAT traversal of combinations

NAT Traversal Techniques

Several NAT behavior-based solutions aim at NAT behavior and are proposed to solve NAT traversal problem. The following lists few solutions required for peer-to-peer connections on UDP and TCP in common use.

UDP NAT traversal

STUN

STUN (Figure 2(a)) is the most commonly used technique to solve UDP NAT traversal problems. It lets NAT applications search for external NAT types [2] and test the public address and port number that NAT allocates before giving the information to the host for automatic connection. STUN evolved from RFC 3489(Simple Traversal of User Datagram Protocol) to RFC 5389(Session Traversal Utilities for NAT), it was redefined a tool that is utilized as part of NAT traversal techniques, and support TCP now.

TCP NAT traversal

STUNT

The approach is proposed in [10], the endpoint sends out a low-TTL SYN packet, then sender aborts the connection attempt and creates a passive TCP socket on the same address and port number. The other then initiates a TCP connection, as illustrated in Figure 2(b). The endpoint needs to select an appropriate TTL value that is large enough to cross NAT and the NAT must not close mapping if it receives ICMP error. It also requires that the NAT accepts an inbound SYN following an outbound SYN – a sequence of packets not normally seen.

P2PNAT

In [6], the authors take advantage of the simultaneous open defined in the TCP specifications [13]. Both endpoints initiate a connection by sending SYN packets. If the SYN packets cross in the network, both the endpoint stacks respond with SYNACK packets establishing the connection. If one end's SYN arrives at the other end's NAT and is dropped before that end's SYN leaves that NAT, the first endpoint's stack ends up following TCP simultaneous open while the other stack follows a regular open, Figure 2(c). In the latter case, the packets on the wire look like the STUNT approach without the low-TTL and associated ICMP.

The STUNT approach requires that the NAT accept an inbound SYN after an outbound SYN. In addition, the approach requires the endpoint to retry failed connection attempts in a tight loop until a timeout occurs. If instead of dropping the SYN packet, a NAT responds to it with a TCP RST then this approach devolves into a packet flood until the timeout expires.

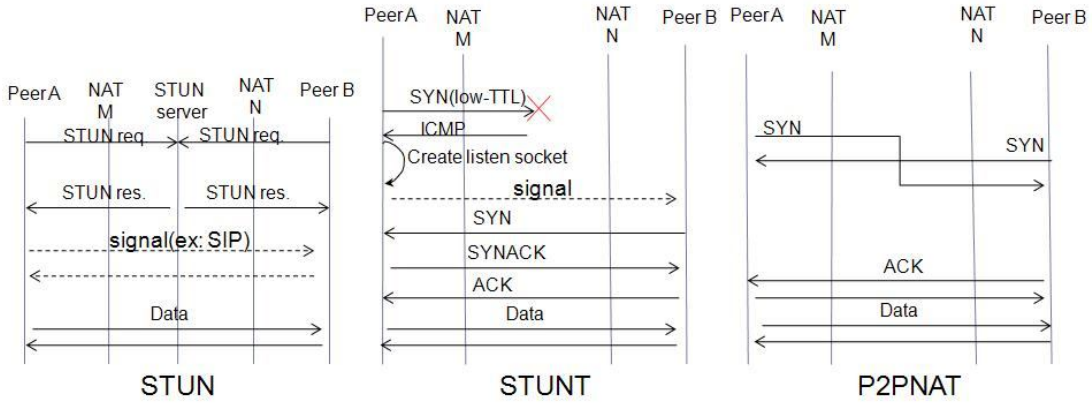


Figure 2. NAT traversal techniques for peer-to-peer connectivity, (a)STUN, (b) STUNT, (c) P2PNAT.

ICE-UDP and ICE-TCP

Since several techniques are proposed, all advantage and disadvantage which make each one optimal in some network topologies, but a poor choice in others. Therefore, we need a single solution which is flexible to work well in all situations.

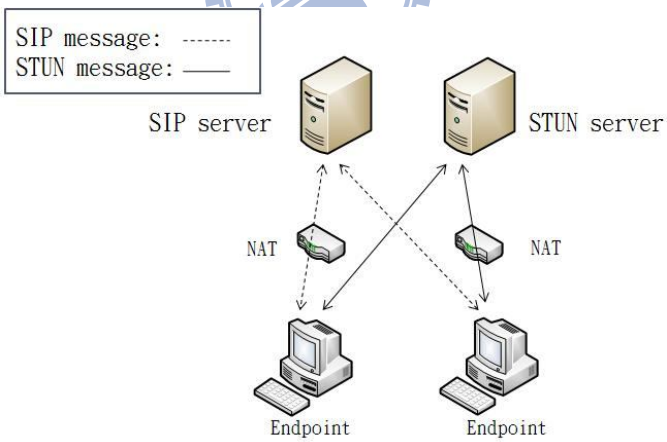


Figure 3. ICE deployment scenario

ICE is a technique for NAT traversal for UDP-based and TCP-based streams established by the offer/answer mechanism used by protocols such as the Session Initiation Protocol (SIP), combines STUN, STUNT, and P2PNAT techniques for peer-to-peer connectivity, and works by multiplicity of IP addresses and ports derived from STUN in SDP offers and answers, then tests connectivity checks and nominates

the valid connections. Figure 3.show the ICE deployment. In the following we introduce the above ICE procedure in three main parts.

ICE-UDP vs. ICE-TCP

Gathering Candidates

For executing ICE, endpoint needs to identify all of its address candidates. CANDIDATE is an address combines IP and port. ICE-UDP draft [5] defines few types of candidates derived via STUN [2] for peer-to-peer connection. In ICE-UDP, it defines three candidates for peer-to-peer connectivity: address obtained directly from a local interface (LOCAL CANDIDATE), translated address on the public side of a NAT (SERVER REFLEXIVE CANDIDATE) in Figure 4, besides, sometime discover a new mapping address assigned by Symmetric NAT (PEER REFLEXIVE CANDIDATE). In ICE-TCP defines address for STUNT unidirectional open (ACTIVE CANDIDATE, PASSIVE CANDIDATE), and address for TCP simultaneous open (SIMULTANEOUS CANDIDATE) in Figure 4.

Connectivity Checks

In [5], after local endpoint gathers all candidates, it sends them to remote endpoint over the signaling channel. The candidates are carried in attributes in the SDP offer. When remote peer receives the offer, it performs the same gathering process and responds with its own list of candidates. At the end of this process, each agent has a complete list of both its candidates and remote peer's candidates. It pairs local and remote candidates up, resulting in CANDIDATE PAIRS, which are possible path for establishing connection. To test which pairs could work, ICE schedules a series of CHECKS. Each check is a STUN request/response transaction that the peer will perform on every candidate pair by sending a STUN request from the local

candidate to the remote candidate on ICE-UDP or ICE-TCP.

The obvious difference of ICE-UDP and ICE-TCP are different connectivity checks. In ICE-UDP, the peer-to-peer check list on endpoint will be “local candidate to server reflexive candidate”, ICE checks the path to remote endpoint behind NAT, and “local candidate to local candidate”, ICE checks weather both endpoints behind identical NATs; in ICE-TCP check list includes “local candidate to local candidate”, “local candidate to simultaneous candidate” and “local candidate to passive candidate”, which are similar with “local candidate to server reflexive candidate”, “active candidate to passive candidate” and “passive candidate to active candidate” are paired to avoid the un-sequence packets causing TCP state filtering. Figure 4, illustrates the different connectivity checks of ICE-UDP and ICE-TCP.

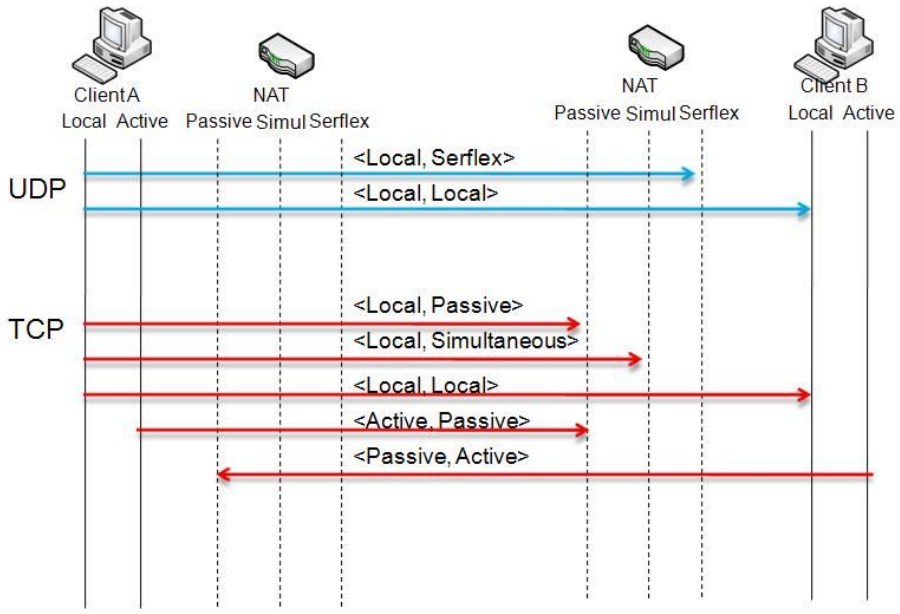


Figure 4. Each candidate location and ICE connectivity check pair in check list

Concluding ICE

ICE assigns one endpoint to be the role CONTROLLING, and another be CONTROLLED. The controlling role is responsible of nominating which candidate pairs will get used for media amongst the ones that are valid. It can do this in one of

two ways – using REGULAR NOMINATION or AGGRESSIVE NOMINATION, the former selects a path after all the checks finish, the later selects one immediately when the check is succeed. Once the nomination completes, the nominated valid pair could provide a path for applications to create peer-to-peer connectivity.

3. ICE Library Implementation for SIP Calls

Architecture

For analyzing the ICE-UDP and ICE-TCP, first thing is that implement ICE on applications to test. ICE cooperates with signaling protocol to exchange necessary transport candidates, we implement ICE-UDP/TCP on VoIP SIP application to carry out ICE process and communicate by SIP protocol to exchange candidate data, the application follows the architecture in Figure 5, includes ICE Library and utilize SIP module to transport ICE candidates.

ICE process start with STUN session and perform UDP/TCP check sessions with remote candidates, the implementation considerations of UDP and TCP are different since ICE-UDP and ICE-TCP are based on UDP and TCP socket layer. ICE-UDP opens one socket which is bound with a local IP and port, gets the candidates and check paths, but the ICE-TCP handles few sockets for different connections if it needs listen and initiate connections to remote peers. The draft [8] notes that the BSD TCP sockets of ICE use socket option SO_REUSEADDR to reuse the local IP port since ICE demands few sockets for different TCP sessions.

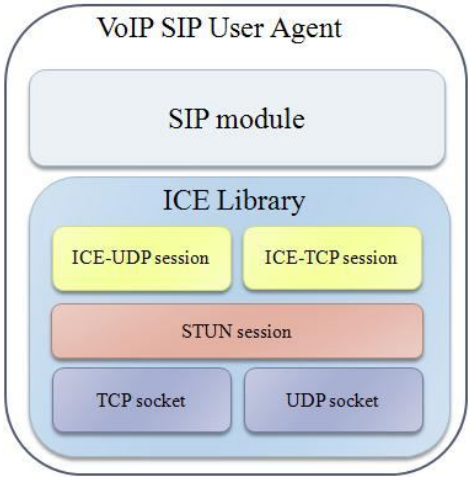


Figure 5. VoIP SIP application with ICE architecture

ICE Library Modules

PJNATH (PJSIP NAT Helper), followed draft-ietf-mmusic-ice-19 [5], is an open source library providing ICE-UDP NAT traversal functionalities. For testing ICE-UDP and ICE-TCP NAT traversal utility, we wrap PJNATH up and modify it to support ICE-TCP [8], provide a ICE Library implementation for SIP calls, the following lists the ICE library and describes the action in Table 1.

Here illustrates the functions related to ICE process in Figure 4, ICE_Init() function needs to input configuration to set the STUN server for discovering candidates, initialize the required data structure and memory allocation. ICE_Getcandidates() perform STUN process to send STUN request to server and collect candidates from STUN response; ICE_StartCheck() needs input the remote peer candidates that are exchanged by SIP, then paring the local candidates and remote candidates according pair list mentioned in chapter 2, and start to check each pair by sending STUN request periodically. For synchronizing the check results between caller and callee, ICE performs a 4-way handshake (caller and callee send and receive STUN request/response for the same pair) to sure that result is in chorus. After all the checks finished, caller sends a STUN request with a flag to callee for nominating pair.

Table 2.ICE Library implementation for SIP calls.

Function	**Description**
void ICE_Init(STUNConfig*)	Allocate memory; initialize the STUN configuration and data structure.
Candidates* ICE_GetCandiates(void)	Collect the candidates for ICE-UDP/TCP
CandidatePair* ICE_StartConnectCheck(Candidates*)	Start ICE connectivity check of ICE-UDP/ICE-TCP and return nominated path

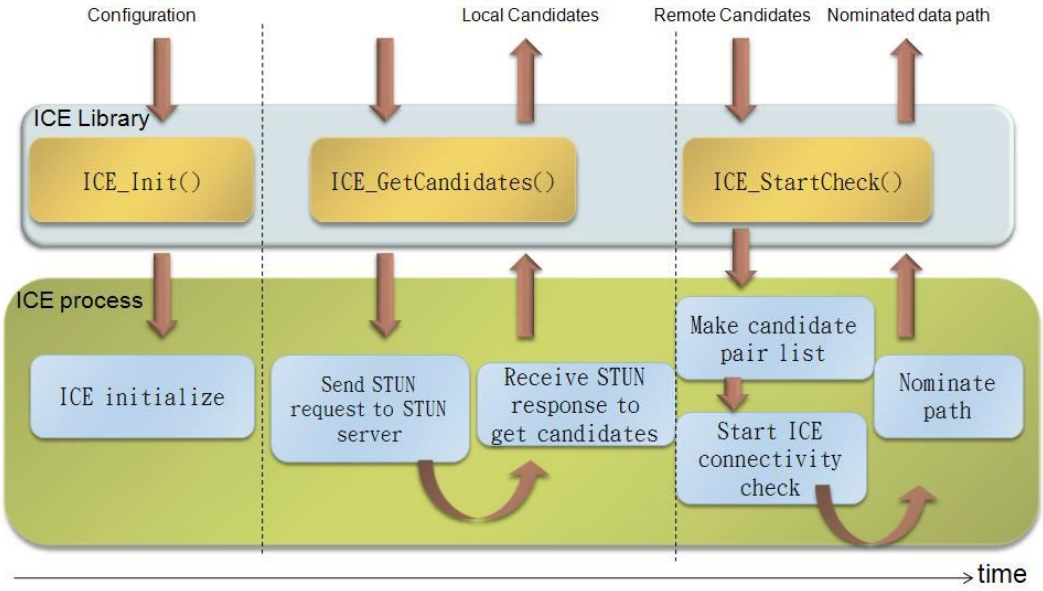


Figure 6. The relations between ICE API and process in time sequence (left to right)

Flow of VoIP SIP application with ICE-UDP and ICE-TCP

This section details the execution sequence of application layer shown in Figure 7.When VoIP SIP application set up, application calls ICE_Init() to initialize the ICE configuration. As caller (controlling) ready to make a call to callee, caller executes the ICE_GetCandidates() to collect candidates, encode candidates to SDP embedded in SIP message and send out SIP INVITE message to SIP server forwarding to callee (controlled). After callee receives and decodes candidates from SDP, callee also executes the ICE_GetCandidate() to get the candidates of itself. After that, callee collected local candidates and remote candidates obtained from caller, encode local candidates to SDP in 200 O.K SIP message and response to caller, finally callee executes ICE_StartConnectCheck() to start period check, and caller do the same action after it receives the SIP response from callee. After connectivity all checks finish, caller sends a STUN request to calle to nominate which candidates pairs will get used, at the end, both caller and callee both they can send (and receive) date

packets peer-to-peer bidirectional through nominated path.

The ICE process in the figure includes ICE-UDP and ICE-TCP. The implementation wraps ICE-UDP and ICE-TCP so we could execute ICE-UDP and ICE-TCP process together. If some applications apply on UDP and TCP protocol, we recommend that the ICE-UDP process should fist since the UDP NAT traversal is easier.

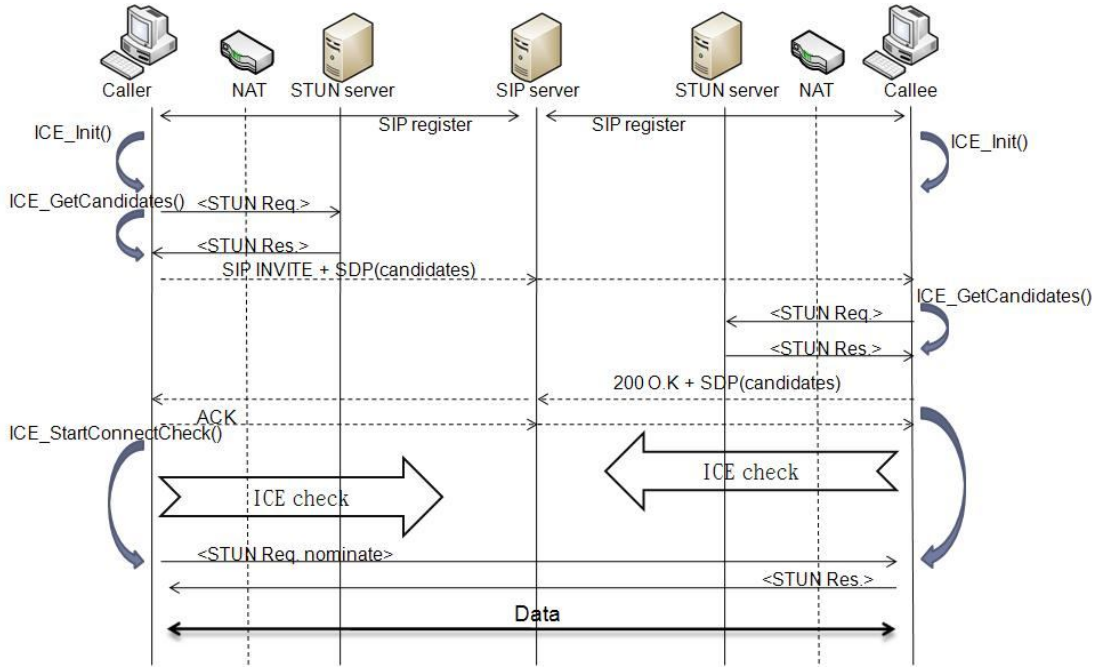


Figure 7. VoIP SIP application applied ICE library to perform NAT traversal.

4. Experiments and Results

Experiments Design

Here we design experiments to analyze the DCR of ICE technique under NATs and observe the characteristics of ICE-UDP/TCP. Because traversal techniques suffer from NAT behavior, we analyze the UDP/TCP behaviors of 15 NATs and classify them into four types according to previous work [2]. For each experiment, the results of different combination of NAT types are different. We infer the upper-bound of DCR under various combinations of NAT types, observe the DCR of ICE in a real NAT environment, and understand whether the DCR of ICE can reach the upper-bound or not.

In our experiment environments as Figure 8 shows, we set two peers, caller and callee, and execute VoIP application for testing ICE DCR under 15 NATs. Also, we set STUN server to get NAT mapping and SIP server to exchange necessary candidates under ICE process.

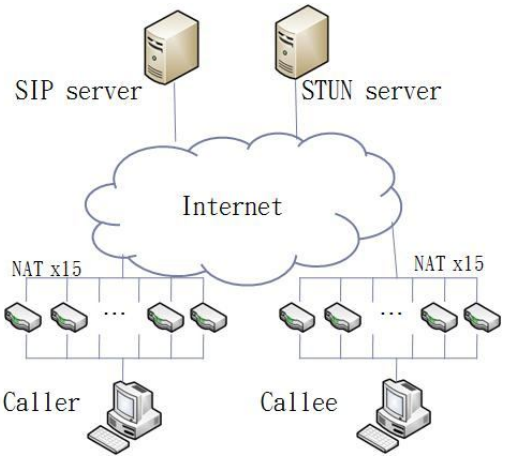


Figure 8. ICE DCR test environment.

In the experiment, we test application under various NAT combinations to

establish peer-to-peer connectivity. There are 210 cases (15*15-15) for ICE-UDP and ICE-TCP respectively. The experiment starts by making a call from caller to callee to establish connection, and performs ICE process to traverse NAT. If both caller and callee establish bidirectional data transmission, the result of test case is success. Otherwise the result is failure. The following is the test case procedure.

Step 1: Caller and callee set up application configuration and execute.

Step 2: Caller and callee register with SIP server and caller make a call to callee.

Step 3: Caller and callee perform ICE-UDP first and then ICE-TCP process. If the traversal is successful, they start to transmit data.

Step 4: Application will start or fail to transmit data.

NAT Classification and Analysis

Before measuring the ICE-UDP/TCP DCR, we have to understand the NAT behavior and further classify NAT types and realize which combination of types ICE could traverse so that we can figure out the DCR upper-bound in advance. Each peer runs the test program NATCheck [6] and Stunt [12] to test the UDP/TCP behavior of NAT and classify NATs. Table 3 shows the results, and we could reference the Table 1 and Figure 1 to estimate the upper-bound of DCR under UDP and TCP in experiment environment.

Table 3. Test results of NAT type

NAT Brand	Model	UDP NAT type	TCP NAT type
NAT 1. Asus	3crwer 100-75	Symmetric	Symmetric
NAT 2. Belkin	F5d8231tw4	Port Restricted	Port Restricted
NAT 3. Corega	Cg-barmx2	Address Restricted	Port Restricted
NAT 4. Draytek	Vigor 2104p	Port Restricted	Port Restricted
NAT 5. D-Link	Di-604	Full Cone	Full Cone
NAT 6. Edimax	Br-6204wg	Port Restricted	Port Restricted
NAT 7. Linksys	Befsr41	Port Restricted	Symmetric
NAT 8. Lemel	Lm-wlg6400	Port Restricted	Port Restricted
NAT 9. Netgear	Pr614	Symmetric	Symmetric
NAT 10. PCI	Blw-54mr	Address Restricted	Port Restricted
NAT 11. SMC	Smcwbr14-g2	Full Cone	Port Restricted
NAT 12. Zyxel	P334	Symmetric	Symmetric
NAT 13. FreeBSD	Pf	Symmetric	Port Restricted
NAT 14. Linux	iptables	Port Restricted	Port Restricted
NAT 15. SMC	WGBR14-N	Address Restricted	Port Restricted

Testing Results

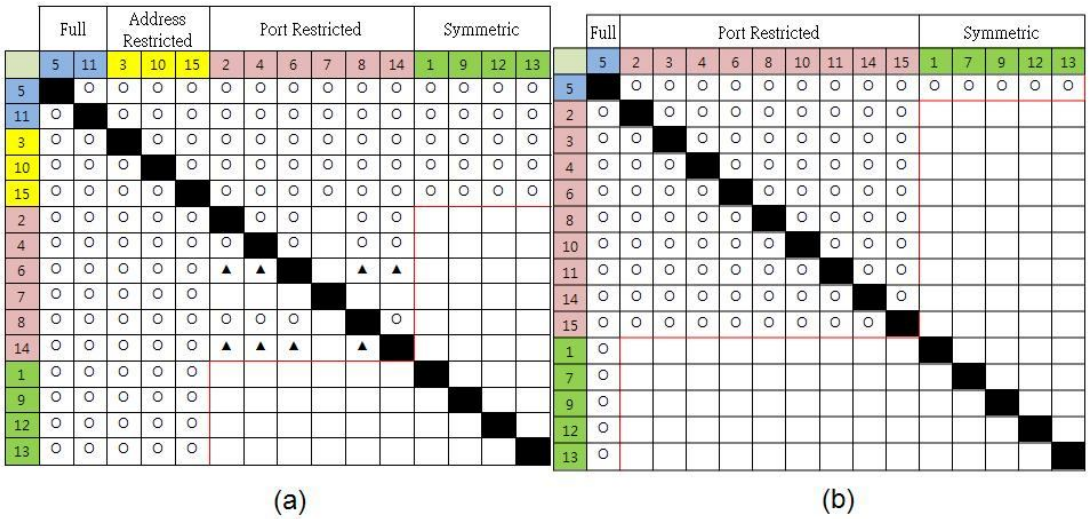
DCR of ICE-UDP and ICE-TCP

Table 4 shows the results and Table 5 summarized them. Table 5 displays the DCR in above four combinations and normalized DCR which means the DCR of entire cases of experiment. ICE-UDP and ICE-TCP techniques almost solve all cases SHOULD pass in combination 1 and 2 so that their DCR approach to the DCR upper-bound, this proves the conjecture of upper-bound is correct. The DCR upper-bound of ICE-UDP is 71.4% which is higher than ICE-TCP because handling TCP connection is complicated. We consider that NAT takes care of reliable and secure TCP connection so there are more Port Restricted NATs and Symmetric NATs for traversing TCP connections, increases the portion of combination 3 and combination 4 and decrease the DCR upper-bound. Because the Combination 3 and Combination 4 are the bottleneck of DCR upper-bound, port predict [13] traversal technique which guess the binding of Symmetric NAT, could be utilized in ICE and

raise the ICE-UDP and ICE-TCP DCR when the port mapping assignment of Symmetric NAT is sequential.

According the ICE-UDP test results, we observe a situation that caller behind NAT 6 and NAT 14, the UDP connection fail to traverse NAT, but it could pass when the role of caller and callee are changed. This situation is called Conntrack Bind problem [17] described in section 4.3.2.

Table 4. Total test cases of (a) ICE-UDP and (b) ICE-TCP, “O” means success case, “▲” means success if we fix Conntrack Bind problem, space means fail case.



(a)

	Full		Address Restricted			Port Restricted						Symmetric			
	5	11	3	10	15	2	4	6	7	8	14	1	9	12	13
5		O	O	O	O	O	O	O	O	O	O	O	O	O	O
11	O		O	O	O	O	O	O	O	O	O	O	O	O	O
3	O	O		O	O	O	O	O	O	O	O	O	O	O	O
10	O	O	O		O	O	O	O	O	O	O	O	O	O	O
15	O	O	O	O		O	O	O	O	O	O	O	O	O	O
2	O	O	O	O	O		O	O		O	O				
4	O	O	O	O	O	O		O		O	O				
6	O	O	O	O	O	▲	▲			▲	▲				
7	O	O	O	O	O										
8	O	O	O	O	O	O	O	O			O				
14	O	O	O	O	O	▲	▲	▲		▲					
1	O	O	O	O	O										
9	O	O	O	O	O										
12	O	O	O	O	O										
13	O	O	O	O	O										

(b)

	Full	Port Restricted									Symmetric				
	5	2	3	4	6	8	10	11	14	15	1	7	9	12	13
5		O	O	O	O	O	O	O	O	O	O	O	O	O	O
2	O		O	O	O	O	O	O	O	O					
3	O	O		O	O	O	O	O	O	O					
4	O	O	O		O	O	O	O	O	O					
6	O	O	O	O		O	O	O	O	O					
8	O	O	O	O	O		O	O	O	O					
10	O	O	O	O	O	O		O	O	O					
11	O	O	O	O	O	O	O		O	O					
14	O	O	O	O	O	O	O	O		O					
15	O	O	O	O	O	O	O	O	O						
1	O														
7	O														
9	O														
12	O														
13	O														

Table 5. (a) ICE-UDP and (b) ICE-TCP DCR testing analysis

Combinations	DCR	Percentage in total cases	Normalized DCR
1. Cone ⇔ Cone	85.45%	52.4%	44.76%
2. Full Cone and Address Restricted ⇔Symmetric	100%	19%	19%
3. Port Restricted ⇔ Symmetric	0%	22.9%	0%
4. Symmetric ⇔ Symmetric	0%	5.7%	0%
Total	N/A	100%	63.76%

(a)

Combinations	DCR	Percentage in total cases	Normalized DCR
1. Cone ⇔ Cone	100%	42.85%	42.85%
2. Full Cone and Address Restricted ⇔Symmetric	100%	4.8%	4.8%
3. Port Restricted ⇔ Symmetric	0%	42.85%	0%
4. Symmetric ⇔ Symmetric	0%	9.5%	0%
Total	N/A	100%	47.65%

(b)

Conntrack Bind Problem

From Table 5(a), we observe that caller under NAT 6 and 14 (Port Restricted NAT) could not traverse Port Restricted NATs, but this case should pass under the NAT combination because they are Corn NATs. This problem is proposed in [17] when one of peer is under Linux-based NAT (Port Restricted NAT); Figure 9 illustrate the packet flows when the problem occurs. When an unsolicited packet attempts to traverse a Linux-based NAT, the NAT will block the old mapping (a) created by STUN session and assign a new mapping (b) in Figure 9 for new outbound packets, such mapping behavior is similar to Symmetric NAT. In Figure 7, we know callee always start connectivity check earlier than caller. After a Linux-based NAT creates the mapping a by STUN server for the caller, it drops an unsolicited STUN check packets from callee. That's why the Conntrack Bind problem occurs.

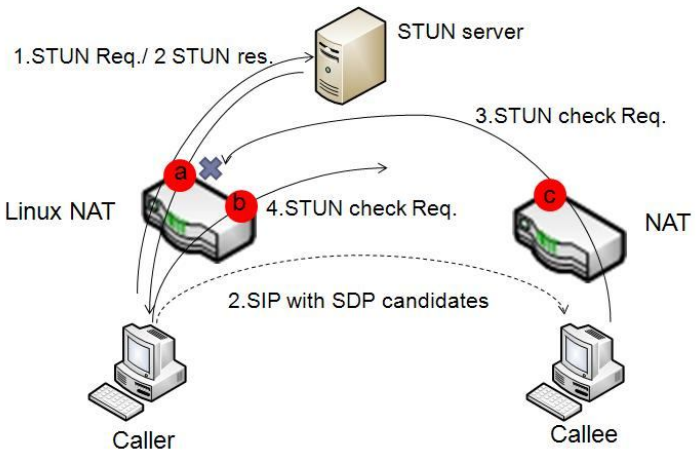


Figure 9. Conntrack Bind problem occurs when peers under Linux-based NAT.

Here we propose a solution: *Forestalling Low-TTL,* solve the conntrack bind problem even if caller and callee under Linux-based NATs. In our solution, we consider that caller should send out a Low-TTL packet applying NAT to create a correct mapping before other side check packets arrive.

We demand the two sets of candidate for twice ICE process. If the fist ICE process fails, we presume that the Conntrack Bind problem occurs and then trigger the ICE process of second candidate set. We collect the extract (second) candidates and encode into SDP with original (first) candidates, after callee receives the SIP message with SDP, callee also collects the extract candidate and original candidates, and pairs two lists for checking. Before callee responses the SDP with candidates to caller, it send out the Low-TTL packet to caller's second server reflexive candidate binding b on NAT in Figure 9, to create a mapping on NAT, and send SIP message to caller and starts the original ICE process. The caller receives callee's candidates from SIP message and immediately sends a Low-TTL packet to callee's second server reflexive candidate b', at this time, two sides NAT have created mapping by outgoing Low-TTL packets. If the fist ICE process fails, we will start the extra ICE process; begin the extra ICE connectivity check after we have sent Low-TTL packet to created correct mapping on NAT. Therefore, even both peers are under Linux-based NATs, Conntrack

Bind problem cases could be solved in Table 4(a), and we improve the ICE-UDP DCR from 63.76% to 67%.

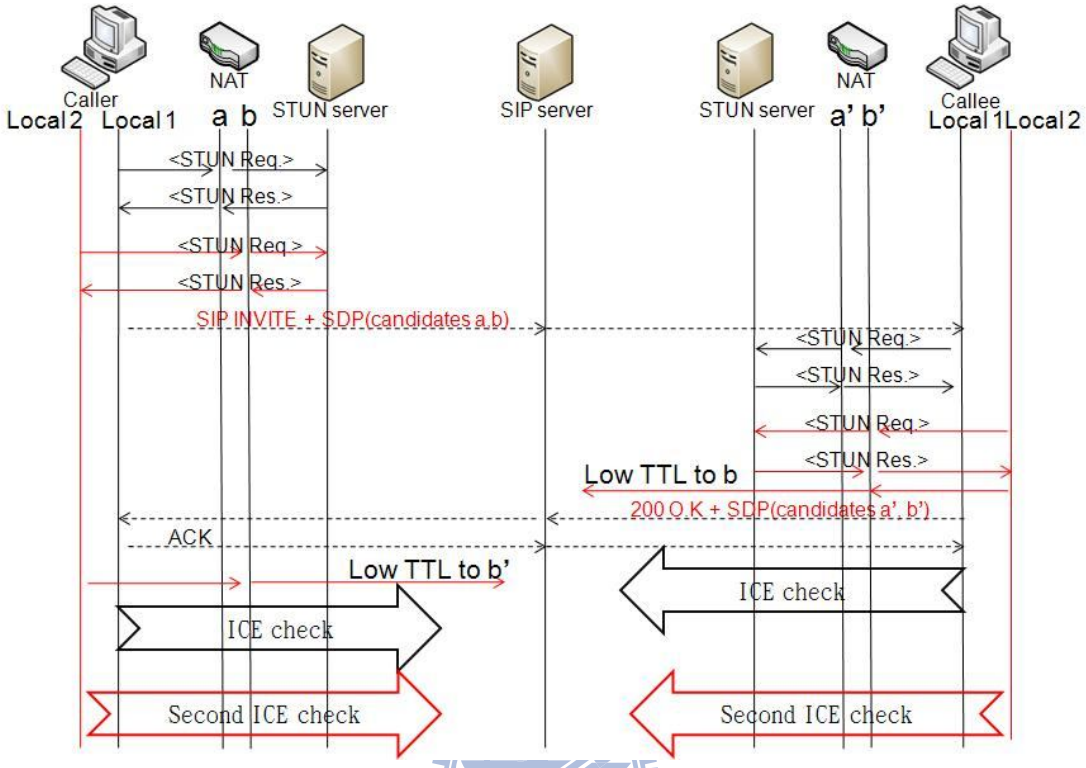


Figure 10. Solution: Forestalling Low-TTL for Conntrack Bind problem.

Although we have solved the Conntrack Bind problem, but the ICE-UDP DCR combination 1 (Table 3.) does not reach 100%. We consider this caused of particular NAT behavior since the DCR test result of NAT7 are similar Symmetric NAT, but NAT 7 doest not match the NAT behavior testing result according to Table 3, this may be a point for a further research.

The results show that the media protocol that runs over UDP and TCP, for example: RTP protocol, it prefer to use UDP protocol first on P2P application for NAT traversal cause of the higher DCR and upper-bound. If endpoint could not traverse NAT on UDP protocol, it attempted to change into TCP protocol to TCP NAT traversal, solve some special cases on UDP protocol such as Conntrack Bind problem.

Bottleneck of the upper-bound of NAT TCP DCR

In Table 2, the types of NAT TCP behavior are almost Port Restricted, this is an issue of implementation. When NAT handles TCP outbound connections, it binds a socket to establish connection to external endpoint, this TCP socket does not accept other SYN packets since TCP are unidirectional establishment, causing filtering behavior are strict. The NAT type of D-Link NAT on TCP is Full Cone NAT, we consider it handles outbound connections and accept inbound connections at the same time by open few sockets, but it has the resource issue that we do not discus here.

5. Conclusions and Future Works

Main Differences on Connectivity Checks and Implementation

In the thesis, we implement the ICE Library, which is used by VoIP application to realize ICE-UDP/ICE-TCP process for UDP/TCP NAT traversal, and analyze and compare their differences. Also, we design experiments to observe the characteristics and measure the ICE-UDP and ICE-TCP NAT traversal abilities with various combinations of NAT types.

The implementation of ICE-UDP and ICE-TCP are different on ICE session layer and socket layer. On socket layer, the implementation of UDP is simpler than TCP because ICE-UDP use the same socket handles few UDP connections while ICE-TCP must handle few TCP sockets, reuse them to establish TCP connections to STUN server, check connectivity, and accept inbound connections. On ICE session layer, ICE-UDP has “local to server reflexive” path for NAT traversal expect “local to local” path, and ICE-TCP has TCP simultaneous and active – passive path to perform TCP NAT traversal.

ICE

According to the experiment results we summarize: (1) It is complicate for NAT to handle TCP connections so that it limits the TCP traversal upper-bound; besides, TCP connection should be by modifying the implementation. (2) So far ICE could traverse the NAT types of combination 1 and combination 2 and almost reach 100% of upper-bound of UDP and TCP. However, the Conntrack Bind problem may occur in the scenario of figure 8. Therefore, this thesis proposes the solution which is Forestalling Low-TTL to solve problems in combination 1 that ICE could not traverse.

As a result, it helps to raise the ICE-UDP DCR. (3) The utilization on media protocol run over on both of UDP and TCP recommends that always use UDP protocol to traverse NAT first and change to TCP protocol if it fail on UDP. (4) The TCP NAT behavior should be reformed by implementation to handling the inbound TCP connections, the TCP DCR upper-bound will be improved.

DCR Upper-bound Bottleneck on NAT implementation

So far the ICE traversal technique almost has reached the upper-bound on UDP and TCP. If we attempt to improve the DCR, combining the port predict technique in ICE could better DCR of combination 3 and combination 4. Furthermore, attacking upon the NAT infrastructure is a good way to enhance DCR. If the proportion of Full Cone and Address Restricted Cone NAT is higher, it could raise the upper-bound in combination 1 since Port Restricted NAT limits the DCR upper-bound when it meets Symmetric NATs. In addition, it needs NAT vendors to follow the RFC 4787 [3] and RFC 5382 [4] to reform the particular behavior like the behavior of NATs NAT 7 and Linux-based NAT.

References

[1] K. Egevang and P. Francis, “The IP Network Address Translator (NAT),” IETF RFC 1631, March 1994.
[2] 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.
[3] E. F. Audet and C. Jennings, “NAT Behavioral Requirements for Unicast UDP,” IETF RFC 4787, January 2007.
[4] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, “Session Traversal Utilities for (NAT) (STUN),” IETF RFC 5389, October 2008.
[5] J. Rosenberg, Interactive connectivity establishment (ICE), IETF Draft, October 2007.
[6] B. Ford, P. Srisuresh and D. Kegel, “Peer-to-Peer Communication Across Network Address Translators,” in USENIX Annual Technical Conference, Anaheim, CA, April 2005, pp.179-192.
[7] J. Rosenberg, C. Huitema, and R. Mahy, “Traversal using relay NAT (TURN),” IETF Draft, February 2009.
[8] J. Rosenberg, TCP Candidates with Interactive Connectivity Establishment, IETF Draft, July 2008.
[9] A. Biggadike, D. Ferullo, G. Wilson, and A. Perrig, “NATBLASTER: Establishing TCP connections between hosts behind NATs,” in ACM SIGCOMM Asia Workshop, Beijing, China, April 2005.
[10] 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.
[11] Y. Takeda, “Symmetric NAT Traveral using STUN,” IETF draft, June 2003.
[12] S. Guha, P. Francis, “Characterization and Measurement of TCPTraversal through NATs and Firewalls,” in ACM Internet Measurement Conference (IMC), Berkeley, CA, Oct 2005, pp. 199-211.
[13] J. Postel, “DoD standard Transmission Control Protocol,” RFC 761, January 1980.
[14] S. Guha, K. Biswas, B. Ford, S. Sivakumar, and P. Srisuresh, “NAT Behavioral Requirements for TCP,” IETF RFC5382, October 2008.

[15] H. Khlifi, Jean-Charles Gregoire and J. Phillips, “VoIP and NAT/Firewalls: Issues, Traversal Techniques, and a Real-World Solution,” IEEE Communication Magazine, July 2006, pp. 93-99.
[16] S. Perreault and J. Rosenberg, “Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations,” IETF Draft, March 2009.
[17] 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,” Technical Report, Dept. D-Link Joint NCTU Research Center, July 2009.
[18] P. Srisuresh, B. Ford and D. Kegel, “State of Peer-to-Peer (P2P) Communication across Network Address Translators,” IETF RFC 5128, March 2008.