國立交通大學

資訊科學與工程研究所

碩士論文

環境感知的NAT穿透機制

CAN – Context Aware NAT Traversal Scheme

研 究 生：劉俊延

指導教授：曾建超　教授

中華民國 九十八 年六月

環境感知的 NAT 穿透機制

CAN – Context Aware NAT Traversal Scheme

研 究 生：劉俊延　　Student：Jyun-Yan Liu

指導教授：曾建超　　Advisor：Chien-Chao Tseng

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis

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

June 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年六月

環境感知的 NAT 穿透機制

研究生： 劉俊延　　指導教授： 曾建超 教授

國立交通大學資訊學院資訊科學與工程研究所

摘　要

本論文針對互動式連線建立機制(ICE - Interactive Connectivity Establishment)，以多個連線網路位址交換組合成多個連線路徑的特性為基礎，提出一套利用 Network Address Translation (NAT) 環境的資訊來幫忙 ICE 進行連線路徑的測試。本機制在 NAT 環境資訊的收集不僅完善，且可以詳加利用 NAT 資訊的幫忙來幫助 ICE 選擇出需要測試的連線路徑藉此減少不必要的測試路徑以及連線測試的延遲時間，更可以利用特殊的測試連線方式來提昇直連的比例。

NAT 的存在對於點對點的應用程式，例如: VoIP、BT 等，是一個阻礙，而大家將這問題定義為 NAT 穿透問題。為了解決 NAT 穿透問題，便有很多方法被提了出來，而目前最為大家普遍公認的方法為「互動式連線建立機制」，但此套機制還是有其不足以及缺陷的地方。

ICE 的缺陷在於說對所有的連線路徑進行了一個系統化的測試，雖然全面，但是卻會增加不少連線的延遲時間，且 ICE 沒辦法對於 Linux-based 的 NAT 設備有特殊的應對機制。因為，Linux-based 的 NAT 設備本身就有 ConnTrack Binding 的問題存在，但是 ICE 對於這個問題完全沒有應對機制，導致原本可以直連的路徑會測試失敗轉而使用比較差的 Relay 路徑。

為了提出一套比 ICE 更好的 NAT Traversal 方法，我們以 ICE 為基礎加入了 NAT 資訊感知的功能，稱為 Context Aware NAT Traversal Scheme (CAN)。CAN 的中心思想為讓想要進行點對點連線的雙方，利用 out-of-band 的通訊協定交換 NAT 資訊，並且讓雙方利用 NAT 資訊來協助連線路徑的測試。CAN 不但可以有效改進 ICE 的連線延遲時間更可以一舉提高直連比率。

本論文對於所提出的方法，透過實做實測的方式來和原有的 ICE 進行了一連串的比較。而所有實做的工具和測試環境皆來自「友訊交大聯合研發中心」。在實際測試中發現，CAN 確實在延遲時間和直連率優於 ICE，而且也確實的解決的 ConnTrack Binding 這項問題。

總結以上，CAN 可良好的提供點對點應用程式一套簡易的穿透 NAT 建立連線的機制，低延遲時間、高直連率、易於瞭解連線失敗的原因並以此加以分析。CAN 更具有擴張性，以後可以跟其他特殊的 NAT Traversal 的機制作結合來提高直連率。

關鍵詞：NAT、NAT Traversal、ICE、UDP

CAN – Context Aware NAT Traversal Scheme

Student : Jyun-Yan 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 proposed a Context Aware Network Address Translation (NAT) Traversal scheme that can use NAT information to adjust its behavior when it tries to establish a connection across NATs. Many researchers have proposed techniques to tackle the NAT traversal problem. Among the previous proposals of NAT traversal techniques, Interactivity Connectivity Establishment (ICE) is the most acknowledged approach to establishing a connection across NATs. Although ICE is a very powerful connectivity establishment mechanism, it still possesses some drawbacks. First, ICE performs a systematical and exhaustive test procedure to find a connection from all possible paths between two peers; however, this procedure introduces a long delay or excessive message exchanges for setting up a connection. Second, ICE may fail in finding a direct connection

that exists between the two communicating peers.

In order to shorten the connectivity check delay, reduce the number of message exchanges, and increase the overall direction rates, we propose a Context Aware NAT (CAN) traversal scheme for finding a connection between two communicating peers behind NATs. The main idea of CAN is that user agents (UAs) exchange the NAT information, such as NAT types, Hairpin Capability and Connection Tracking and Binding feature, that can help UAs to eliminate unnecessary connectivity checks, shorten the delay of connectivity checks, and increase Direct Connection Rate (DCR). We have implemented CAN and compared the performance of CAN and ICE. The experimental results show that CAN outperforms ICE in terms of latencies and message exchanges for connectivity checks, and direct connection rate.

Furthermore, CAN is compatible with standard NAT traversal mechanisms, such as TURN and STUN, and can work with other mechanisms for increasing DCR.

Keywords : NAT, NAT Traversal, ICE, UDP

致　謝

從未想過自己真的可以獨立完成這篇碩士論文，這期間除了自己的努力之外還幸虧了很多人的幫助，我僅以這篇論文來感謝所有曾經在我碩士班求學路上幫助過我的人。

首先感謝指導教授增建超老師，要不是有他的辛苦教導不會有這篇論文的產生，更重要得是老師教導了我很多更重要的知識，學術的完整性、學術的道德、以及健康的求學知識等，很高興可以成為老師的學生。其次，要感謝辛苦的口試委員曹孝櫟教授、王讚彬教授、嚴力行教授，由於他們對本論文非常詳細之指正與建議，使得本篇論文內容能更加完善。

特別感謝 WINLAB 所有的學長們，RT 學長、GUNTER 學長、GARY 學長、家梁學長，感謝他們曾經給過得指導，WINLAB 的學弟妹和同儕們，由於你們的幫助讓我在被論文搞得要死不活的時候還可以有快樂的情緒。歡樂同心的 WINLAB 是我碩士兩年最好的歸處，不管以後大家會是在何方，我都會記住我們曾經在一起過得這些日子。

當然，我還要感謝我所有的家人們，因為有你們的支持我才可以順利完成我碩士的生活。感謝媽媽、大姐、兩位堂妹、堂弟、叔叔嬸嬸和爺爺奶奶們，我終於不負你們的期望完成了學業。

最後要感謝一下張懸、TIZZY BAC、YUI、和所有我曾經聽過你們音樂的歌手們，因為你們的音樂讓我在無數個孤獨奮鬥的夜晚都能有所期盼。

僅以此微薄的論文獻給以上所有的人，還有那一直在我身邊默默保護我的父親。

目　錄

圖目錄

表目錄

第一章 緒論

1.1. 研究動機

在現今的網路環境中 NAT(Network Address Translation) 已經是個相當普遍的技術，特別是很多家庭中所購買的無線網路寬頻路由器，其實都是具有 NAT 的功能，可以這樣說，如果家中裝設有無線網路的話，大概很大部份其實都是利用 NAT 來幫助上網。

NAT 定義於 RFC1631 [1], 基本上它是在 Router 中進行一個偷換 IP Header 的動作，以便讓多台電腦能共用一個 IP 連上網路的一項技術。最初，NAT 的提出只是為了解決 IPv4 位址不足的一項 Short-Term Solution, 不過沒想到 NAT 的方便性會讓它很快的在幾年內佔有相當大的位置。

通常以一個家庭的網路拓樸為例，如圖 Figure 1-1 所示，NAT 位於家中 private network 和外界的網際網路的中間幫忙進行 IP 的轉換，以便可以讓家中的多台電腦共用同一 Public IP 上網。如圖 Figure 1-1 所示，家中的 IP 網段大都為 Private Address Space 所描述的範圍 [2]。

由 Figure 1-1 可知道 NAT 就正好位於 Public Network 和 Private Network 之間幫忙進行 IP 的轉換，這時會有個問題就是通常由內部網路發起的封包要到達網路上的某一個端點是可以經由 NAT 的轉換並且順利路由到達目的地，但若是網路上的某一個端點想要傳送資料給位於 Private Network 內的機器很顯然是無法直接在目的地填上對方的 Private IP，因為 Public Network 是無法對於 Private IP 進行路由的動作。

針對如何讓位於 Private Network 外部的端點去直接和位於 Private Network 內部的端點直接建立連線便是一項很有趣且重要的問題。而這問題便被稱為 NAT Traversal。我們希望可以研究出一套有效率且方便的 NAT Traversal 技術，使得當任一端點想要和位於 Private Network 內部的端點進行連線是一項簡單的事情。

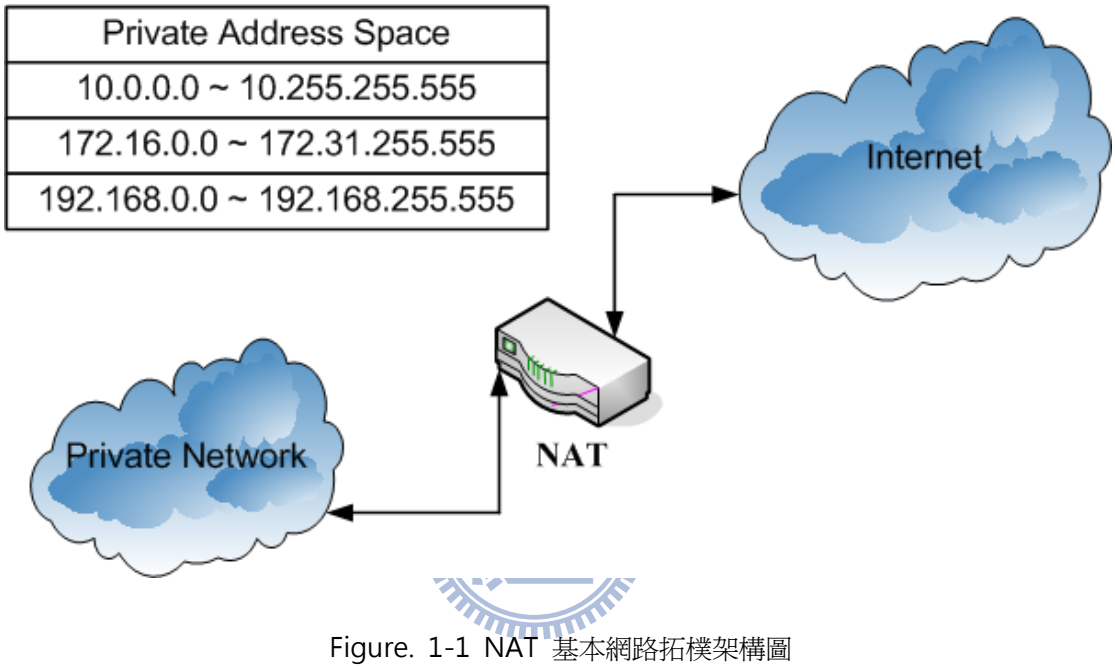

Figure. 1-1 NAT 基本網路拓樸架構圖

1.2. 研究目的

針對 NAT Traversal 會因為 Transport Layer 的不同而有所不同，所以 TCP NAT Traversal 以及 UDP NAT Traversal 在本質上就有蠻大的相異之處，而本論文主要論述的要點集中在 UDP 方面的討論；這是因為 TCP 的 NAT Traversal 尚需要考慮到 TCP 本身的 3-way Handshake，較為複雜。。本論文希望可以提出一個對於 UDP NAT Traversal 比較全面性的解決方法。

UDP 如大家所認識是一個相當輕量級的 Transport Layer Protocol，而也因為 UDP 的即時性的特色所以被很多注重 Real Time 的影音串流軟體所使用，例如當紅的 VoIP 即是以 UDP 為主要傳輸的 Protocol。特別是 VoIP 類的軟體，當它在傳送聲音串流資料的時候其實是額外再建立一條 Peer-to-Peer 的串流通道來傳送此 Media Stream。當如前述的 Peer-to-Peer 通道要建立的時候如果遇到通話雙方剛好在 NAT 底下的話便會產生問題。以下我們以 Figure 1-2 來述說問題為何發生。

如 Figure 1-2 所示，當 node B 想要打電話給 node A 的時候，會發出 Invite 這個訊息並且帶有自己本地端的 IP 和 Port(即 Figure 1-2 中的 IPB/PortB)並經由 SIP Server 轉送給 node A，當 node A 決定回應這次通話便會回覆 200 OK 的訊息給 node B 一樣是經由 SIP Server 轉送並且帶有自己本地的 IP 和 Port(即 Figure 1-2 中的 IPA/PortA)。但是，如之前提過的 Private IP 是無法在 Public Network 進行路由，而 node A 和 node B 都位於 NAT 底下，很顯然的在第三步驟要建立的 RTP 語音串流必然會失敗，通話雙方將無法聽到彼此的聲音。

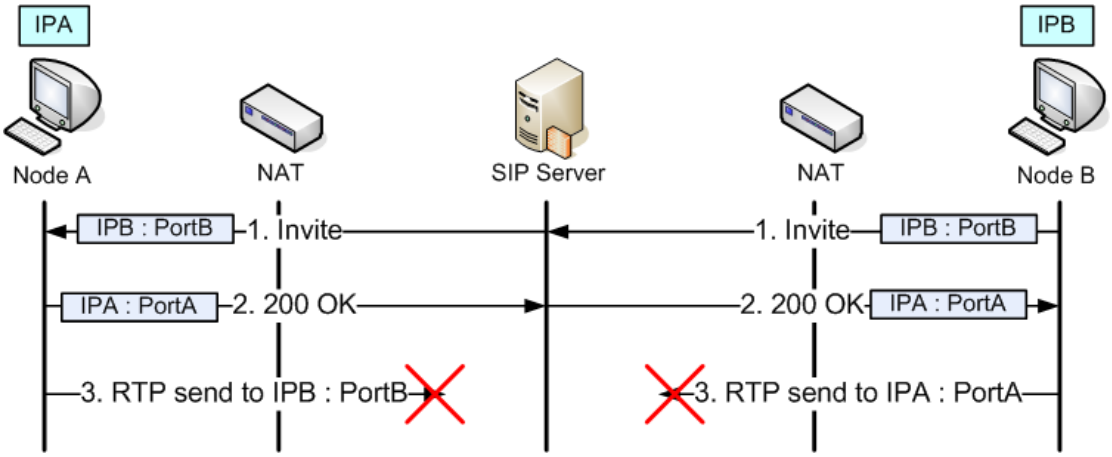

Figure. 1-2 SIP-based VoIP 建立通話流程圖

Figure 1-2 的例子說明瞭位在 NAT 底下的 Private Node 並無法知

道自己是否在 NAT 底下，且就算知道自己在 NAT 底下也無法去得到一個可用的 Public Mapping 好讓自己可以順利接收到外界傳送進來的封包。

目前針對如何去得到在 NAT 上的 Public Mapping，已經有一套相當成熟的技術稱為 STUN [3]，不過就算我們可以順利得到這個 Public Mapping 封包也不一定可以順利傳送進來給 Private Node。以下舉例 Figure 1-3 來說明這種狀況。

首先，我們定義所謂的 Public Mapping，其實是 Private Node 的一組 Private IP/Port 去對應到 NAT 上的 Public IP/Port，NAT 就是利用這種 Public Mapping 來進行 IP/Port 的轉換。當 Figure 1-3 中的 node A 利用 STUN 機制去獲得了 Public Mapping，並且將此 Public Mapping 經由第三方 Server 告訴了 node B。如 Figure 1-3 所示，當 node B 傳送封包至此 Public Mapping(即圖上的 140.113.21.7:5000)，NAT 是否會順利轉送此封包至 node A 端看此 NAT 的 Filtering 特性，而為了讓此封包可以讓 node A 順利收到便有人提出了一項技術 UDP Hole Punching [4]。

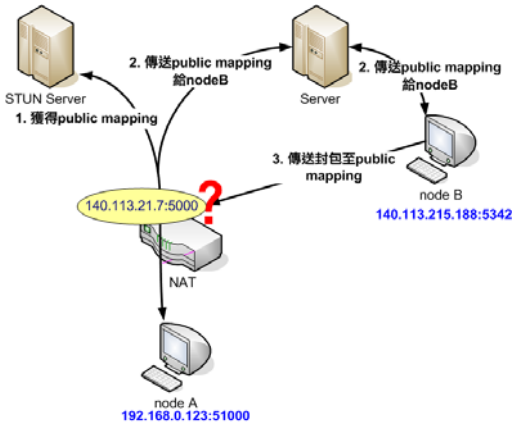

Figure. 1-3 NAT Filtering 示意圖

但是 UDP Hole Punching 還是無法解決全部 NAT 的問題，而且在本質上似乎有些 NAT 先天上就是無法和外部進行 Peer-to-Peer 的行為，所以便有人提出 TURN [5]，其實就是利用 Relay Server 在端點之間幫忙轉送封包，而此方法最大缺點就是需要架設 Server 且需要付錢給 ISP 去購買一定的頻寬，除非萬不得已，否則大部分的人都不會採取此項方法。

總結以上，要在 NAT 之間建立一個 Peer-to-Peer 的連線通道，必須作到下列幾件事情：

1. 設法得到 Public Mapping
2. 利用 Hole Punching 的技術順利建立連線通道
3. 當無法順利建立直連通道，可以使用 Relay 的方式替代

Interactive Connectivity Establishment (ICE) [6]，就是一套整合 STUN 以及 TURN 來完成上述的三項事情的技術，已是現今大家認為比較好的一個 Solution。不過 ICE 在實際測試上還是有一些不足的地方，例如建立連線的時間有點久，以及無法實際去提高 Direct Connection Rate (DCR: 所謂的 DCR 定義為，給定一固定數量的 NAT 配對組合，在這組合群當中所能達到直連而非使用 Relay 的成功率)，更因為缺乏資訊無法去判斷失敗原因，因為 ICE 的測試為 UDP 可能會有封包遺漏的情形而導致測試失敗。

有些研究文獻也對於 ICE 的 Delay Time 問題提出探討，並且提出了一套方法來達成 NAT Traversal [7]。

為了有效的增進 ICE 的連線測試效率以及增加直連率，本篇論文期望可以引進一種攜帶資訊幫助進行連線測試的方式，來讓 ICE 的功能更加的

強大。

而有關利用資訊來幫助解決 NAT Traversal 的方法，也有相關文獻進行過研究。所謂的 CDCS [7]，就是一套利用攜帶 NAT 資訊來達成 NAT Traversal 的方法，但是此方法還是有其缺點存在，不過其「利用資訊幫助 NAT Traversal」的這項特性，卻是一項很大的優點。。

所以，我們引用了 CDCS 的「利用資訊幫助 NAT Traversal」的這項特性，來融合原本的 ICE，如此一來便可以期待去增進 ICE 的 NAT Traversal 的能力，而這也是本篇論文的最大目的。

1.3. 章節簡介

- 第一章　緒論

  簡介本論文的研究動機以及目的

- 第二章　研究背景與相關研究

  介紹相關背景知識以及相關領域的研究概況

- 第三章 Context Aware NAT Traversal Scheme (CAN)

  闡述本篇論文所提出的 NAT Traversal 技術，包括該收集那些 NAT 資訊以及如何使用這些資訊

- 第四章 CAN的實做

  描述實做內容的架構以及實做結果的討論

- 第五章 實機測試與驗證

  描述測試環境、以及測試內容

- 第六章 結論與未來工作

  結論、本篇論文的總結及外來可進行的研究方向

第二章 研究背景與相關研究

2.1. Network Address Translation (NAT)

在本小節將會詳細描述 NAT 的基本的運作原理，並對現有市面上 NAT 所具有的特性加以分析以及說明這些特性對於 NAT Traversal 的運作有哪些影響。

Figure 2-1 為一簡單的 NAT 工作原理示意圖，一台 NAT 通常會有兩個以上的介面，一個對內部網路一個對外部網路。正因為 NAT 位於內部和外部網路的中間恰好扮演了一個橋樑的角色，所以所有從內部流向外部網路，或者從外部網路流向內部網路的封包都會經過 NAT，而 NAT 就在這樣的環境下發揮它的公用。

如 Figure 2-1 所示，所有的 NAT 其內部都會維護一個稱為 Mapping Table 的資料結構，詳細記載流通於內部和外部之間的封包是如何被 NAT 進行轉換的。如 Figure 2-1 中的 192.168.0.7:5100 這個內部封包將會在流向外部網路世界的時候被轉換成 140.113.215.188:12345。

Figure 2-1 中只畫出內部流向外部的封包轉換過程，但如果這時候 外部的位址，以圖上的 140.113.21.88:747 為例，當此位址的機器想要將封包回覆給內部網路的時候，便會將封包的目的地填為 140.113.215.188:12345，而 NAT 在接收到這樣的封包的時候便會根據 Mapping Table 將目的地作轉換成 192.168.0.7:12345，並將此封包順利路由到內部的機器之中。

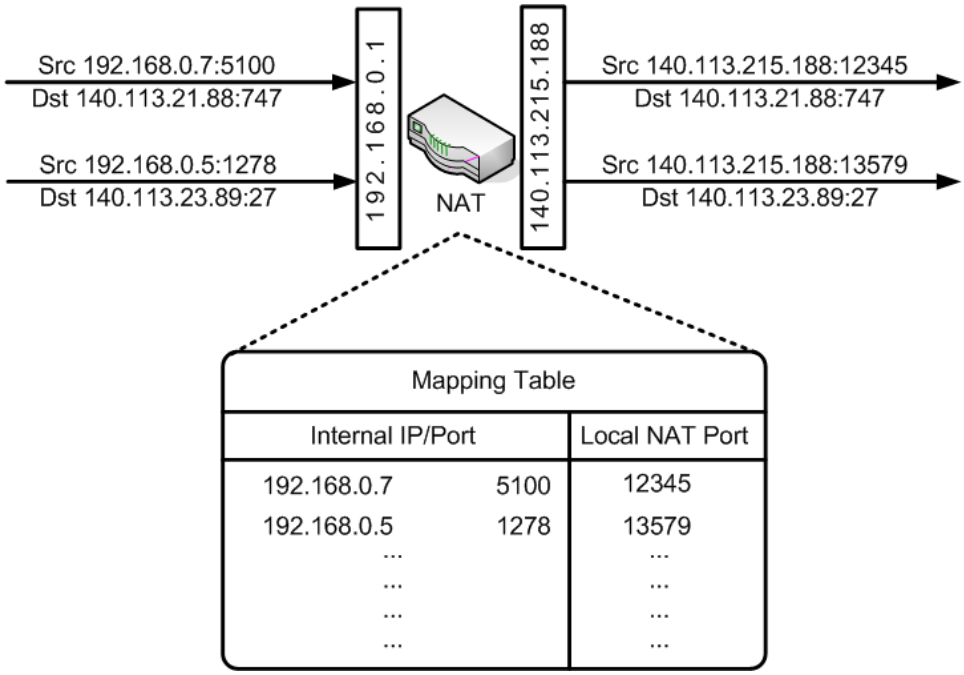

Figure. 2-1 NAT 工作原理示意圖

以上為一個 NAT 的簡單運作過程，若考慮實際上的實做可能會因為各家廠牌的不同而有些不同，例如維護 Mapping Table 的方式各家廠商所使用的演算法可能會不一樣。若果是更複雜的 TCP 的連線維護，各家廠商的差異性可能更大，不過 TCP 就不在本篇論文的討論範圍之內。

2.1.1. NAT 特性及分類

一如前述，NAT 所具有的許多特性都會影響到 NAT Traversal 的成功與否，所以在本小節將會提出 NAT 所具有的哪些特性會影響到 NAT Traversal 以及詳述這些特性。

本小節將在接下來的內容介紹 NAT 的幾個重要的特性，特別是會影響到 UDP Hole Punching 的結果： NAT Mapping and Filtering Rule、

Hairpin、ConnTrack Binding。

2.1.1.1. Mapping and Filtering

- Mapping Rule: 當 internal node 向外發起封包，NAT 如何去維護 public mapping 的行為。可分為 Independent Mapping 和 Dependent Mapping 兩種。

參見 Figure 2-2，當同樣一個內部 node A (位址為 192.168.0.7:5100) 的封包發出，雖然目的地不一樣，一個發向 node B (位址為 140.113.21.88:747)、一個發向 node C (位址為 140.113.23.89:27)，但是在 NAT 上所配發的 Public Mapping 都是 140.113.215.188:12345。這種只要是內部端點的 IP/Port 固定 Public Mapping 也就固定的行為稱作「Independent Mapping」，而具有這種特性的 NAT 也稱為「Cone NAT」。

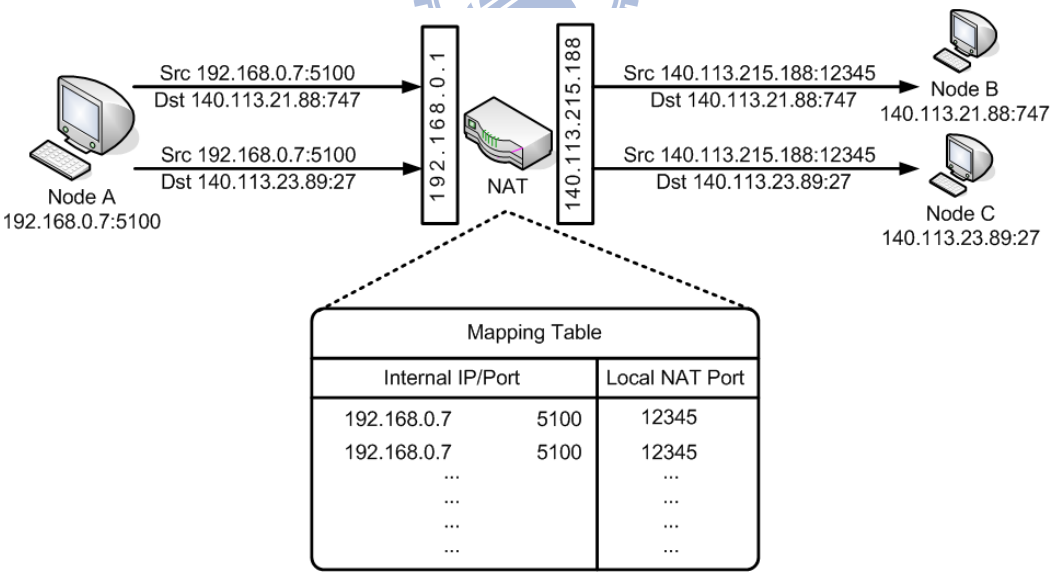

Figure. 2-2 Independent Mapping 示意圖

參見 Figure 2-3，當同樣一個內部 node A (位址為 192.168.0.7:5100)

的封包發出，目的地不一樣，一個發向 node B (位址為 140.113.21.88:747)、一個發向 node C (位址為 140.113.23.89:27)，而在 NAT 上所配發的 Public Mapping 也不一樣，一個是 140.113.215.188:12345、一個是 140.113.215.188:13679。這種雖然內部端點的 IP/Port 固定，但卻會因為目的地的位置不同而導致 Public Mapping 不同的行為稱作「Dependent Mapping」，而具有這種特性的 NAT 也稱為「Symmetric NAT」。

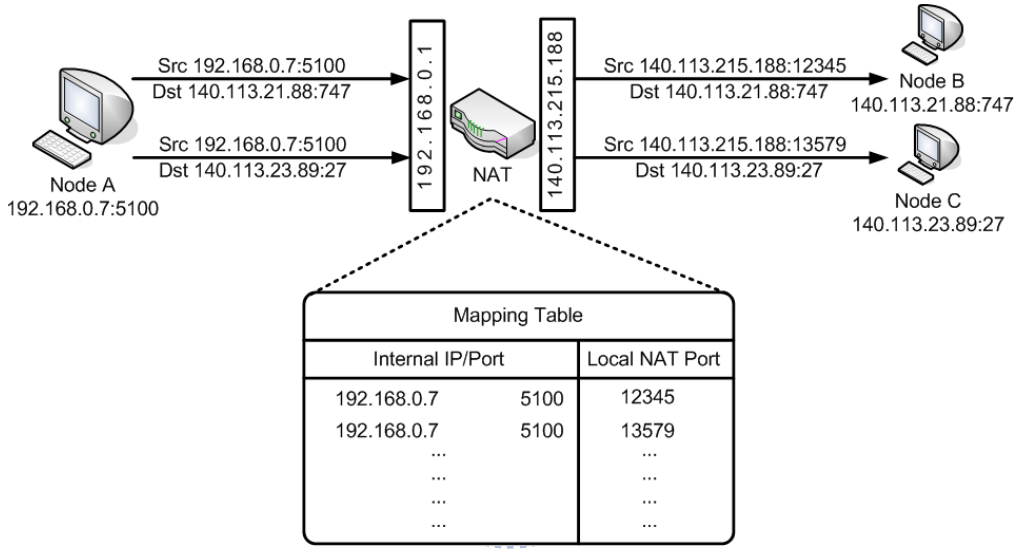

Figure. 2-3 Dependent Mapping 示意圖

- Filtering Rule: 當外部網路封包到達 NAT 上的某一個 Public Mapping 的時候，NAT 的行為。可分為 Independent Filtering、Address Dependent 和 Address and Port Dependent 三種。

參見 Figure 2-4，當內部端點 node A 向外部端點 node B 發起一個封包並且順利在 NAT 上建立了一個 Public Mapping，這時若果有其他外部端點知道了這個 Public Mapping 都可以將封包丟到這個位置並且順利的被內部端點 node A 所收到，以 Figure 2-4 為例，node 3 將封

包丟到了 140.113.215.188:12345 並且順利的經由 NAT 的轉換而被 node A 接收到。這種特性稱為「Independent Filtering」。

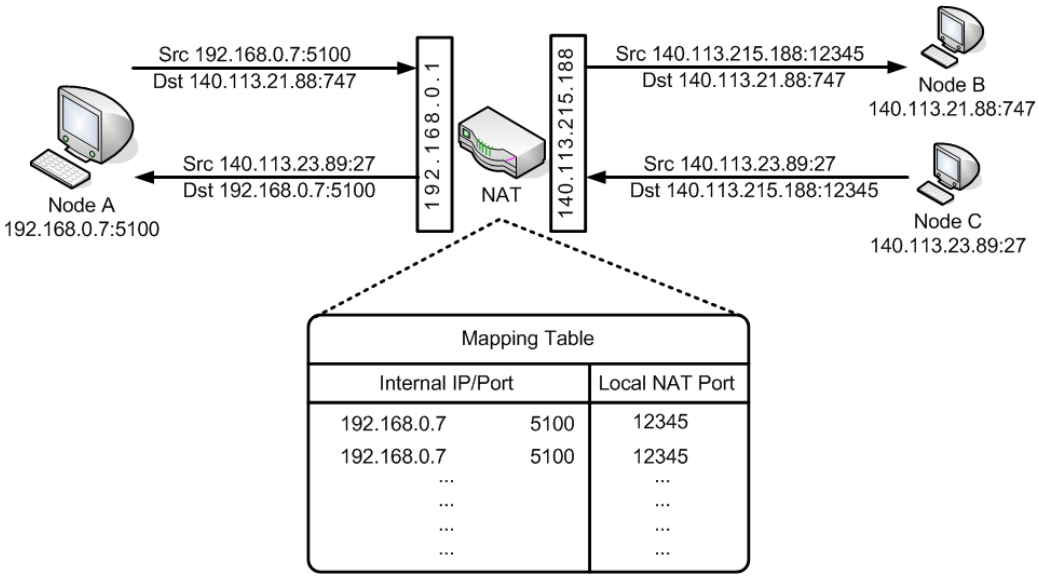

Figure. 2-4 Independent Filtering 示意圖

參見 Figure 2-5，當內部端點 node A 向外部端點 node B 發起一個封包並且順利在 NAT 上建立了一個 Public Mapping，這時只有 node B 的 IP 所發起的封包丟到這個 Public Mapping 可以順利的 node A 所順利收到。

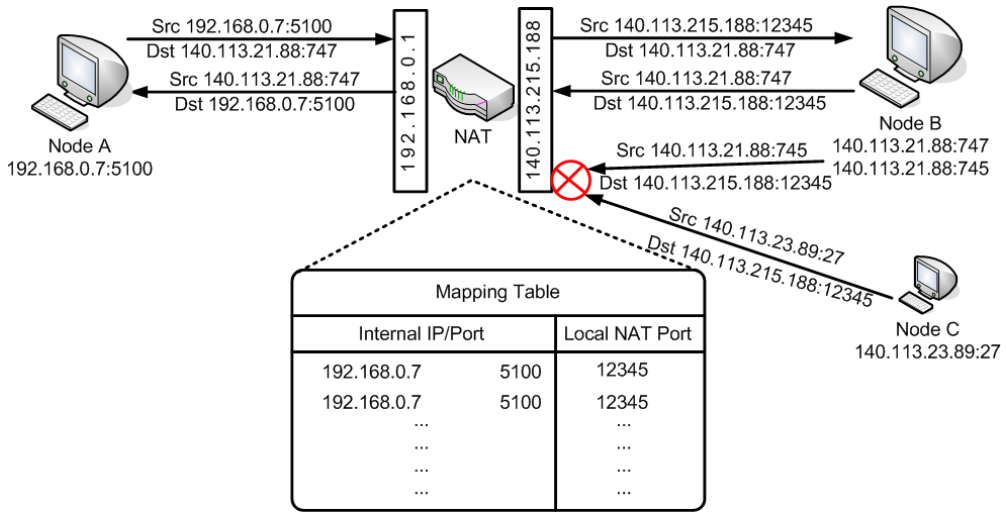

Figure. 2-5 Address Dependent Filtering 示意圖

以 Figure 2-5 為例，node B 從另一個 socket，140.113.21.88:745 發起封包丟到 140.113.215.188:12345 此封包就可以順利的經由 NAT 的轉換而被 node A 所接收到，但是 node C 丟到 140.113.215.188:12345 的封包就會被 NAT 擋下來，而無法為 node A 所接收到。這種會被外部端點 IP Address 所限制住的 Filtering 特性稱作為「Address Dependent Filtering」。

參見 Figure 2-6，當內部端點 node A 向外部端點 node B 發起一個封包並且順利在 NAT 上建立了一個 Public Mapping，這時只有 node A 所發起的封包中的目的地 IP/Port 所發起的封包丟到這個 Public Mapping 可以順利的 node A 所順利收到。以 Figure 2-6 為例，node B 從另一個 socket，140.113.21.88:745 發起封包丟到 140.113.215.188:12345 此封包就會被 NAT 擋下，而 node C 丟到 140.113.215.188:12345 的封包也會被 NAT 擋下來，而無法為 node A

所接收到，只有 node C 的 140.113.21.88:747 這個 Socket 所丟出的封包才會被 NAT 順利轉換而被 node A 所接收到。這種會被外部端點 IP/Port 所限制住的 Filtering 特性稱作為「Address and Port Dependent Filtering」。

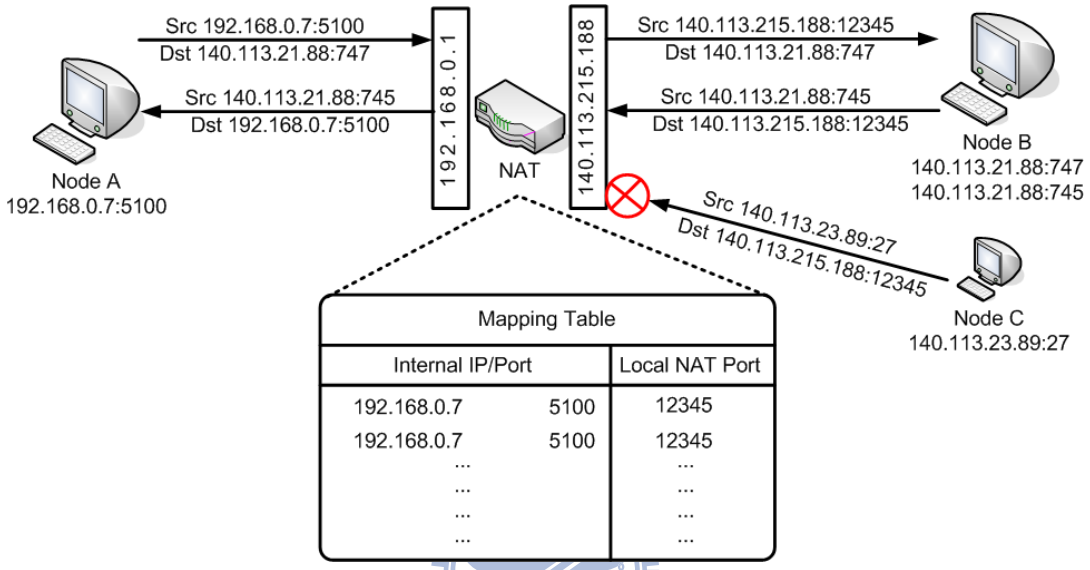

Figure. 2-6 Address and Port Filtering 示意圖

總結以上，我們以下列表格來做個結論：

表格 2-1 NAT 特性分類表

	Independent Filtering	Address Dependent Filtering	Address and Port Dependent Filtering
Independent Mapping	Full Cone	(Address) Restricted Cone	Port Restricted Cone
Dependent Mapping			Symmetric

以下，再針對四種 NAT Type 描述一下:

- Full Cone NAT: Public Mapping 不會隨著外部端點的改變而跟著改變，並且任一外部端點皆可以藉由傳送封包到此 Public Mapping 而讓內部端點收到。
- Restricted Cone NAT: Public Mapping 不會隨著外部端點的改變而跟著改變，但是只有內部端點曾經傳送封包過的 IP 位址可以藉由傳送封包到此 Public Mapping 而讓內部端點收到。
- Port Restricted Cone NAT: Public Mapping 不會隨著外部端點的改變而跟著改變，並且只有內部端點曾經到達過的 IP/Port 才可以藉由傳送封包到此 Public Mapping 而讓內部端點收到。
- Symmetric NAT: Public Mapping 會隨著外部端點的改變而跟著改變，並且只有內部端點曾經到達過的 IP/Port 才可以藉由傳送封包到此 Public Mapping 而讓內部端點收到。

2.1.1.2. Hairpin

當一個內部的網路拓樸是由一個以上的 NAT 所組成的時候，NAT 是否具有 Hairpin 這項特性便相當的重要因為這會關係到在同一個 NAT 底下的端點如果要進行通訊連線是否可以達成直連。接下來以 Figure 2-7 來說明 Hairpin 的特性。

Figure 2-7 為一多層的 NAT 網路架構圖，node A 已經在最外層 NAT (NAT A)上面開了一個 Public Mapping，140.113.215.188:5689，此時若是位在同一最外層 NAT 底下的 node B 發起封包並且丟向了這個 Public Mapping，若是封包可以被 NAT 接受並且順利的 node A 收到，我們稱這種行為是具有 Hairpin 的特性。以 Figure 2-7 為例，若是 NAT

A 具有 Hairpin 的話，那 node A 和 node B 便可以達到直連的效果，否則 node A 和 node B 之間就只能靠著 Relay 來進行通訊。

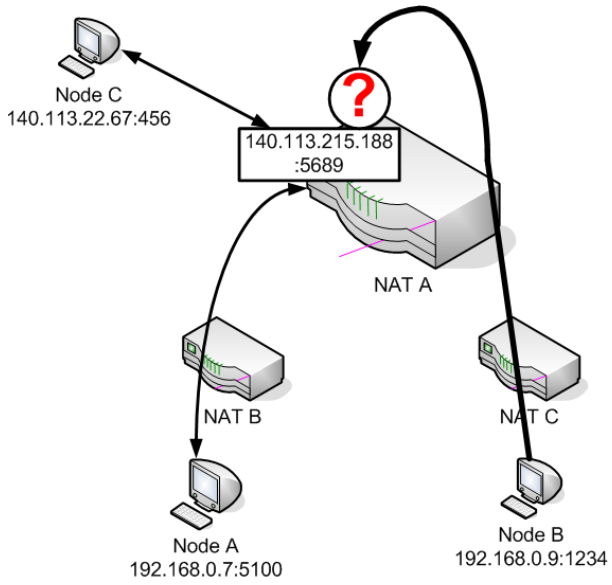

Figure. 2-7 Hairpin 特性示意圖

2.1.1.3. ConnTrack Binding

ConnTrack Binding 為一個新發現的問題，引述於 [11]。ConnTrack Binding 問題的關鍵點在於它會讓一個屬於 Cone 類型的 NAT 發生了 Mapping 改變的現象，特別是進行 UDP Hole Punching 這項技術的時候，往往會因為這特性而讓 NAT Traversal 失敗，關於 ConnTrack Binding 是如何導致 UDP Hole Punching 失敗，將會在之後的第三章加以詳細的介紹，以下則是簡述 ConnTrack Binding 的行為是怎樣。

簡言之，ConnTrack Binding 就是 NAT 在某一個 Public Mapping 上收到了一個未知位址的封包，這時 NAT 會對此未知的位址有一個 Block 的動作，下次若是要送封包到此位址 NAT 將會開啟一個新的 Public Mapping 而不再是使用原本的 Public Mapping。以 Figure 2-7 來當作例子說

明。

參見 Figure 2-7，node A 利用 Socket A (192.168.0.7:1234) 和 node B 連線並且在 NAT 上面建立了一個 Public Mapping，通訊阜為 51000，此時若有一個外界的 node C 知道了此 Public Mapping，並且發起封包丟到這個 Public Mapping，由於此 NAT 為 Restricted Cone NAT 所以封包會被 NAT 丟棄，但是當 node A 再次從 Socket A 發出封包到外界的時候卻發生了 Mapping 由之前的 51000 變換成 1024。如果 NAT 具有這種特性的稱為具有 ConnTrack Binding 的特性，而通常會具有這種特性的 NAT 大都為 Port Restricted Cone NAT 或者是 Restricted Cone NAT。

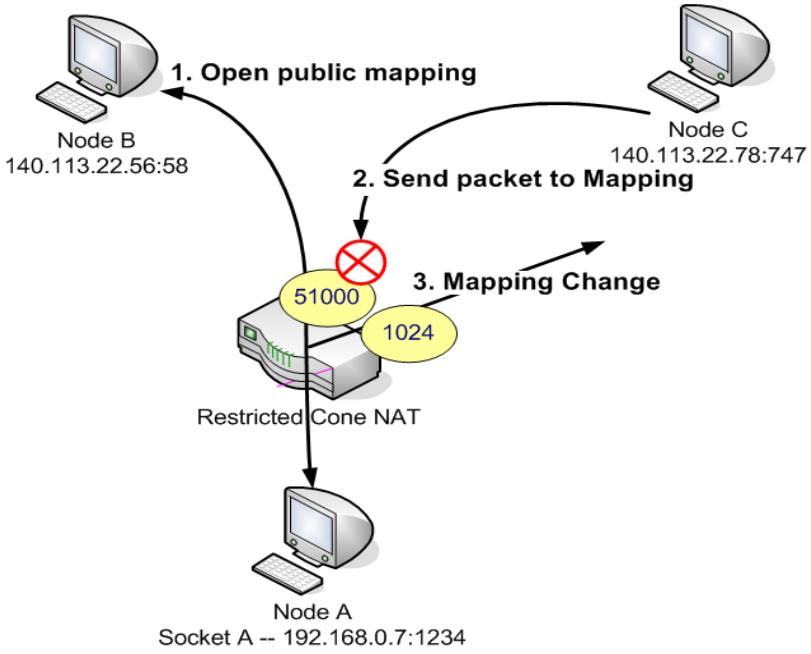

Figure. 2-8 ConnTrack Binding 示意圖

2.2. NAT Traversal Method

目前有關 NAT Traversal 的研究文獻相當的多，不過可以從大方面歸納出以下三種類型 [8]：

1. Application-layer approaches: 只需改動應用程式，便可達到 NAT Traversal 的目的。例如 TURN [5]、ICE [6] 便是屬於此類型的方法。

2. NAT Box approaches: 直接對於 NAT devices 進行修改。所以此類方法除了要改動應用程式之外還要 NAT devices 有支援。例如 UPnP IGD [9], [13]。

3. Mid-layer approaches: 直接去變動現有的 Network 或者 Transport Layer Protocol 的架構，比較全面的解法，不過在現實中難以實現。例如 IP 4+4 [10]。

總結以上三種類型，現實環境中比較常用的就是第一種類型的方法，而本篇論文所提出的方法是屬於第一種類型，所以接下來本篇論文將會介紹第一種類型中的幾種重要的方法。

2.2.1. STUN and TURN

STUN [3] 和 TURN [5] 都是 IETF 提出來的 RFC 標準，目前除了 STUN 已經確定之外，TURN 目前尚在繼續在制定中尚未正式定案。一如第一章所提到的 STUN 為一項可以偵測出某一 NAT 上面所開得 Public Mapping，而應用程式可以利用此 Public Mapping 配合 Hole Punching

達成 NAT Traversal，而 TURN 則為一 Relay 的技術，要特別強調的是 TURN 所幫忙 Relay 的是應用層(Application Layer)的資料，所以其底層還是走 TCP 或 UDP，不過本篇論文所提到的 TURN 皆是使用 UDP 為傳輸底層。

Figure 2-9 是一個標準的 STUN 的架構，STUN Server 需要備有兩個 IP Interface，並且搭配兩個 Port Number (3478 and 3479) ，所以 STUN Server 應該會開有四個 Socket，跟 Figure 2-9 圖示中一樣。

參照 Figure 2-9，位在 NAT 底下的 Private Node 可以建立起一個 Socket 並且向 STUN Server 發起 Binding Request，而 STUN Server 在接收到這樣的一個封包之後會做出一些處理，並且將封包來源的 IP/Port 放入 Binding Response 之中回傳給此 Private Node，如此 Private Node 接收到此 Response 之後便可以從封包之中知道自己的 Public Mapping。不過 STUN[3] 對於 Symmetric NAT 所得到的 Public Mapping 並無法使用，因為 Symmetric NAT 會因為目的端的不同而 Public Mapping 有所不同。

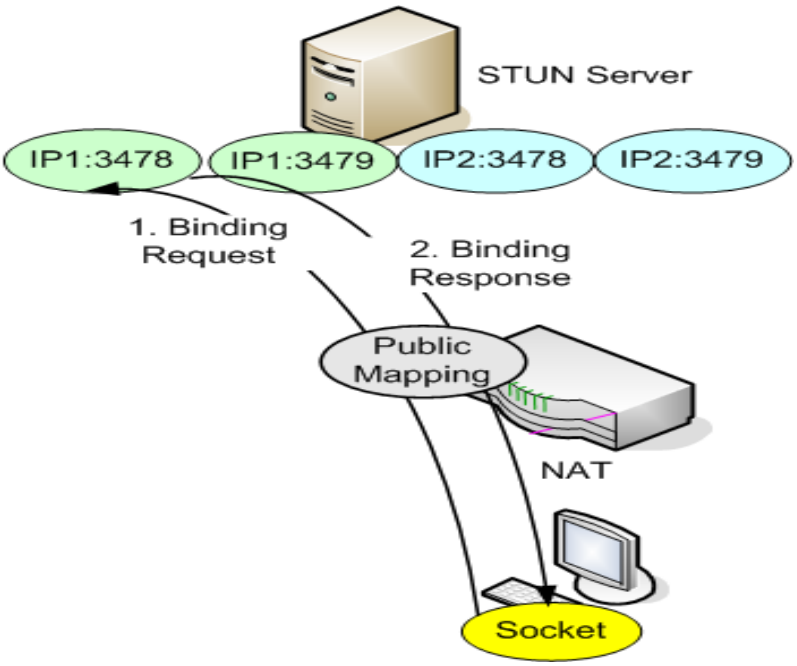

Figure. 2-9 STUN 架構圖

當然 STUN 的功能並不僅止於偵測 Public Mapping，它也定義了一個演算法可以幫忙偵測出目前的網路環境為 Public Network 或是 Private Network，若是 Private Network 更可以偵測出目前此 NAT 的類型為何，有關 STUN 的演算法可以參考 STUN 在此便不再贅述。

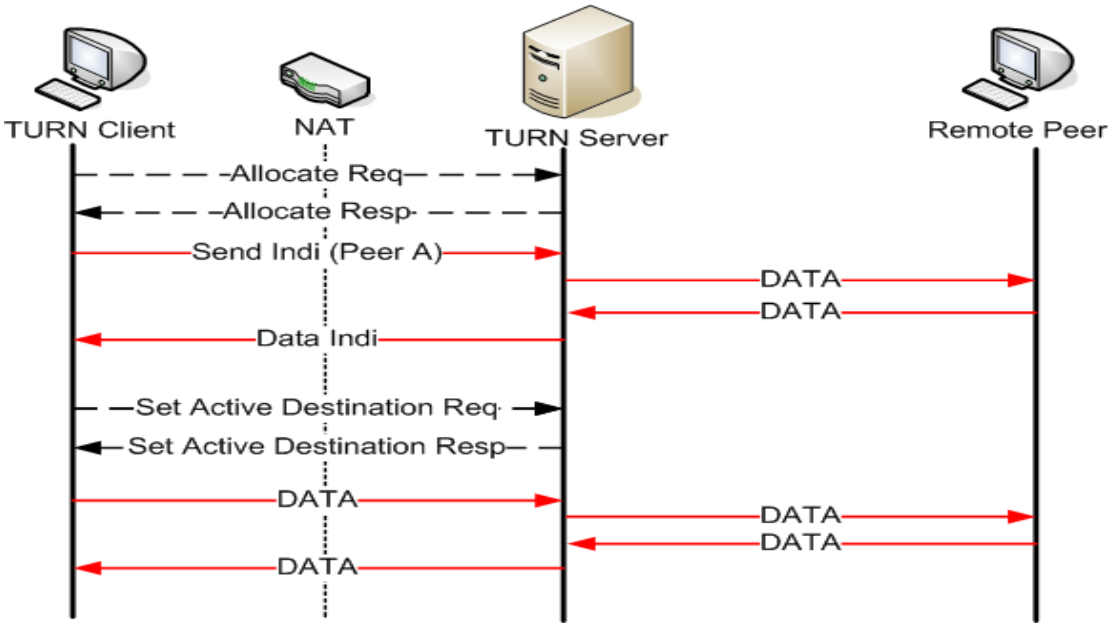

Figure. 2-10 TURN 運作流程圖

下面以 Fig 2-10 來描述 TURN 的機制為何。一開始 TURN Client 向 TURN Server 發出 Allocate Request 要求一個 Relay Resource，在 Server 許可之後會以 Allocate Response 回覆內容中帶有 Server 所開啟的 Relay Port，而之後 TURN Client 所要溝通的 Remote Peer 都是將資料丟到這個 Relay Port；而 TURN Client 如果要送資料給 Remote Peer 則是將資料和 Remote Peer 的 IP/Port 包裹在 Send Indication 這個 Message 當中送到 Server，Server 就會幫忙轉送，若有資料要給 TURN Client，Server 會以 Data Indication 這個 Message 並將資料包裹在其中傳送給 TURN Client。

一旦 TURN Client 以 Set Active Destination 和某一個 Remote Peer 綁定之後，資料便不需再藉由 Message 包裹，而可以直接接收和傳送資料，如 Figure 2-10 一樣。

特別注意一點，如果有某一個端點知道了一個 Client 所開啟的 Relay

Port 而故意傳送封包到此 Relay Port，Relay Server 會將使封包丟棄，這是因為 Relay Server 並沒有關於此封包應該丟向哪一個 Client 的紀錄，所以，Relay Server 的 Filtering 有點像是 Port and Address Dependent Filtering。

2.2.2. Interactive Connectivity Establishment (ICE)

ICE [6] 是 IETF 所制定的 NAT Traversal 的技術,ICE 結合了 STUN 和 TURN 提供了一個對於 SIP (Session Initiation Protocol) 完整的穿越 NAT 的方案，不過當然這項技術也可以加以改變應用於其他傳輸協定上。

ICE 的主要精神在於說對於每一個位在 NAT 底下的 Private Node 理論上都會有兩個 IP address，一個為 Private Node 本身的 Local Address，另一個則為 NAT 上的 Server Reflexive address，而再加上利用 TURN 去得到的一個 TURN Relay Address，那總共會有三個 IP/Port，ICE 定義這樣的一個 IP/Port 為一個 Candidate。

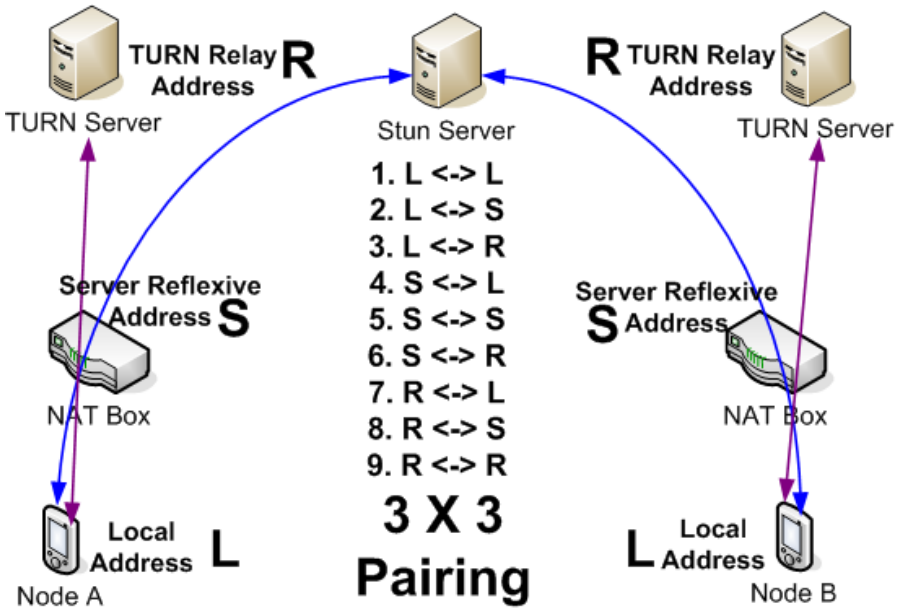

Figure. 2-11 ICE Candidates 示意圖

參照 Figure 2-11，當 Node A 和 Node B 都具有這樣的三種 IP/Port，兩者互相交換之後再進行三對三的配對，總共會產生九種組合，也就是會有九條通訊路徑，而 ICE 再對這九條路徑進行連線測試，以找出可用的路徑。

有一點需要注意，上述的九條路徑其實是有「路徑對稱性」存在。因為 node A 和 node B 都各自有著這樣的九條路徑的紀錄，所以其實 node A 的第二條路徑，L-to-S，其實就是 node B 的 S-to-L，所以，如果沒弄清楚這一點的話在實做的時候可能會產生最後選擇路徑錯誤的問題。

最新版的 ICE 多了一個步驟,可以將九條路徑簡化為六條路徑。也就是 Figure 2-11 中的 1-3 以及 6-9 這六條路徑，相關細節部份可以參閱 ICE 的文件。

Figure 2-12 為 ICE 的完整通訊流程，ICE 利用 SIP 中的 Invite 和 200 OK 來攜帶所要交換的 Candidate 資訊，當雙方交換完 Candidate 之後便會進行 Pairing 的動作，接下來會對於所有路徑進行 Connectivity Check，最後就是選定一條可用的路徑傳送 RTP 資料。

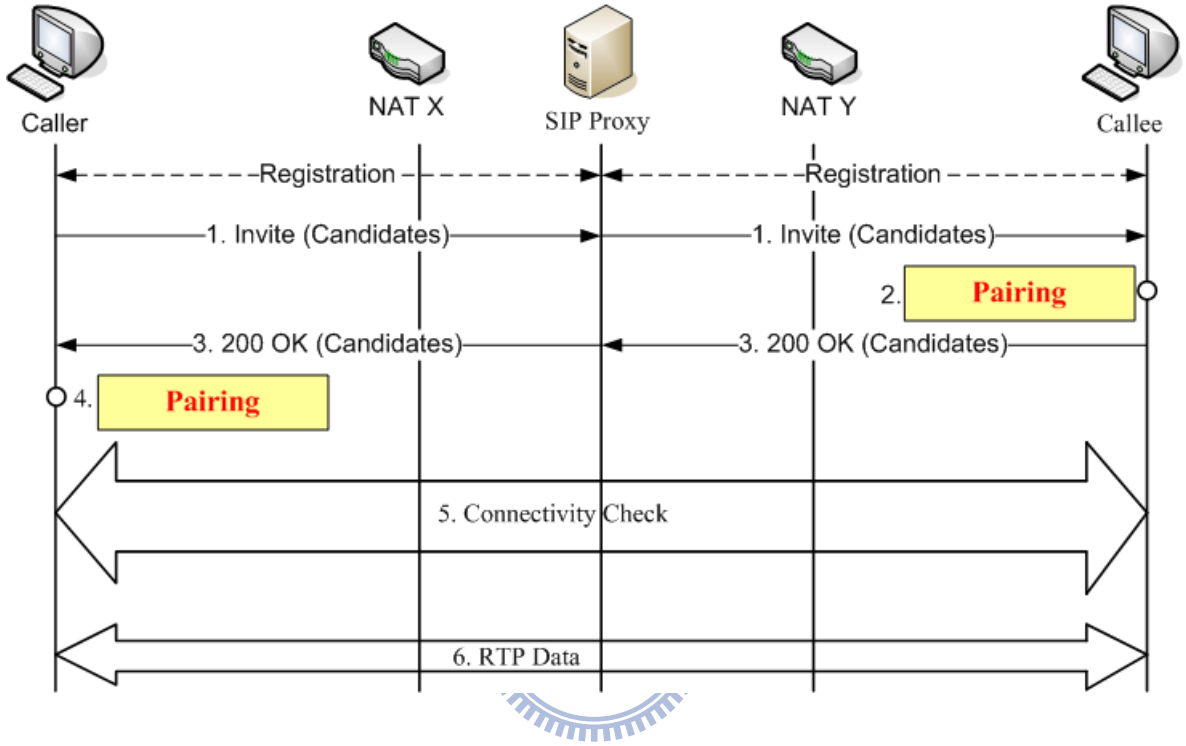

Figure. 2-12 ICE 訊息流程圖

2.2.3. CDCS

CDCS[7] (Case-based method for call setup)，提出來主要是為了希望可以藉由收集 NAT 資訊來進行各種 Hole Punching 的技巧，依照這些 NAT 資訊來對於各種不同的網路拓樸進行各個擊破的方式來達成 NAT Traversal，但是此方法在 NAT 資訊的收集上有不足之處，並且它將資訊放至一個中央的 Server，並且由 Server 進行全權的處理，所以哪種

網路情況該用哪種方式來達成 NAT Traversal 皆是由 Server 來通知要進行連線的兩個 Client，等於是額外增加了 Server 的工作。

CDCS[7] 的主要運作過成為下列這三部份：

1. Client 收集自己的 NAT Type 並且向 Server 註冊，Server 便儲存此向資訊。
2. 當 Caller 要向某個 Callee 通訊的時候，便會發出訊息由 Server 幫忙轉送給 Callee。
3. 當 Server 發現 Caller 和 Callee 確定要通訊的時候，便會依照雙放的 NAT Type 的情況，適當的要求 Caller 或 Callee 去做出一些動作來達成 NAT Traversal。

至於 CDCS 的細節部份本篇論文便不再贅述，有興趣的讀者可以參閱 Reference [7]。

2.3. 單元總結

總結以上，本篇論文介紹了 NAT 的主要特性，並且將其依照 Filtering 和 Mapping 規則的不同而分成四種類型，詳細請參照

	Independent Filtering	Address Dependent Filtering	Address and Port Dependent Filtering
Independent Mapping	Full Cone	(Address) Restricted Cone	Port Restricted Cone
Dependent Mapping			Symmetric

。

並且本論文也介紹了對於 NAT Traversal 有影響的兩大特性，一個為已有人提出的 Hairpin 特性，另一個則為新發現經由實驗得出來的特性 ConnTrack Binding 特性，尤其是 ConnTrack Binding 的解決方法以及在進行 Hole Punching 時候 ConnTrack Binding 的影響皆會是本篇論文的重點之一。

ICE 以及 CDCS 的優缺點也在本章節一併有了描述，並且也將承續兩種方法的優點來使本篇論文提出的方法更加的有用，並且改善兩者的不足之處。下列章節第三章將會詳述本篇論文的方法以及到底是取用了 ICE 以及 CDCS 的哪些優點。

第三章 Context Aware NAT Traversal Scheme (CAN)

3.1. 基本精神與目標

總結第二章節所提到的，ICE 的優缺點以及 CDCS 好壞處：

- ICE:
  - 優點: 完整的 NAT Traversal Solution、 IETF 所制定的標準
  - 缺點: 連線測試時間的冗長、無特殊機制去提昇直連率
- CDCS:
  - 優點: 利用「NAT 資訊」幫助進行 NAT Traversal
  - 缺點: 利用太少的 NAT 資訊、將決策機制交給 Server 增加了 Server 的負擔、需變動原有 Server 的架構 (例如 SIP Proxy Server)

有鑑於 ICE 的不足之處，本篇論文利用額外攜帶的 NAT 資訊來幫助 ICE 進行連線測試，以期能提高直連率(DCR) 以及降低連線測試時間；和 CDCS 不同之處在於，本篇論文提出的方法收集和利用了更多的 NAT 資訊來幫忙判斷連線情況，並且不是將 NAT 資訊交給 Server 處理，而是由連線的雙方交換並且保有此資訊，如此便不會如 CDCS 一般額外增加了

Server 的負擔。

我們希望本篇論文所設計出來的機制，是一個有彈性的機制，可以依照連線雙方的網路狀況來選擇最佳的路徑，並且針對某些特殊的 NAT 運作特性，可以因為事先的資訊收集來幫助克服原先 ICE 所無法達成直連的問題，藉此來提昇直連率。

以下本篇論文將以 SIP based VoIP 為例來說明整個運作的機制，當然，本篇論文所提出的方法不只可以套用於 SIP，任何可以夾帶資訊給要進行連線雙方的網路協定都是可以被支援的。

3.2. 基本機制

本篇論文所設計的機制還是基於 ICE 的基本精神(Multiple Addresses)之上，所以運作模式會類似於 ICE，不過因為有我們創立的新的機制，所以，以下就會對於我們的方法與 ICE 方法的相異之處詳細加以說明。

如上一小節所說，本篇論文的方法的重點在於讓所要進行連線的兩個端點進行 NAT 資訊的交換，而這兩端點再根據這些 NAT 資訊判斷如何進行連線測試，而本篇所會用到的 NAT 資訊有下列這幾項：

- 是否在 Public Domain
- NAT Type：若是位在 NAT 底下，則此 NAT Type 為何。有四種型別，Full Cone、Restricted Cone、Port Restricted Cone、Symmetric。
- Hairpin Support：若是位在 NAT 底下，則此 NAT 是否具有支援 Hairpin 這項特性。

◆ ConnTrack Binding Existence：若是位在 NAT 底下，則此 NAT 是否具有 ConnTrack Binding 這項特性。

本篇論文所提出的方法是由以下這幾項步驟所組成：

1. 收集 NAT 資訊： 當使用者將程式開啟的時候，程式將會使用一個稱為 NAT Check 的模組進行收集 NAT 資訊的工作，並且儲存此項資訊以待要進行建立連線的時候取用。

2. 交換 NAT 資訊： 當使用者想要和某個端點進行連線的時候，通常會有個機制是可以將 Control Signal 帶給兩個要進行連線的端點，例如 SIP。以 SIP 為例子，使用者會利用 SIP Signal 中的 Invite 將自己的 NAT 資訊帶給遠方的端點，而遠方的端點利用 200 OK 將它的 NAT 資訊帶給使用者，如此一來雙方都有了完整的 NAT 資訊。

3. 進行連線測試：一如之前所說，本篇論文提出的方法是以 ICE 為基礎，所以 ICE 的 Candidates 的交換也在上述的第二步驟的時候一併進行了交換，所以雙方也都會有 Figure. 2-11 圖中所示的九條路徑。此時，雙方在根據 Context Aware NAT Traversal Algorithm 來決定出所要測試的路徑為哪一條路徑。

4. 連線建立的結果： 如果沒有例外狀況產生，連線將會建立成功，這時雙方端點便可以進行資料的傳輸。若是，連線建立失敗，那便是產生了例外狀況。

以下 Figure 3-1，為簡單的本篇論文的基本機制說明圖(以 SIP Based VoIP 為例子)，首先會執行 NAT Check 去收集資訊並且將資訊儲存起來，當要進行 Call Setup (打電話)的時候，便會取用所儲存起來的 NAT 資訊。

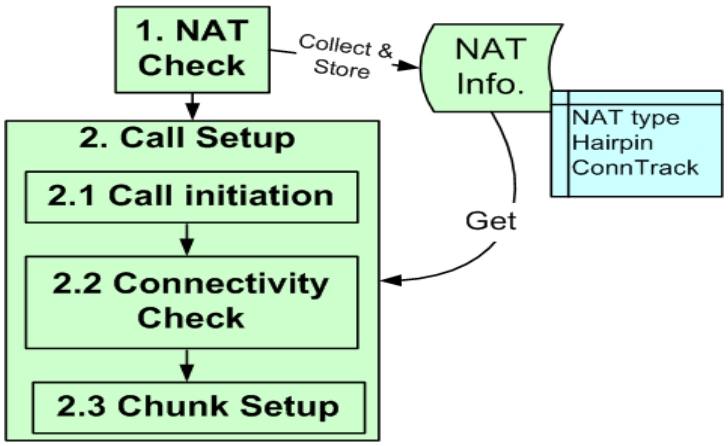

Figure. 3-1 CAN 基本架構圖

我們再用 Figure 3-2 來說明整個完整的訊息流程。Caller 和 Callee 雙方藉由 Invite 和 200 OK 交換 NAT 資訊，當雙方都有了彼此的 NAT 資訊之後，便利用 Context Aware NAT Traversal Algorithm 來決定所要進行測試的路徑，當測試成功之後便可以利用此路徑來傳送 RTP 資料。

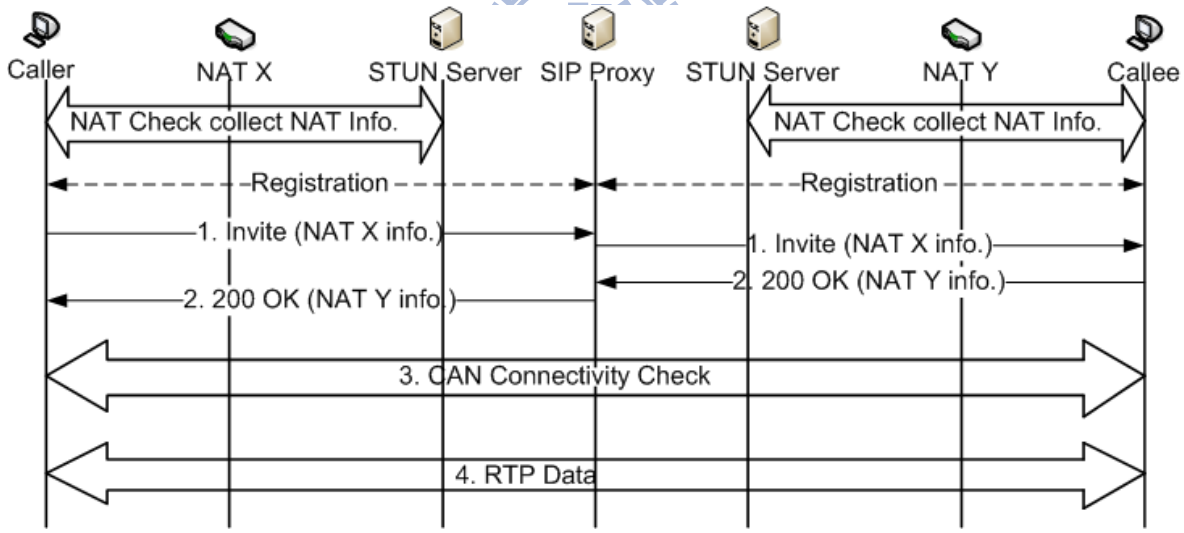

Figure. 3-2 CAN 完整訊息流程圖 (SIP Based VoIP 爲例)

總結以上，可以知道我們的機制為在 ICE 的原本機制之上加上 NAT 資訊的收集以及參照 NAT 資訊去選擇出最佳路徑來進行測試，如此不但可以縮減測試路徑的總數，更可以瞭解測試失敗的原因。

3.3. Context Aware NAT Traversal Algorithm

以下本節內容將要詳細闡述,本篇論文是如何應用 NAT 資訊來決定所要測試的路徑為何。

本篇論文將利用 NAT 資訊來決定所要測試的路徑為何的演算法稱為 Context Aware NAT Traversal Algorithm (CANA)。以下我們以 Figure 3-3 來說明 CANA 的基本流程。

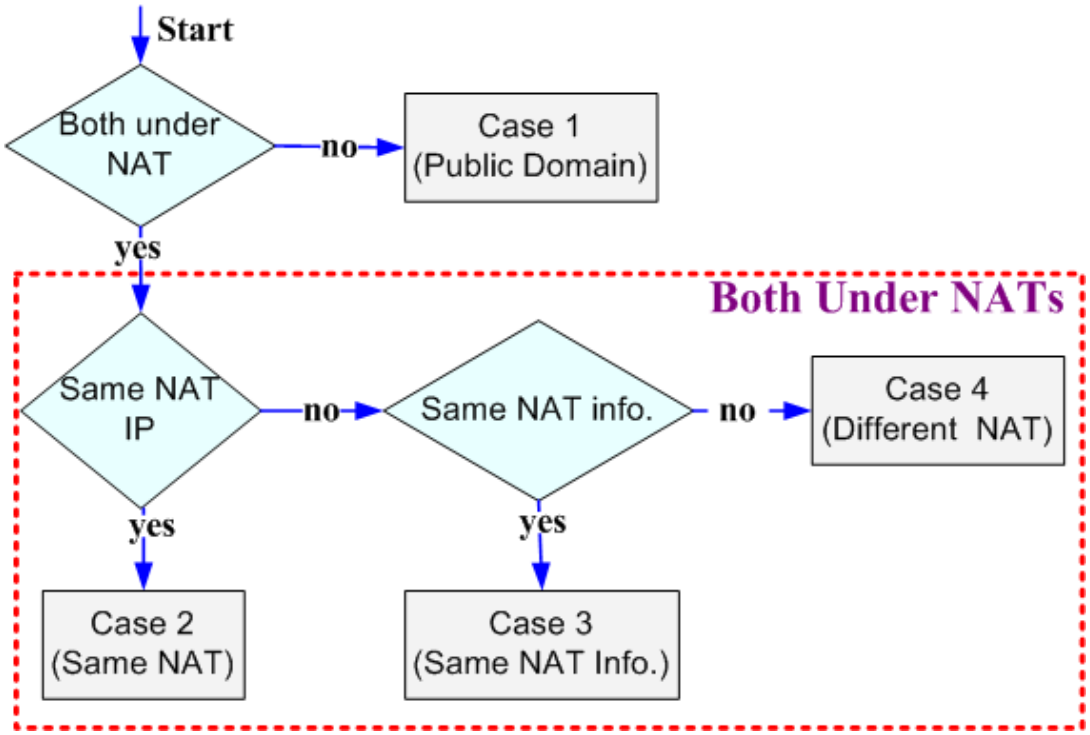

Figure. 3-3 CANA 基本流程圖

首先 CANA 會判斷說本身與遠方要進行連線的端點是否都在 NAT 底下，如果都在 NAT 底下的話那便是屬於 Figure 3-3 中虛線內的 Case，否

則便是 Case 1 的情況，屬於 Public Domain 的配對情況。

當本身和遠方端點具有同樣的 NAT Public Mapping IP (Server Reflexive Address)的時候，表示雙方位於同樣一個 NAT 底下，此時便是屬於 Case 2 的情況；接下來再判斷雙方式否有同樣的 NAT 資訊，如果是的話便是屬於 Case 3，雙方可能會在同樣一個 NAT 底下的情況，否則便是 Case 4，雙方位於不同的 NAT 底下的情況。

綜觀以上，CANA 將連線雙方的網路拓樸情況分為四種類行，並依照各種類型的不同而給以不同的解決方案，藉此可以減少原先 ICE 所要測試的路徑數目，以及也可以確定兩者所選定的路徑為唯一且固定的而不會有路徑對稱性的問題，在原本 ICE 中實做上可能遇到的一個問題為路徑對稱性問題，此問題已在第二章中有介紹過了，在此便不再對此贅述。

接下來將會在四個小節中，分別詳細介紹四個 Case 的運作機制。

3.3.1. CASE 1 (Public Domain)

Case1 的情況為 Public Domain 的情況，在此情況下有可能有兩種情況第一種為兩者都在 Public Domain；第二種為一者在 Public Domain，一者在 Private Domain。不過，不論哪兩種，在這個 Case 1 的情況下絕對會用到對於 Local Addresses 的測試，而在 Case 1 的情況下最後選出來的測試路徑只會有一條。

在原本 ICE 的情況之下，此種 Case 依然會進行九條 Case 的測試，那便會有了一種無形的浪費，而也因此浪費了 TURN Server 的資源，因為在此狀況下 TURN Server 是完全不需要用到的，所以 CANA 在 Case

1 的處理不僅可以縮短測試的路徑，也可以因此減少 TURN Server 的負擔。底下以 Figure 3-4 來說明整個流程。

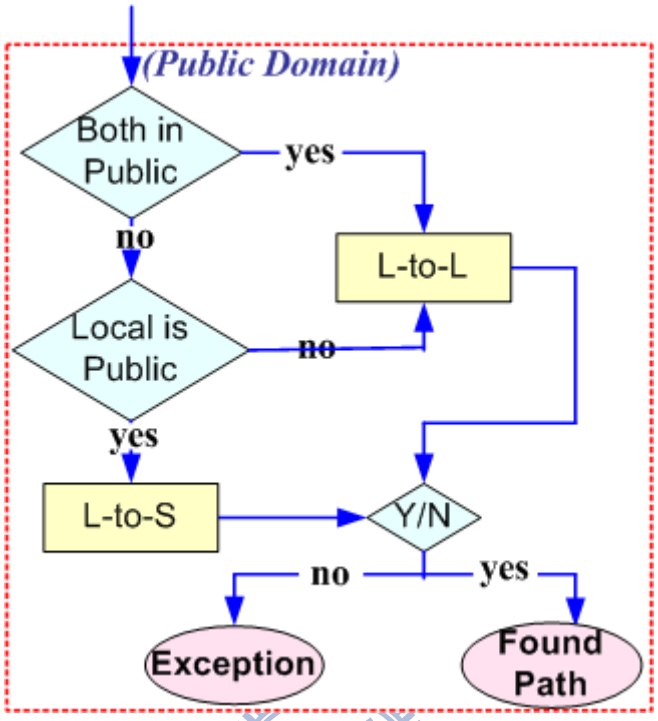

Figure. 3-4 CANA Case1 流程圖

在 Case 1 一開始會先判斷說，要進行連線的雙方使否都是位在 Public Domain，如果是的話那便是上述提及的第一種狀況，反之的話便是上述提及的第二種狀況。

在第一種情況下因為要連線的雙方都是位在 Public Domain，所以雙方都用自己的 Local Address 去和對方的 Local Address 進行連線的測試，而如果沒有例外情況的話那此條路徑便會建立起來，以供後續的使用。

在第二種情況下，一者位於 NAT 底下，一者位於 Public Domain，位於 NAT 底下的將會用自己的 Server Reflexive Address 去和位在 Public Domain 的 Local Address 進行連線測試。所以，便需要去判斷說，到底本身是不是位在 NAT 底下，如果是的話那便是用 Local Address 去和對方的

Local Address 進行連線測試，反之的話，則是用 Local Address 去和對方的 Server Reflexive 進行連線。

3.3.2. CASE 2 (Same NAT)

Case2 的情況為雙方都在同一個 NAT 底下的情況，在此狀況下會有兩種可能性，第一種為兩者都位於同樣一個 NAT 底下，且雙方都可以利用 Private IP 去路由封包到對方，例如 Figure3-5 左圖。第二種為雙方雖然都位於同樣一個 NAT 底下，但是雙方無法藉由 Private IP 路由封包給對方，例如 Figure 3-5 的右圖。

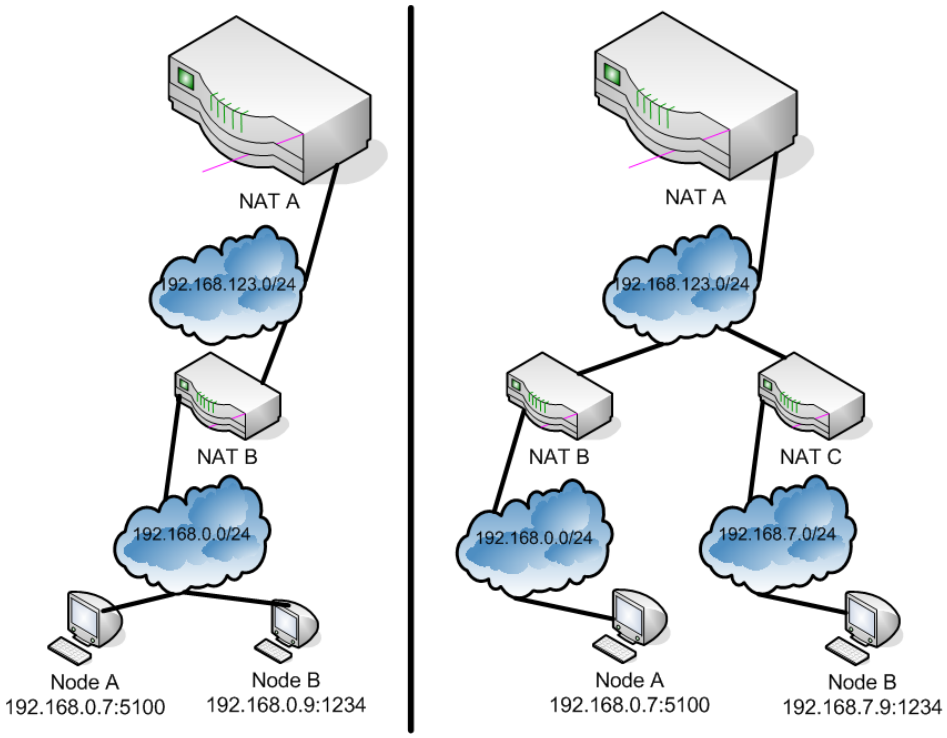

Figure. 3-5 同一 NAT 底下網路拓樸狀況

在 Figure 3-5 情況下左圖為比較友善的網路拓樸狀況，因為 Node A 可以運用 Private IP 路由封包給 Node B，所以雙方如果要進行連線的話

利用 Local Address 來進行連線測試一定可以成功；若是右圖的話 Node A 和 Node B 要是想進行連線的話，利用 Local Address 的話一定會失敗，因為無法利用 Private IP 去路由到對方,在此情況下,次好的情況為 NAT A 有支援 Hairpin 那便可以利用 Server Reflexive Address 來達成直連，否則的話便只能利用 Relay 的方式來幫助雙方建立連線。所以，在 Case 2 的情況之下 CANA 將會測試 1-2 條的路徑。底下以 Figure 3-6 以及 Figure 3-7 來說明 CANA 在 Case 2 下的完整流程

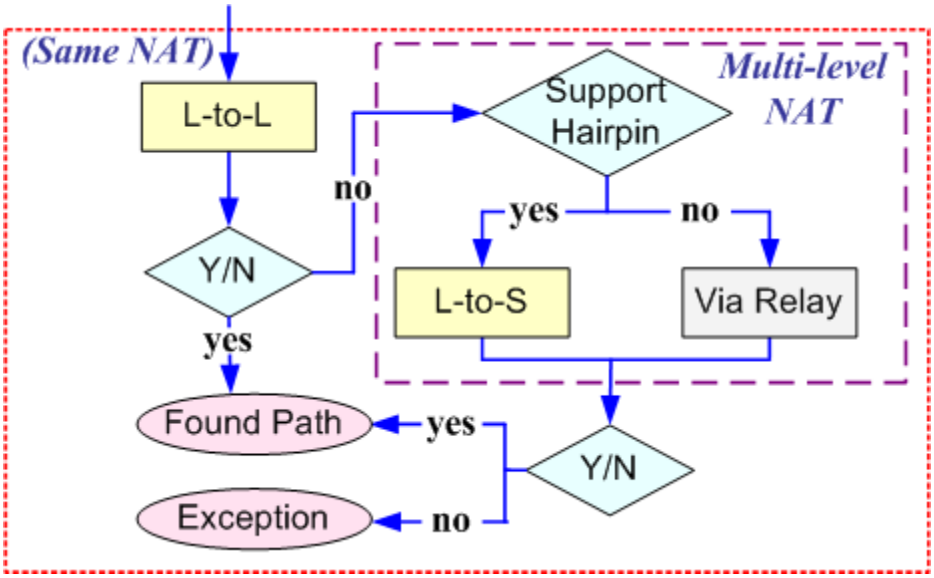

Figure. 3-6 CANA Case2 流程圖

首先一開始在 Case 2 的狀況下，本身會利用 Local Address 和對方的 Local Address 進行連線測試，當測試成功便是找到路徑，否則的話便會進行接下來的動作。在 Local Address 對 Local Address 無法達成直連的狀況下，便會判斷說是否可以利用 Hairpin 的特性來達成另一種方式的直連。所以會判斷最外層的 NAT 是否支援 Hairpin，若果有支援的話本身便可以利用 Local Address 來和對方的 Server Reflexive Address 來達成直連；反之，若是不支援的話，便只能利用 Relay 的方式來讓雙方達成連線

的建立，所以會利用 Via-Relay 的方式來找出所要進行 Relay 的路徑。

要去 Via-Relay 的原因除了要找出 Relay 路徑之外，另一個原因是為了要解決「路徑對稱性」的問題，詳細情況在第二章有說明，在此便不再贅述。CANA 在進行 Via-Relay 時，首先會找出完全 Relay 的方式，也就是本身和遠方端點都必須用 Relay Address 來進行連線，當確定不必使用完全 Relay 的路徑之後，接下來便是選定半邊 Relay 的方式，在此狀況下便是需要決定到底雙方要誰來用 Relay 而誰不用 Relay，而 CANA 的設計機制是由 Callee 來使用 Relay。

我們以 Figure 3-7 來說明決定 Relay 路徑的流程。當 NAT 為 Symmetric NAT 或者是具有 ConnTrack Binding 的特性時，雙方便只能利用完全 Relay 的方式來達成直連。而 CANA 會判斷本身是否為 Caller，若是為 Caller 的話，那便是利用 Local Address 的方式和對方的 Relay Address 進行連線；反之，則是利用 Relay Address 來和對方的 Relay Address 進行連線。

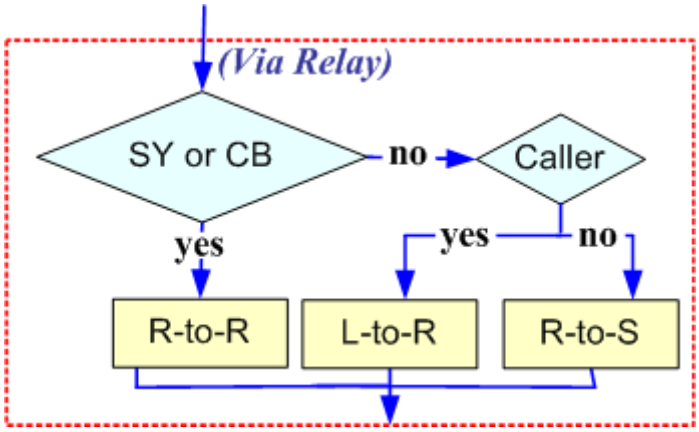

Figure. 3-7 CANA Via-Relay 流程圖

3.3.3. CASE 3 (Same NAT Info)

現實環境中有些 NAT 具有兩個 Interface 各自接到不同的 ISP,因此 NAT 上會有多個 Public IP,而 Case 3 是為了解決此種狀況的 NAT 所延伸出來的情況,因為若果雙方同時位於此種 NAT 底下但因為 Server Reflexive IP 的不同導致判斷為非在同一 NAT 底下,而直接套入 Case 4 的情況便會有無法預測的錯誤產生,因此便有了 Case 3 的產生。

在 CASE 3 的狀況下表示雙方的 NAT 資訊一模一樣,但是這只是表示有可能雙方位於同一個 NAT 底下,而不是肯定就在同一 NAT 底下,所以為了避免會損失掉最佳的路徑測試,於是便直接將 Server Reflexive Address 對 Server Reflexive Address 列入一開始的測試項目,而不是像 Case 2 一樣,判斷完是否有支援 Hairpin 之後才決定是否測試。

以 Figure 3-8 為例子,當 Node A 和 Node B 都具有相同的 NAT 資訊,如果以下圖帶入 Case 2 的話,當然是沒問題可以順利找到最佳路徑;如果以上圖帶入 Case 2 便會找到 Relay 的路徑,但其實雙方是可以達成直連的,假設上圖之 NAT 不是 Symmetric NAT。

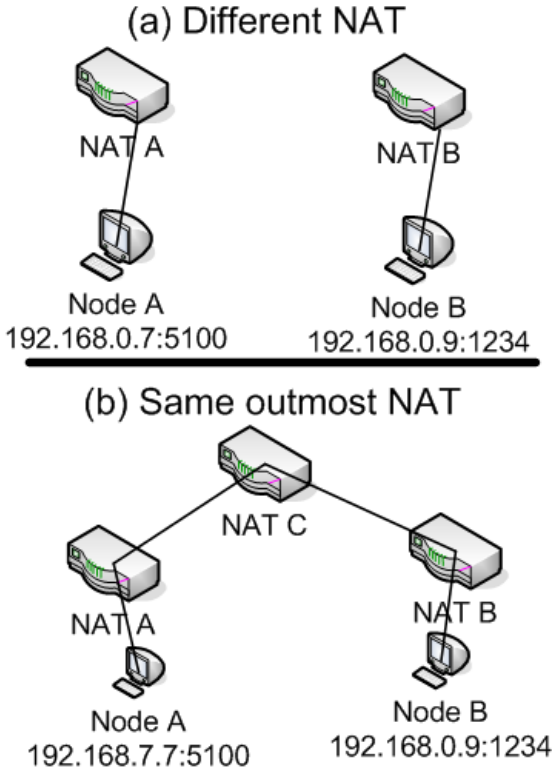

Figure. 3-8 CANA Case 3 兩種網路拓樸但是具有同樣 NAT 資訊

接下來以 Figure 3-9 來說明 CANA 在整個 Case 3 的流程。可以看出 Case 3 和 Case 2 的唯一差別在於說 Case 2 在判斷完是否支援 Hairpin 之後才決定是否要測試 L 對 S 條路徑，而在 Case 3 就直接測試。Via-Relay 請參考 Figure 3-7，在此便不再贅述。

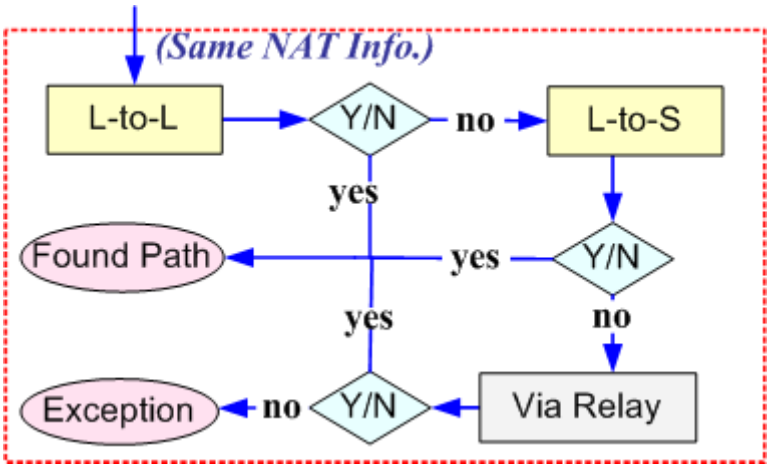

Figure. 3-9 CANA Case 3 流程圖

總結以上，其實 Case 3 的狀況為一種特殊狀況，實際上在實做的時候其實可以衡量是否要實做這部份，不過為了要讓演算法可以有個完整性所以便有 Case 3 的產生。

3.3.4. CASE 4 (Different NAT)

Case 4 的狀況為雙方都在各自不同的 NAT 底下，在此情形下大部分都可以達成直連，所以在這個 Case 就和之前的 Case 2 和 Case 3 最後才找出 Relay 不一樣，在 Case 4 的情況下會先找出 Relay 的路徑，並且在 Case 4 中會有測試的直連路徑是否要開啟 『Waiting』 的機制，而這機制是為了要避免掉 ConnTrack Binding 的問題而可以順利達成直連，ICE 因為沒有這機制而導致直連失敗。

以下以 Figure 3-10 來說明 CANA 在整個 Case 4 的流程圖。首先，決定要選定怎樣的 Relay 路徑。屬於 Symmetric NAT 配對 Symmetric NAT、PR/CB (Port Restricted Cone with ConnTrack Binding) 配對 PR/CB、PR/CB 配對 Symmetric 的狀況，在這些狀況下都只能利用

完全 Relay 來建立連線。

接下來會找出使用半 Relay 的方式,會使用到半 Relay 的方式就只有 Symmetric 配對 Port Restricted Cone 的狀況，在此狀況下應該由位於 Symmetric 下的一方使用 Relay Address 來和另一方的 Server Reflexive Address 進行連線測試。

當判斷為非上述需要 Relay 的狀況的時候，便會進入 Direct Connection (DC) 的狀況，理論上是一定會有直連路徑。這是因為餘下情況不論是 Symmetric 配對 Full Cone 、 Symmetric 配對 Restricted Cone 或是 Cone 對 Cone 的狀況，都是可以達成直連。

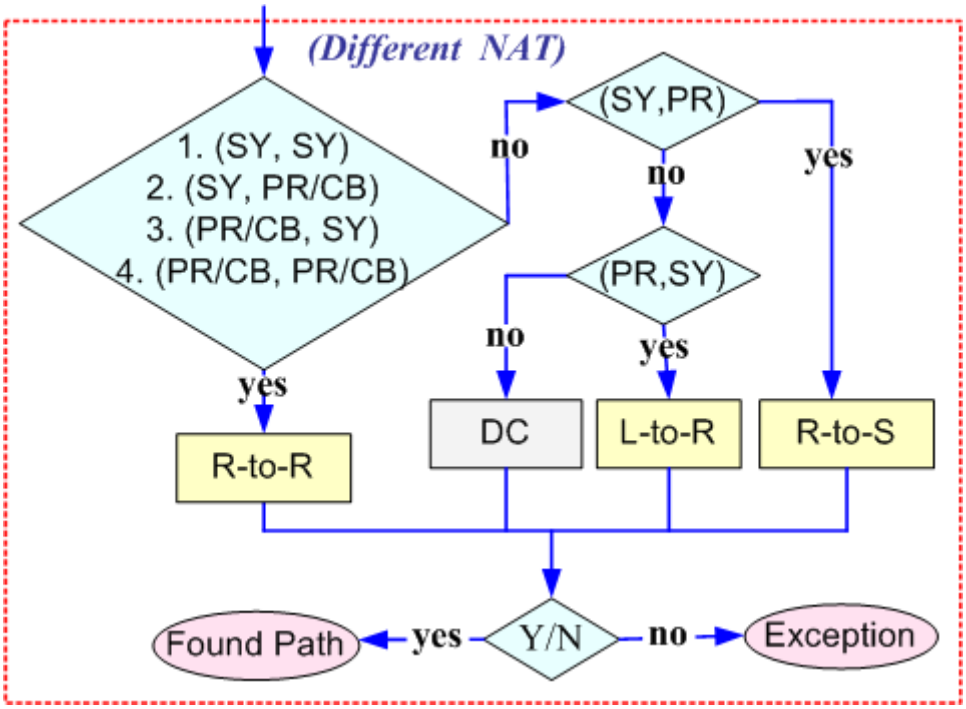

Figure. 3-10 CANA Case 4 基本流程圖

接下來我們以 Figure 3-11 來說明 CANA 在 Case 4 是如何做出直連的選擇。

當確定不要 Relay 之後，便會決定接下來要測試的是一般的直連路徑還是要開啟 「Waiting」 機制的直連路徑。主要會有三種情況會需要啟動「Waiting」的機制：

1. 本身為 Port Restricted Cone 並且有 ConnTrack Binding (PR/CB) 的 NAT，而對方為 Restricted Cone 並且有 ConnTrack Binding (AR/CB) 的 NAT： 在此狀況下，雖然本身的 PR/CB 會被引出 ConnTrack Binding 的特性，但是只會導致 Port 被改變而 IP 還是不變，所以測試封包不會被對方的 Restricted Cone 的 NAT 丟棄，因此可以順利達成直連。

2. 本身為 Port Restricted Cone (PR)，而對方為具有 ConnTrack Binding (CB) 的 NAT： 在此狀況下，應該要由位於 CB 底下的端點先發出測試封包，如此就可以避免具有 CB 的 NAT 被引出 CB 的特性。

3. 本身為 Symmetric (SY)，而對方為 Restricted Cone 並且有 ConnTrack Binding (AR/CB) 的 NAT：在此狀況下，應該要由位於 AR/CB 底下的端點先發出測試封包，如此就可以避免被引出 CB 的特性，而位於 SY 底下的端點發出的測試封包雖然改變了 NAT Mapping，但是只改變 Port，IP 並沒有改變，因此測試封包不會被 AR 丟棄。

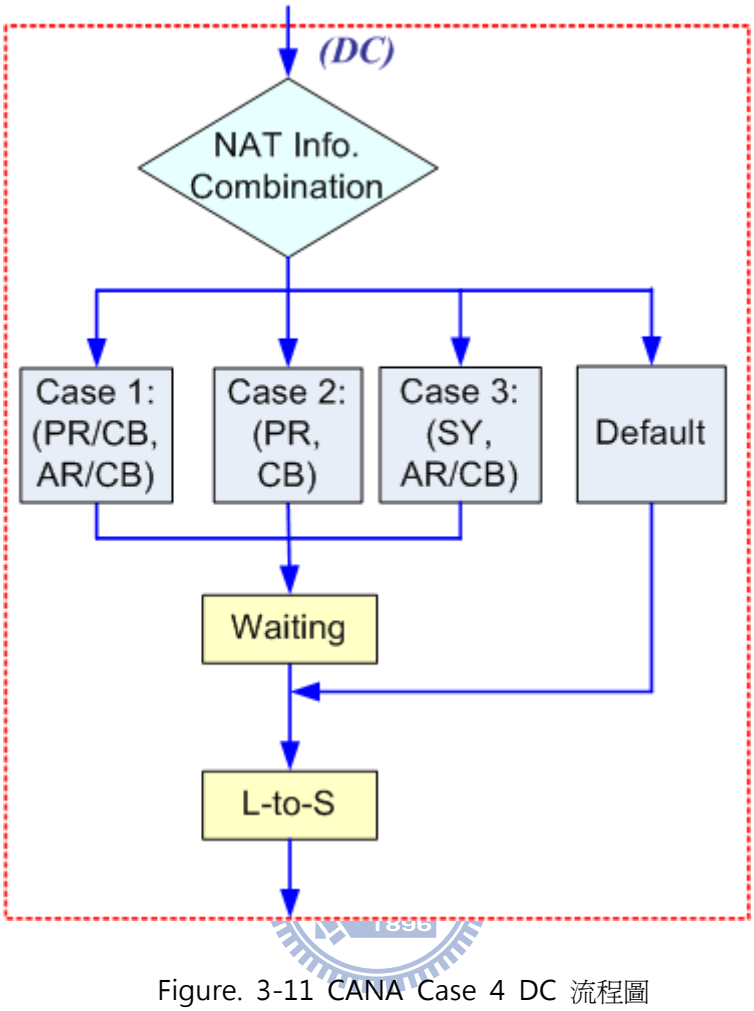

Figure. 3-11 CANA Case 4 DC 流程圖

以下我們以 Figure 3-12 說明在原先 ICE 下，ConnTrack Binding 所造成的問題。

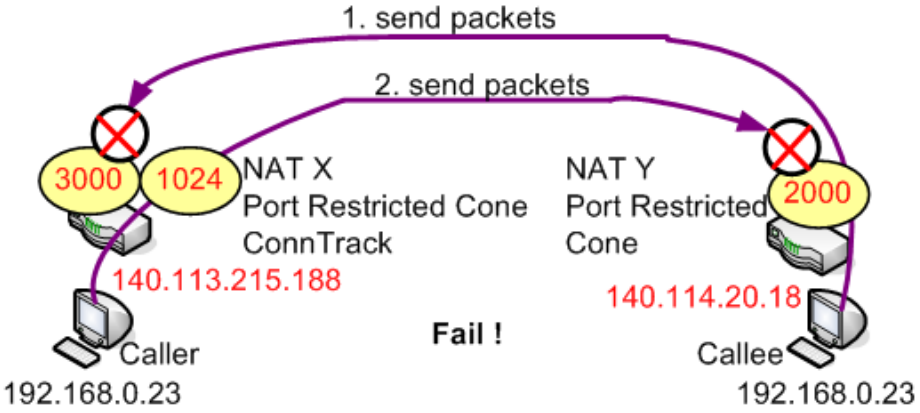

Figure. 3-12 ConnTrack Binding 在 ICE 中 Hole Punching 的問題

在 Figure 3-12 中，caller 和 callee 互相交換完 candidate 之後已經知道互相要測試的 IP/Port 為何，但這時 NAT Y 先送出了測試封包給 NAT X 導致 NAT X 的 ConnTrack Binding 的行為被引出來，所以當 NAT X 邀送出測試封包的時候便會從另外一個 Public Mapping 送出，但是 NAT Y 並沒有為 NAT X 的這個 Public Mapping 有先做任何 Hole Punching 的動作，導致直連的路徑測試失敗。

但若是，我們可以事先讓 NAT X 發出測試封包，就可以避免 NAT X 產生 ConnTrack Binding 的行為，如此便可以順利達成直連，如 Figure 3-13 所示。

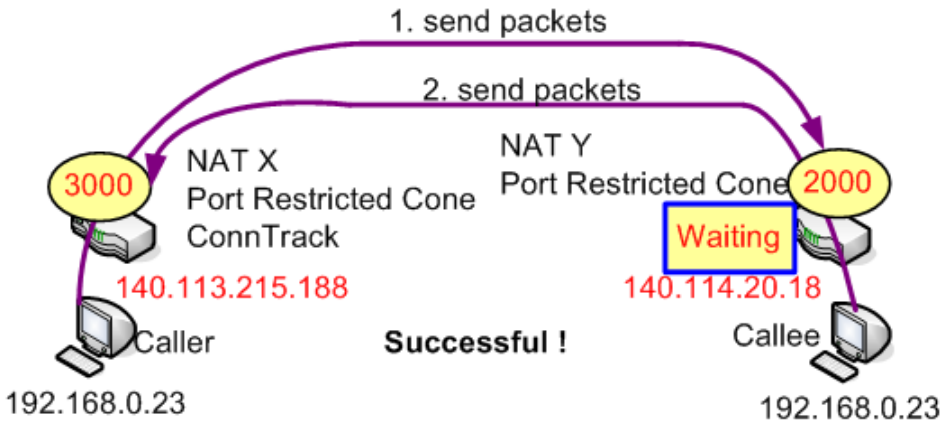

Figure. 3-13 Waiting 機制解決 ConnTrack Binding

第四章 CAN的實做

4.1. 簡介

本系統採用「友訊交大聯合研發中心」所開發的一套「DLINK-ICE」當作主要基底。DLINK-ICE 已經具備了簡單的 UA 和 ICE 功能，在本章節將會簡要描述 SIP UA 和 ICE 之間的運作流程，並且詳細介紹 CAN 的實做過程。

DLINK-ICE 的實做環境和運用到的函式庫，如下表所列：

- 作業系統: Ubuntu 8.04
- SIP Protocol : OSIP2-3.1.0 和 EXOSIP2-3.1.0
- RTP Protocol: ORTP-0.13.1
- TUNR Module: PJNATH-0.8.0

我們運用 OSIP2 和 EXOSIP2 去建構所有的 SIP UA 的 SIP 訊息的處理，並且利用 ORTP 去進行音訊串流的傳送。由於，DLINK-ICE 的 TURN 模組有問題，所以我們以 PJNATH 的 TURN 模組來換掉原先的 TURN 功能。

接下來，我們以 Figure 4-1 來簡要說明整個 ICE 流程的函式架構。

SIPUA 在進行 ICE 的程序之前會先向 ICE 模組（也就是圖中的 ICE Lib）進行一些初始化的動作，之後才可以順利的引用 ICE 模組所提供的功

能。圖中的左邊的 SIP UA 在要進行播打電話的時候會進行一項「addCandiToSDP」的動作，此動作得主要目的為收集 ICE 中的 3 種 Candidate 並且 將此項資訊包裹成 SDP 的模式交給 SIP Protocol 傳送給受話方。這項動作將會需要使用到 ICE 模組中的收集 Candidate 的功能。

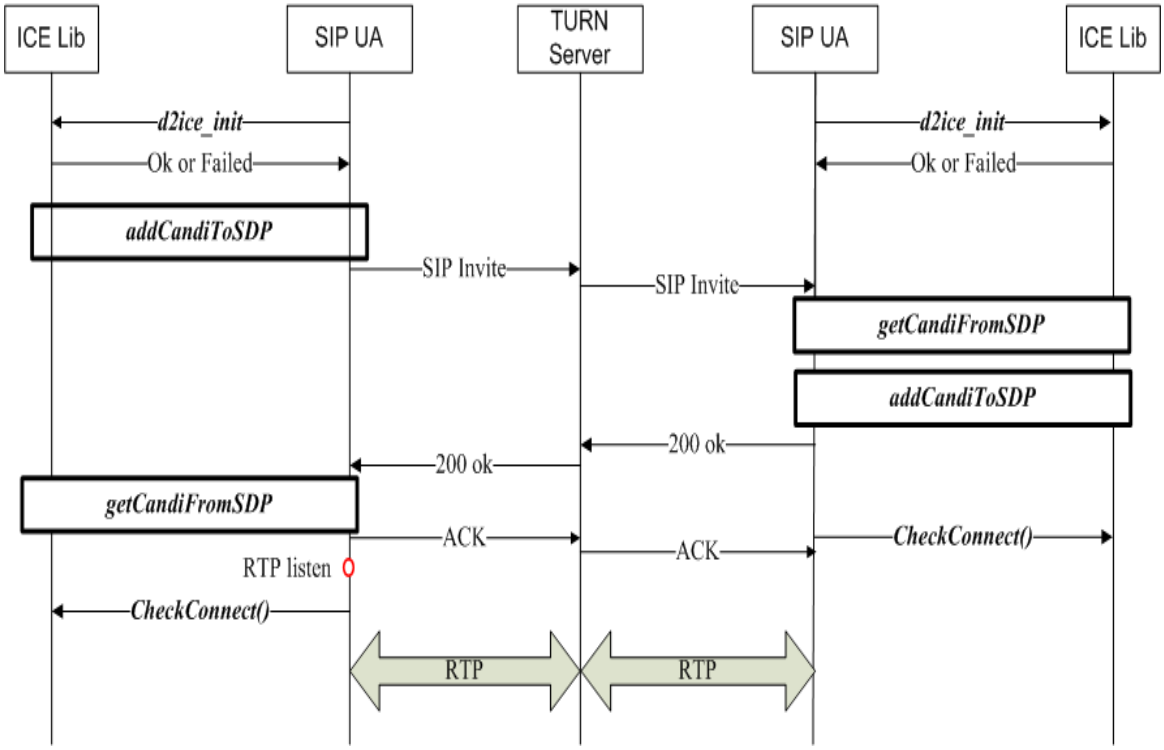

Figure. 4-1 DLINK-ICE 流程圖

緊接著當受話方，也就是圖中右邊的 SIP UA，接受到通話要求的訊息 SIP Invite 之後，會先啟動「getCandiFromSDP」這項動作，將 SIP 訊息中有關發話方的 Candidate 資訊收集起來，並且自身也再起動「addCandiToSDP」這項動作，將自己本身的 Candidate 資訊收集起來並且傳送給發話方。

當發話方和受話方都擁有了彼此的 Candidate 資訊之後，便會使用 ICE 模組中的 「CheckConnect」的動作，去進行連線測試，當測試完成之後便會使用可用路徑中的最好路徑來傳送音訊串流。

以上為，一個 ICE 的簡單實做流程說明，下一節將會說明 CAN 是如何實做上去與原有的 ICE 做結合。

4.2. 實做上的方法

CAN 最重要的兩大核心部份為 NAT 資訊的偵測，以及測試連線路徑的演算法。在 NAT 資訊測試方面我們採用 STUN 所制定的演算法來實做，此套演算法可以偵測出 NAT 的型別，而且在實做上稍稍加以改變便可以順便偵測出 Hairpin 的行為，但是 ConnTrack Binding 這項行為卻尚未有相關方法提及檢驗方式，所以在 4.2.1 小節將會詳述，我們時如何加上這項偵測功能。

我們將原先 DLINK-ICE 中實做 STUN 演算法的 STUN Lib 來當作基底，並且加上 ConnTrack Binding 的行為偵測，而這整套的模組我們稱為『NAT Check』模組。

測試連線路徑的實做方式，將會在 4.2.2 小節加以詳細說明，基本上主要的架構就是 CANA 演算法。

4.2.1. ConnTrack Binding 偵測

我們以 Figure 4-2 來說明整個 ConnTrack Binding 的測試流程。

第一步，向 STUN Server 的綁定的四個 Socket 中的第一個進行詢問 Public Mapping 的動作，此動作將會導致 NAT 開啟了一個 Mapping 我們稱為 Mapping1。

接著，一樣向 STUN Server 的第一個 Socket 發送 Request，不同於第

一步，這次的 Request 將會要求 STUN Server 從另外一個 IP 和 Port 回傳此 Response。到此，如果此 NAT 有 ConnTrack Binding 的行為的話，那麼接下來將會發生 Mapping 改變的狀況，否則的話就是沒有 ConnTrack Binding 這項行為。

最後，就是在詢問一次 Mapping，我們稱為 Mapping2，如果 Mapping2 和 Mapping1 不一樣的話，那麼顯然的就是有 ConnTrack Binding 的特性存在。

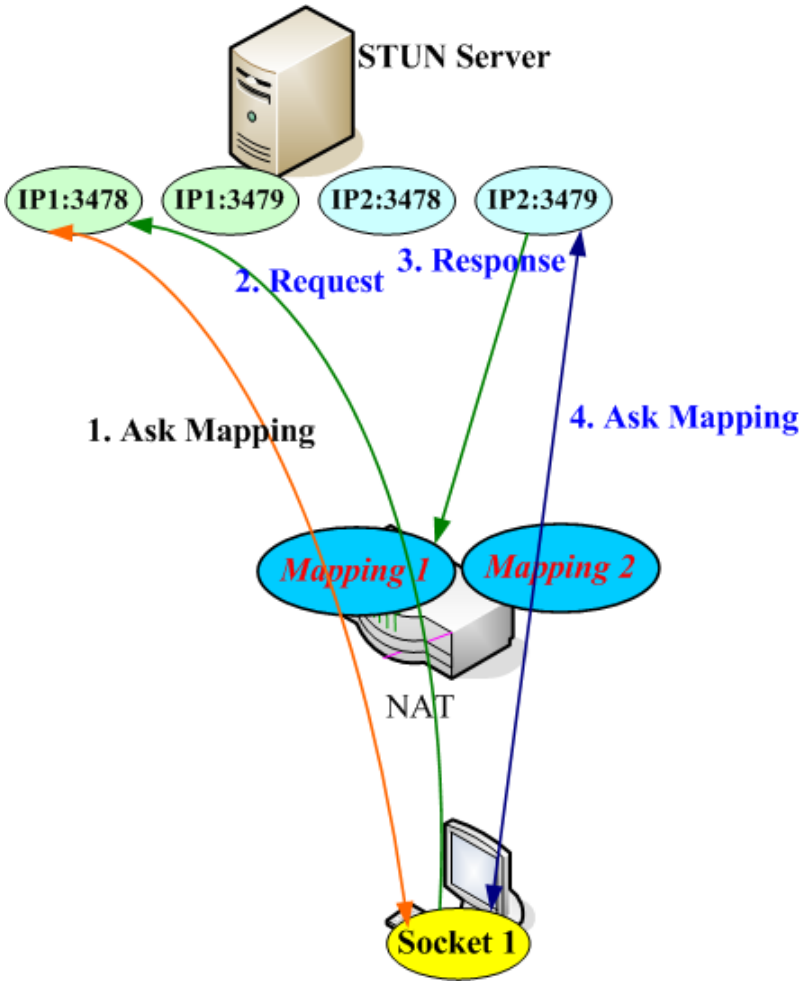

Figure. 4-2 ConnTrack Binding 測試流程

最後，我們以 Figure 4-3 來說明整個 NAT Check 模組的架構為何。

當 SIP UA 想要獲得當下的 NAT 資訊，可以利用 「getNATInfo」這個函式呼叫 NAT Check 模組，NAT Check 模組將會利用 STUN Lib 的功能去獲得 NAT 的型別和是否有支持 Hairpin 的特性，當 NAT Check 模組察覺到 NAT 型別為 Restricted Cone 或者為 Port Restricted Cone，將會在啟動 ConnTrack Binding 的特性偵測。

最後，NAT Check 會將所有的 NAT 資訊回傳給 SIP UA。

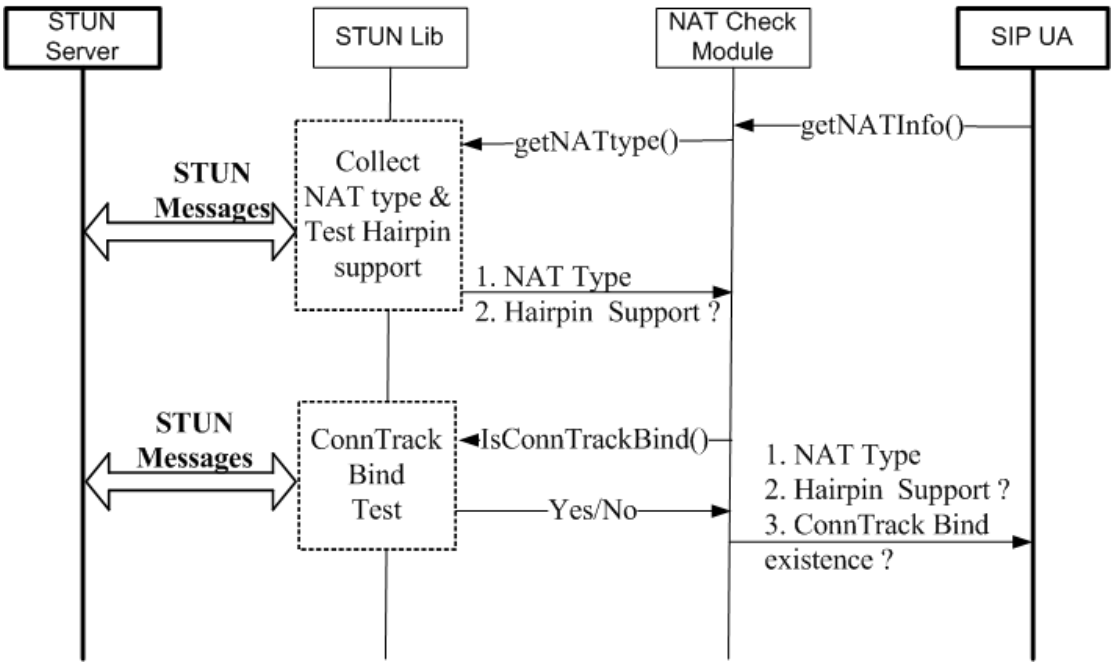

Figure. 4-3 NAT Check 架構

4.2.2. CANA 實做

CANA 的實做主要是參照 CANA 演算法下去實做，並且將新的 CheckConnect 稱為『CAN_CheckConnect』。

有關演算法細節請參考第三章。

4.3. Implementation Library 介紹

這一小節將會介紹，實做的簡易 ICE 模組。將會分兩部份介紹，第一部份為原本的 ICE 流程，在第二部份將會介紹 CAN 的流程。

首先,我們介紹原本的 ICE 流程。要注意的是,我們這邊定義的 ICE 流程已經不單單止於 SIP Protocol，任何可以交換 Candidate 的通訊協定都是可以被接受的。

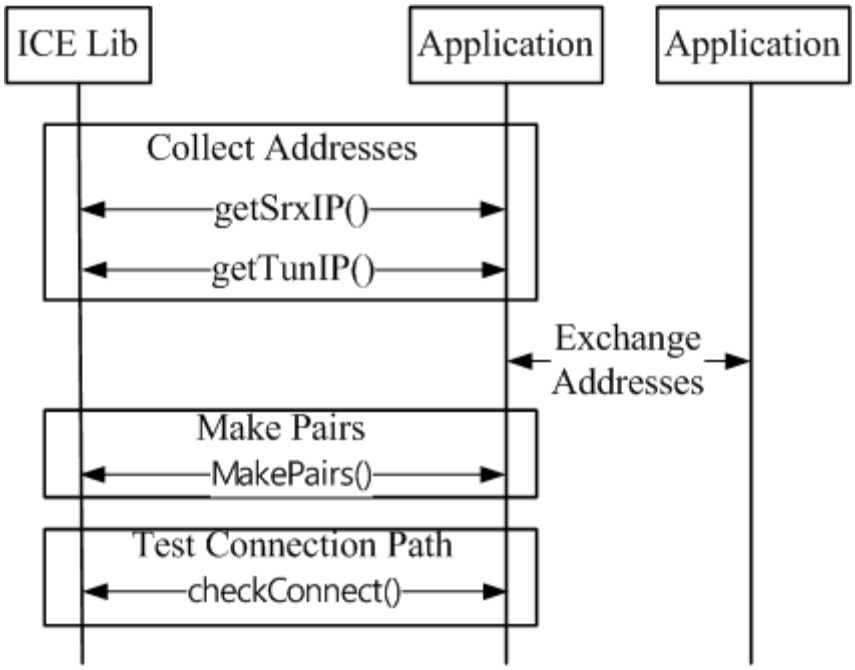

Figure. 4-4 Original ICE 函式流程

如 Figure 4-4 所示，在我們設計出來的簡易 ICE Library 中，其實只有三個功能面。

首先就是收集 Candidate 也就是圖中的 Collect Addresses，其中的 getSrxIP 為收集 NAT 上 Public Mapping，而 getTunIP 則是收集 TURN Relay 位址，當這些 Candidate 都被應用程式擁有的時候，應用程式可以運

用任何通訊協定去和遠方想進行連線的端點交換這些資訊，可以是 SIP 也可以是其他方式。

而當交換完 Candidate 之後，便可以利用 MakePairs 的組合路徑，並且將這些路徑交由 checkConnect 進行測試，以便找出可用且最好的路徑。

接下來，以 Figure 4-5 說明 CAN 的函式流程。

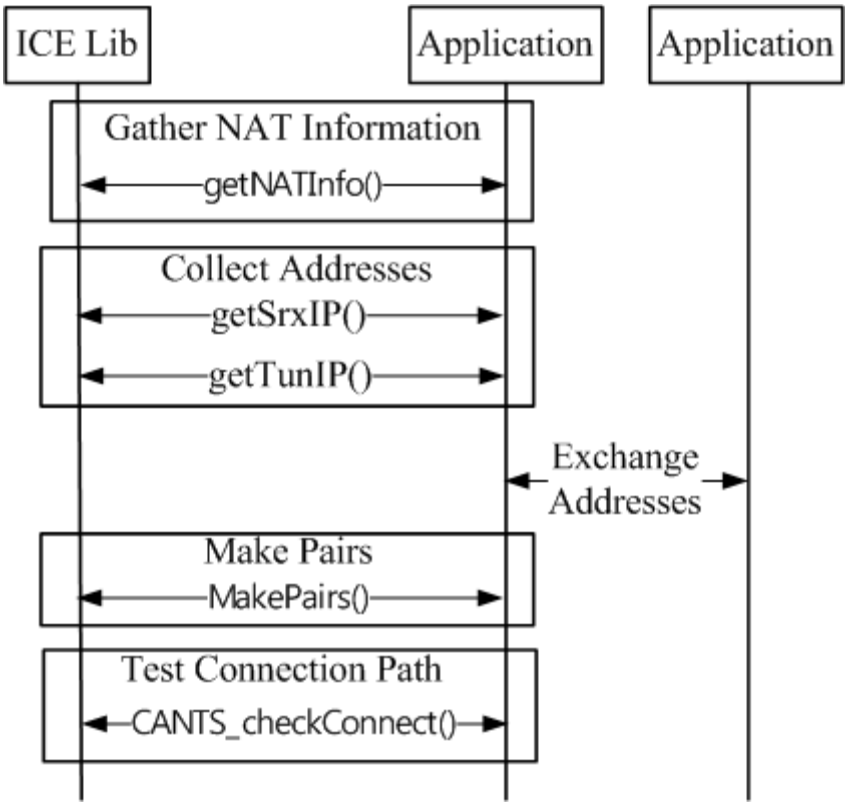

Figure. 4-5 CAN 函式流程

Figure 4-4 最大的不同為，Figure 4-5 多了收集 NAT 資訊的這項動作，由於 CAN 的機制需要用到 NAT 的資訊，所以在一開始應用程式便需要去收集 NAT 的資訊，並且在交換 Cnadidate 這項步驟的時候一併交換 NAT 資訊。

而在最後進行連線測試的時候，便呼叫 CAN_checkConnect 來進行

路徑的測試。

以上，便是實做之後的簡易版 ICE 功能函式庫，可以讓使用者更加快速使用的一套 NAT Traversal 的函式庫。

4.4. 單元總結

總結以上，我們詳述了整個 CAN 的實做內容，並且將函式庫的使用方式簡單的介紹了一下。

當然此套函式庫並不只是可以用於 CAN 這套 NAT Traversal 的技術，也可以單獨取用其中的函式來進行使用，例如可以單讀取用 getNATInfo 這函式來獲得想要的 NAT 資訊。

第五章 實機測試與驗證

5.1. 驗證環境介紹

在驗證 CAN 與 ICE 的差別點，我們啟用 17 台 NAT 來進行測試比較。而我們的環境示意圖，如下圖 Figure 5-1。

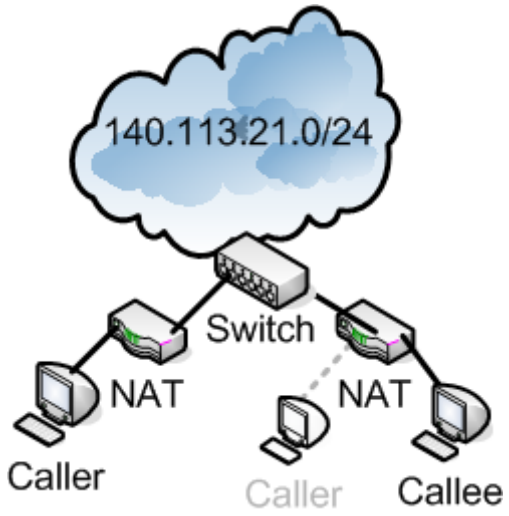

Figure. 5-1 測試環境示意圖

Caller 和 Callee 位在總共 17 台 NAT 底下，並且可以隨時切換到指定的 NAT 底下，而這 17 台 NAT 的 Public 位置都接到 140.113.21.0/24 這網域下。

測試的過程，將會是 Caller 播打電話給 Callee 並且在進行完路徑測試之後，Caller 和 Callee 將會在找到的路徑上面各自收發一段音訊串流檔案，以 RTP 通訊協定傳送。

實驗環境的 17 台 NAT，總計有 2 台 Full Cone、4 台 Restricted

Cone、6 台 Port Restricted Cone、5 台 Symmetric。我們下面以表格列出各個 NAT 的廠牌以及其具有的特性。

表格 5-1 Full Cone NAT 列表

Full Cone		
Brand	ConnTrack	Hairpin
D-Link	No	Yes
SMC	No	Yes

表格 5-2 Restricted Cone NAT 列表

Restricted Cone		
Brand	ConnTrack	Hairpin
Linksys	Yes	No
Corega	No	No
PCI	No	No
SMS	No	Yes

表格 5-3 Port Restricted Cone NAT 列表

Port Restricted Cone		
Brand	ConnTrack	Hairpin
BelKin	No	No
Draytek	No	Yes
Edimax	Yes	No
Lemel	No	No
Linux	Yes	No
PCI (W)	No	No

表格 5-4 Symmetric NAT 列表

Symmetric		
Brand	ConnTrack	Hairpin
Abocom	No	No
Asus	No	No
Netgear	No	No
Zyxel	No	No
Freebsd	No	No

由上述四個表格可知，總共有三台 NAT 具有 ConnTrack Binding 的特性，一台為 Restricted Cone、兩台為 Port Restricted Cone。總共四台具有 Hairpin 的特性，兩台為 Full Cone、一台為 Restricted Cone、一台為 Port Restricted Cone。

5.2. 驗證項目

在本測試環境中主要測試項目為三個，一個為直連率，一個為延遲時間，一個為連線測示訊息量，測試步驟如下：

1. Caller 切換到 NAT1 底下
2. Callee 切換到 NAT1 底下
3. Caller 向 Cellee 發出發話通知，並且開始一連串 ICE 的動作，而程式將會將 Caller 和 Callee 的訊息給記錄下來存成 LOG 檔案，而且也會把雙方連線過程的網路封包給測錄下來。
4. 當完成一次測試之後，Callee 會切換到 NAT2 而 Caller 一樣維持在 NAT1。並且再進行一次步驟 3。如此，一直到 Caller 切換到

NAT17，才換完成一輪測試。

5. 當 Callee 完成一輪的 NAT 切換，Caller 將會切換到下一台 NAT，一直到 Callee 切換到 NAT 17 才會完成整個測試流程。

總計以上步驟，總共會有 17 X 17 種測試狀況，所以會有 289 種測試狀況。

5.2.1. Direct Connection Rate 直連率

直連率數據方面，總共會有 17 X 17 – 17，計有 272 個狀況，之所以忽略 17 種是因為這 17 種為 位在同一個 NAT 底下，在數據上表現是一定會成功達成直連的，是故沒有比較價值。

表格 5-5 16 種 NAT type 組合

	Full Cone	Restricted Cone	Port Restricted Cone	Symmetric
Full Cone	1	2	3	4
Restricted Cone	5	6	7	8
Port Restricted Cone	9	10	11	12
Symmetric	13	14	15	16

如表格 5-5 所示，我們根據 Caller 和 Callee 的 NAT 型別的配對，把情況分為 16 種狀況，表格 5-5 的直排為 Caller 的 NAT 型別，橫列為 Callee 的 NAT 型別。

我們將這 16 種狀況各自佔有的 Case 以及其比率以表格 5-6 來表示。由表格 5-6 可知，Case7、Case8、Case10、Case11、Case12、Case14、Case15、Case16 佔有比較大的比重。

表格 5-6 16 種 NAT type 組合各自所佔比例

	Full Cone	Restricted Cone	Port Restricted Cone	Symmetric
Full Cone	2	8	12	10
	0.74%	2.94%	4.41%	3.68%
Restricted Cone	8	12	24	20
	2.94%	4.41%	8.82%	7.35%
Port Restricted Cone	12	24	30	30
	4.41%	8.82%	11.03%	11.03%
Symmetric	10	20	30	20
	3.68%	7.35%	11.03%	7.35%

接下來我們以表格 5-7 來表示 CAN 在這 16 種狀況下的直連率。

表格 5-7 16 種 NAT type 組合下 CAN 直連率

	Full Cone	Restricted Cone	Port Restricted Cone	Symmetric
Full Cone	100.00%	100.00%	100.00%	100.00%
Restricted Cone	100.00%	100.00%	100.00%	100.00%
Port Restricted Cone	100.00%	100.00%	93.33%	0.00%
Symmetric	100.00%	100.00%	0.00%	0.00%

再來我們以表格 5-8 來表示 ICE 在這 16 種狀況下的直連率。

表格 5-8 16 種 NAT type 組合下 ICE 直連率

	Full Cone	Restricted Cone	Port Restricted Cone	Symmetric
Full Cone	100.00%	100.00%	100.00%	100.00%
Restricted Cone	100.00%	100.00%	75.00%	75.00%
Port Restricted Cone	100.00%	100.00%	66.67%	0.00%
Symmetric	100.00%	100.00%	0.00%	0.00%

從表格 5-7 以及表格 5-8 可以很清楚知道，CAN 和 ICE 的差距在於，Case7、Case8 以及 Case11。在這三個 Case 的情況下，可以發現 CAN 是優於 ICE 的。而這主要的原因在於，ConnTrack Binding 的問題在 CAN 被

解決了，但在 ICE 還是依然存在這問題。

若是以整體的直連率而言，69.85% 明顯優於 ICE 的 62.86%。CAN 以 190 個成功 Case 優於 ICE 的 171 個。

最後以表格 5-9 和表格 5-10 來表示全部的直連狀況。

表格 5-9 ICE 直連率整體表現

	D-Link	SMC	Linksys	Corega	PCI	SMS	Belkin	Draytek	Edimax	Lemel	Linux	PCI(W)	Abocom	Asus	Netgear	Zyxel	Freebsd
D-Link		★	★	★	★	★	★	★	★	★	★	★	★	★	★	★	★
SMC	★		★	★	★	★	★	★	★	★	★	★	★	★	★	★	★
Linksys	★	★		★	★	★											
Corega	★	★	★		★	★	★	★	★	★	★	★	★	★	★	★	★
PCI	★	★	★	★		★	★	★	★	★	★	★	★	★	★	★	★
SMS	★	★	★	★	★		★	★	★	★	★	★	★	★	★	★	★
Belkin	★	★	★	★	★	★		★	★	★	★	★					
Draytek	★	★	★	★	★	★	★		★	★	★	★					
Edimax	★	★	★	★	★	★											
Lemel	★	★	★	★	★	★	★	★	★		★	★					
Linux	★	★	★	★	★	★											
PCI(W)	★	★	★	★	★	★	★	★	★	★	★						
Abocom	★	★	★	★	★	★											
Asus	★	★	★	★	★	★											
Netgear	★	★	★	★	★	★											
Zyxel	★	★	★	★	★	★											
Freebsd	★	★	★	★	★	★											

表格 5-10 CAN 直連率整體表現

	D-Link	SMC	Linksys	Corega	PCI	SMS	Belkin	Draytek	Edimax	Lemel	Linux	PCI(W)	Abocom	Asus	Netgear	Zyxel	Freebsd
D-Link		★	★	★	★	★	★	★	★	★	★	★	★	★	★	★	★
SMC	★		★	★	★	★	★	★	★	★	★	★	★	★	★	★	★
Linksys	★	★		★	★	★	★	★	★	★	★	★	★	★	★	★	★
Corega	★	★	★		★	★	★	★	★	★	★	★	★	★	★	★	★
PCI	★	★	★	★		★	★	★	★	★	★	★	★	★	★	★	★
SMS	★	★	★	★	★		★	★	★	★	★	★	★	★	★	★	★
Belkin	★	★	★	★	★	★		★	★	★	★	★					
Draytek	★	★	★	★	★	★	★		★	★	★	★					
Edimax	★	★	★	★	★	★	★	★		★	★	★					
Lemel	★	★	★	★	★	★	★	★	★		★	★					
Linux	★	★	★	★	★	★	★	★	★	★		★					
PCI(W)	★	★	★	★	★	★	★	★	★	★	★						
Abocom	★	★	★	★	★	★											
Asus	★	★	★	★	★	★											
Netgear	★	★	★	★	★	★											
Zyxel	★	★	★	★	★	★											
Freebsd	★	★	★	★	★	★											

5.2.2. Delay Time 延遲時間

延遲時間數據方面，總共會有 17 X 17，計有 289 種狀況，我們依然依照，表格 5-5 ，來做分析。

以表格 5-11 和 表格 5-12 表示 ICE 的 Caller 和 Callee 進行連線測試的延遲時間(單位為秒)。

表格 5-11 ICE Caller 延遲時間

	Full Cone	Restricted Cone	Port Restricted Cone	Symmetric
Full Cone	4.10s	4.28s	4.26s	4.23s
Restricted Cone	4.266s	4.19s	4.32s	4.34s
Port Restricted Cone	4.24s	4.29s	4.33s	4.08s
Symmetric	4.18s	4.28s	4.31s	4.07s

表格 5-12 ICE Callee 延遲時間

	Full Cone	Restricted Cone	Port Restricted Cone	Symmetric
Full Cone	4.33s	4.46s	4.45s	4.50s
Restricted Cone	4.50s	4.50s	4.50s	4.45s
Port Restricted Cone	4.50s	4.50s	4.50s	4.30s
Symmetric	4.50s	4.49s	4.49s	4.08s

總和以上，ICE 在 Caller 會有平均 4.24s 的延遲時間，而 Callee 又會比 Callee 慢個 0.2s 左右，大約是 4.44s。這是因為從 ICE 的流程當中，Callee 會是最晚結束連線測試的一端。

以表格 5-13 和 表格 5-14 表示 CAN 的 Caller 和 Callee 的延遲時間。

表格 5-13 CAN Caller 延遲時間

	Full Cone	Restricted Cone	Port Restricted Cone	Symmetric
Full Cone	0.01s	0.03s	0.01s	0.2s
Restricted Cone	0.04s	0.01s	0.01s	0.2s
Port Restricted Cone	0.03s	0.01s	0.01s	0.01s
Symmetric	0.01s	0.01s	0.01s	0.01s

表格 5-14 CAN Callee 延遲時間

	Full Cone	Restricted Cone	Port Restricted Cone	Symmetric
Full Cone	0.32s	0.35s	0.33s	0.47s
Restricted Cone	0.36s	0.34s	0.33s	0.48s
Port Restricted Cone	0.35s	0.34s	0.34s	0.34s
Symmetric	0.33s	0.34s	0.33s	0.34s

總和以上，CAN 在 Caller 會有平均 0.03s 的延遲時間，而 Callee 又會比 Callee 慢個 0.3s 左右，大約是 0.35s。

在第三章有提到，為了解決 ConnTrack Binding 的問題，所以啟用了『Waiting』的機制，此舉導致了在有些情況下 CAN 在 Caller 和 Callee 各會有 1.67s 和 2.00s 的延遲。如表格 5-15 以及 表格 5-16 灰色部份所示。

表格 5-15 Waiting 機制導致的 Caller 的延遲

	Full Cone	Restricted Cone	Port Restricted Cone	Symmetric
Restricted Cone (ConnTrack Binding)	0.01s	0.01s	1.62s	1.69s
Port Restricted Cone(ConnTrack Binding)	0.01s	0.01s	1.69s	0.01s

表格 5-16 Waiting 機制導致的 Callee 的延遲

	Full Cone	Restricted Cone	Port Restricted Cone	Symmetric
Restricted Cone(ConnTrack Binding)	0.34s	0.34s	2.00s	2.00s
Port Restricted Cone(ConnTrack Binding)	0.33s	0.34s	2.01s	0.34s

我們以 Figure 5-2 和 Figure 5-3 來表現 CAN 和 ICE 在 Delay

Time 的差異性，可明顯看到兩著有蠻大的差距，CAN 明顯優於 ICE。

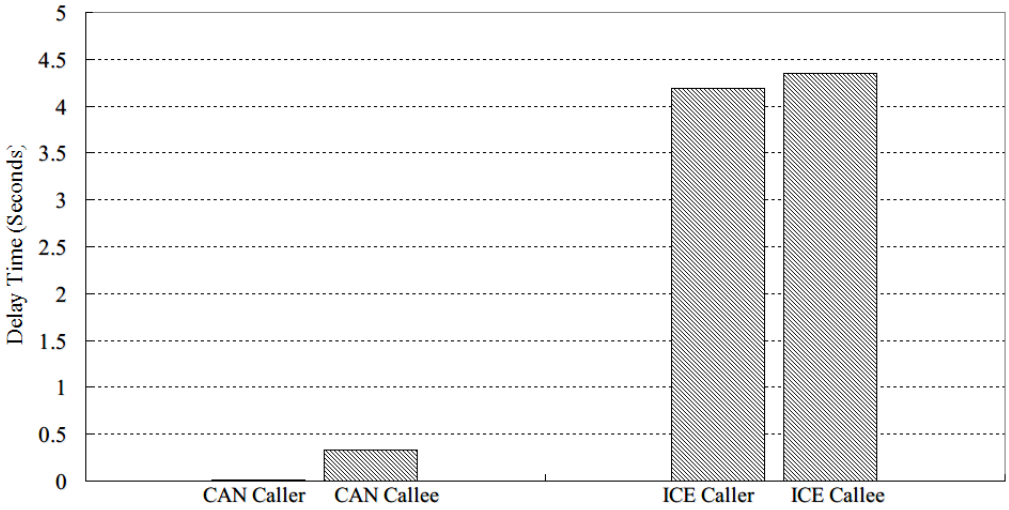

Figure. 5-2 CAN and ICE Delay Time under different NATs

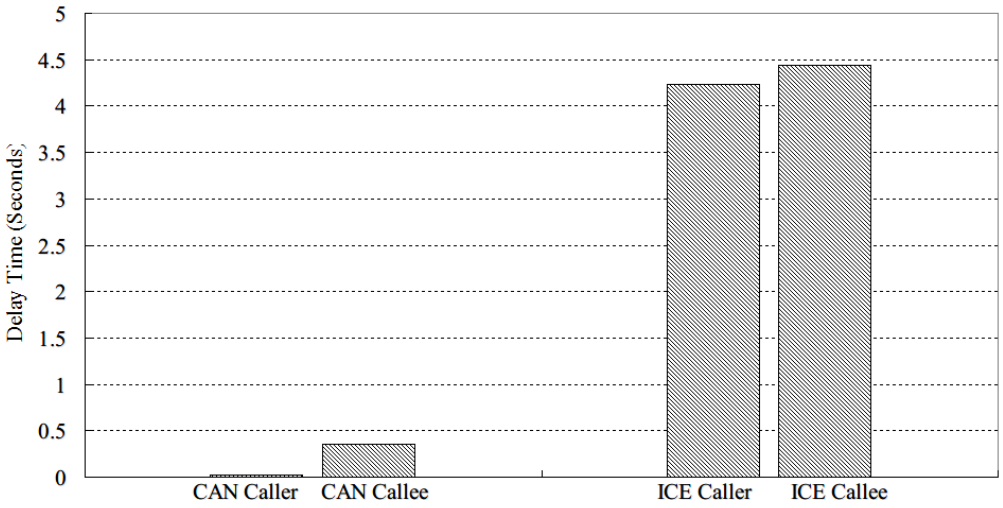

Figure. 5-3 CAN and ICE Delay Time under same NAT

ICE 在延遲時間會高達 4~5 秒主要是因為在實做上對於測試封包的重複傳送，這數值可以經由調整而變小，不過調整之後還是會介於 1~2 秒之間；這是為了要判斷一個 UDP 封包的遺失是因為 NAT 的問題還是單純因為封包的遺失。

我們以 Figure 5-4 來表現 CAN 啟用 Waiting 機制和 ICE 在 Delay Time 的差異性，可明顯看到 CAN 明顯優於 ICE。這是因為 CAN 在一開始就啟用了等待機制解決問題，但是 ICE 卻是在連線失敗之後才啟用等待機制，造成了延遲時間的堆疊，當然 ICE 也可以利用平行化測試的方式避免時間的堆疊，但是還是會有訊息量多的問題，而且因為 ICE 的測試連線的 State Machine 的關係，要融合等待機制和 ICE 進行平行化測試將會提高程式開發者的設計難度。

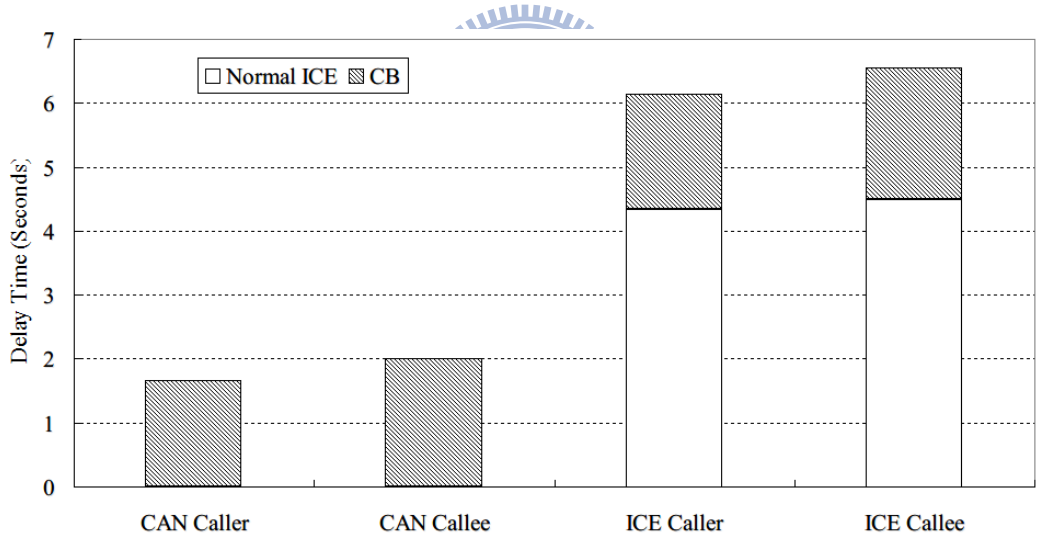

Figure. 5-4 CAN (Waiting) and ICE Delay Time in Caller

總結以上，CAN 有效的降低了連線測試所造成的延遲時間，並且保證找到的路徑為當時網路拓撲狀況下的最佳路徑。

5.2.3. Connection Test Message 連線測試訊息量

ICE 和 CAN 在測試路徑的時候，皆是以 STUN Messages 來當作測試封包，在測試中，一條測試失敗的路徑將會產生 6 個 STUN Messages，而測試成功的路徑將會有 4 至 6 個 STUN Message 依照當時 NAT 的組成而定。

我們將 CAN 和 ICE 在上述的 289 種測試種類下所產生的 STUN Messages 進行加總而成 Figure 5-5。

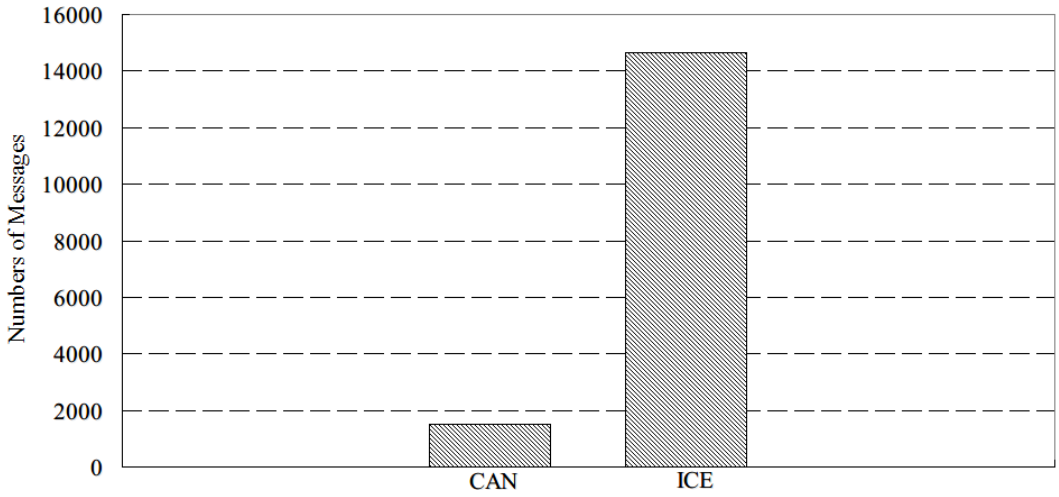

Figure. 5-5 CAN (Waiting) and ICE Delay Time in Callee

由 Figure 5-5 可以清楚看到 CAN 所產生的 STUN Message 明顯低於 ICE。

第六章 結論與未來工作

6.1. 結論

在本篇論文中，我們針對 NAT 所造成 Peer-to-Peer 的連線問題，提出一套有效率且高直連率的解決機制；有了 ICE 的系統化測試的優點，更避免了 ICE Delay time 的問題，也解決了現實世界中 NAT Traversal 存在的一些問題，因此而增加了直連率。綜述而言，本篇論文中所探討的這套機制是以 UDP 這個 Transport Protocol 為主，希望可以讓 UDP 的 Peer-to-Peer 建立連線更加簡單的有效率，而且可以盡量避免掉 Relay 的使用。

我們提出的這套機制是採取一種分散式的 NAT 資訊運用，NAT 資訊是由所要進行連線的雙方進行交換和持有，而不是跟 CDCS 是一種中央伺服器管理的方式，因此本方法將不會額外增加伺服器的負擔。首先，我們讓應用程式在一執行的時候便啟動 NAT Check 收集 NAT 資訊，而這收集到的資訊便會被應用程式所儲存起來，每當嘗試要進行 P2P 連線的時候才會被應用程式取出來進行使用，因為收集 NAT 資訊的動作和進行 P2P 連線的動作在時序上是分開的，所以並不會因此而增加了 P2P 連線的 Latency。

另外本論文也針對如何使用 NAT 資訊提出一套演算法，本演算法完善的使用 NAT 資訊，並且根據 ICE 的九條路徑選出最適當地路徑進行連線測試；因為這些 NAT 資訊的幫助，此套機制可以很清楚的掌握這一次連線

測試的狀況。本機制可以知道說，這一條路徑的失敗是因為實際上 NAT 的無法連通，還是因為測試路徑的 UDP 封包的遺漏（UDP 封包在網路上會有遺漏的狀況）。

在本論文的第四章和第五章，將本方法的實做成品以及測試結果提出了報告，藉此可以驗證我們這套機制確實可以有效的優於 ICE 原先的表現。並且有效的解決了 ConnTrack Binding 所造成的問題，藉此提高了不小的直連率。

6.2. 未來工作

未來，我們希望可以將本機制導入 TCP，而不是讓本方法單純的運用於 UDP。在 TCP 本機制的導入首先要克服的第一個問題在於 TCP 的 NAT 資訊取得不易，現有的模組在測試 NAT 的 TCP 特性往往會花費不少時間，和 UDP 的快速收集到資訊有明顯的差距，所以如果要導入 TCP 的話這會是一個不小的問題。

另外，在實做過程中發現有些 NAT Device 的 NAT type 會變換有不穩的現象，推測是 NAT 本身的不穩定，還好有問題的機器才一兩台，所以建議本方法的 NAT Check 可以每隔幾分鐘再測試一次，讓收集到的 NAT 資訊可以是最符合當下的 NAT 狀態。

最後，我們希望所收集到可用的 NAT 資訊可以越來越多，或許還有些特性是我們未發現的；而且也可以將其他的 NAT Traversal 方法融入本方法，例如將 Port-Prediction [14] 導入，如此可以再將直連率進行提昇，也是一個未來可行的方式。

參考文獻

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

[2] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. de Groot, Address Allocation for Private Internets, RFC 1597, March 1994

[3] J. Rosenberg, J. Weinberger, C. Huitema, R. Mahy, “STUN - Simple Traversal of User Datagram Protocol (UDP)”, March 2003

[4] Bryan Ford, Pyda Srisuresh, Dan Kegel, “Peer-to-Peer Communication Across Network Address Translators”, Pp. 179–192, USENIX Annual Technical Conference, March 2005

[5] J. Rosenberg, R. Mahy, P. Matthews, C. Huitema, “Traversal Using Relays around NAT (TURN)”, draft-ietf-behave-turn-07, February 2008

[6] J. Rosenberg, “Interactive Connectivity Establishment (ICE)”, draft-ietf-mmusic-ice-15, March 2007

[7] Mustapha GUEZOURI, Abdelhamid MELLOUK, “CDCS: a New Case-Based Method for Transparent NAT Traversals”, Journal of Computer Science & Technology, April 2007

[8] Hidekazu Suzuki , Yuji Goto, Akira Watanabe, “External Dynamic Mapping Method for NAT Traversal”, Communications and Information Technologies, ISCIT'07, 2007

[9] UPnP IGD, UPnP Fourm, http://www.upnp.org/standardizeddcps/igd.asp

[10] Zoltan Turanyi, Andras Valko, Andrew T. Campbell, “4+4: An Architecture for Evolving the Internet Address Space Back Toward Transparency”, ACM SGICOMM, 2003

[11] 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 Magazine, 2009

[12] F. Audet, Ed., C. Jennings, “Network Address Translation (NAT) Behavioral Requirements for Unicast UDP”, draft-ietf-behave-nat-udp, January 2007

[13] Hideo Yoshimi, Nobuyuki Enomoto, Zhenlong Cui, Kazuo Takagi, “NAT Traversal Technology of Reducing Load on Relaying Server for P2P Connections”, Consumer Communications and Networking Conference, 2007. CCNC 2007. 4th IEEE, January 2007

[14] Yong Wang , Zhao Lu, Junzhong Gu, “Research on Symmetric NAT Traversal in P2P applications”, Computing in the Global Information Technology, 2006. ICCGI '06. International Multi-Conference on, August 2006

[15] P. Srisuresh, K. Egevang, “Traditional IP Network Address Translator”, RFC 3022, January 2001

[16] D. Senie, “Network Address Translator (NAT)-Friendly Application”, RFC 3235, January 2002