國立交通大學

資訊科學與工程研究所

碩 士 論 文

匿 名 網 路 之 管 理 與 連 線 分 析 機 制

The management and connection analysis of anonymous network

研 究 生：陳冠志

指導教授：楊　武　教授

中 華 民 國　九 十 六　年 七 月

匿 名 網 路 之 管 理 與 連 線 分 析 機 制

The management and connection analysis of anonymous network

研 究 生：陳冠志　　Student：Kuan-Chih Chen

指導教授：楊　武　　Advisor：Wuu Yang

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 Information
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

July 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年七月

匿名網路之管理與連線分析機制

學生：陳冠志　　指導教授：楊　武 博士

國立交通大學資訊科學與工程研究所

摘　　要

隨著電腦網路快速發展，在安全性技術上，最新的加密技術，已經能針對傳輸封包的內容提供相當程度的保護。然而即使不知道通訊的內容，若是得知通訊雙方的網路位置，亦即得知通訊雙方的身份，依然有可能帶來安全性的威脅。

因此在本篇論文中提出一個匿名網路的系統，對送信端與接收端之外的第三者隱藏其雙方的身份，同時相容於現有的平台與應用軟體，並且在必須且經過認可的情況下提供警察單位還原出相關的路徑與連線資訊。

The Management and Connection Analysis of Anonymous Network

Student：Kuan-Chih Chen **Advisor：Dr. Wuu Yang**

Department of Computer Science

National Chiao Tung University

Abstract

As the computer network is developed fast, latest encryption methods, can already offer certain protection to the contents of package. If someone gets the network positions of both sides of communication, namely knows the identities of both sides, even if he does not know the content of the communication, it might still bring the threat of the security.

We propose an anonymous network system which attempts to hide the sender and receiver's identities and positions. The whole system is compatible with multi platform and existing application software. It also provides the authority to restore the original connection information when necessary.

This work was supported in part by Taiwan Information Security Center (TWISC), National Science Council under the grants NSC 95-2218-E-001-001, and NSC95-2218-E-011-015

致謝

首先誠摯的感謝指導教授楊武博士，老師悉心的教導使我得以一窺匿名網路領域的深奧，不時的討論並指點我正確的方向，使我在這兩年中獲益匪淺。老師對學問的嚴謹更是我學習的典範。

本論文的完成另外亦得感謝一起參與專案的志誠的大力協助。因為有你的體諒及幫忙，使得本論文能夠更完整而嚴謹。感謝不厭其煩的指出我研究中的缺失，且總能在我迷惘時為我解惑，同時也感謝實驗室同學癸夫、文均還有學弟們的幫忙，讓我順利完成這篇論文。

另外特別要感謝清大資工石維寬教授實驗室，中興電機黃穎聰教授實驗室，逢甲資工許慶賢教授實驗室，中山電機黃宗傳教授實驗室，東華資工雍忠教授實驗室，政大資科所王乃昕學長，中央資工所陳炯廷學弟，慷慨借我們主機與網路環境供實驗使用。

承蒙國科會資通安全人才培育計畫(II)-資通安全人才培育計畫(II)之計畫編號：NSC95-2218-E-001-001 與資通安全人才培育計畫(II)-資通安全研究與教學中心(II)之計畫編號：NSC95-2218-E-011-015 補助，謹此致謝

最後，謹以此文獻給我摯愛的家人以及女友給我的鼓勵。有你們的支持，使我更能專心於論文的研究，順利完成我的碩士學位。願將此論文獻給他們，並致上最誠摯的感謝！

論文目錄

圖目錄

表目錄

第1章 緒論

1.1 研究動機

隨著電腦網路快速發展，在安全性技術上，最新的加密技術，已經能針對傳輸封包的內容提供相當程度的保護。然而即使不知道通訊的內容，若是得知通訊雙方的網路位置，亦即得知通訊雙方的身份，依然有可能帶來安全性的威脅。

但是在另外一方面，猶如現有電話網路，在顧全一般人通訊隱私的前提之下，當有罪犯利用電話勒索時也必須有一套機制能夠有效的追查出嫌疑犯所在位址，或是裝設錄音與監聽的相關機制。

所以我們希望能夠提出一個在現有網際網路上容易使用匿名性強、執行效率好、相容性佳，且具有自我成長能力，不因使用者數量而受到限制的匿名網路系統，另外具有管理功能，能夠在必要的情況下，防範此系統用於非法用途所造成的傷害。

1.2 研究目標

我們希望能夠提供一個匿名網路的環境，使得傳輸的時候只有發送端與接收端雙方知道是誰在跟對方通訊，而其他第三者都無法得知。加入 Peer To Peer 概念，能夠藉由使用者的加入，自動將整個匿名網路建構起來，並且要能夠相容於現有的應用軟體達到秘密通訊目的。另外在必要情況例如警察單位需要監控或是還原路徑時能夠提供相關的資訊。

綜合以上我們列出下面幾項系統的研究目標：

讓第三者的連線分析與竊聽難度提高

匿名通訊的其中一個目的就是讓第三者無法得知雙方的身份，首先必須讓連線分析與竊聽的困難度提高，這樣才能有效隱藏住發送者與收信者的身份。

將連線路徑與封包的標頭分離

在一般傳統的 IP 網路裡，封包傳送的目的地會直接記錄在標頭裡，在傳輸的過程中任何人截取到封包的標頭內容，立刻能夠知道該封包的最終目的地，對安全性造成相當大的威脅，所以必須將這類的標頭做處理。

能夠支援現有的應用軟體

由於一般現有應用軟體並沒有匿名通訊的效果，所以我們的系統必須要提供一個方便且立即的解決方案，讓這些軟體可以不需要大幅度的變動而達到匿名通訊的目的。

有效的匿名網路節點管理機制

根據[1]這篇論文所提出的網路架構，我們的系統可以經由使用者的加入自動做群組的分配，讓各系統的負載不至於全部集中在少數節點上面。

在必要的情況下提供路徑還原機制

我們的匿名網路系統必須提供一個能夠顧及使用者隱私，同時供檢警單位偵訊與辦案相關線索之管理機制。

1.3 論文架構

第 2 章 相關研究

介紹目前現有的相關匿名網路系統與我們實作系統時所用到的一些模組和相關研究。

第 3 章 系統設計與實作

介紹我們系統[1]的詳細的網路架構、系統模組架構與其運作流程。

第 4 章 管理設計與連線分析

提出我們的管理的目的與策略，並且對於路徑還原機制的不同情境做介紹。

第 5 章 實驗設計與結果

針對我們的匿名網路系統提出實驗設計，分別從匿名性分析、傳輸效能分析與路徑追蹤結果來評估系統。

第 6 章 結論與未來展望

總結匿名系統特點，並且針對實驗的數據與結果提出未來可以改進的方向。

第2章 相關研究

2.1 現有匿名網路系統

早期的匿名系統主要是透過類似代理伺服器（Proxy Server）的方式來幫使用者做加密或是轉傳的動作來達到匿名或是增加安全的目的，而固定的 Proxy 供大家使用，較容易被攻擊者鎖定固定之目標進而入侵或是攻破。之後發展慢慢朝向 P2P 的概念，每個使用者可能是 Proxy 也同時可能是普通節點的角色，讓攻擊者的目標範圍增大，分散其被攻陷的風險。

目前已有的匿名通訊包括ANON.[2]、Crowds[3]、Mixes[4]、Onion Routing[5]、Tarzen[6]等，採用rerouting或是padding機制，來達到匿名保護。所謂的rerouting，如同上述類似Proxy的概念，即提供間接通信傳送，在一次通信傳輸中，配合多部host節點轉送訊息，由發送至接收構成一條由多個安全通道組成的虛擬路徑，此爲rerouting path。在這個路徑上傳送的IP封包，外部的攻擊者無法獲得真實的發送端或是接收端的IP address。使用這種方式，通信者的身份資訊（IP address）就可以有效地隱藏。而padding機制則是將封包的大小固定，若小於該設定的大小則補上空白或是無意義的資料來補齊，讓攻擊者無法在通訊時藉由封包大小來猜測相關的資訊，以提高匿名通訊的安全性。

另外還有特別針對電子郵件的匿名通訊系統Mixmaster[7]，郵件寄出後會由 server 將信拆成多份（fixed size），分別交給不同remail server來做傳送，最後再合起來送給收信人，適合較不需要即時回應的系統。

2.2 SOCKS Protocol

SOCKS是一種網路通訊協定，主要用於Client端與外部網路Server端之間通訊的中間傳遞，而SOCKS是 "SOCKetS" 的縮寫。在這裡我們利用Socks4a protocol[8] 來達到目前應用軟體例如：web browser、ftp client等支援匿名通訊的目的。

Socks4a的傳輸流程如 圖 2-1 所示：Client端會先向Socks4 Server送出一個SOCKS4_REQUEST（其封包格式如 圖 2-2 所示），其中的欄位包含VN表示SOCKS的版本（Version Number），這裡固定爲 4。CD欄位指的是連線的指令（CommanD），一般都是使用Connect指令（其CD欄位值爲 1），若遇到特殊情況如ftp使用active mode，client需要listen在某個port等待遠端server連過來才需要使用到bind指令（其CD欄位值爲 2）。接下來DST PORT、DST IP則分別爲欲連接目的地的Port與IP Address，最後則是認證時會用到的USER ID欄位，在我們的系統實作上因爲考量到只允許本機端的連線，所以並沒有用到這個欄位。

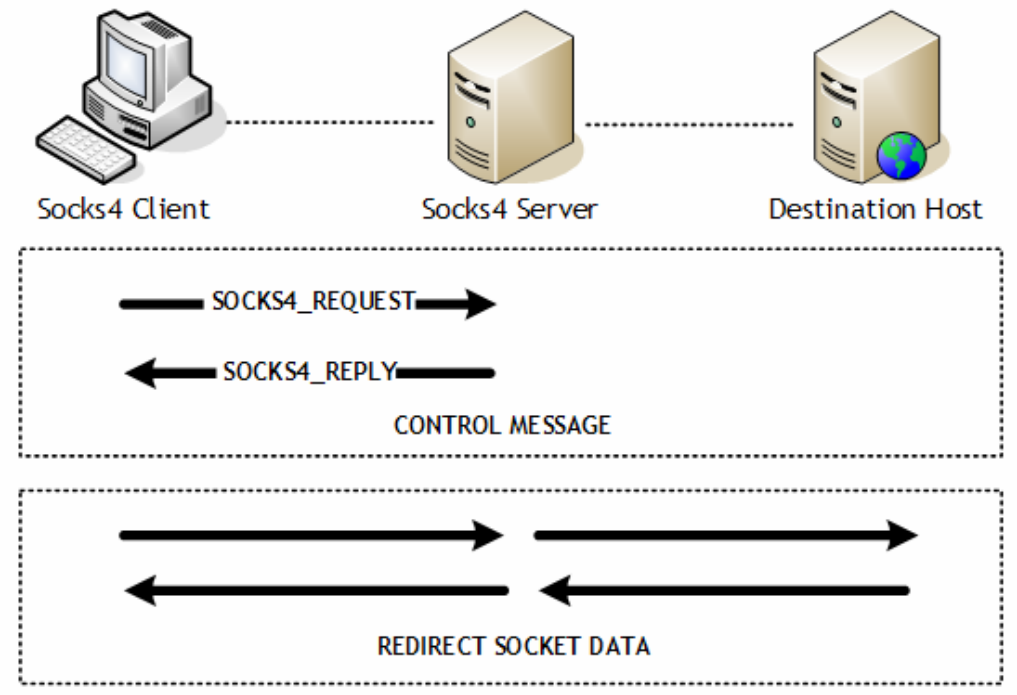


圖 2-1: Socks4a 傳輸流程

Socks4a Server 收到之後會判斷是否允許，回應 SOCKS4_REPLY 給 Client 端，跟 SOCKS4_REQUEST 類似一樣有，VN、CD、DST PORT、DST IP 欄位，但其中 CD 則代表剛剛的請求是否允許，若許可的話 Socks Server 則會幫 Client 建立一個轉送的 socket 來幫忙轉傳資料，完成一個轉傳的連線。

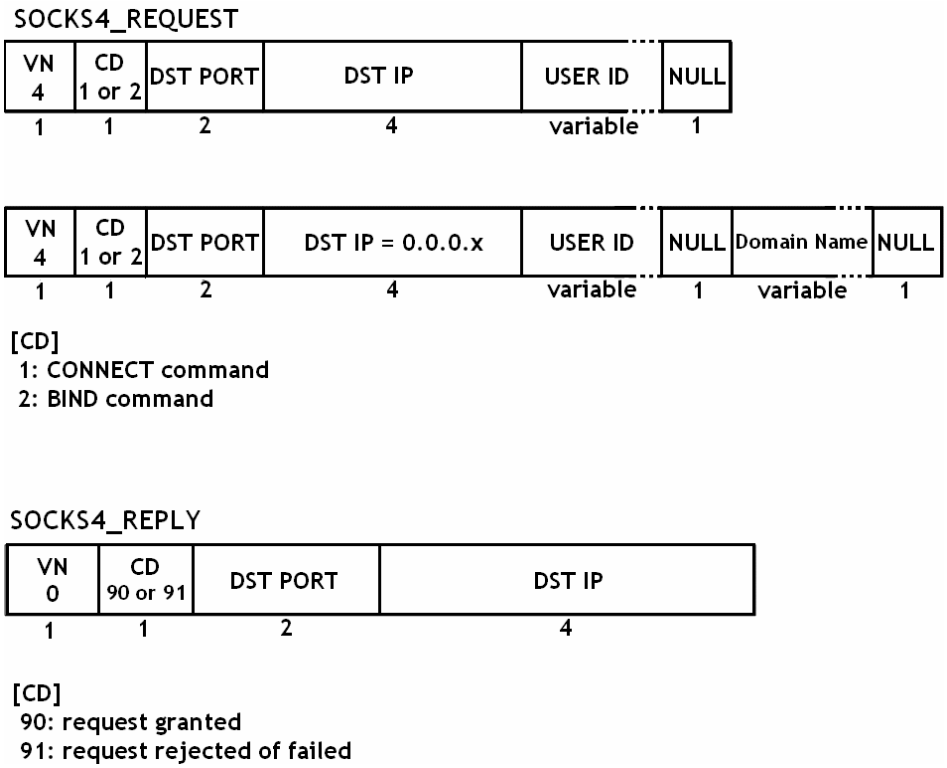


圖 2-2: Socks4a 封包格式

2.3 SSL 加密模組

SSL是Secure Sockets Layer的縮寫。它是一個在 Internet 上進行安全通訊的標準，在封包加密的機制上我們選用一個Open Source的加密套件OpenSSL [9] 加密模組，這是一套開放原始碼的SSL套件，其函式庫是以C語言所寫成，實作了基本的傳輸層資料加密功能。此軟體是以Eric Young以及Tim Hudson兩人所寫的 SSLeay [10] 爲基礎所發展的。支援多種對稱式DES、IDEA、RC2、RC4、RC5、Blowfish、CAST，與非對稱式RSA、DH、DSA的加密演算法，並且在UNIX系統下可以很方便的安裝與使用，例如在我們主要實作的系統 ubuntu Linux 下只要使

用指令 ”apt-get install libssl-dev” 就可以利用apt套件來安裝編譯程式時所需要的OpenSSL加密模組。

OpenSSL 將各種加密演算法各自獨特地方封裝在內部，對外則使用了統一的呼叫介面，因此外部應用只需指定使用何種加密演算法，就可以用相同方法呼叫加解密函數而不用考慮其內部實作方式的差異。EVP 系列函數就是 OpenSSL 統一格式的 interface，定義包含在 evp.h 裡面，這是一系列封裝了 OpenSSL 加密庫裡面所有演算法的函數。透過這樣的統一封裝，使得只需要在初始化參數的時候做少許的改變，就可以使用相同程式呼叫格式，但採用不同加密演算法進行資料的加密和解密。

實際使用的情況以對稱式加密演算法 Blowfish 爲例，只要在 C/C++的原始碼最前面引用 Blowfish 與 EVP 的標頭檔：

```
#include <openssl/blowfish.h>
#include <openssl/evp.h>
```

與宣告加解密所需要用到的 key 與 initial vector。然後在 main function 開頭初始 key 與 schedule，其中在 EVP_BytesToKey () 例子裡，指定好要給哪種演算法（EVP_bf_cbc）所使用的金鑰，產生金鑰雜湊的演算法（EVB_md5），亂數種子（範例中給一個空的值 NULL），最後是一個字串 "plaslab.cis.nctu.edu.tw"，與其長度，讓系統產生加密所需的金鑰。

```
unsigned char key[16];
unsigned char iv[8];

char *str = “plaslab.cis.nctu.edu.tw”;

EVP_BytesToKey(EVP_bf_cbc,EVP_md5,NULL,str,strlen(str),key,iv);
```

加密與解密的流程是一樣的，差別在第 2 到第 4 步驟所呼叫的加密/解密函數不一樣，以下是加密函數詳細流程：

1. EVP_CIPHER_CTX_init

初始化一個 EVP_CIPHER_CTX 資料結構，只有初始化後該資料結構才能在下面介紹的函數中使用。

2. EVP_EncryptInit （解密則是呼叫 **EVP_DecryptInit**）

設置並初始化 EVP_CIPHER_CTX 資料結構。其中，參數 ctx 必須在調用本函數之前已經進行了初始化。參數 type 通常通過函數類型來提供參數，例如下面所使用的 EVP_bf_cbc () 函數形式。最後是加密金鑰 key，與初始化向量 iv。

3. EVP_EncryptUpdate （解密則是呼叫 **EVP_DecryptUpdate**）

對 in 指標所指長度爲 ilen 的字串做加密，將結果儲存在 outbuf 中，當輸入資料超過該加密演算法 block size 的時候，本函數會自動處理。

4. EVP_EncryptFinal （解密則是呼叫 **EVP_DecryptFinal**）

因爲加密演算法必須以固定大小 block 爲單位（例如這裡的 Blowfish 爲 64 bits）做加密，若遇到最後面除不盡剩下來的資料做 padding 特別處理。

5. EVP_CIPHER_CTX_cleanup

清除一個 EVP_CIPHER_CTX 結構中所有資料，並釋放該結構佔用的所有記憶體空間，這樣可以避免一些敏感訊息遺留在記憶體中造成安全上的疑慮。

以下是使用 Blowfish 演算法實作加密函數的範例：

```
int encrypt(int ilen,unsigned char *in,int *olen,unsigned char *out)
{
    int tlen, n;
    unsigned char outbuf[OP_SIZE];

    EVP_CIPHER_CTX ctx;
    // 初始化加密結構，給之後的函數使用
    EVP_CIPHER_CTX_init (&ctx);
    // 設定加密的各項參數
    EVP_EncryptInit(&ctx, EVP_bf_cbc (), key, iv);
    // 對 in 指標所指的資料做加密，結果存在 outbuf 中
    EVP_EncryptUpdate(&ctx, outbuf, olen, in, ilen);
    // 處理剩餘 block 的資料
    EVP_EncryptFinal(&ctx, outbuf + *olen, &tlen);

    // 計算加密後真正資料的長度，並寫回 output 指標
    *olen += tlen;
    memcpy(out, outbuf, *olen);
    // 清除加密結構，去掉存在記憶體中可能的對安全造成威脅資訊
    EVP_CIPHER_CTX_cleanup (&ctx);
    return 1;
}
```

2.4 應用層與網路層實作方式比較

參考之前的匿名網路系統[11]，其中實作層級主要在TCP中的網路層，藉由直接修改作業系統核心原始碼來達到匿名通訊的目的架構如 圖 2-3，此種作法的優點是能夠有效的直接相容於現有的應用軟體，而不需要額外的做任何修改，在效率方面也相對的比較高，但是也因爲必須要對系統的核心直接做修改，跨平台與可用度變得比較低。

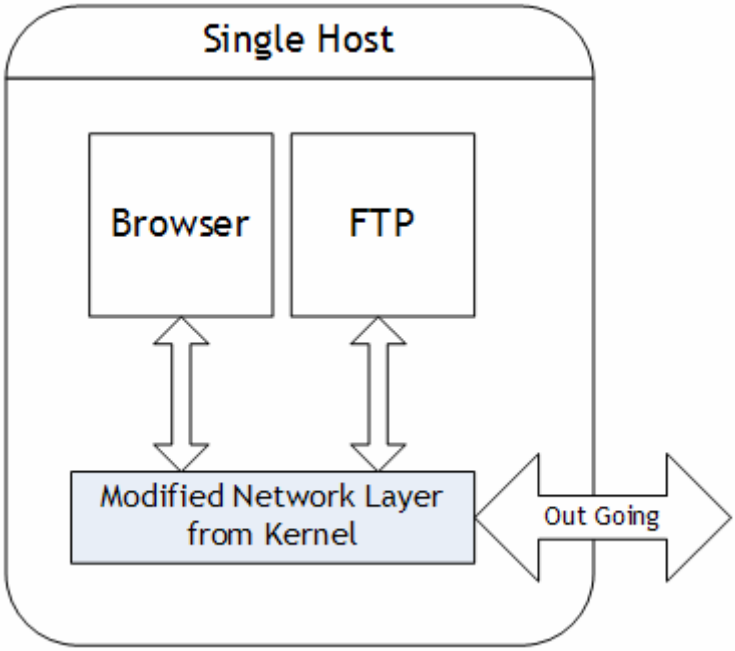


圖 2-3: 實作於網路層的匿名網路

所以在這篇論文裡我們提出另外一種想法與架構，轉而從應用層著手，如 圖 2-4 藉由前面 2.2 節所提到Socks4a Protocol技術來達到相容現有應用程式的目的，再從Socks Server傳送做修改，使其傳送出加密處理過的訊息。且修改與節點間合作機制也更有彈性與完整，而其傳輸效率可能較差的先天性缺點相信也在網路硬體設備的傳輸速度快速進步之下能夠在不久的將來得到一定程度改善。

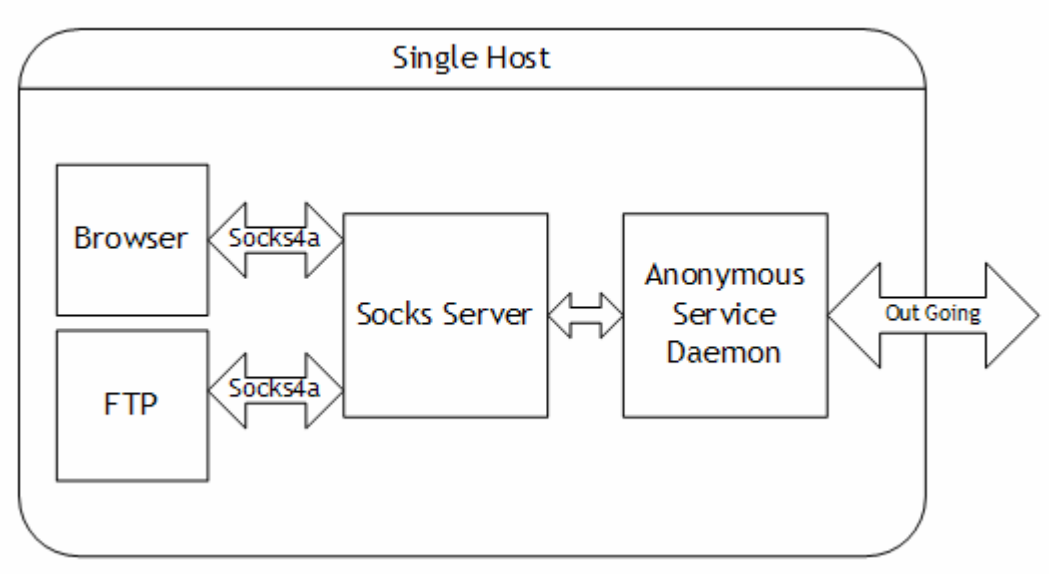


圖 2-4: 實作於應用層的匿名網路

第3章 系統設計與實作

3.1 系統架構概述

如 圖 3-1 所示，整個匿名網路系統主要分成三個部份：Local Host，Onion Server Daemon，Optional Part。

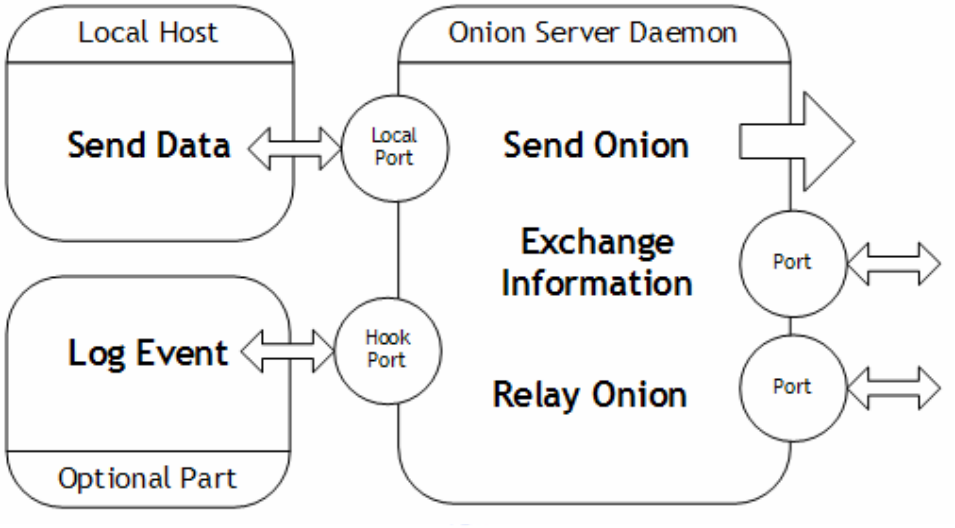


圖 3-1: 系統架構

3.1.1 Local Host

就是使用者目前所使用的環境，一般應用程式如Web，FTP，將其設定成經過 Socks Server(如附錄)，若不支援則在Windows平台上可以透過SocksCap [12] 軟體；Linux等UNIX平台則可透過TSOCKS [13] 的支援來經由Socks4a Protocol 連出去，進而達到匿名的效果。

3.1.2 Onion Server Daemon

匿名網路的主要程式區塊，一般使用者藉由安裝這套程式來達成匿名網路成員的目的，其中提供傳送匿名通訊封包、交換節點資訊等等主要功能，在後面

章節會再詳細介紹。

3.1.3 Optional Part

可額外安裝的部份，讓使用者自行選擇是否要使用，可以記錄該節點收發 Onion 的資訊，收集一定數量節點後可經由分析得到整體傳輸狀況。

3.2 加密與傳送機制

爲了達到隱藏傳送路徑的目的，我們在傳送之前，依照後面會在提到的節點交換機制所得到資訊，先建立好一組節點，連同傳送目的地位址跟資料，把封包使用與該節點的Secret Key一起加密，像洋蔥（Onion）一樣層層包起來。就像 圖 3-2所示，當Sender要跟Receiver傳送資料M時，最內層會先用Sender與Receiver的Secret Key：R先做一次加密，接著依照Tunnel所指示的節點順序依序再以Sender與A還有Sender與B的Secret Key做加密，最後將整個Onion包裝完成。每當一個節點收到一個Onion時，會用自己相對的Secret Key來解密，得知下一站的目的地後往下傳，直到最後Receiver收到爲止。

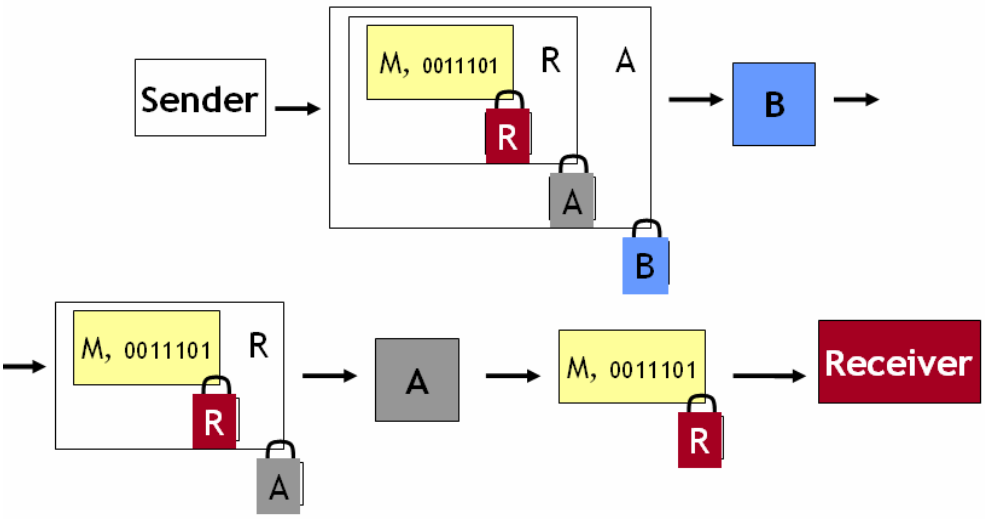


圖 3-2: 封包加密機制(Onion)

另外在傳輸機制上，我們在系統上加入了Peer To Peer的概念，如 圖 3-3所示：每個節點都可能成爲發送端、接收端，或是中間幫忙轉傳的節點，使用者在本機端透過Socks Server與Onion Creator做出Onion封包發送出去，沿途一層一層解密拆開，往下一站送，最後到圖示中Server節點，完成一次匿名傳輸。其中只有發送端跟接收端彼此知道互相的身份，而中間幫忙轉傳的節點只知道該路徑中的前一站與後一站，並不清楚整個傳輸路徑與頭尾究竟是誰在做秘密通訊。

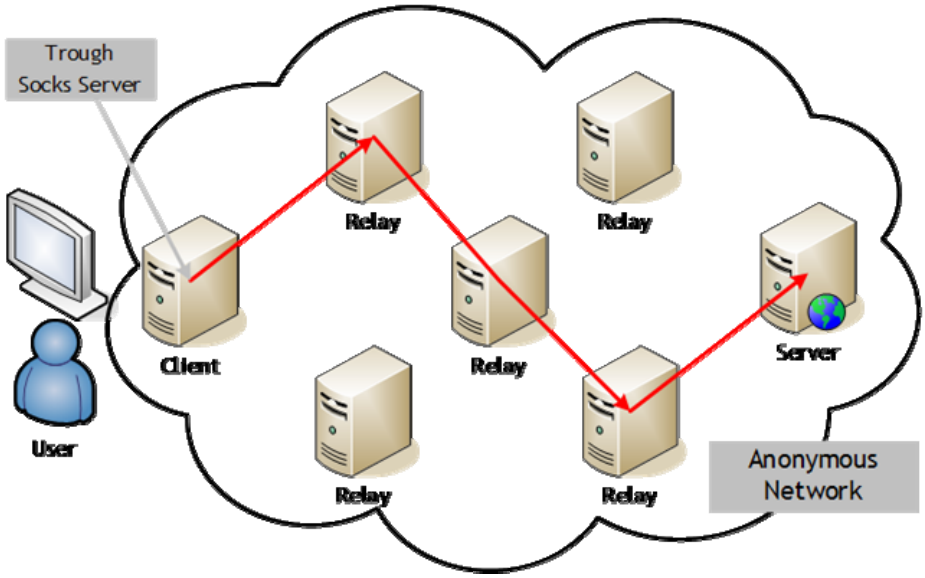


圖 3-3: 匿名網路傳送機制

3.3 標頭加密機制

考慮對整個完整的封包做加密後傳輸必須付出的額外負擔較大，我們提出另一種加密模式。在原本的機制下所傳送的資料會被重複加密多次，在加密演算法有一定的強度之下，我們的目標在於保護封包的目的地，所以我們採取對資料只用與接收端的秘密金鑰加密一次，而對記錄有該封包繞徑資訊得標頭做層層加密，詳細的封包格式在後面會做介紹。

3.4 節點分層機制

我們將節點分成三種，如 圖 3-4 所示：分別爲Static Subscribe Server，Dynamic Subscribe Server與Common Node。其中Static Subscribe Server其概念類似DNS Root Server，是固定的幾台機器，一方面可以平衡負載，另一方面也可以有備援的效果，三種節點間也會有相對應的Protocol來溝通與交換資訊，以下各節會分別介紹這三種節點的作用與實作方式。

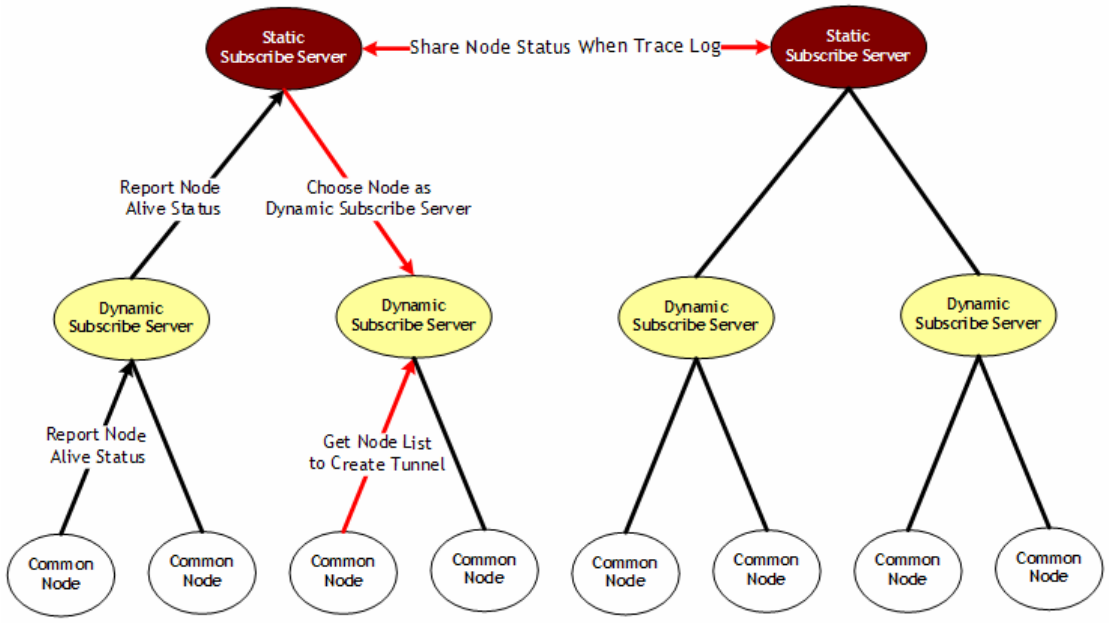


圖 3-4: 匿名網路節點架構圖

3.4.1 Static Subscriber Server

最上層的節點，類似 DNS Root Server 的概念，爲固定數量 Server，各自擁有自己旗下的節點清單，在需要的時候會互相交換節點資訊，同時接受旗下 Dynamic Subscribe Server 與 Common Node 回報節點資訊，在水平向其他 Static Server 交換資訊，達到擁有整個網路拓樸的目的。

- 固定的數台，在本篇論文中我們固定爲三台，以達平衡負載的效果。
- 接受 Common Node 上線之後註冊資料，Common Node 選擇其中一台 Static Subscribe Server 登記，Timeout 才換下一台。
- 在Common Node清單中，依據其註冊資訊中的頻寬，處理速度等參數來挑選出Dynamic Subscribe Server。
- 接受 Dynamic Subscribe Server 回報該節點是否還可用。

3.4.2 Dynamic Subscriber Server

第二層的節點，剛開始也只是 Common Node，在一定數量節點跟 Static Subscribe Server 註冊後，Static Subscribe Server 從中決定出某個升級成 Dynamic Subscribe Server，它擁有全部 Static/Dynamic Subscribe Server 清單，但只管理旗下的 Common Node，定期接受它們回報節點狀況。

3.4.3 Common Node

第三層，同時也是最基層的節點，一般使用者剛加入匿名網路都是從 Common Node 身份開始，經過 Static Subscribe Server 挑選後有可能升格成爲該區的 Dynamic Subscribe Server，在建立匿名網路的路徑時主要跟上層的 Dynamic Subscribe Server 請求得到節點資訊來建立 Tunnel。

3.5 模組實作

如 圖 3-5所示，各個部份以下是詳細各個模組的功能與實作方式：

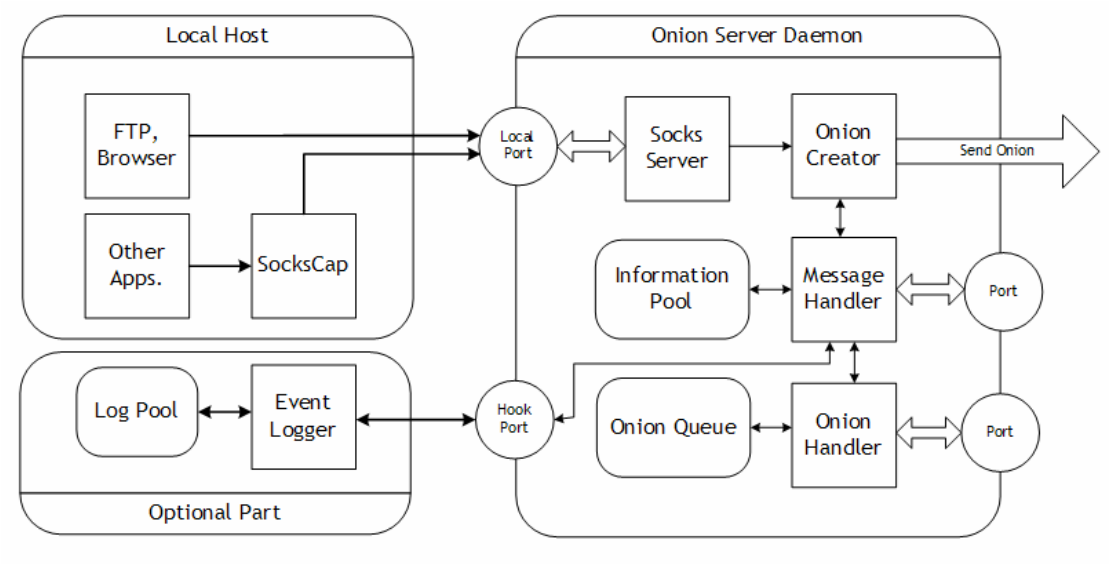


圖 3-5: 匿名網路模組架構圖

3.5.1 Socks Server

接收由本地端應用程式送來的封包，將其傳送給 Onion Creator。其中 Socks Server 設定有防火牆機制，每當有建立連線的請求時，會去檢查該連線的來源位址、連接埠，目的位址、連接埠，確認是否爲合法連線再決定是否放行。在我們匿名網路系統中預設只允許來自本機端的連線，以避免外部不法的連線藉由 Socks Server 連往外部的網路，而產生安全性問題。其防火牆的設定檔 socks.conf 如下：

```
allow c 127.0.0.1 - - -
deny c - - - -
```

表格 3-1: 設定檔 socks.conf

第一行表示允許 socks4a protocol 中的 connect 連線，來自 127.0.0.1(本機端)，的任意連接埠，連往任意目的地位址與連接埠，其中 “ - “ 表示任意的意思。第二行表示拒絕其他任何的連線，由於防火牆採用 First Match 原則，所以可以達到只允許來自本機端連線的目的。

3.5.2 Onion Creator

跟 Message Handler作溝通，取得建立 Tunnel 中各節點相關資訊（IP位址、各節點的加密金鑰）後，如 圖 3-6 所示，若某條Tunnel的傳送路徑爲：Node1 → Node2 → Node3 （其中Node1 的IP Address1，Node2 爲IP Address2，Node3 爲IP Address3）。依據傳送的順序把封包層層加密，先用與Node3 節點的金鑰做最內層的加密，接著把它當成資料用與Node2 的金鑰再加密一次，最外層的也用與Node1 的金鑰加密，完成後在最外層補上讓OnionHandler辨認的ID_Bits與當初建立Tunnel時產生的Serial Number 其作用與機制會在 4.2 節 做介紹）完成 Onion 並傳送出去。而單純加密標頭的模式如 圖 3-7，標頭加密的方式剛上面所提到的一樣，但是資料的部份只有使用接收目的端與傳送端的加密金鑰加密一次。

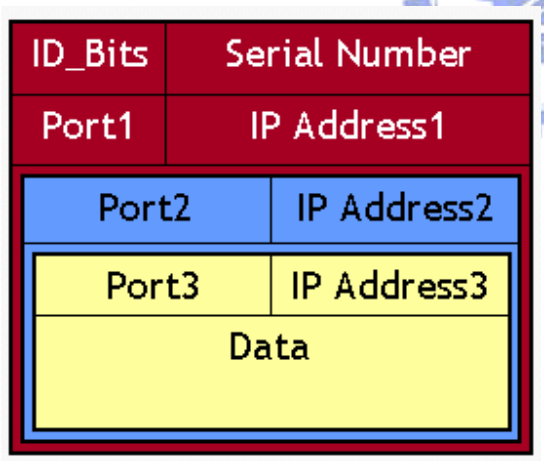


圖 3-6: Onion 格式圖

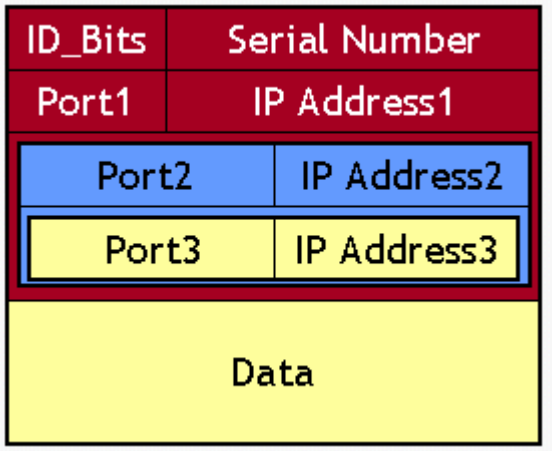


圖 3-7: Onion 格式圖(只加密標頭)

3.5.3 Message Handler

負責在一開啓系統時跟 Static Subscribe Server 做註冊，將本機資料如 IP Address、Port、節點的效率值上傳上去。其中 Port 資訊可以由使用者自訂，修改設定檔 onion.conf 即可，而節點效率值則根據系統的處理器、記憶體與網路

頻寬等參數測定出一個相對數值。

接下來取得節點清單儲存在 node.list 中供以後傳輸時建立 Tunnel 使用。平常每隔一定時間向 Dynamic Subscribe Server 更新節點資訊，存取 Information Pool 裡面所維護與記錄的各式 Table 資訊（詳細種類在 3.4.5 節介紹），負責回應其他 handler 所需要的資訊。並且經由外部 Port 跟外界交換資訊，並且更新 Information Pool 中的資料。

3.5.4 Onion Handler

經由 Port 處理外部送過來需要轉傳的Onion。 收到Onion會讀取Serial Number先記下來，然後選擇對應的加密金鑰做解密，如 圖 3-8 與 圖 3-9 所示，得到下一站資訊，然後將新的Serial Number填回原本的欄位，傳送到下一站節點去。

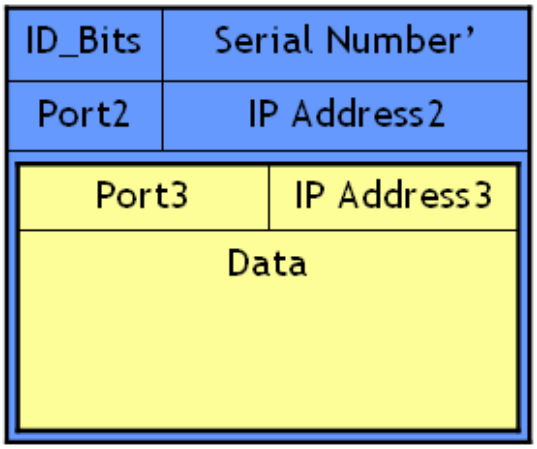


圖 3-8: 經過一站後的 Onion

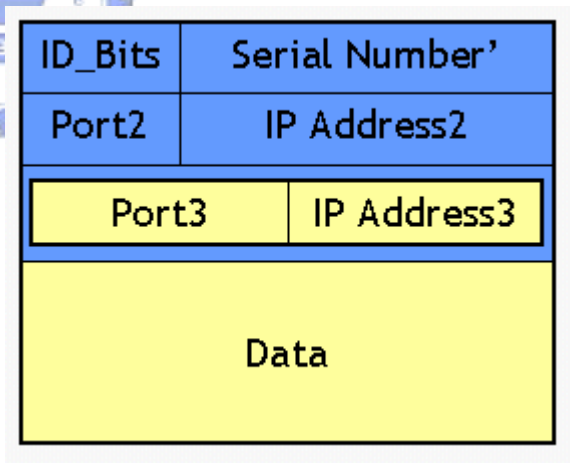


圖 3-9: 經過一站後的 Onion(只加密標頭)

3.5.5 Information Pool

儲存各種 Table 與節點資訊如 表格 3-1所示：只能允許給 Message Handler作查詢或更新動作。

設定檔	功能與用途
serv.list	儲存 Static Subscribe Server 資訊，預設有固定三台
node.list	儲存 Common Node 節點資訊
tunnel.list	儲存 Tunnel 與 Key Number 資訊
SN.key	儲存與其他 Common Node 的共同 Secret Key 加密金鑰，一檔案存一組 Key，檔名爲 Serial Number

表格 3-2: Information Pool 中的 Table 列表

如 表格 3-3 所示，serv.list中儲存預設的固定三台Static Subscribe Server的資訊，在系統剛啓動的時候會去找第一台做註冊，若無回應則換下一台。

```
140.113.166.21
140.113.166.24
140.113.166.25
```

表格 3-3: 設定檔 serv.list

如 表格 3-4 所示，node.list儲存可供建立Tunnel時所需要的節點，第一欄爲IP Address，第二欄是該節點所使用port，第三個欄位則是該節點的效能指標值。經過Message Handler交換節點資訊後，離開的節點則在該筆資料前加上一個 # 記號，標記其節點目前爲不可使用的狀態。

```
140.113.166.24 1089 10
140.113.166.25 1089 10
#140.113.166.31 1089 10
140.113.179.205 1089 10
140.113.179.206 1090 10
140.113.179.207 1091 10
```

表格 3-4: 設定檔 node.list

3.6 執行流程

從系統啓動開始，逐步介紹匿名網路系統的各個步驟與其目的：

1. 取得本機資訊

取得本機IP Address，經由設定檔socks.conf得知以後要由哪個Port交換資訊，並且執行Benchmark函數，計算本機端的運算能力，送出一筆如 表格 3-4 所示的node.list中的一筆資料。

2. 節點註冊與交換訊息

經由前個步驟所得到本機相關資訊，向記錄在設定檔 serv.list 中的 Static Subscribe Server 註冊，而 Static Subscribe Server 會在接受註冊的節點群中，依據 IP Address 與運算能力數值決定出一台 Dynamic Subscribe Server，爾後則由該節點向旗下各 Common Node 做訊息交換，例如其他節點的連線資訊、是否目前還可以使用（儲存於 node.list 中）。

3. 啓動本機軟體

啓動 Socks Server，供本機端應用軟體透過 SOCKS 通訊協定做匿名通訊傳送封包，詳細的使用方式如 附錄 所介紹；接著啓動 Onion Handler，供其他節點轉傳封包使用。而其他固定週期時間執行的程式則交由系統 crontable 來自動執行，例如向 Dynamic Subscribe Server 要求更新節點資訊的程式，還有之後 4.3 節會提到的 Log 上傳程式。

4. 建立 Tunnel 清單

在傳送 Onion 之前，系統會事先從節點清單（node.list）中依據運算能力數值、來源、系統設定的匿名繞站次數（預設值爲 3）配合非對稱加密金鑰機制

跟選定的各個節點交換共同的加密金鑰（Secret Key），將 Tunnel 資訊儲存在 tunnel.list 中。

5. 傳送資料

當 Socks Server 收到來自本機端的連線請求時，會從前個步驟建立好的 Tunnel 清單中取出一條來，加上連線請求的目的端節點，以各自對應的金鑰層層加密成一個 Onion 傳送到第一站去。Onion 的實際格式如 **錯誤! 找不到參照來源。** 所示。

6. 轉傳資料

當Onion Handler接收到由其他節點傳送過來的Onion時，會用自己對應的加密金鑰先解密，如 圖 3-8 所示，若下一站的IP Address不是自己，則幫忙傳往下一站去。

第4章 管理設計與連線分析

4.1 管理目的與 Trade off

雖然匿名通訊的主要目標是隱藏通訊雙方之身份，但是如同電話網路的狀況一樣，當有不法人士例如綁架勒贖的歹徒，利用電話向受害者家屬勒索贖金，此時檢警單位也會利用相關機制去追蹤發話端位址，以期更進一步得知歹徒的行蹤與可能藏匿的地點。類似情形也可能發生在網路世界裡，所以我們的匿名網路系統必須提供一個能夠顧及使用者隱私，同時供檢警單位偵訊與辦案相關線索之管理機制。

爲達到上述目標，我們採取分散通訊記錄策略，每個節點各自記錄該站的傳輸或是幫忙轉傳的記錄，像是前一站與下一站資訊，或者是該事件發生時間。如果不法人士攻破某個節點或是獲得該點資訊，只要沒有獲得一定數量以上節點，則無法完整的還原當初連線全部內容，如此一來便可以相當程度保護匿名通訊記錄。

舉例來說，若某個相近時間區間內，有兩個人 A 與 B 正在做匿名通訊，而他們傳輸的路徑爲 A → W → X → Y → Z → B，依照前面所介紹分散通訊記錄策略，每個節點只會記錄前一站與下一站資訊，例如節點 W 只會記錄 A → W → X 的關係。如此一來若攻擊者所掌握的資訊在此路徑上必須中斷的點數低於兩個節點才有可能正確的還原當時的情況。舉例來說，若攻擊者已經攻破 W 跟 Z 兩個節點（缺少 X 與 Y 兩節點資訊），則他可以掌握到 A → W → X 與 Y → Z → B 這兩筆傳送的資料，但是他不能夠確定 X 與 Y 之間是否真正在通訊，而無法還原

整個正確資訊。

而在只缺少一個節點的情況下如 圖 4-1所示，當缺少的那個節點是交會的部份，如圖中的X節點，則攻擊者也無法正確知道究竟該路徑是A → X → B或者是C → X → B，當經過X的連線有*n*條，則攻擊者猜對正確路徑的機率也變成 *1/n*。

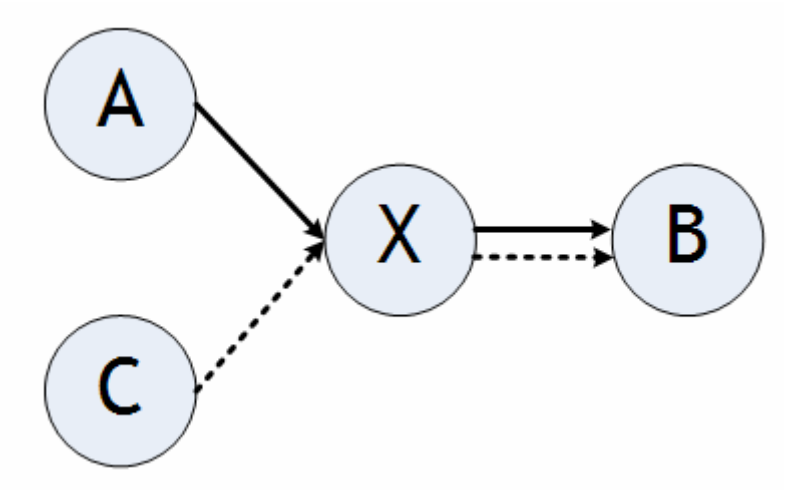


圖 4-1: 匿名傳通訊節點交會範例

4.2 Serial Number 機制

接續上一節所提到的連線記錄，會在每個節點記錄時間、前一站、下一站的資訊，在一般的情況之下取得該時間範圍內所有連線記錄，即可還原出整個傳送的路徑，但是當整個網路的傳輸量大到一定程度時，很有可能發生同個時間內有某段路徑有多個不同的連線經過的情況發生，如 圖 4-2 所示，在這個時間點上，分別有下列兩個連線發生：

連線 1： W → A → C → D → E → Y（實線部份）

連線 2： X → B → C → D → F → Z（虛線部份）

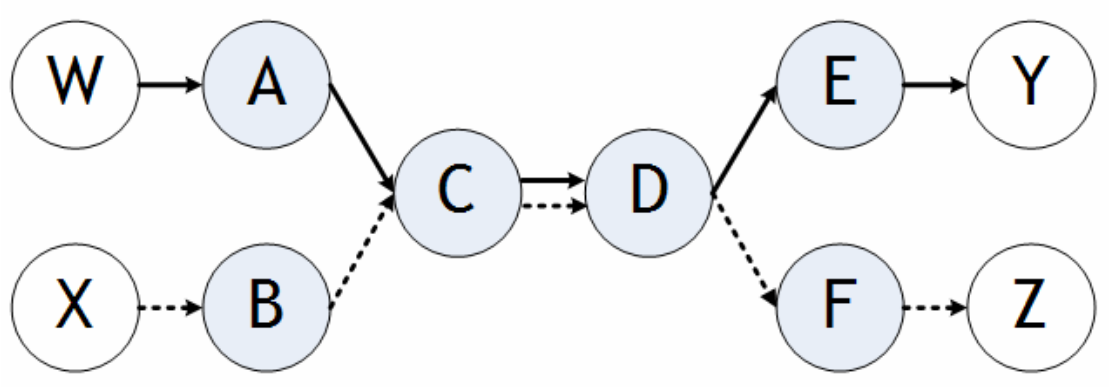


圖 4-2: 匿名傳送重疊路徑範例

而中間各點的連線記錄情況(擷取上一站與下一站資訊)會如同 表格 4-1 所示:

節點編號	A	B	C	D	E	F
記錄資訊	W → C	X → C	A → D B → D	C → E C → F	D → Y	D → Z

表格 4-1: 重疊路徑節點記錄資訊

當警察單位取得這份記錄，嘗試著想還原出這兩筆連線時由於時間點太過接近，同時間都有經過 C → D 這段，所以單就連線 1 的情況而言，無法正確判定該路徑是 W → A → C → D → E → Y，還是 W → A → C → D → F → Z，造成目的地的不確定而無法正確的還原整個連線的原貌。

爲了避免這種不確定的因素，我們加入一個Serial Number的參數來確定該連線的唯一性，並且顧及到此參數若中途被第三者所攔截不可以直接透露出連線的路徑，我們採取每經過一個節點轉傳時Serial Number便會變更一次，而其相對應的關係只有該節點有記錄，如 表格 4-2 所示： 其中粗體的數字都是Serial Number，實際資料類型爲一個integer。

節點編號	A	B	C	D	E	F
記錄資訊	W → C **23 → 65**	X → C **49 → 67**	A → D **65 → 43**	C → E **43 → 84**	D → Y **84 → 24**	D → Z **13 → 90**
			B → D **67 → 27**	C → F **27→ 13**		

表格 4-2: 重疊路徑節點記錄資訊(增加 Serial Number)

以先前提到的連線 1 作爲例子，當節點 A 收到一個 Onion 時，解開發現其下一站的目的地是節點 C，且 Serial Number 爲 23，在送往下一站之前，會把這個欄位改成 65 並且把這個 23 → 65 的對應記錄下來，後面的節點依此類推。如此一來當需要還原路徑時，就可以從這個對應的關係分別出連線 1 與連線 2 的差異，而且就算在傳輸的當中第三者竊取到任意一個 Onion 得到其 Serial Number 欄位，沒有取得完整得沿路節點資訊，也無法猜出正確的路徑。

實際某個節點的連線記錄如 表格 4-3 所示，列出六筆資料：

Date	Last Hop	Next Hop	Serial Number1	Serial Number2
1184092125	2356258335	2356301680	648906234	535086567
1184092147	2356258329	2356258328	1100509170	1354948435
1184092174	2356258335	2356258328	2078030442	1734305903
1184092199	2356261838	2356258329	168408188	570888359
1184092202	2356258329	2356261839	409975485	737582962
1184092208	2356258328	2356258329	732033291	2128692284

表格 4-3: 節點的連線記錄

第一欄爲時間，格式爲time_t的integer，是從 1970-1-1 日開始到現在的秒數，第二欄與第三欄分別代表該次連線的上一站與下一站節點的IP Address，其格式也是integer（IP: a.b.c.d => a x 256^3 + b x 256^2 + c x 256 + d），最後兩個欄位

記錄的是收到的Serial Number與改變後傳出去的Serial Number，格式一樣爲integer。

4.3 Log 的分散式與傳輸架構

連線記錄 Log 的儲存方式分兩層架構，下層是各節點轉傳與連線時記錄的Log，上層則是固定幾台的 Static Subscribe Server（在本系統中預設爲 3 台）。而依照 Log 傳送的方向可分成由下往上定期傳送（事後分析/被動模式）與由上往下詢問（事先預測/主動模式）的方式兩種。

在由下往上定期傳送方面，各節點所記錄的Log每天會將本地端的資料壓縮起來，送到所屬的Static Subscribe Server。由 表格 4-3 可以得知，每筆資料都是integer大約有 10 位的整數字，加上分隔的Tab字元，所以一行大概有 55 Bytes。以 2007.07.06 這天的 140.113.179.206 這個節點實際的記錄資訊來看，總共記錄了 3517 筆傳送資料，平均一分鐘會發送或是轉傳兩到三筆資料，而整個Log檔案大小未壓縮的純文字大小爲 192 Kbytes，經過壓縮成.tgz檔後只剩下 8.38 Kbytes，節省了將近 77%的空間，爲每天上傳到Static Subscribe Server上面節省不少頻寬。

當 Static Subscribe Server 收集到旗下節點的傳輸資訊後，便可以對剛剛所建立的資料庫做分析，例如想要知道某某時間點上，某台電腦正在和誰做通訊，透過連線記錄的時間與來源欄位，便可以迅速的查出相關的資訊。

在由上往下詢問方面，當警察單位掌控了某些特定資訊，例如在電信網路的例子中，歹徒下一次勒贖訊息會在某個時間範圍發生，且地點可能在某段 IP 網域中發出，如此一來 Static Subscribe Server 便可以發出相關的線索給旗下節

點，只要一有發生類似的連線事件就會立即通報上去，相對於每天上傳機制更能達到即時反應的效果。

4.4 三種情境模式的分析

以實際例子來介紹連線分析會遇到的情況，分成事前、事後還有混合模式三種，分別在下面的三節做詳細的介紹：

4.4.1 事後分析(被動模式)

事後分析的實際的運作步驟如 圖 4-3 所示，每個Common Node會各自記錄傳輸的資訊。每天（安裝時排入固定的時程，例如加入UNIX上的crontable，交給系統自動執行）會將自己的Log整理，壓縮起來以節省傳輸時所需要的頻寬與時間，送到自己所屬Static Subscribe Server，在實作上面我們直接使用ftp來傳送。當Server收集到Client傳來的資料，會先解壓縮，然後再跟自己Log Pool做合併排序後儲存，因爲各節點的資訊已經是以時間做排序來記錄，所以合併起來在做一次排序時間複雜度也相對降低，最後以天爲單位將Log做好分類。最後，遇到需要做查詢時，即可在Static Subscribe Server執行查詢程式，針對相對應時間的Log做搜尋與比對。

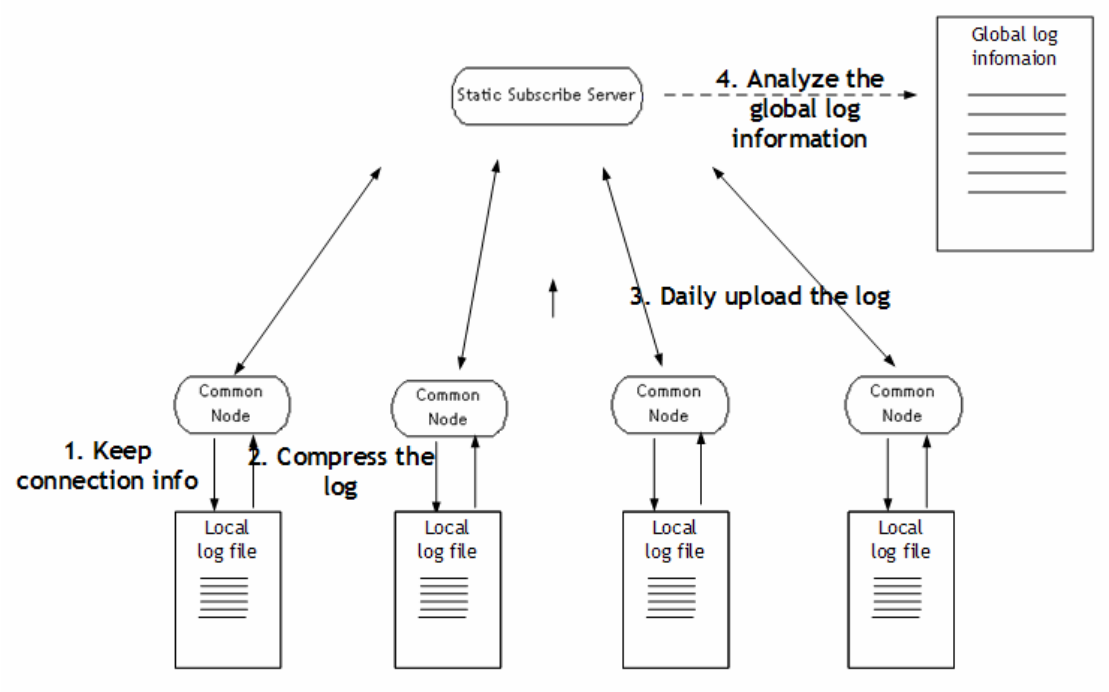


圖 4-3: 被動模式的 Log 分析步驟示意圖

4.4.2 事先預測(主動模式)

事先預測實際運作模式如 圖 4-4 所示，相對於前一節提到被動等Common Node定期送連線資訊而言，事先預測模式可以先把想要讓大家注意的時間範圍或是特定的節點資訊傳給旗下所有Common Node，讓大家一起在傳送或是轉傳的時候都去檢查是否符合。當有可疑的連線行爲符合給定的線索時，該節點會即時回訊息給Static Subscribe Server，就可以有效掌握不法人士的連線記錄，同時也兼具時效性。

舉例來說，若 Static Subscribe Server 指定在 2007.07.09 這天，若有任何連線有經過 140.113.179.206 這個疑似有不法行爲的位址時，立即回報。如此一來，各節點在每筆連線寫入 Log 之前，就會先去比對是否符合上面的條件，如果一發現有可疑的連線，馬上就回報給所屬的 Static Subscribe Server，立即取得連線的記錄。

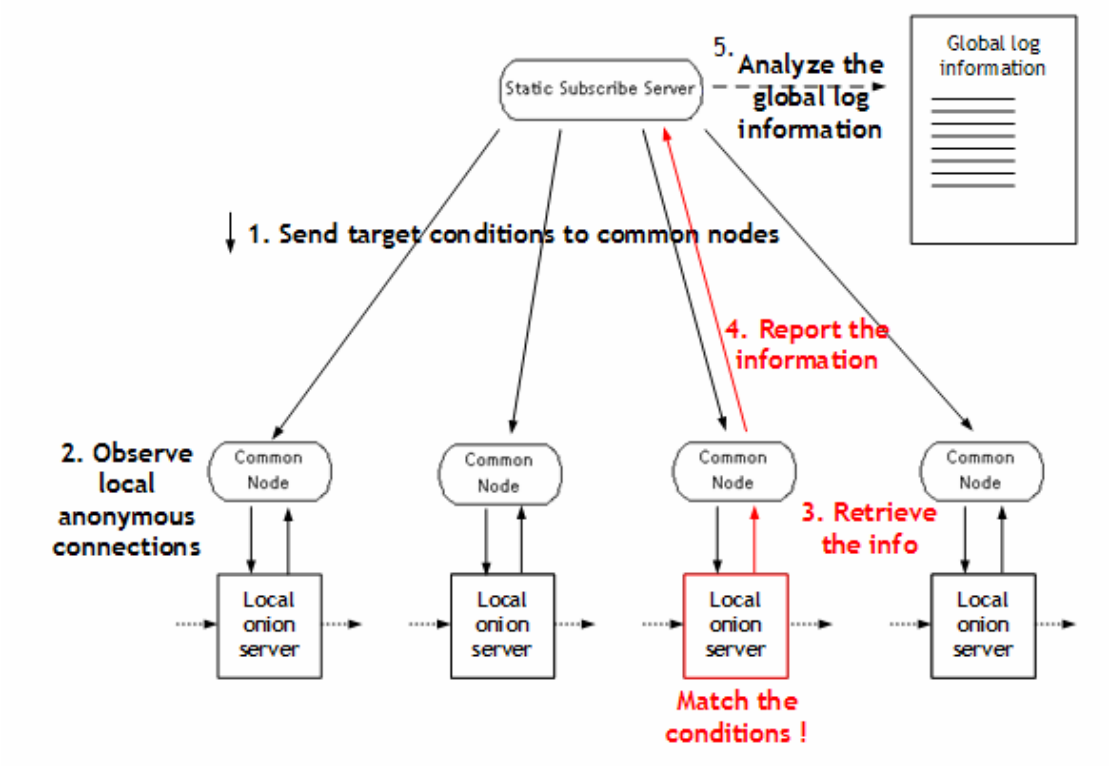


圖 4-4: 主動模式的 Log 分析步驟示意圖

4.4.3 混合模式

除了前面兩種模式之外，若加上簡單的學習或是規則的設定，融合兩種模式的特性，便可以達到更有彈性的追蹤效果。例如訂定一個規則，同時間點內，某個節點若是連線數超過一定的量，便可能推測該節點可能是中毒或是被當成跳板之類所以產生此類可疑的連線行爲，在 Static Subscribe Server 中事後的分析可以自動化的去偵測可能發生的源頭，之後在自動的發送相關可能的線索，例如某個可疑的節點位址，或是在某時間點發作，如此一來更能加強本系統的分析功能。

4.5 連線分析演算法

這邊的連線分析情況與演算法主要針對事後的分析來做討論，以現有的資料來做探勘與分析，盡可能的還原出案發當時的連線情況。又因爲種種原因例如

某個節點因爲匿名性考量並未安裝上記錄連線 Log 的 Optional Part，或是因爲傳送的問題或是其他原因造成 Static Subscribe Server 上面的 Log Pool 上可能沒有完整的節點連線資訊，會有以下幾種狀況在後面的三節分別做討論。

分析時所給定的線索主要分成時間或是節點位址做爲查詢的參數，而時間可能是一個不確定的時間範圍內都是目標，位址方面也可能是模糊的一小段子網域內的節點，必須考慮到這方面的搜尋需求。

4.5.1 完整節點

在理想的狀況下，在分析前可以獲得所需要的全部節點資訊，無論是本機端就直接可以存取到的，或是向別台 Static Subscribe Server 做查詢。依照站與站之間的傳輸關係，在配合 Serial Number 把重疊路徑的連線區別開來，按照下面介紹的*演算法 1* 即可得到所需要的結果。

我們將所有的連線記錄存放在 struct log pool[MAXLOG]; 這個結構裡：

```
struct log{
    unsigned long int time;     // Time Stemp
    unsigned long int last_ip; // Last Hop IP
    unsigned long int this_ip; // This Hop IP
    unsigned long int next_ip; // Next Hop IP
    unsigned long int sn1;
    unsigned long int sn2;
};
```

如 圖 4-5 虛線的方框，要確定節點A在路徑上的下一站是否是節點B，必須檢查

三件事情： 1. A.this_ip == B.last_ip, 2. A.next_ip == B.this_ip, 3. A.sn2 == B.sn1 若皆成立表示B爲A在該路徑上的下一站。

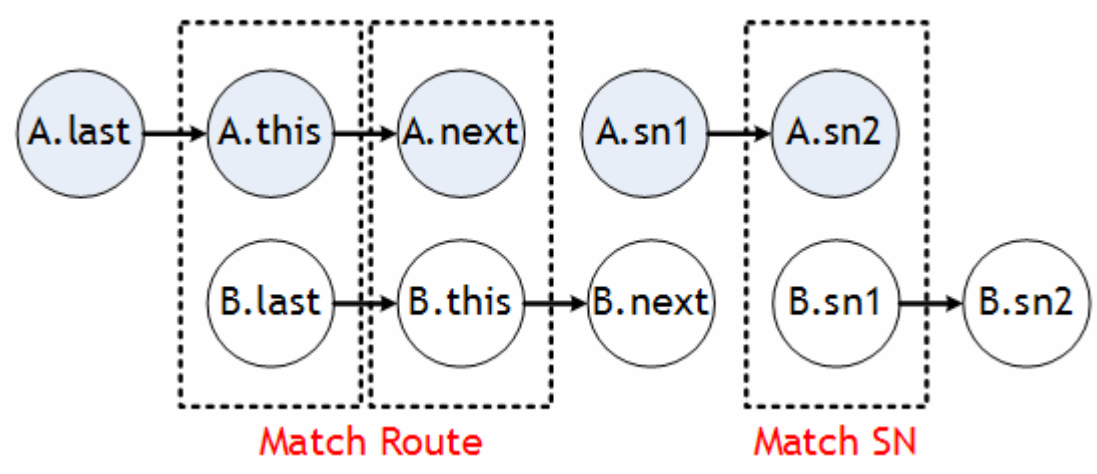


圖 4-5: 確認路徑上的下一個節點

寫成函數的型態則如下：

```
Function FindNext(int index) {
      for(i=index; i<MAXLOG ; i++){
          if (pool[i].last_ip == pool[index].this_ip &&
              pool[i].this_ip == pool[index].next_ip)
                matchRoute = true; // 路徑順序符合
          if(pool[i].sn1= pool[index].next_sn)
                matchSN=true;       // Serial Number 對應符合
          if(matchRoute && matchSN)
                retrun i; // 如果兩者皆符合，回傳該點爲該路徑下一個節點
          else
                return -1; // 找不到符合節點
      }
}
```

//演算法 1：取得下一站節點（完全比對）

1. 從限制條件中（特定時間或是特定節點）過濾，取得指定的節點

```
pool[index].this_ip
```

2. 尋找該節點的下一站：

```
next = FindNext(index);
while( next != -1)
{
  將 next 這個節點加入節點清單;
  next = FindNext(next);
}
```

3. 將節點清單中的節點依序取出，得到一條完整的連線路徑記錄。

4.5.2 缺少單一節點

先前提到有時候在還原路徑的時候會發生缺少節點的狀況，若在上一節所提到的演算法中遇到無法由前後站與 Serial Number 對應找到下一站，也就是 FindNext() 中搜尋完整個節點集合，卻找不到 matchRoute 且 matchSN 的節點，則判定爲缺少節點，則中途再呼叫*演算法 2*，在此節點後面在由前後站的關係，找到下一站可能的節點集合，再次套入我們的*演算法 1*，找到所有可能的節點。

如 圖 4-6 所示：當缺少一站，節點A要找路徑上下下站的節點資訊，只剩下A.next == C.last 這個關係來確認，少了Serial Number的幫助，我們只能從時間的間隔判斷，將所有符合的節點都列出來，分別去尋找各個點之後的路徑。如下面的演算法 2：

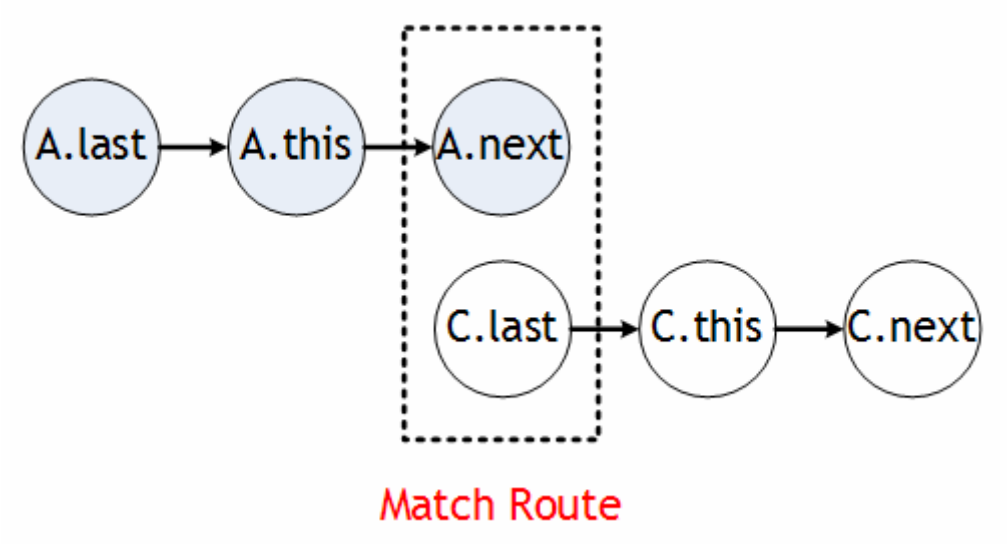


圖 4-6: 確認路徑上的下下站節點

//演算法 2：模糊比對前後站節點資訊，取出可能節點

1. 取得指定的節點。
2. 只取出該節點的後一站資訊
3. 比對節點集合中的路徑資料

```
for(i=index; i<MAXLOG ; i++)
{
    if ( pool[i].last_ip == pool[index].next_ip )
        return i; //  路徑順序符合，立即回覆該節點
}
```

4.5.3 缺少多個節點

如之前 4.1 節所提到，當攻擊者缺少兩個以上的節點，則可能無法有效的還原當時完整的路徑。作爲管理者身份，能夠擁有更多的節點資訊，可以使用刪去法去掉比較不可能的節點，配合機率的估計將可能的節點都列舉出來，配合之前提到的主動模式，在往後的時間裡事先預防，持續做追蹤。

第5章 實驗設計與結果

5.1 實驗設計

為了測試我們的匿名網路系統，我們拿目前在網際網路上應用最廣泛的網頁瀏覽來做主要的測試情境，搭配伺服器端的應用程式如：PHP等，來測試來源的IP Address是否有達到繞徑的效果；同時透過同網段相鄰的電腦使用Ethereal [14] 這套可以擷取網路上封包的軟體來檢查封包是否有加密，達到匿名與秘密通訊的效果。而在效率的測試我們採用FTP傳檔的方式，記錄其傳輸速率，比較其一般通訊與開啓匿名通訊後的差別來當作連線效能的依據。

每台節點主機皆有遠端登入的功能，並且提供檔案伺服器，網頁伺服器的服務，讓匿名連線能夠在其中做測試，盡可能提供有圖形化介面與遠端遙控（VNC [15]）可以使用web browser做測試，若只能提供純文字介面遠端登入的節點就單純拿來當中間轉傳的relay node實驗。以各節點之間的網頁瀏覽與檔案存取連線為主，測試其效能，並且記錄其log。

以下是我們的實驗環境：

作業系統：Linux/FreeBSD 等 UNIX 作業系統

應用軟體： SSHd、FTPd(vsftpd)、HTTPd(apache2.0+php5)、
WEB Browser(Firefox)、FTP Client(Firezilla/IglooFTP)、Ethereal

節點清單：

地點	IP 位址	作業系統	運算能力
交大電資	140.113.179.205	Ubuntu Linux	CPU: P3-500 Mem: 128MB
交大電資	140.113.179.206	Ubuntu Linux	CPU: P3-500 Mem: 128MB
交大電資	140.113.179.207	Ubuntu Linux	CPU: P3-500 Mem: 128MB
交大工三	140.113.166.24	FreeBSD 4.11 Stable	CPU: P4-1.6G Mem: 768MB
交大工三	140.113.166.25	Ubuntu Linux	CPU: AMD 1800+ Mem: 768MB
交大工三	140.113.166.31	FreeBSD 4.10 Stable	CPU: AMD 900 Mem: 256MB
清大資工	140.114.79.112	Ubuntu Linux	CPU: AMD 3000+ Mem: 512MB
中央資工	140.115.50.36	FreeBSD 6.2 Stable	CPU: P3-1G Mem: 256MB
中央資工	140.115.50.53	FreeBSD 6.2 Stable	CPU: P3-735 Mem: 256MB
中山電機	140.117.164.33	RedHat Linux	CPU: AMD 900 Mem: 256MB
中山電機	140.117.167.236	RedHat Linux	CPU: P3-667 Mem: 192MB
政大資科	140.119.164.48	FreeBSD 6.2 Stable	CPU: P3-1G Mem: 512MB
中興電機	140.120.108.93	RedHat Linux	CPU: AMD 3500+ Mem: 3GB
中華資工	140.126.130.33	Fedora Linux	CPU: AMD 2800+ Mem: 128MB
中華資工	140.126.130.34	Fedora Linux	CPU: AMD 2800+ Mem: 128MB
中華資工	140.126.11.2	Fedora Linux	CPU: AMD 2800+ Mem: 443MB
東華資工	134.208.3.103	Ubuntu Linux	CPU: P4 - 1.8G Mem: 256MB
東華資工	134.208.3.107	Ubuntu Linux	CPU: Celeron - 1.2G Mem: 1036MN

表格 5-1: 實驗節點清單

在網頁伺服器上，我們利用 PHP 與 apache 所提供的函數寫一個顯示伺服器端與訪客來源 IP address 與 hostname 的網頁程式，擷取其部份原始碼如下，

```
…

<body>
<p align="center" class="style1"> HTTP Test Server<br>

//顯示本機端的 domain name
<? echo gethostbyaddr($_SERVER["SERVER_ADDR"]); ?>

//顯示本機端的 IP address
(<? echo $_SERVER["SERVER_ADDR"]; ?>)</p>

<p> </p>
<p align="center" class="style2">
 You came from <span class="style3">

//顯示訪客端的 domain name
<? echo gethostbyaddr($_SERVER["REMOTE_ADDR"]); ?>

//顯示訪客端的 IP Address
(<? echo $_SERVER["REMOTE_ADDR"]; ?>)!!</span> </p>

<p align="center" class="style2"> </p>
<p align="center" class="style2">
<img src="./plaslab.jpg"></p>
```

透過這樣的網頁程式，如果我們在plaslab25.cis.nctu.edu.tw這台電腦上面使用瀏覽器，連往安裝有上面提到的網頁程式主機R516-01.eic.nctu.edu.tw這台機器上，可以得到下面 圖 5-1 的結果：可以清楚看出來我們最後是經由plaslab25.cis.nctu.edu.tw這台電腦連往R516-01.eic.nctu.edu.tw這台網頁伺服器。

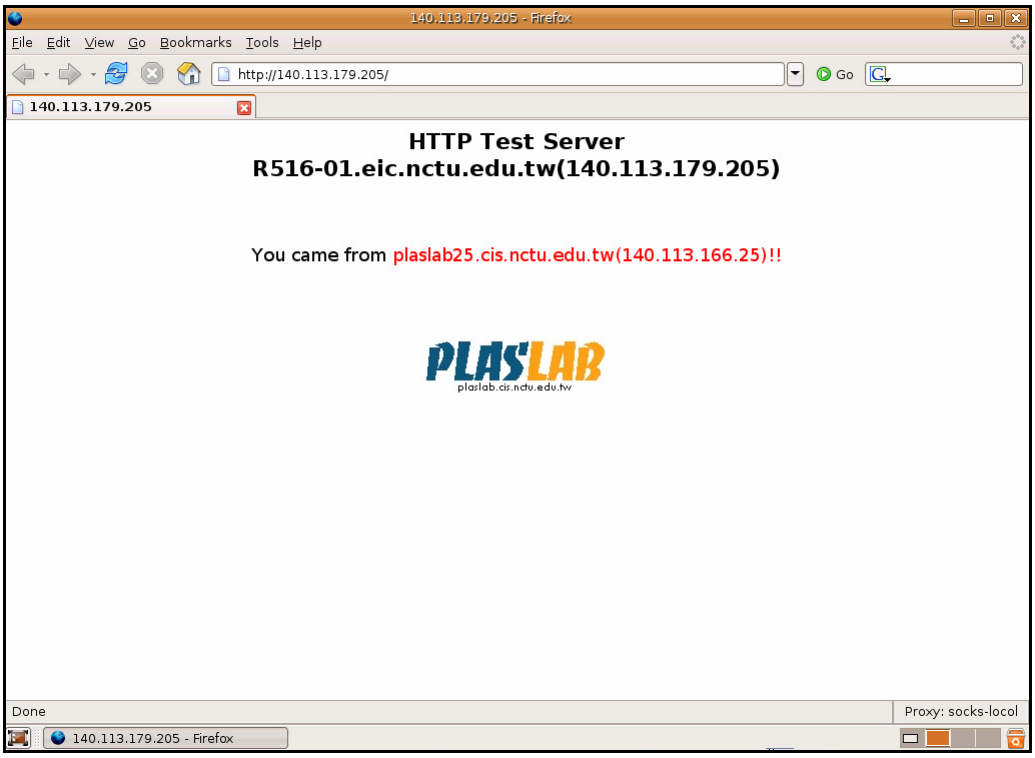


圖 5-1: 顯示訪客來源的網站

5.2 匿名性分析

以實際的情況來看，使用者 A 從 140.114.79.112 這部電腦，透過我們的匿名網路系統想要跟 B（140.120.108.93）這台網頁伺服器做通訊，此時 A 與 B 之間互相知道對方的存在，可是中間幫忙轉傳的節點只知道經過該站的前一站與後一站，並不知道連線的兩端是誰。在一次的通訊裡，由於 Tunnel 是由發送端，也就是使用者 A 來決定，我們從中記錄下來這次的連線是由：

140.114.79.112 → 140.113.166.25 → 140.113.179.206 → 140.113.179.207 → 140.120.108.93 (目的地)

並且使用者A在網頁上可以看到如 圖 5-2 的結果，顯示經由繞徑中的最後一站 140.113.179.207 連上該網站，顯示其匿名的效果：

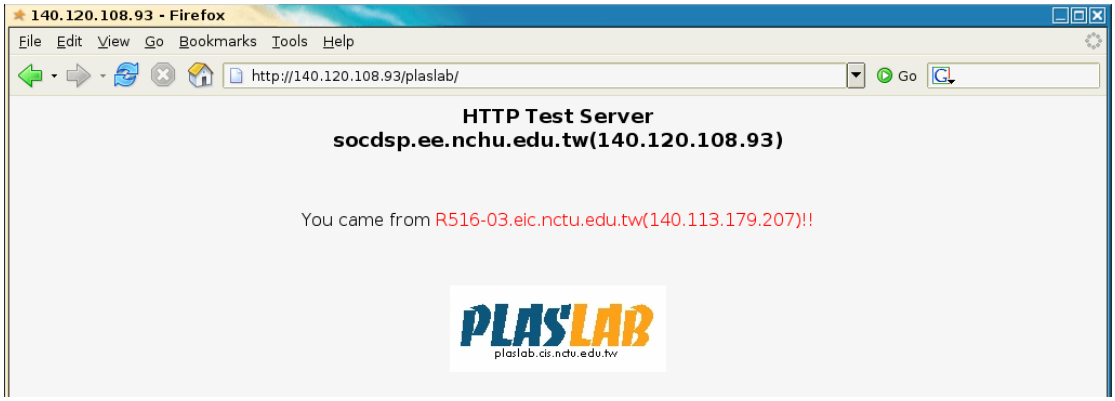


圖 5-2: 訪客來源位址測試（從 140.114.79.112 上抓圖）

而我們在路徑其中一個節點 140.113.166.25 上面以Ethereal來聽取正在做匿名傳輸的封包內容，並且測試其資料加密與連線來源隱藏的效果。以下是在做匿名通訊但是尚未啓動加密時，由Ethereal來聽取 140.113.166.25 與該路徑下一站 140.113.179.206 之間的其中一個封包內容的結果如 圖 5-3 所示：

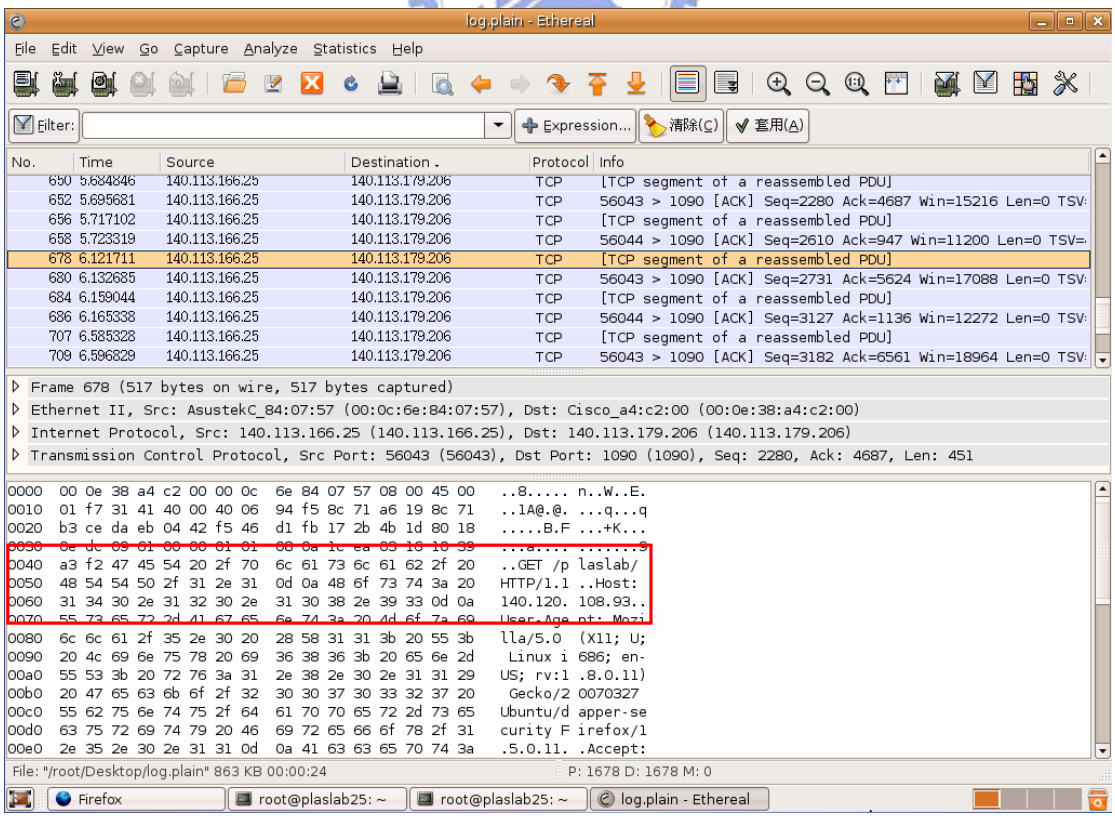


圖 5-3: 尚未啓動匿名通訊的封包內容

在紅色框框處可以看出使用者A正以明文的方式將http的指令送往目的端網頁伺服器 140.120.108.93 的內容，雖然已經有達到繞徑的效果，但是在不加密的情況下如果有攻擊者在中間被動的聽取封包，通訊的內容還是會有可能洩漏出來。此時啓動Onion加密的機制後，如 圖 5-4，一樣在由 140.113.166.25 這個節點聽取往下一站（140.113.179.206）時，可以看出紅色框框內，傳輸的訊息都是已經加密過的內容。

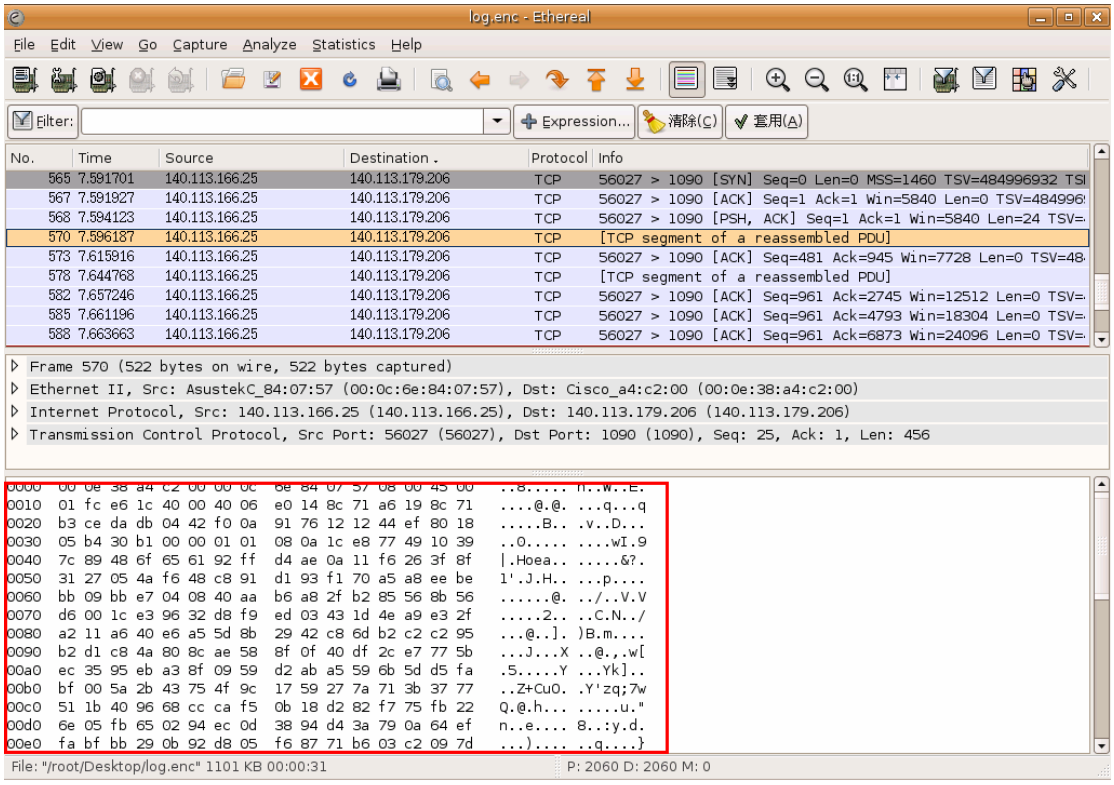


圖 5-4: 啓動匿名通訊後的封包內容

5.3 傳輸效能結果

如附錄的介紹，我們在 Linux 上使用 IglooFTP PRO 這套 FTP 傳輸軟體，並搭配 TSOCKS 來支援我們的匿名通訊。傳輸的效能分析目的在測試不使用匿名通訊，使用匿名通訊但不加密，與使用加密機制這三者間頻寬與流量的變化與比較，以下是我們實驗的結果：

實際傳送的路徑爲：

140.113.166.25 (IglooFTP + TSOCKS)

→ 140.113.179.205 → 140.113.179.206 → 140.113.179.207 →

140.113.166.24 (FTP Server)

中間轉傳的三個節點的運算能力如 表格 5-2 所示，節點位址皆在交大校內。

地點	IP 位址	作業系統	運算能力
交大電資	140.113.179.205	Ubuntu Linux	CPU: P3-500 Mem: 128MB
交大電資	140.113.179.206	Ubuntu Linux	CPU: P3-500 Mem: 128MB
交大電資	140.113.179.207	Ubuntu Linux	CPU: P3-500 Mem: 128MB

表格 5-2: 匿名傳輸效能測試節點列表

以兩種檔案大小 179MB與 1035MB，分別記錄未加密直接傳輸、未加密匿名繞徑三站、加密一層、兩層與三層匿名繞徑三站傳輸速率，在下午 3 點開始與晚上 10 點開始分別做兩次實驗，記錄三次取平均後結果如 表格 5-3：

加密 / 檔案大小	未加密 直接傳輸	未加密 匿名繞 3 站	加密 1 層 匿名繞 3 站	加密 2 層 匿名繞 3 站	加密 3 層 匿名繞 3 站
179 MB (15:00)	10.19 MB/s	2.04 MB/s	0.59 MB/s	0.54 MB/s	0.50 MB/s
179 MB (22:00)	10.01 MB/s	2.00 MB/s	0.53 MB/s	0.51 MB/s	0.48 MB/s
1035 MB (15:00)	9.97 MB/s	1.92 MB/s	0.56 MB/s	0.53 MB/s	0.50 MB/s
1035 MB (22:00)	9.82 MB/s	2.00 MB/s	0.65 MB/s	0.54 MB/s	0.48 MB/s

表格 5-3: 匿名傳輸效能結果

由結果可發現，整體上白天（15:00）的傳輸速度稍微較晚上（22:00）快一點，推測跟網路使用的尖峰時間有關係，但影響不大。由傳輸速度可知，受到單純繞徑後頻寬的影響，大約剩下 20%，加密一層後剩下 6%，加密兩層剩下 5%，加密三層後更只有剩下 4%。

加密 / 檔案大小	未加密 直接傳輸	未加密 匿名繞 3 站	加密 1 層 匿名繞 3 站	加密 2 層 匿名繞 3 站	加密 3 層 匿名繞 3 站
179 MB	17 sec	85 sec	293 sec	319 sec	345 sec
1035 MB	100 sec	515 sec	1520 sec	1831 sec	1977 sec

表格 5-4: 匿名傳輸時間結果

若由時間來看，179MB 的檔案在未加密直接傳輸所花的時間平均爲 17 秒，而透過匿名通訊繞徑三站後傳輸時間變爲 85 秒，也就是說有 (85-17)/85 = 80% 的時間是花在轉傳上面。加密一層、兩層與三層所花的傳輸時間分別爲 293 秒、319 秒與 345 秒，所以其加密所增加的負載百分比分別爲 (293-85)/85 = **245%** 與 (319-85)/85 = **275%**，(345-85)/85 = **306%**由此可見得加密的效率佔傳輸速率一個很大的因素，而檔案大小影響的成份較小。

因此若把中間節點換運算能力較高如 表格 5-5 再作同樣的實驗結果記錄在 表格 5-6，實際傳送的路徑爲：

140.113.166.25 (交大)(IglooFTP + TSOCKS)
→ 140.114.79.112(清大) → 140.120.108.93(中興) → 140.126.130.33(中華)
→ 140.113.166.24 (交大)(FTP Server)

地點	IP 位址	作業系統	運算能力
清大資工	140.114.79.112	Ubuntu Linux	CPU: AMD 3000+ Mem: 512MB
中興電機	140.120.108.93	RedHat Linux	CPU: AMD 3500+ Mem: 3GB
中華資工	140.126.130.33	Fedora Linux	CPU: AMD 2800+ Mem: 128MB

表格 5-5: 匿名傳輸效能測試節點列表 2

加密 / 檔案大小	未加密 直接傳輸	未加密 匿名繞 3 站	加密 1 層 匿名繞 3 站	加密 2 層 匿名繞 3 站	加密 3 層 匿名繞 3 站
179 MB (15:00)	10.19 MB/s	3.97 MB/s	2.44 MB/s	1.71 MB/s	1.66 MB/s
179 MB (22:00)	10.01 MB/s	3.99 MB/s	2.48 MB/s	1.71 MB/s	1.66 MB/s
1035 MB (15:00)	9.97 MB/s	4.11 MB/s	2.42 MB/s	1.71 MB/s	1.66 MB/s
1035 MB (22:00)	9.82 MB/s	4.13 MB/s	2.47 MB/s	1.71 MB/s	1.66 MB/s

表格 5-6: 匿名傳輸效能結果 2

由於節點運算速率提昇的關係，雖然連線路徑繞的比較遠，但是未加密單純繞徑的傳輸速度較前一次的實驗的 20% 提昇到 40%；加密一層、兩層與三層也分別從 6%、5% 與 4% 提升到 24%、17% 與 16%。

加密 / 檔案大小	未加密 直接傳輸	未加密 匿名繞 3 站	加密 1 層 匿名繞 3 站	加密 2 層 匿名繞 3 站	加密 3 層 匿名繞 3 站
179 MB	17 sec	44 sec	71 sec	101 sec	104 sec
1035 MB	100 sec	240 sec	400 sec	561 sec	578 sec

表格 5-7: 匿名傳輸時間結果 2

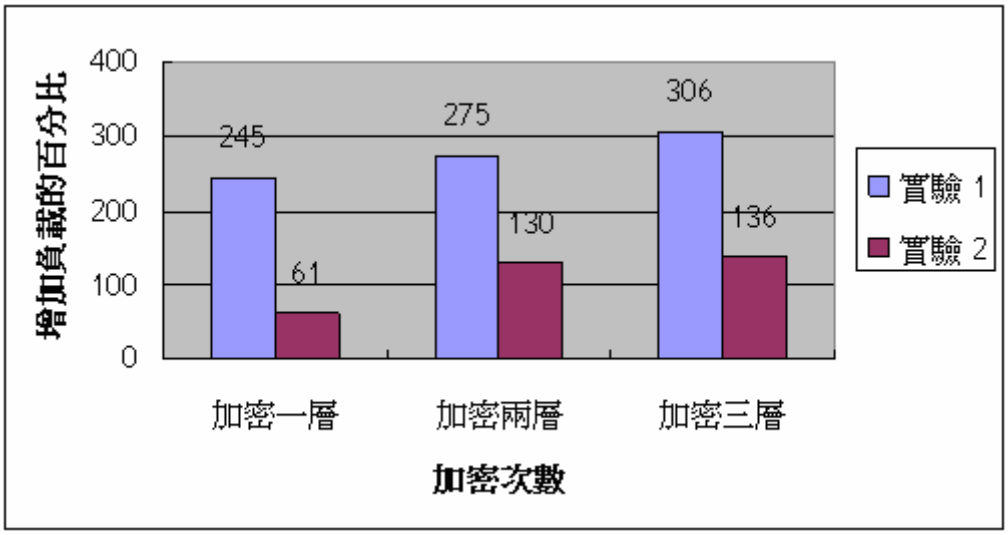


圖 5-5: 加密負載比較圖

另外從傳輸時間的角度來看，1035MB 的檔案透過匿名通訊繞徑三站後傳輸時間縮短爲 44 秒，也就是說花在轉傳上面的時間從原本的 80% 降爲 (44-17)/44 = 61%。而加密一層、兩層與三層所花的傳輸時間也分別縮短爲 71 秒、101 秒與 104 秒，所以其加密所增加的負載也從原本的 **245%**、**275%**與 **306%**降爲 (71-44)/44 = **61%**、(101-44)/44 = **130%** 與 (104-44)/44 = **136%**。可見其加解密的效率對傳輸的速率有相當程度的影響。

另外在單純加密標頭繞徑三站的模式下，傳輸速度與未加密繞徑三站的速度相差不大，在實驗一與實驗二的傳輸速度平均爲 2MB/s 與 4MB/s，可見只加密標頭的負載並不高。

5.4 路徑追蹤結果

被動模式方面我們觀察 2007/07/14 這一天的傳輸，各節點的 Log 經過壓縮統一回報傳輸資訊之後，從 Static Subscribe Server 端所收集到的資料量與實際的情況：

```
[plaslab@plaslab24:~/anonymity]> ls -lh
-rw-r--r--  1 plaslab plaslab  617 Jul 14 03:13 140.113.166.24.070714.tgz
-rw-r--r--  1 plaslab plaslab  600 Jul 14 03:14 140.113.166.25.070714.tgz
-rw-r--r--  1 plaslab plaslab  723 Jul 14 03:27 140.113.166.31.070714.tgz
-rw-r--r--  1 plaslab plaslab  666 Jul 14 03:14 140.113.179.205.070714.tgz
-rw-r--r--  1 plaslab plaslab  744 Jul 14 03:14 140.113.179.206.070714.tgz
-rw-r--r--  1 plaslab plaslab  660 Jul 14 03:14 140.113.179.207.070714.tgz
-rw-r--r--  1 plaslab plaslab  823 Jul 14 04:22 140.114.79.112.070714.tgz
-rw-r--r--  1 plaslab plaslab  562 Jul 14 03:23 140.115.50.53.070714.tgz
-rw-r--r--  1 plaslab plaslab  708 Jul 14 03:29 140.117.164.33.070714.tgz
-rw-r--r--  1 plaslab plaslab  690 Jul 14 03:26 140.119.164.48.070714.tgz
-rw-r--r--  1 plaslab plaslab  588 Jul 14 03:19 140.120.108.93.070714.tgz
-rw-r--r--  1 plaslab plaslab  719 Jul 14 04:15 140.126.130.33.070714.tgz
-rw-r--r--  1 plaslab plaslab  774 Jul 14 04:15 140.126.130.34.070714.tgz
```

解壓縮經過合併排序後，得到一個整合的資料集 log.070714.pool，其欄位分別是 integer 型態的 Date, Last Hop, Next Hop, Serial Number1, Serial Number2，列出前十筆資料如下：

```
[plaslab@plaslab24:~/anonymity]> head -10 log.070714.pool
1184351737   2356258329   2356301680   2356258335   6089324      2042332805
1184351737   2356258335   2356261838   2356301680   864151640    1349845072
1184351737   2356301680   2356258335   2356261838   2042332805   864151640
1184351746   2356258329   2356261837   2356359733   1118755894   1559195214
1184351746   2356261837   2356359733   2356519969   1559195214   2053226419
1184351746   2356359733   2356519969   2356258328   2053226419   97216093
1184351751   2356258329   2356359733   2356258335   1901543769   2053310454
1184351751   2356258335   2356651056   2356520940   1280979534   1901526437
1184351751   2356359733   2356258335   2356651056   2053310454   1280979534
1184351761   2356258329   2357101089   2356261838   549163492    1711694990
```

假設某天我們想要查詢 2007/07/14 這天 03:05:45，140.114.79.112 這台電腦正在跟誰做匿名通訊，我們可以透過 Static Subscribe Server 上的查詢程式來達到目的，實際的步驟如下：

選擇日期與欲查詢的IP Address如 圖 5-6，然後可以選擇Trace的方向是要往前（Forward）或是往後（Backward），就會列出目前可能的節點，選擇可能的時間點後系統會去比對前後站與Serial Number的關係把路徑還原出來：

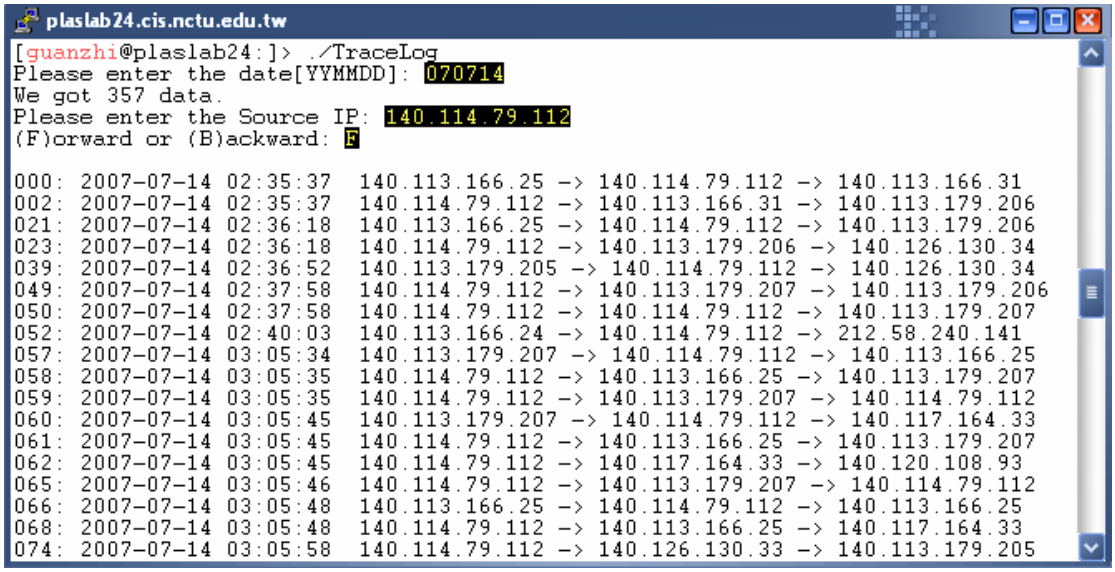


圖 5-6: 輸入還原路徑參數

選取可能時間範圍內的節點後，即可經由連線記錄還原出當時的連線路徑，如 圖 5-7 所示，可以得到原來的傳輸路徑爲：

140.114.79.112 → 140.113.166.24 → 140.126.130.33 → 140.119.164.48 → 140.117.167.236

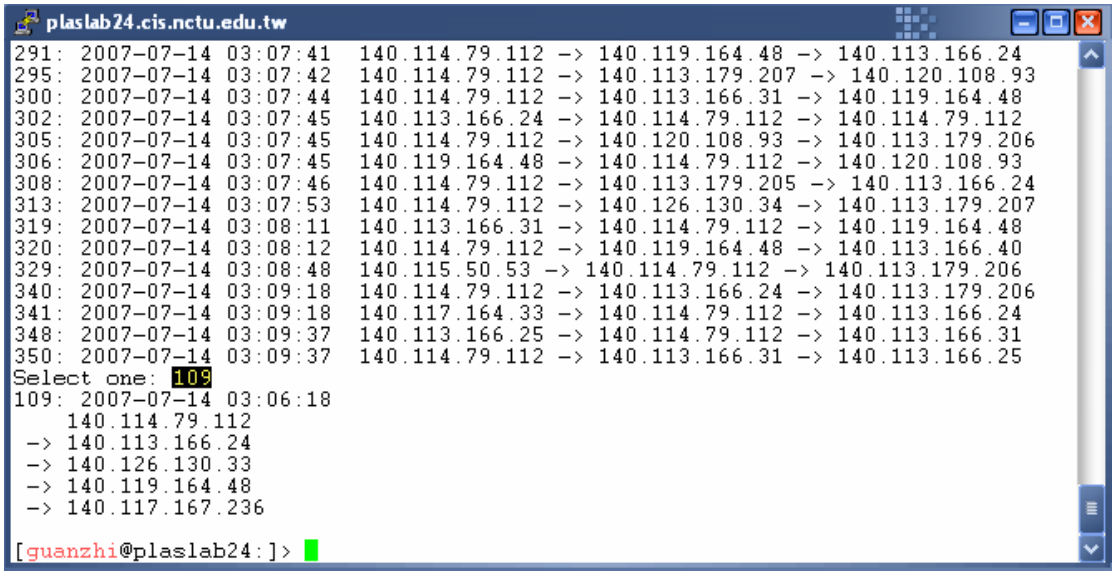


圖 5-7: 還原路徑結果

在主動模式方面，我們事先鎖定特定目標，進行匿名連線後，立即偵測其匿名連線的完整路徑，測試其回報路徑的功能。例如我們鎖定 2007/07/17 這天下午 3 點鐘左右，如果有來自 IP Address 是以 140.114 開頭的連線，立刻回報，以下是實際實驗的狀況：

如 圖 5-8 所示，輸入想要追蹤的時間點與範圍，這裡設定爲 2007/07/17 的 15:00，時間範圍是前後各 30 分鐘。目標IP Address可以是確定的位址，也可以像圖中的範例一樣設定爲一個子網域（140.114.0.0/16）之後會產生一組主動追蹤的條件，格式爲：前兩欄爲時間的上界與下界，後兩欄爲IP Address的範圍。傳送到各節點去請他們幫忙注意是否有符合的連線發生。然後啓動EventServer 等待各節點回報狀況。

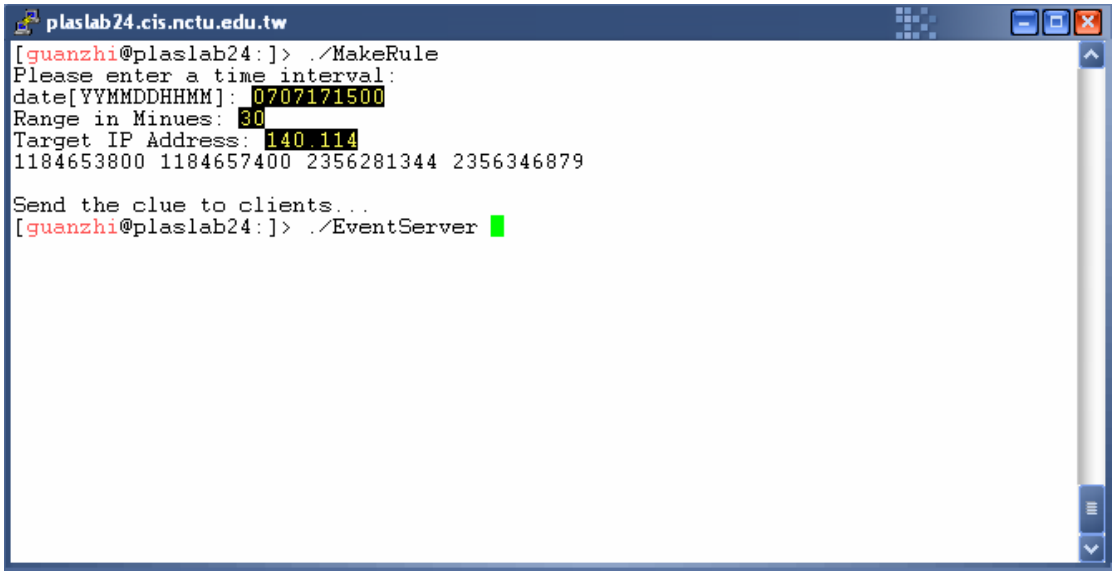


圖 5-8: 設定主動追蹤條件

當各節點設定好之後開始正常傳輸，在某次連線裡如 圖 5-9 所示：有兩個節點 140.113.179.205 與 140.117.167.236 分別在 14:52 的時候即時回傳他們收到符合的連線資訊，所以我們從這兩筆資料即時可以確定出連線的部份路徑：

140.113.166.25 → 140.113.179.205 → 140.114.79.112 → 140.117.167.236 → 140.119.164.48

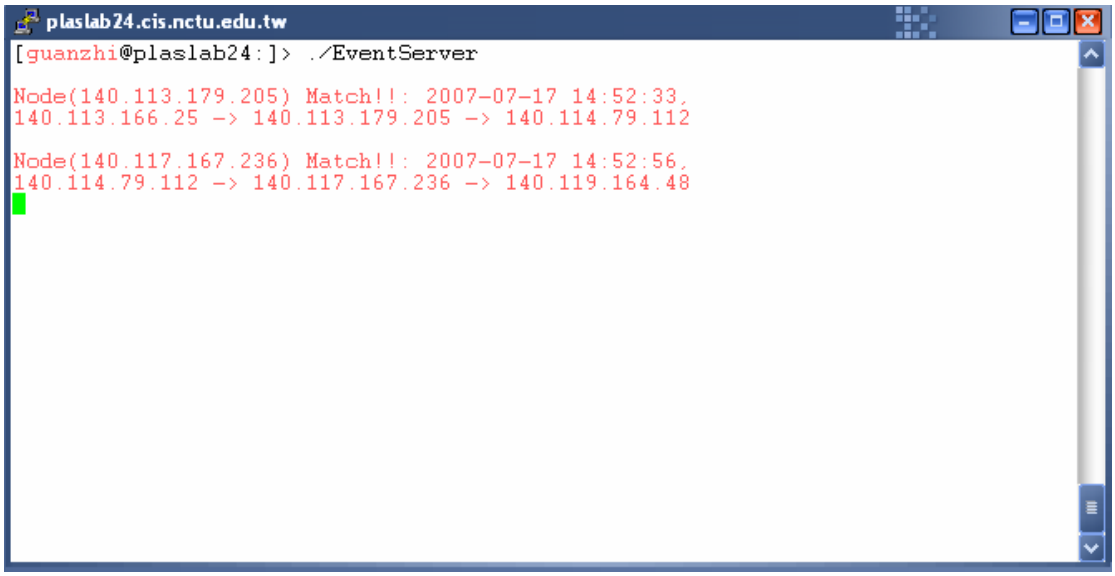


圖 5-9: 主動追蹤模式回報結果

5.5 不完整路徑資訊追蹤結果

在測試不完整路徑時，我們延續之前 5.4 節 的結果先將其中一個節點 140.120.130.33 的連線資訊移除，不讓該節點回報給 Static Subscribe Server，測試其分析的結果。

圖 5-10 所示上面的紅色方框裡顯示確定的路徑爲 140.114.79.112 → 140.113.166.24 → 140.126.130.33，之後可能的連線如下面的紅色方框裡，列出所有的可能性，依照時間先後排序，分別在將他們往後的路徑找出來，越前面的路徑可能性越高。

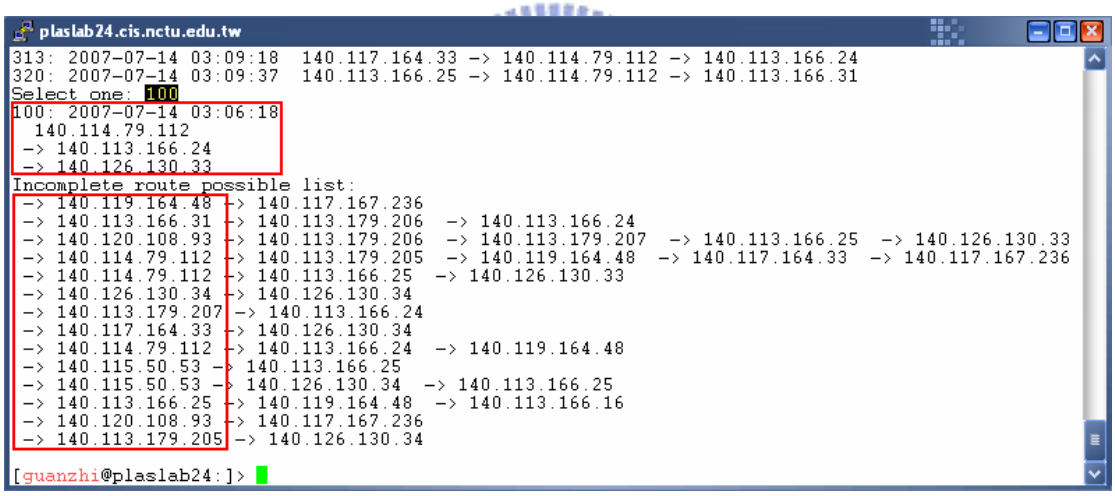


圖 5-10: 不完整路徑資訊追蹤結果

第6章 結論與未來展望

6.1 系統特點

由前面章節的介紹與實驗可以得知，我們的系統可以提供連線加密的機制，加強網路通訊的安全性，並且同時藉由 Onion 的機制來達到將連線的路徑與封包的標頭分離的效果，使得第三者的連線分析與竊聽的難度提高。並且透過 SOCKS4a 的標準通訊協定能夠讓現有的應用程式例如瀏覽器或是檔案傳輸得到匿名通訊的支援。

在管理與路徑分析方面，我們的系統在有需要的情況下，也能夠提供管理者做事後分析路徑還原機制。若管理者掌握了部份的可疑資訊，也能夠透過事先預測的模式，先將線索發送給各節點，一旦發生符合線索的連線行為則能夠立即採取行動，讓管理更加有彈性。

6.2 未來發展與改進方向

對於目前的系統我們嘗試提出未來可以發展或是改進的方向：

節點傳輸效率

實作在應用層相較於網路層來說效率較為低，除了期待未來又更新更快的硬體或是加密演算法之外，在節點的效率值除了參考本機 Benchmark 與頻寬之外，未來可以加入考慮各節點之間的頻寬，使得匿名路徑的規劃上更有效率，傳輸速度也可望有所提昇。

節點記錄資訊種類

目前我們記錄的資訊只有時間與上下站的位址，若能夠更進一步記錄連線類型或是其他更詳細的資料，則會對路徑還原與監控更加有效率，但同時可能也會降低使用者的匿名性。

時間同步問題

在分散式架構中，各自以自己系統時間記錄連線資訊，在整合的時候若時間無法有效同步則會造成路徑還原的時候，判斷可能會有誤差，尤其是在連線記錄缺少節點或是同個時間範圍內有大量連線的時候影響會更大。目前系統的作法是每個節點定時去跟同一台 Time Server 作對時，未來可以朝更有效統一各節點時間的方向做改善。

圖形介面改進

目前我們的系統只提供純文字的使用者介面，在路徑還原的節點選取與呈現對一般的使用者而言還不夠直觀，未來可以朝向更友善的圖形化介面來發展，讓程式的可用性更高。

參考文獻

[1]. 張志誠, “匿名網路系統之架構與金鑰管理”, 國立交通大學資訊科學與工程研究所碩士論文

[2]. Chen-Mou Cheng; H.T. Kung; Koan-Sin Tan; Scott Bradner, “ANON: an IP-layer anonymizing infrastructure”, in *proceedings DARPA Information Survivability Conference and Exposition*, Volume: 2 , 22-24 April 2003

[3]. Michael K. Reiter; Aviel D. Rubin, “Crowds: anonymity for Web transactions”,ACM Transactions on Information and System Security (TISSEC) Volume 1 , Issue 1 (November 1998)

[4]. Marc Rennhard; Bernhard Plattner, “Introducing MorphMix: peer-to-peer based anonymous Internet usage with collusion detectionWorkshop On Privacy In The Electronic Society.”, Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society.

[5]. Paul Syverson, “Onion routing for resistance to traffic analysis”, in *proceedings DARPA Information Survivability Conference and Exposition*, 2003.Volume 2, 22-24 April 2003 Page(s):108 - 110 vol.2

[6]. Michael J. Freedman; Robert Morris, “Tarzan: a peer-to-peer anonymizing network layer”, Conference on Computer and Communications Security. Proceedings of the 9th ACM conference on Computer and communications security.

[7]. Mixmaxter, http://mixmaster.sourceforge.net/

[8]. Ying-Da Lee, *SOCKS: A protocol for TCP proxy across firewalls* http://www.socks.nec.com/protocol/socks4.protocol

[9]. OpenSSL, http://www.openssl.org/

[10]. SSLeay, http://www.columbia.edu/~ariel/ssleay/index.html

[11]. 彭垂業、許志行、楊武 (2005)，網路秘密通訊機制，*Proc. 2005 National Computer Symposium*, Tainan, Taiwan, December 2005.

[12]. SocksCap, http://www.socks.nec.com/sockscap.html

[13]. TSOCKS, http://tsocks.sourceforge.net/

[14]. Ethereal, http://www.ethereal.com/

[15]. RealVNC, http://www.realvnc.com/

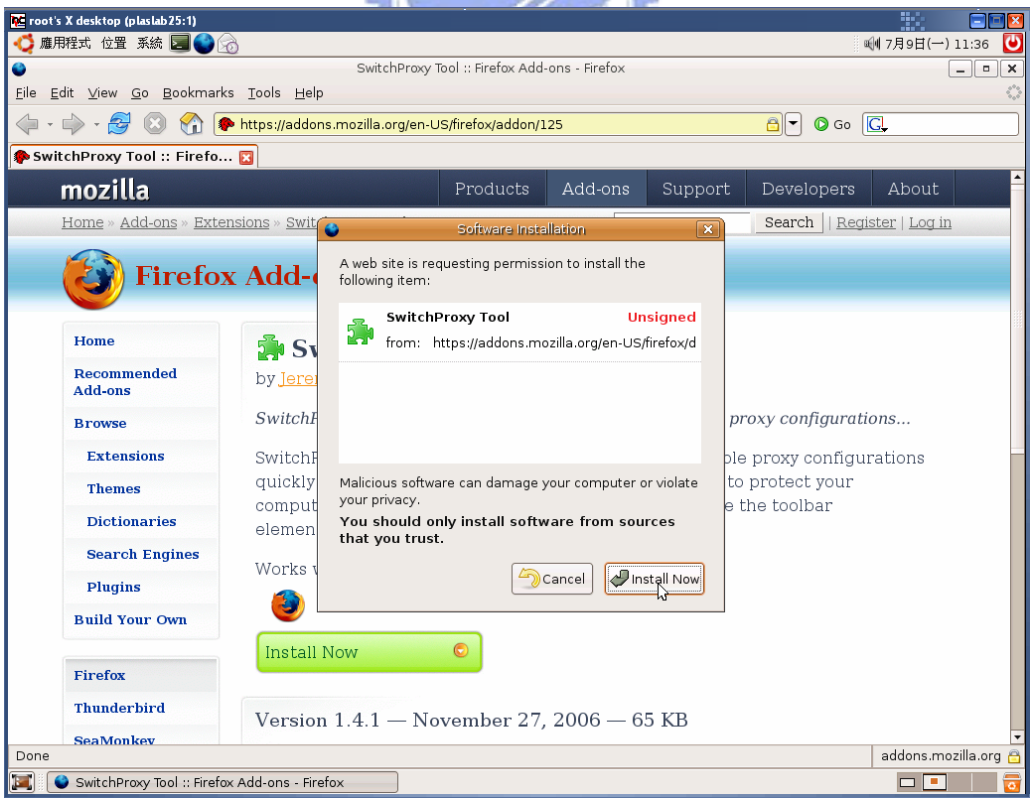
附錄: 透過 Socks Server 做匿名通訊

介紹在 Linux 的環境下使用 Firefox 瀏覽器搭配 SwitchProxy 套件與 Filezilla 經由 tsocks 透過 SocksServer 做匿名通訊：

A. Firefox 網頁瀏覽器

1. 安裝 SwitchProxy 套件

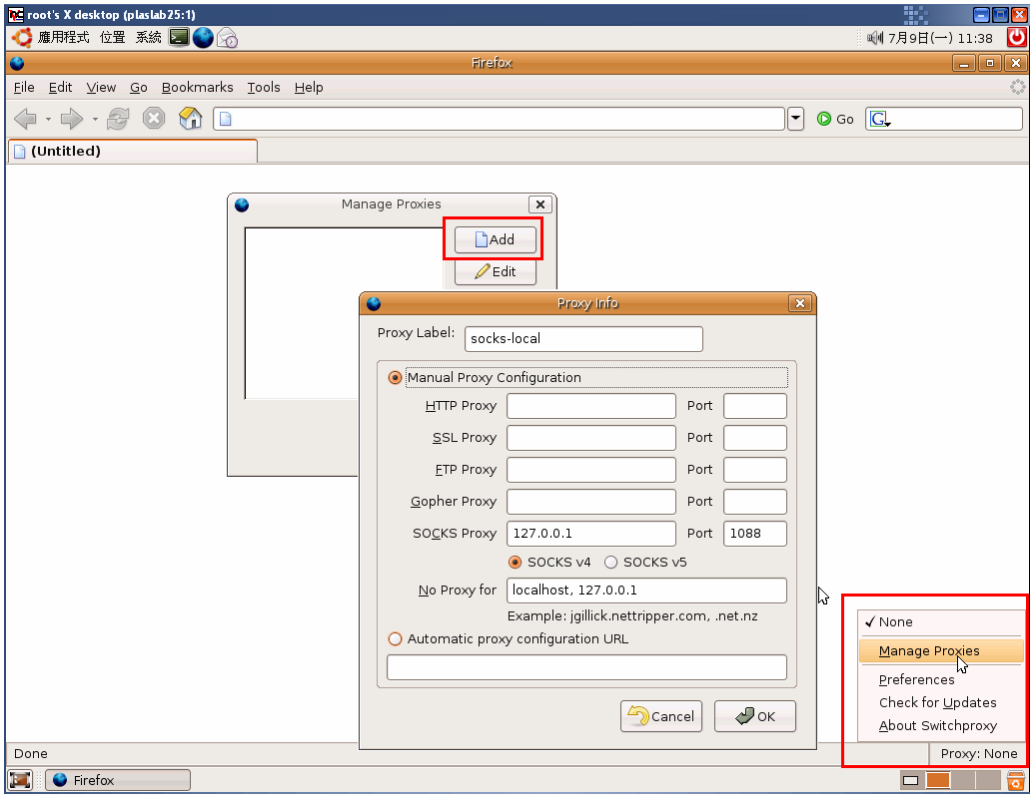
如圖，用 Firefox 開啓 https://addons.mozilla.org/en-US/firefox/addon/125 直接點選 Install Now 連結，線上安裝 SwitchProxy。



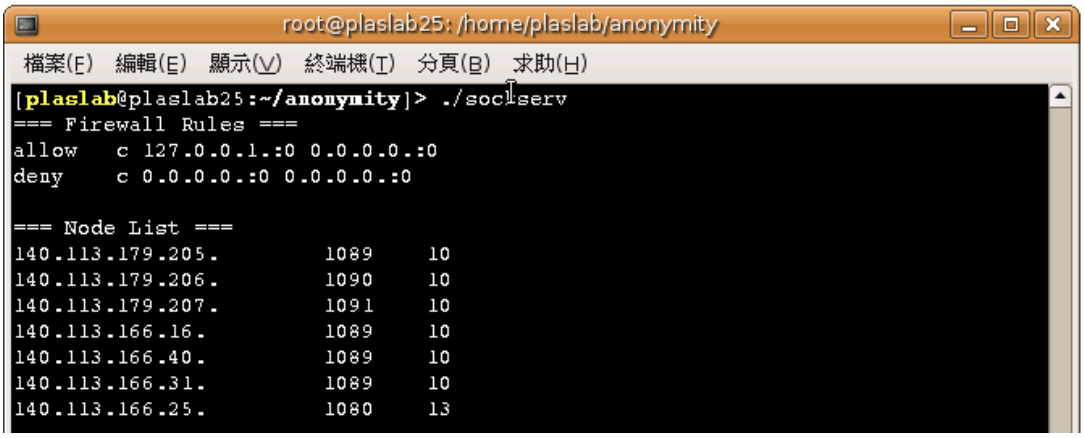
安裝完成後重新啓動 Firefox 讓套件生效。

2. 設定 SwitchProxy

如圖，在視窗右下角的 Proxy 標籤案右鍵選 Manage Proxies，新增 Proxy 設定，將 SOCKS Proxy 設定爲本機端 127.0.0.1，並選擇類型 SOCKS v4。

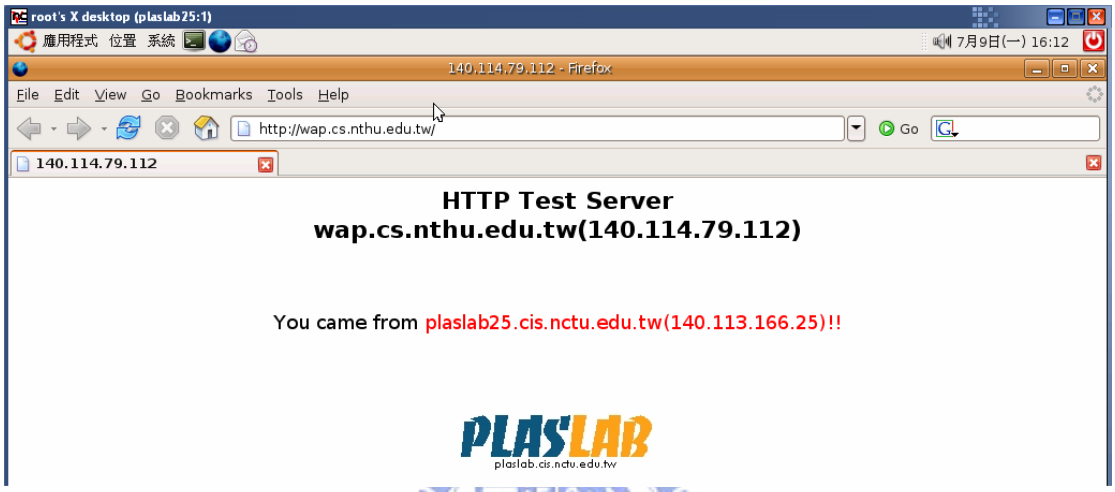


3. 啓動 Socks Server

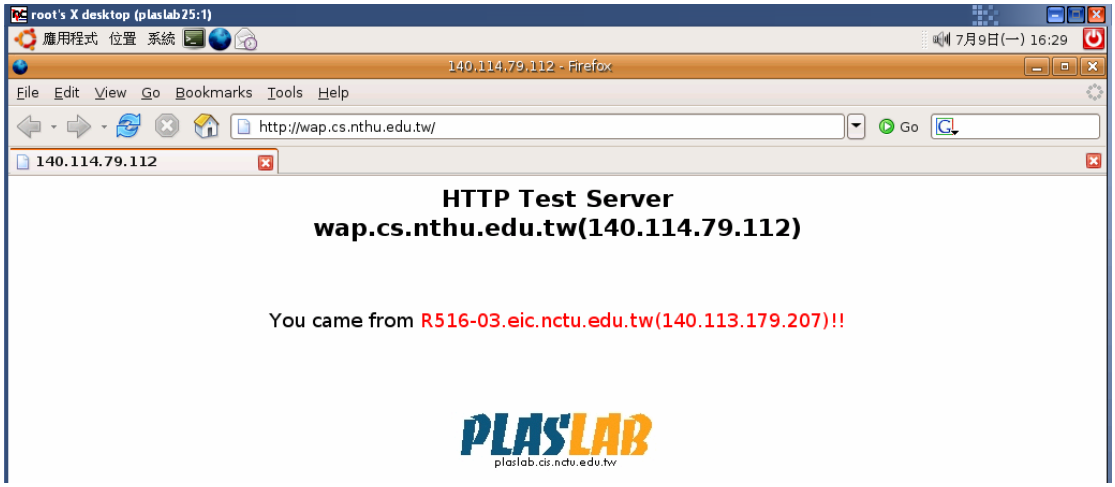


4. 開始匿名通訊

連往測試網站 http://wap.cs.nthu.edu.tw，在正常的模式下可以看出來源爲目前的本機端位址 plaslab25.cis.nctu.edu.tw



啓動匿名通訊後，可看到來源位址變成來自交大電資館的節點，網域名稱爲 R516-03.eic.nctu.edu.tw

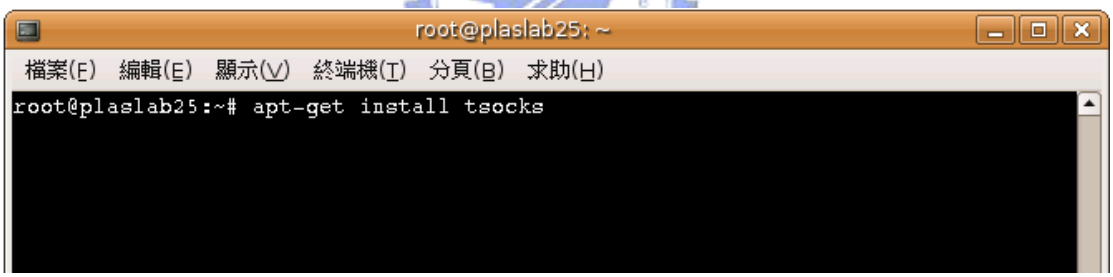


B. Filezilla 檔案傳輸

Linux 上的 FTP 軟體例如 Filezilla3 並沒有直接支援 SOCKS 通訊協定，必須要在透過 TSOCKS 的幫忙，間接啓動 Filezilla3，TSOCKS 的做法是去攔截應用軟體底層 connect() 的呼叫，在建立連線之前先幫它透過 SOCKS 通訊協定，再傳出去。以下是實際的步驟：

1.安裝 TSOCKS

到 TSOCKS 的官方網站的下載區 http://tsocks.sourceforge.net/download.php 抓取原始碼回來 compile 後安裝，或是在 ubuntu Linux 下面可以使用 apt 套件管理直接下指令“apt-get install tsocks”安裝：



2.設定 /etc/tsocks.conf

安裝完成 TSOCKS 後必須手動設定未來要連線的 SOCKS Server 的連線資訊，用文字編輯器開啓/etc/tsocks.conf 這個設定檔，修改以下欄位後存檔：

```
server = 127.0.01

server_port = 1088

server_type = 4
```

3.安裝 Filezilla

Linux 版本的 Filezilla 可以到 http://filezilla-project.org/nightly.php ，選擇 i586-linux-gnu 的版本，如下圖點選 Download 連結

Target	Status	Download
i586-linux-gnu	Successful	Download
i586-mingw32msvc	Successful	Download #1 Download #2
i686-apple-darwin8	Successful	Download
powerpc-apple-darwin8	Successful	Download

會得到一個 Filezilla3.tar.bz2 的壓縮檔，使用 bzip2 的解壓縮工具來解壓縮，指令爲： “ bzip2 -cd Filezilla3.tar.bz2 | tar xvf - “ 會得到一個 FileZilla 的資料夾。

4.透過 TSOCKS 啓動 Filezilla 做匿名通訊

切換目錄到 Filezilla/bin 下，如下圖執行指令： ” tsocks ./filezilla”

即可使用 Filezilla 透過 Socks Server 來做匿名通訊。