國立交通大學

資訊科學系

碩 士 論 文

在 網 路 上 的 秘 密 通 訊 機 制 之 研 究

The secret and private network communication

研 究 生：彭垂業

指導教授：楊　武　教授

中 華 民 國　九 十 四　年 六 月

在網路上的秘密通訊機制之研究
The secret and private network communication

研 究 生：彭垂業　　Student：Chui-Yeh Peng

指導教授：楊　武　　Advisor：Wuu Yang

國立交通大學

資訊科學系

碩士論文

A Thesis

Submitted to Institute of Computer and Information Science

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer and Information Science

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

在網路上的秘密通訊機制之研究

學生：彭垂業　　指導教授：楊 武 博士

國立交通大學資訊科學研究所

摘要

現今的網路環境最大的缺失就是安全性不足，因此近年來成為熱門的研究方向。而大部分改善網路安全性的問題，是使用 Secret Key 對 Packet 內部 data 的部份做加密。此方法可以讓第三者無法得知 packet 的內容，但是卻可經由 IP Header 得知某兩端正在通訊，沒有隱密性。因此希望網路環境不但有安全性也要有隱密性。

本論文的研究目標，是使用 FreeBSD 開放原始碼的特性，將現有的 TCP/IP 做一些修改，並不需要修改現有的應用程式，來達成網路的隱密性。所謂的隱密性是利用混淆的方式，透過其他的電腦來轉送 packet，讓第三者很難去判斷 packet 真正的目的地。最後的實作結果，系統可以正常運作，效能也有不錯的表現。

The secret and private network communication

Student : Chui-Yeh Peng Advisor : Dr. Wuu Yang

Department of Computer and Information Science

National Chiao Tung University

Abstract

One of the most significant deficiencies of the network environment nowadays is lacking of security. It becomes, therefore, a popular research topic in the recent years. Of the resolutions to improve internet security, a large proportion of them are using Secret Key to encrypt the data included in the packets. This method is able to prevent the content of the packets against being obtained by third party, but it lacks of privacy because of the communication between certain two computers would be exposed via IP Header. Thus a network environment with not only security but privacy is well expected.

The research goal of this thesis is using the traits of free source code in the FreeBSD to modify existing TCP/IP without making over existing applications for achieving anonymity on internet. The so-called “Anonymity” is using a confusing strategy by means of relaying packets through other computers to make third party difficult to evaluate the exact destination of packets. The final result of tentative runs demonstrated that the system is capable of working smoothly and has good performance on efficiency

致謝

首先我要感謝我的指導教授 楊武博士的教導，在研究上指引我正確的方向以及在論文寫作給我很多的指導，使我能夠順利完成這篇論文。

接著我要感謝實驗室的學長、同學以及室友給我很大的鼓勵與建議。最後要感謝我的家人給我經濟上與精神上的幫助與支持，使我能夠無後顧之憂地專注於研究，順利完成我的碩士學位。

目錄

圖表目錄

第一章 緒論

1.1 研究動機

近年來網際網路最大的問題就是安全性太薄弱，對於現今的電子交易，會在網際網路上傳送重要且隱私的個人資料，因此網際網路的安全性是越來越重要。而絕大多數都是使用密碼學來解決安全性的不足，並且都針對於 Data 的部份作保護，對於兩端通訊者的身份卻還是暴露在網際網路上。

因此希望在現有的網際網路架構上，能提供雙方通訊間的隱密性。所謂的隱密性，是不要讓第三者輕易的就能得知通訊者身分。以現有網際網路架構上，只能用混淆的方式來增加困難度。

1.2 研究目標

利用 FreeBSD 開放原始碼的特性，將原本的 TCP/IP 做修改來達成所要的目的。對於混淆的方式，能透過其他特定的電腦，藉由在它們之間來傳遞封包，達成混淆的效果，而在這些特定電腦的接收與傳送封包使用 FreeBSD 所提供的函式庫來達成。在隱藏的部份，透過對稱式加密法的方式來達成，使用 OpenSSL 所提供相關的加解密函式庫。

秘密通訊機制要有以下特性：

1. 能保有原來的 TCP/IP 的連線方式，與修改過的 TCP/IP 的連線方式，兩者並存，且能互相切換。
2. 能保有 TCP 的所有機制。
3. END TO END 的架構。
4. Sender 自己決定路徑，並且能自動更新路徑。
5. Application 不須要更改，即可使用秘密通訊。

1.3 系統 overview

系統分成三個主要部份，分別為 Host、Onion Server、Onion Router。兩端 Host 在進行秘密通訊時，透過 Onion Router 的協助，來混淆兩端 Host 封包的目的位址。Onion Server 管理 Onion Router 並且協助 Host 取得 Onion Router 的資訊。如圖 1.1。

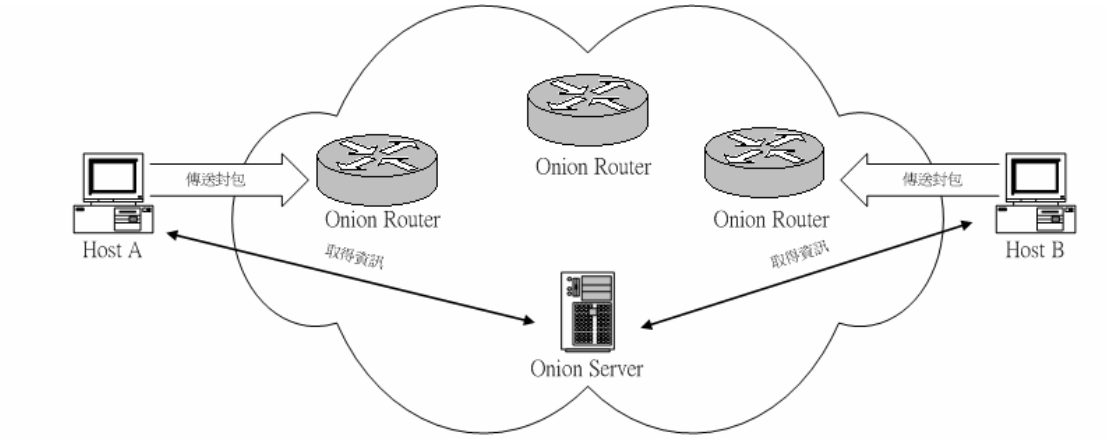

圖 1.1 秘密通訊系統架構圖

1.4 論文架構

本論文一共分為五個章節，第一章為緒論，對整篇論文做一個簡單的介紹。第二章將會介紹與論文有關的各項研究，其中主要為關於 Mbuf 的使用與 Raw Socket 的運用。第三章則說明系統的架構、設計的原理與系統如何實作。第四章則是實作的結果，並且對系統效能與正常模式做比較。第五章為本篇論文的總結，並且對未來的發展提出一些意見。

第二章 相關研究

2.1 Raw Socket

Raw Socket 可以直接存取在 Network layer 的部份資料，因此可以讓我們收送特定的封包。

2.1.1 Raw Socket 概論

Raw Socket 提供三個正常 TCP 與 UDP socket 所沒有個功能：

1. Raw Socket 提供我們讀寫 ICMP、IGMP 封包。
2. 可讓 kernel 不處理的 IPv4 protocol 欄位之 IP 封包，透過 Raw Socket 來處理 ，由於大部分 kernel 只處理 protocol 欄位值為 1(ICMP)、2(IGMP)、6(TCP)、和 17(UDP)，因此我們可以藉由 Raw Socket 來處理非上述 4 項 protocol 的 IP packet。
3. 利用 Raw Socket，可以使用 IP_HDRINCL SOCKET 選項，建構自己的 IP Header。因此可利用此項特性來建立自己想要的封包格式。

2.1.2 Raw Socket 的產生

產生 Raw Socket 需要以下幾個步驟：

1. 將 socket 函式的第二個參數設為 SOCK_RAW。第三個參數設為想要的 Protocol 的值，例如 TCP 就設為 6。

```
int sockfd;
sockfd = socket(AF_INET, SOCK_RAW, protocol);
```

2. 在 setsockopt 函式的第三個參數設為 IP_HDRINCL，表示要自行建造 IP Header。第四個參數為此資料的起始位置。

```
const int on = 1;
```

```
if(setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on))<0)
error
…
```

2.1.3 Raw Socket 的輸出

Raw Socket 的輸出有以下幾個規則：

1. 正常的輸出是呼叫 sendto 函式。第二個參數為要傳送的資料，會直接送到 kernel 處理，第三個參數為此資料的長度，第四個參數為目的端。

```
sendto (sockfd, sendbuf, len, dst, dstlen);
```

2. 如果沒有設定 IP_HDRINCL 選項，kernel 會自行建構 IP Header，並且會附在傳給它的資料(sendto 函式的第二個參數)之前。

3. 如果有設定 IP_HDRINCL 選項，則要傳給 kernel 的資料開頭必須是 IP Header。因此必須要自行建構整個 IP Header，除了以下兩個欄位以外：
   (1) 識別欄位可以設成 0，表示讓 kernel 去設定這個值。
   (2) Kernel 一定會去計算並且儲存檢查碼。

4. 如果封包的長度超過離開介面的 MTU，kernel 會加以分割。

2.1.4 Raw Socket 的輸入

Kernel 收到封包後，對於 Raw Socket 有以下幾個規則：

1. kernel 收到 TCP 封包與 UDP 封包絕對不會傳給 Raw Socket。
2. 大部分的 ICMP 封包在 kernel 處理完 ICMP 訊息後，都會傳給 Raw Socket。
3. 所有的 IGMP 封包在 kernel 處理完 IGMP 訊息後，都會傳給 Raw Socket。
4. 所有包含 kernel 不了解的 protocol 欄位的 IP 封包都會傳給 Raw Socket，kernel

會針對這類的 IP 封包進行最起碼的確認，包含 IP 版本、檢查碼是否正確、header 的長度以及目的 IP 位址。

5. 如果封包以 IP fragmentation 抵達，kernel 會一直等到所有的 IP fragmentations 都抵達，並且重組完成後，才會傳給 Raw Socket。
6. 如果產生的 Raw Socket 指定非 0 的 protocol(socket 函式的第三個參數)，收到封包的 protocol 欄位必須符合這個值，否則封包就不會傳給這個 Raw Socket。如果指定的值為 0，表示接受所有的 protocol 的值。
7. 如果 Raw Socket 使用 bind 函式來繫結一個本機的 IP 位址，則收到封包的目的 IP 位址必須符合這個繫結的位址，否則封包就不會傳給這個 Raw Socket。
8. 如果 Raw Socket 使用 connect 函式指定了一個遠端的 IP 位址，則收到封包的來源 IP 位址必須符合這個連線的位址，否則封包就不會傳給這個 Raw Socket。

2.2 Memory Buffer

Memory Buffer 是使用在網路上的資料結構，存放網路上所需要的資訊。

2.2.1 Memory Buffer 的簡介

Memory Buffer 又稱作 Mbuf，主要是用來存放與處理使用者的資料，從 Process 一路到網路介面，從原本的 data 一層一層的封裝到 Frame，都一直在 Mbuf 內完成的。

Mbuf 的結構如圖 2.1、2.2：

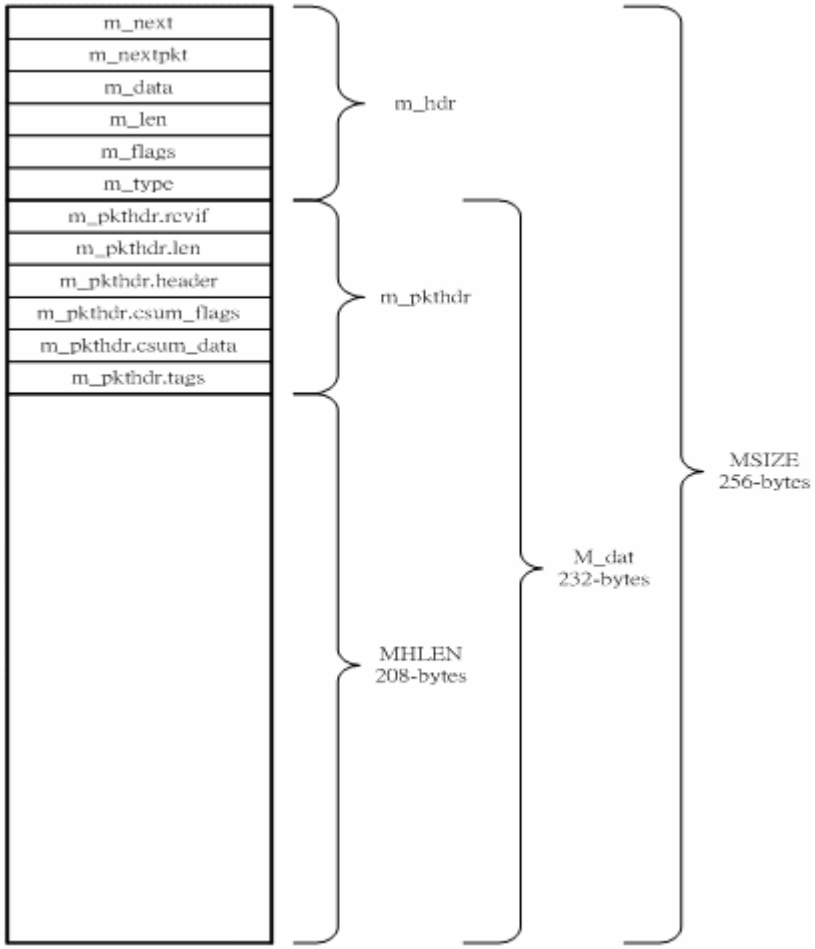

圖 2.1 Mbuf 的資料結構(一)

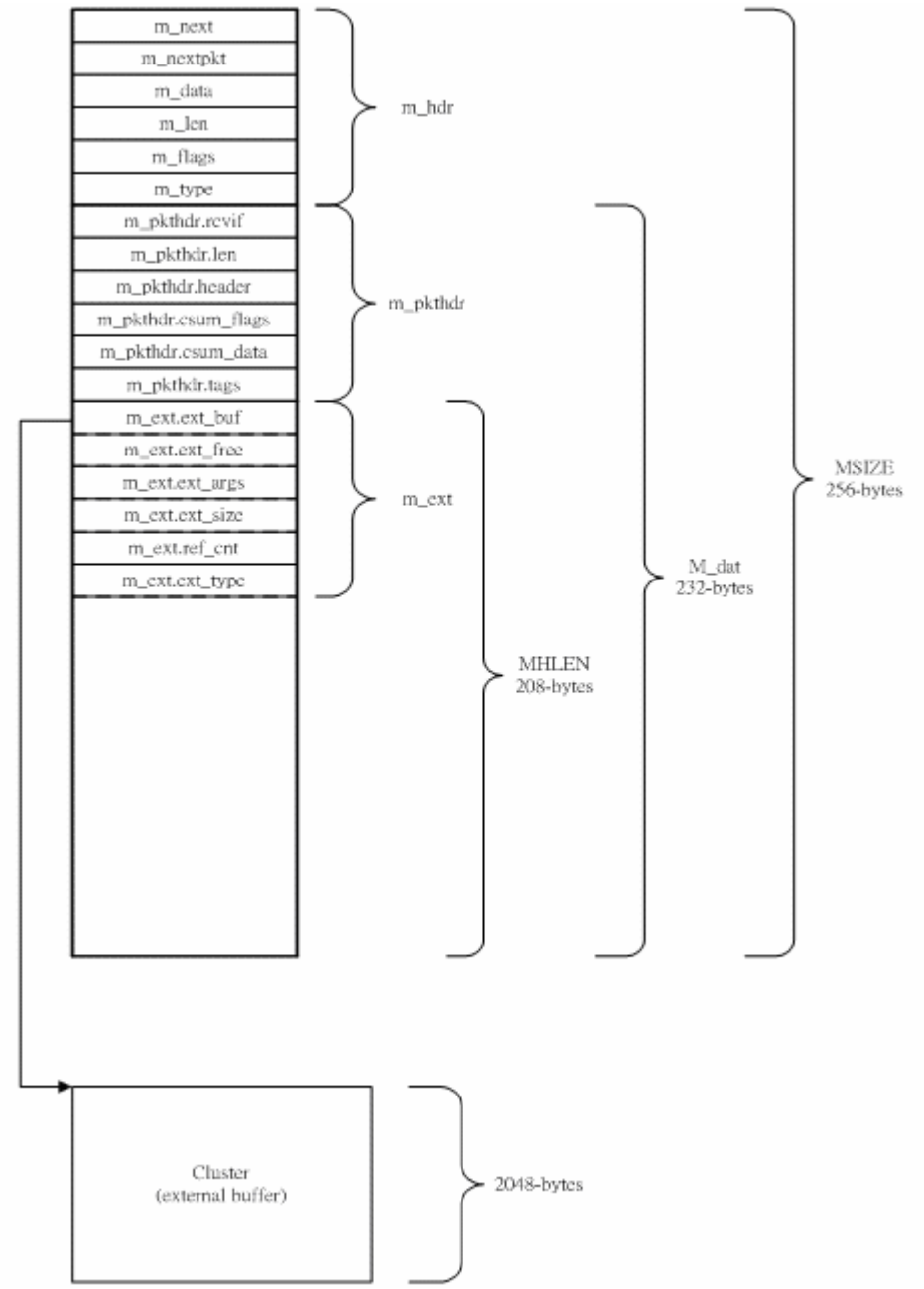

圖 2.2 Mbuf 的資料結構(二)

m_next：連結到下一塊 Mbuf，一個封包或者一段 record，可能會被放置在不同的 Mbuf 上，利用 m_next 將同一個封包或者同一段資料連結起來。這一串連結稱作 Mbuf Chain。

m_nextpkt：將多個封包或者 records 連結在一起形成一個 queue。每一個封包在 queue 內可以是單一 Mbuf 或是 Mbuf Chain。

m_data：Mbuf 內 data 的起始位址。如果 data 是在 Cluster 內部，它會指向到 Cluster

內 data 的起始位址。

m_len：Mbuf 內的 data 長度，0~232 bytes。

m_flag：

1. M_PKTHDR：表示 Mbuf 內包含 Packet Header，record 的開始點。
2. M_EOR：record 的結束點。
3. M_EXT：表示 Mbuf 有指向一個 Cluster，會將 data 放在 Cluster 內。

m_type：儲存在 Mbuf 內 data 的類型。

1. MT_DATA：動態 data 配置。
2. MT_HEADER：packet header。
3. MT_SOCKET：socket structure。
4. MT_PCB：protocol control block。
5. MT_SONAME：socket name (socket address structure)，包含 IP address 和 port number。

m_pkthdr.rcvif：送出的封包不會使用，而是接收到的封包會去指向接收此封包介面的 ifnet structure。

m_pkthdr.len：Mbuf Chain 的 data 長度，表示封包的大小。

m_pkthdr.header：指向 packet header。

m_pkthdr.csum_flags：記錄 checksum 的各種訊息。

m_pkthdr.csum_data：存放 header 內 checksum 欄位與 header 起始位置的相對距離。

m_ext.ext_buf：指向 Cluster 的起始位址。

m_ext.ext_size：Cluster 的長度。

m_ext.ext_type：Cluster 的類型。

2.2.2 **Memory Buffer 的使用**

當應用程式要傳送大小為 50 bytes 的 data 時，便將這 50 bytes 的 data 配置到

Mbuf 內。接著 transport layer，以 TCP 為例，將會再產生一塊新的 Mbuf，並且配置好 IP Header 與 TCP Header 的空間。判斷 data 是否可以容納於剩餘的空間，如果可以將會呼叫系統函式 m_copydata()，將 data 拷貝至 TCP 所產生的 Mbuf 內，如圖 2.3。最後將 Header 欄位填入，便將 TCP 所產生的 Mbuf 往下層傳送。接下來的各層，便在 Mbuf 內封裝成各層的格式。

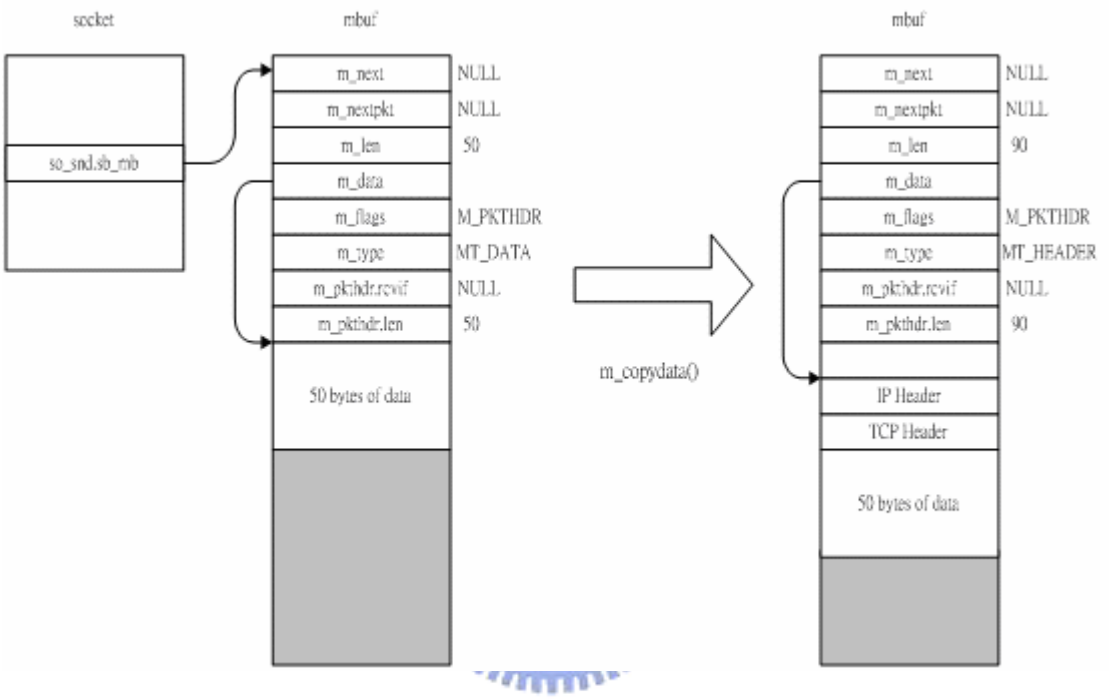

圖 2.3 Mbuf 的使用(一)

如果 data 大於剩餘空間，以下圖為例，data 的大小為 2048 bytes。在 Ethernet，IP 封包長度最大為 1500 bytes，data 必須要放置在兩個封包，分別為 1460 bytes 與 588 bytes。呼叫系統函式 m_copy()，將會產生新的 Mbuf 來存放 data，並且傳回此 Mbuf 的位址，與 TCP 所產生的 Mbuf 鏈結在一起，形成 Mbuf chain。一個 Mbuf chain 表示一個封包，在這裡會有兩個 Mbuf chain，如圖 2.4。

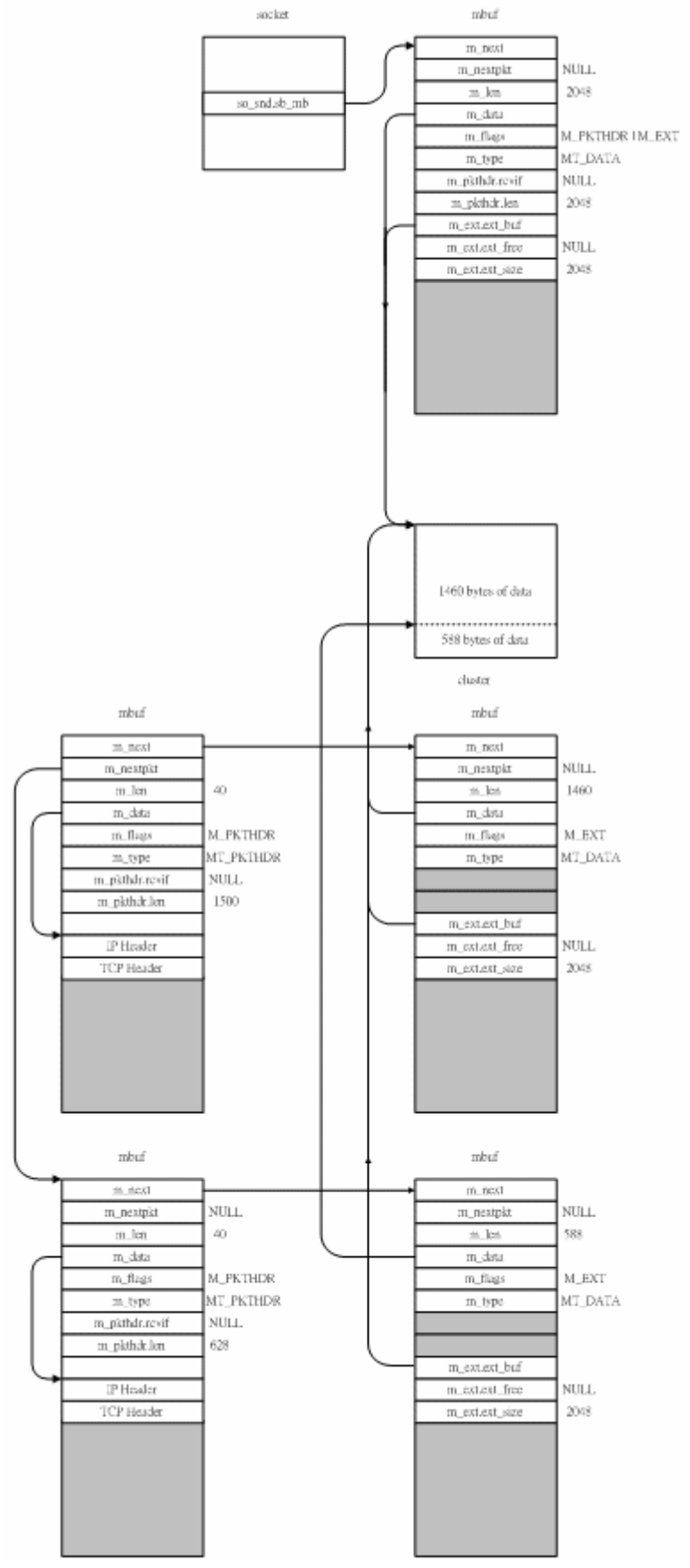

圖 2.4 Mbuf 的使用(二)

2.3 Thread

Thread 又稱為 Lightweight process，一個 process 內的所有的 thread 都會共享一個相同的全域記憶體空間，使得 thread 之間的資訊分享非常容易，但這個容易性的背後需要注意「同步」的問題。一個 process 內的所有的 thread 之間除了可共享全域變數之外，還可以共享：

- Process 的指令
- 大部分的資料
- 開啟的檔案
- 信號處理函式和信號的處置
- 目前的工作目錄
- 使用者與群組 ID

每個 thread 各自擁有的部份：

- Thread ID
- 暫存器，包括 program counter 以及 stack pointer
- 堆疊(放區域變數以及函式返回位址)
- errno
- signal mask
- 優先權

2.3.1 基本的 thread 函式

1. pthread_create

   產生一個 thread。

```
int    pthread_create (pthread_t *tid, const pthread_attr_t *attr,
                       void *(*func)(void *), void *arg);
```

   第一個參數是 thread 的 ID，當這個函式成功建立一個 thread 時，thread 的 ID 將會透過 tid 指標傳回來。

第二個參數是 thread 的屬性，優先權、初始的堆疊大小、是否為 Daemon thread 等。可以設定 pthread_attr_t 變數的值來初始化這些屬性，以取代系統的預設值。一般都可接受預設值，只需將 attr 參數設為 NULL 即可。

第三個參數是 thread 的起始函式，當建立 thread 後，會去執行此函式。Thread 的生命週期就是從呼叫這個函式開始，一直到明確的結束(呼叫 pthread_exit 函式)，或者讓起始函式返回。

第四個參數是起始函式的參數，如果這個起始函式需要多個參數，則必須要將它們包裝成一個結構，然後將結構的位址當作唯一的參數傳給此起始函式。

2. pthread_join

藉由呼叫 pthread_join 函式來等待某個 thread 的結束。

```
int    pthread_join (pthread_t    tid,    void ** status);
```

第一個參數是指定所要等待 thread 的 ID，如果要等待任何的 thread 就設為 -1。

第二個參數是此 thread 的傳回值所存放的位置。

3. pthread_self

呼叫此函式可以傳回自己的 thread ID。

```
Pthraed_t    pthread_self (void);
```

4. pthread_detach

thread 有兩種類型，joinable(預設的類型)與 detached。joinable 的 thread 結束時，它的 thread ID 和結束的狀態會一直保留著，直到 process 的其他 thread 對他呼叫 pthread_join 為止。而 detached 的 thread 一但結束，它所有的資源都會被釋放，也無法等待它的結束。

```
int    pthread_detach (pthread_t tid);
```

參數是要設為 detached 的 thread ID，通常是自己。

5. pthread_exit

呼叫 pthread_exit 函式是結束 thread 的一種方法。

```
void    pthread_exit (void *status);
```

2.4 Make

make 是個命令產生器，只要使用一個描述檔便可以建立一系列可供 UNIX shell 執行的命令。這些命令通常與軟體專案中檔案的維護有關。所謂「維護」係指整批一系列的工作，其範圍從狀態回報與暫存檔案的清除，到最後建構多組程式的可執行版本。

要處理檔案之間的相依關係，make 是最適合的工具。即使是小型的軟體專案通常也會牽涉到許多檔案，更不用說 FreeBSD 的核心，而且檔案間又會以不同的方式相互關聯著。舉例來說，一個可執行程式的產生，必須經過目的檔與函式庫的連結，而它們又分別必須先以組合語言或是高階語言加以建立。如果更動過一個以上的原始檔就必須重新編譯它們，再重新進行連結。在專案的發展過程中，通常會不斷地進行這種選擇性建構動作。Make 大為簡化了這種煩瑣的建構程序。只要針對具關聯性的所有檔案做過一個記錄，此記錄要放在 makefile 或 Makefile 檔內，這個檔案稱作描述檔，便可讓 make 自動執行所有必要的動作。

2.4.1 描述檔

撰寫一個程式，其中包含了：

- 三個 C 語言的原始檔--- main.c、iodat.c、dorun.c。
- 一個由組合語言所寫成的 lo.s，它會被某個 C 原始檔所呼叫。
- 一組置放於 /usr/fred/lib/crtn.a 的函式庫常式。

則描述檔的撰寫如下：

```
1    program: main.o iodat.o dorun.o lo.o /usr/fred/lib/crtn.a
2            cc -o program main.o iodat.o dorun.o lo.o /usr/fred/lib/crtn.a
```

```
3     main. o : main. c
4           cc -c main. c
5     iodat.o : iodat.c
6           cc -c iodat.c
7     dorun.o : dorun.c
8           cc -c dorun.c
9     lo.o : lo.s
10          as -o lo.o lo.s
```

第 1 行說明了 program 相依於 main.o、iodat.o、dorun.o 與 lo.o 檔案，以及 /usr/fred/lib/crtn.a 函式庫。第 2 行所指定的編譯器命令則可以根據其「前提」，建構 program。第 3 行指出 main.o 相依於 main.c，因此，如果 main.c 在 main.o 上一次建構之後有了更動，make 便會執行位於第 4 行的編譯命命，以便建立一個最新版本的 main.o。第 5~10 行與第 3、4 行一樣。

2.4.2 FreeBSD 的核心編譯

核心的原始檔案都位於 /usr/src/sys 目錄內，可將檔案內的原始碼進行修改。在進行編譯前，首先要產生 Makefile 檔案，必須進入 /usr/src/sys/i386/conf 目錄內，執行 config GENERIC，之後會在 /usr/src/sys/i386/compile/GENERIC 目錄內產生 Makefile 檔案。最後在/usr/src/sys/i386/compile/GENERIC 目錄內執行 make depend && make all install 進行編譯。

2.5 Anonymous Connections

在網際網路上的安全性大部分的焦點都著重於防止被竊聽，但是利用加密的方法仍然會被追蹤，會透露誰和誰在通訊。這種追蹤方式稱為 traffic analysis，可能會揭露更重要的訊息。

Anonymous Connections 使用 onion routing 的方式，來防止網路的各項服務

的竊聽與 traffic analysis。主要的技巧是讓攻擊者識別連線中的資訊更困難，例如讀取 IP Header。

Onion routing 是在 onion router 網路上動態建構一條 anonymous 路徑，應用密碼學的方法來轉變要傳送的訊息，然後傳送這些訊息到下一個目的地，並且讓每一個訊息的長度都是固定的。對於這些特定的訊息很難經由某些特定的 bits 或者大小以及前後順序來分析，更不用說能知道是誰和誰在傳輸這些訊息。

Onion proxy 是一個建構與管理 anonymous circuits 的設備，它會選擇路徑與決定 key，因此在此系統內必須信任它。它會將收到的 data streams 轉換建構成可在 onion routing 網路上辨識的獨立格式。

Data 在經過網路有三個步驟：circuit setup、data movement、circuit tear-down。首先是 circuit setup 階段，會產生 onion，onion 是一層一層的資料結構，每一層都包含路徑的資訊，包括下一站的位址與 symmetric onion key，使用 onion router 的 public key 加密。每一個 onion router 將會用它的 private key 解密它收到的 onion，便可取得 onion key 以及下一個 onion router 的位址。Onion router 必須要填塞 embedded onion 使長度維持固定大小，再往下一個 onion router 傳送。當抵達最後一台 Onion Router 之後便按照原路徑返回，便完成 circuit setup 。接著是 data movement 的階段，從 onion proxy 使用多把 onion keys 將 data 做多次的加密之後，便可以將 data 傳送出去，經過一台 onion router 就會解密掉一層，最後到達接收者就是原始 data。返回時，按照原路徑再一層一層做加密。當完成 data 的傳送便可以進入 circuit tear-down 階段，將之前設定的路徑終止。

第三章 系統設計

3.1 系統架構

系統由三個部分組合而成別是 Host、Onion Server、Onion Router。

架構圖：

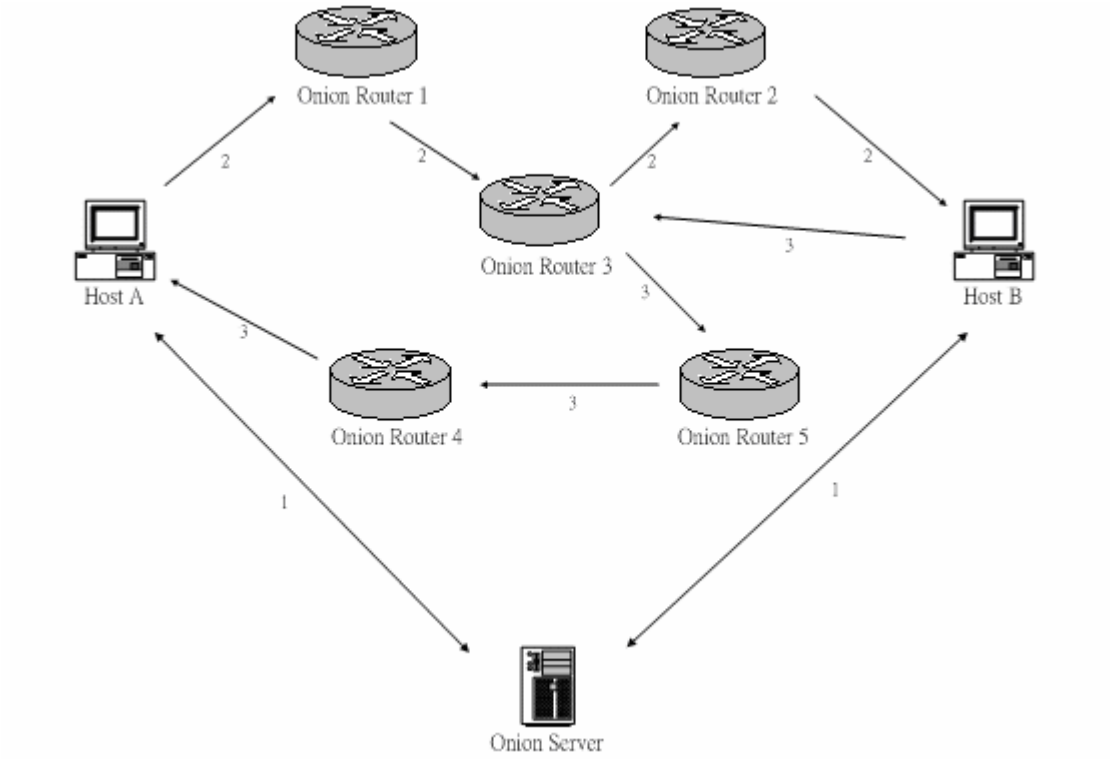

圖 3.1 秘密通訊架構圖

Onion Server：負責管理 Onion Router 的狀態。

Host：向 Onion Server 取得 Onion Router 的資訊，並且自行決定路徑。

Onion Router：負責轉送封包，由轉送的過程來混淆真正的目的端。

首先 Onion Router 必須先向 Onion Server 註冊，讓 Onion Server 得到所有 Onion Router 的資訊。接著 Host A 與 Host B 分別向 Onion Server 取得 Onion Router 的資訊，Onion Server 回應部份 Onion Router 的資訊給 Host A 與 Host B，分別是

Onion Router 1、2、3 與 Onion Router 3、4、5，如圖 3.1 的編號 1。

當 Host A 取得 Onion Router 的資訊，便可以決定封包的路徑，如圖 3.1 的編號 2，Host A → Onion Router 1 → Onion Router 3 → Onion Router 2 → Host B。Host B 決定的封包路徑為 Host B → Onion Router 3 → Onion Router 5 → Onion Router 4 → Host A，如圖 3.1 的編號 3。

3.2 封包格式

原始的 IP datagram 仍然會保留，但必須加密起來，外面再加入一層 IP Header，其中 protocol 欄位為 111 表示為秘密通訊的特定封包，而目的地 IP Address 為第一個經過的 Onion Router。外層的 IP Header 與原始的 IP Header 之間放入第二個至最後一個經過的 Onion Router 的 IP Address 與另一端 Host 的 IP Address，見圖 3.3。

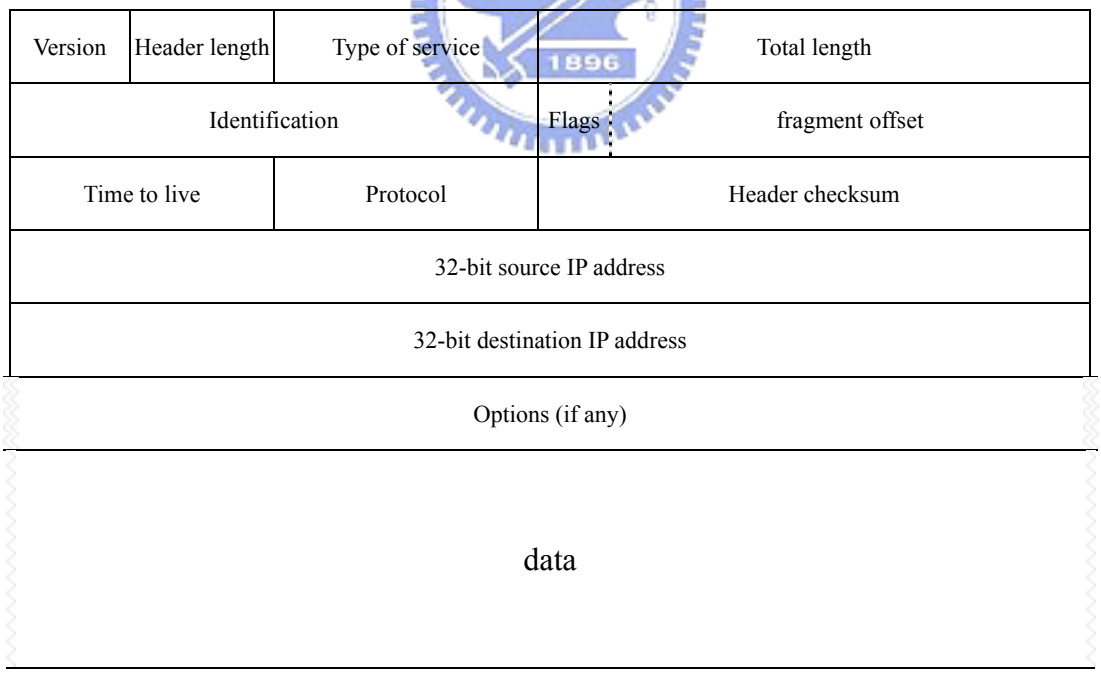

圖 3.2 IP datagram

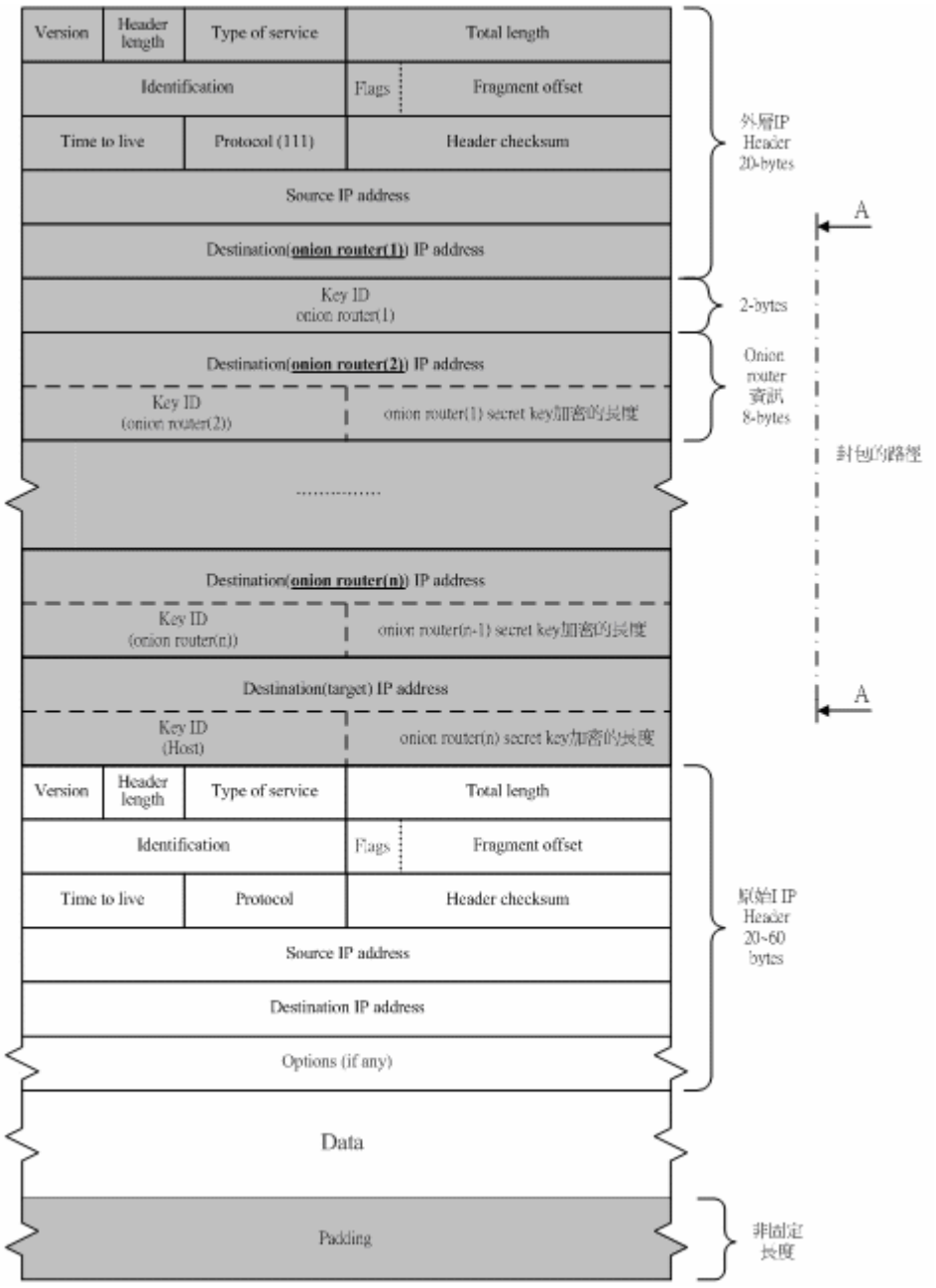

圖 3.3 秘密通訊的封包格式

灰底部分為秘密通訊增加的特定格式，白底部分為原始 IP datagram。

Destination(n) IP Address：存放下一站的 IP Address。

Key ID：識別 Onion Router 的 secret key。由於 Onion Router 的 secret key 不止一把，所以必須記錄使用哪一把 secret key 加密。

secret key 加密的長度：記錄使用 secret key 加密的長度。

3.2.1 分析安全程度

從 Host 送出的封包格式，會將最內部的原始 IP 封包使用雙方 Host 的 secret key 加密，再用 Onion Router 的 secret key 將它之後的路徑資訊與原始 IP 封包一層一層地加密，見圖 3.4。

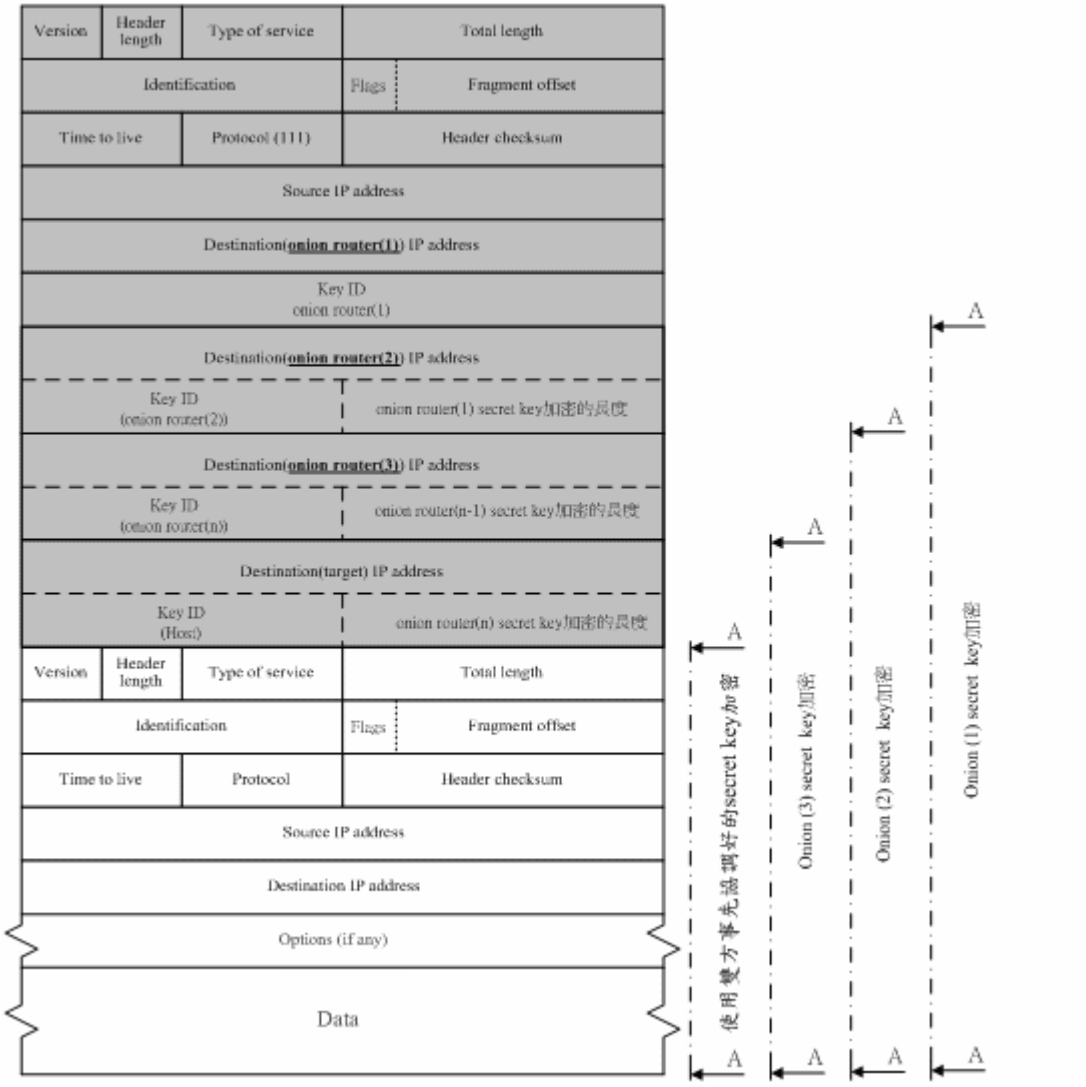

圖 3.4 Host 送出的秘密通訊封包

當 Onion Router 收到封包後，會將一層解密開，接著將封包底部亂數隨機 Padding，如圖 3.5。透過 Onion Router 對封包的處理，會使處理前與處理後的封包，payload 的部份會不一樣，封包長度也不一樣，找不出這兩個封包的相關性，而達到混淆的效果。

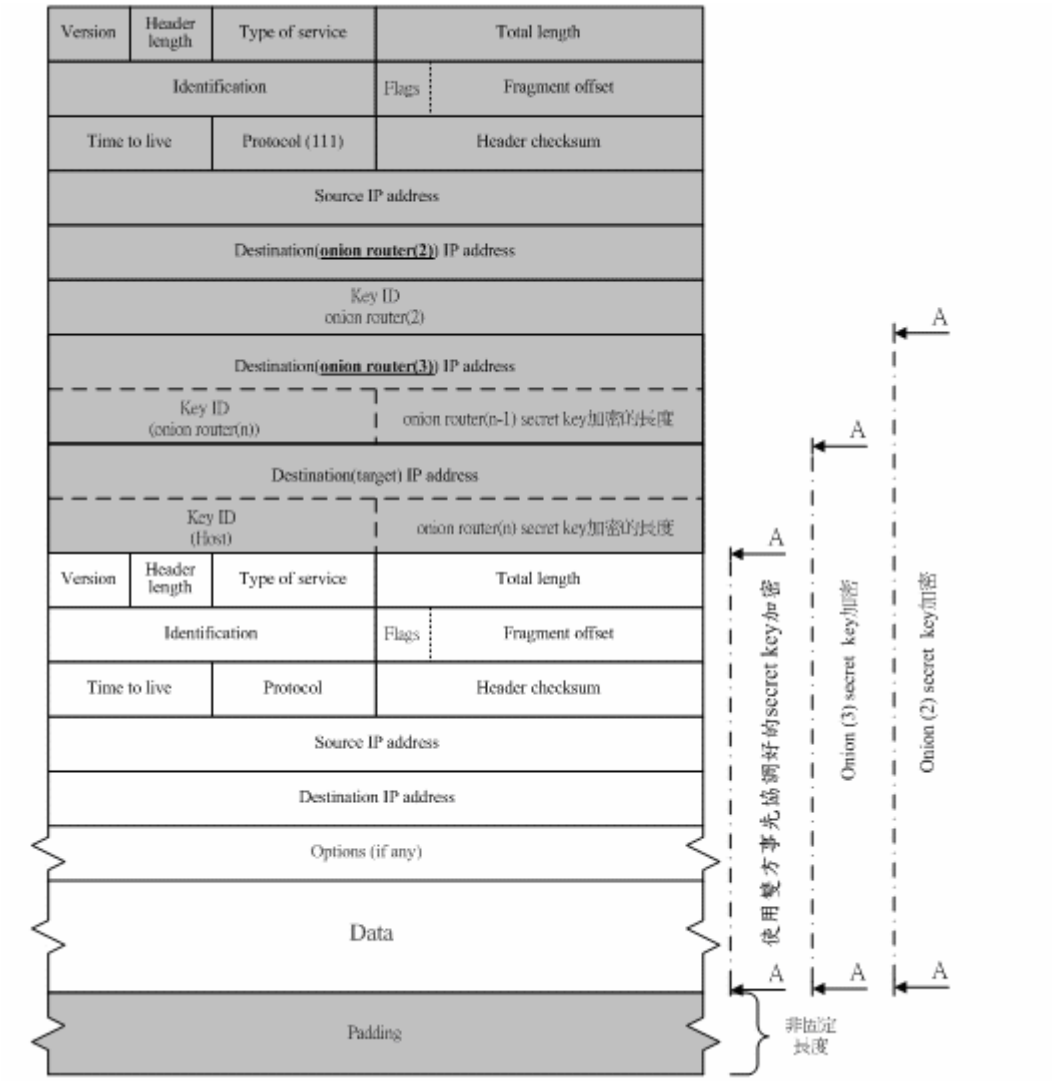

圖 3.5 Onion Router 送出的秘密通訊封包

3.3 介紹 implementation

3.3.1 系統功能

(1) 可設定正常模式或者秘密通訊模式。

當設定為 enable 時為秘密通訊模式，此時所有的封包便會以 3.2 所敘述的封包格式送出。設定為 disable 時，封包便會恢復成原始的格式。

(2) 可設定經過 Onion Router 的數目。

根據需求可設定路徑長度，Onion Router 的數目越多隱密性越高，但是效率

越差。Onion Router 的數目越少隱密性越低，但是效率越好。

(3) 動態調整 segment 大小。

當 Onion Router 的數目越多，IP 封包的長度就會越長，有可能會超過 MTU 的長度。因此根據 Onion Router 的數目，動態地設定 segment 的大小，可讓 IP 封包的長度不會超過 MTU。

3.3.2 implementation

秘密通訊使用特定的封包格式，必須修改 TCP/IP 原始碼，因此使用開放原始碼的作業系統 FreeBSD，版本為 5.1。Host 端修改原始碼的部份會在 3.4 節介紹。Onion Router 使用兩個 thread，一個 thread 使用 Raw Socket 的方式來收送秘密通訊使用特定的封包；另一個 thread 用來與 Onion Server 通訊，會在 3.5 節介紹。Onion Server 使用三種 thread，一種 thread 用來與 Onion Router 通訊，對於每一個 Onion Router 都會產生一個 thread 來與 Onion Router 通訊；另一種 thread 用來與 Host 通訊；最後一種 thread 用來檢查 Onion Router 是否 timeout，會在 3.5 節介紹。

3.4 修改原始碼

為了達成 3.3.1 的各項功能，必須在 Host 端修改 TCP/IP 相關的原始碼，分別為 ip_output.c、ip_input.c、tcp_output.c，再去重新編譯。

3.4.1 ip_output.c

當 TCP/IP 的 transport layer 將資料往 network layer 傳送，將會由 ip_output 負責後續的部份。傳送的資料以及各層的 Header 都會放在 Mbuf 的資料結構內，修改封包格式就必須從 Mbuf 開始著手。

在 transport layer，以 TCP 為例，會將所有的 TCP Header 的欄位都填完，除了 checksum 欄位只計算 PseudoHeader 的部份，其餘的部份會在 ip_output 呼叫

in_delayed_cksum 函式之後才會完全計算完成。並且填好大部分的 IP Header 的欄位，如 Version、Header length、Differentiated services、Total length、Time to live、Protocol、Source IP address、Destination IP address 等欄位。其餘的部份會在 ip_output 填入，Identification、Fragmentation offset、Checksum 三個欄位。其中 Identification、Fragmentation offset、Total length 三個欄位的值會從 host 的格式轉換成 network 的格式。

主要增加與修改的部份：

1. 首先決定 Onion Router 的數目與行走順序後，增加外層 IP Header 以及 Onion Router 的資訊。
2. 將 transport layer 未計算完全的 checksum 部份，會在 ip_output 計算完全。但是加入了外層 IP Header 以及 Onion Router 的位址，導致 Mbuf 內 TCP 或 UTP 的 checksum 欄位的相對位址改變，所以要調整 offset 的值，才不會將計算好的 checksum 填錯欄位。
3. 最後將 Onion Router 的資訊、原始的 IP Header 以及 data 進行加密。一個封包可能會有 1~3 個不等的 Mbuf 所組成，如圖 3.6-1、3.6-2、3.6-3。傳送一個 data 大小為 2072 bytes，在經過兩台 Onion Router 的情況下，會被分成三個封包。第一個封包的第一個 Mbuf 放入 76 bytes 的 Header 與 Onion Router，第二個 Mbuf 指向第一塊 cluster 的 1424 bytes 的 data，如圖 3.6-1。由於第一塊 cluster 剩餘 624 bytes 的 data，將由第二個封包的第二個 Mbuf 來配置，第三個 Mbuf 指向第二塊 cluster 的 800 bytes 的 data，與第一個 Mbuf 的 76 bytes，使封包能達到最大長度 1500 bytes，如圖 3.6-2。最後剩下 24 bytes 的 data，由於 Mbuf 的剩餘空間足夠，便將 data 放置剩餘的空間內，如圖 3.6-3。

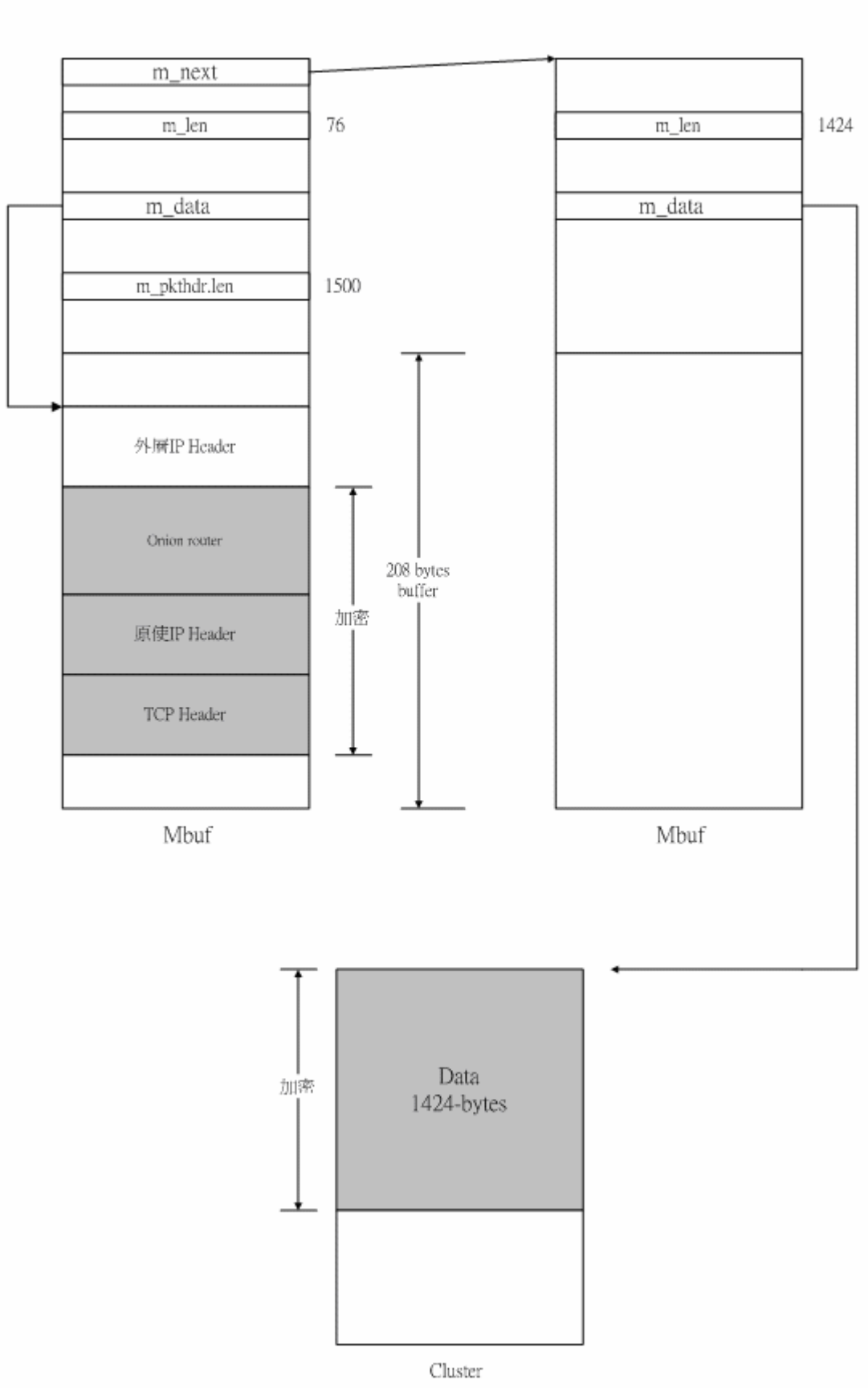

圖 3.6-1 系統內的秘密通訊封包，類型一

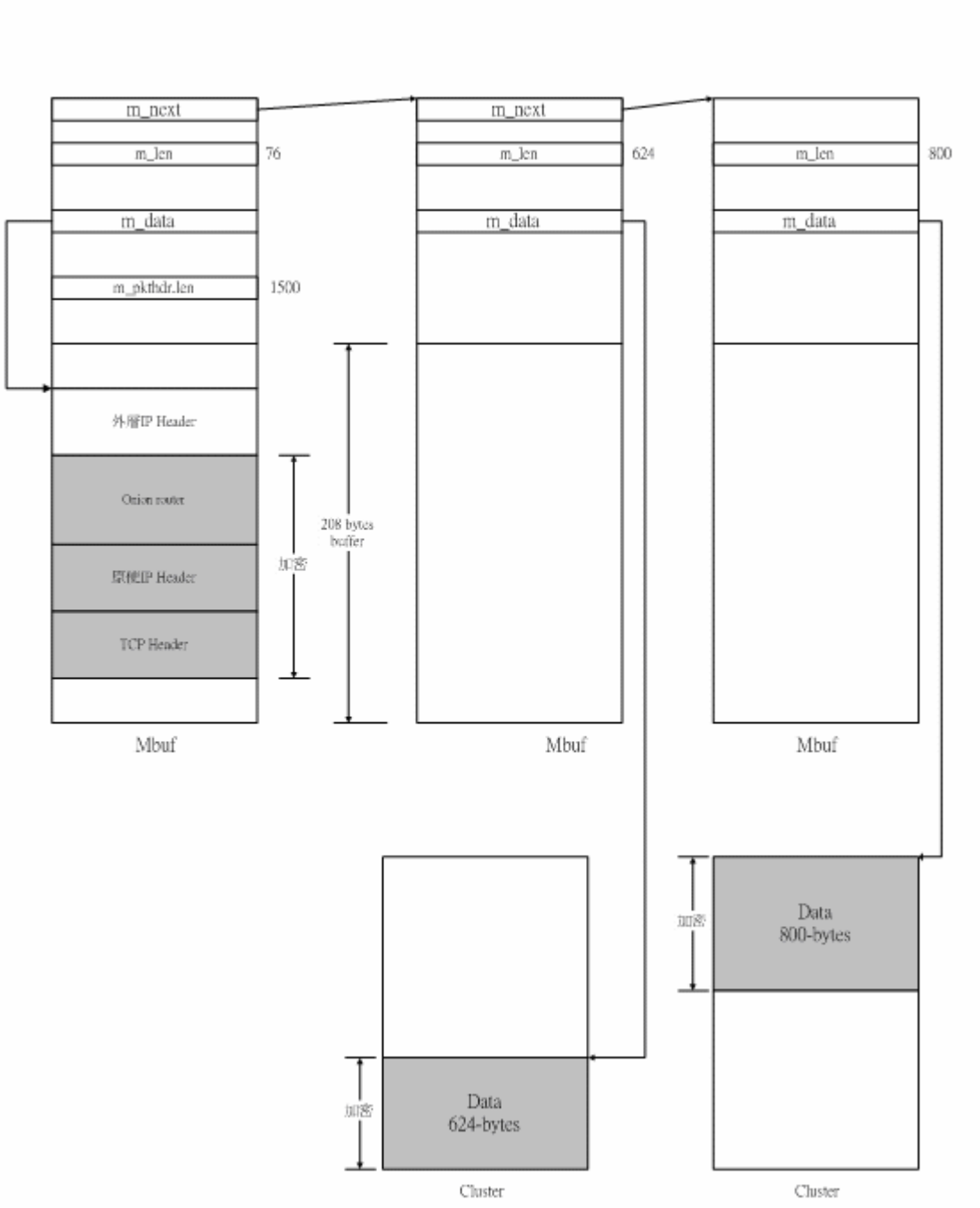

圖 3.6-2 系統內的秘密通訊封包，類型二

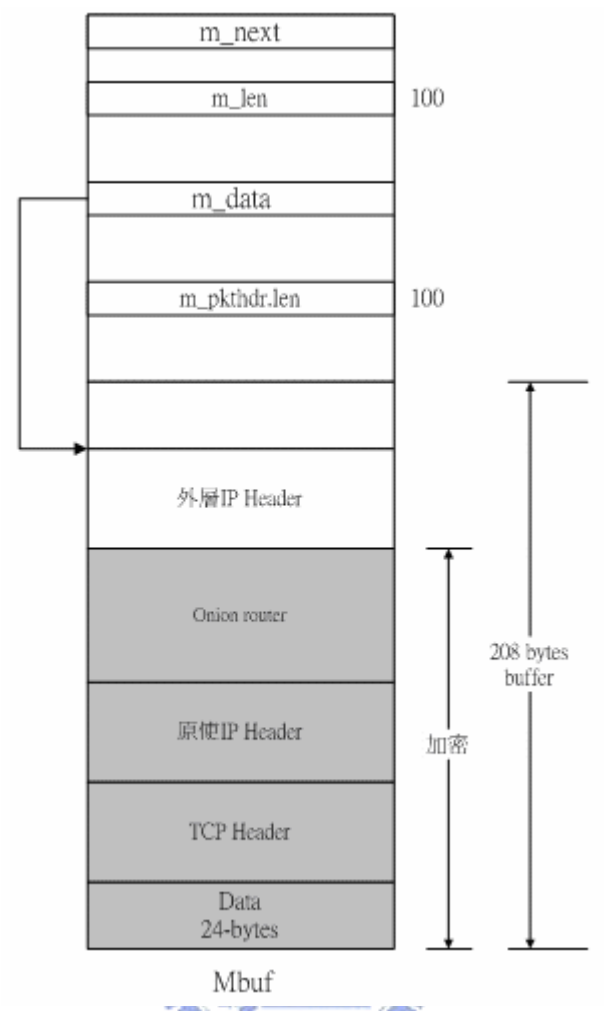

圖 3.6-3 系統內的秘密通訊封包，類型三

3.4.2 ip_input.c

當 TCP/IP 的 Data link layer 將封包往 network layer 傳送，將會由 ip_input 負責後續的部份。當收到封包後，必須先判斷封包的類型，根據不同的類型執行特定的動作。

1. 當收到的封包類型為正常連線，不做任何改變。
2. 封包類型為秘密通訊格式，如圖 3.7，必須先去除外層的 IP Header，再將內部的 IP 封包解密還原成原始的 IP 封包。之後的動作會跟正常連線是一樣的，因此上層並不需要作任何的修改，所有的應用程式也不用作任何的修改，就可以使用秘密通訊。
3. 封包類型為 Onion Router 更新封包，則必需將 Onion Router 的資訊更新。

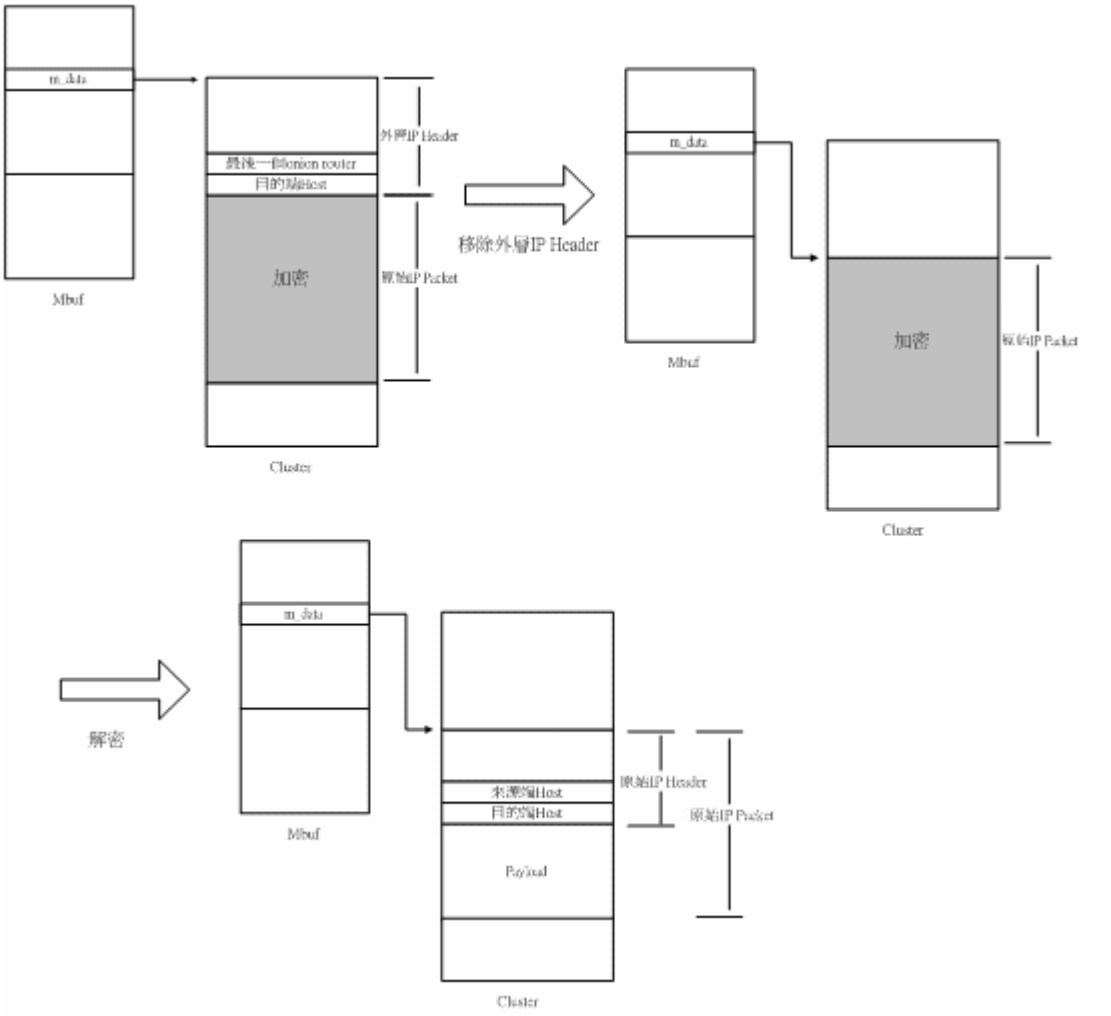

圖 3.7 Receiver 處理秘密通訊封包的運作圖

3.4.3 tcp_output.c

TCP 在建立連線時，Three-Way Handshaking 會去協調雙方的 MSS(Maximum Segment Size)，可讓雙方再傳送封包時，不會被 fragment 來影響傳輸效能。在秘密通訊模式中，會使封包在 ip_output 增加一些額外的資訊，為了不讓封包被 fragment 來影響傳輸效能，因此需要改變 MSS 的大小。

tcp_output 修改的部份如下：

1. 可以根據經過 Onion Router 的數目，動態調整 segment 大小。在 Ethernet 的 TCP segment 最大長度為 1460-bytes，加入 TCP Header 及 IP Header 的長

度為 1500-bytes，剛好是 MTU 的大小。如果是秘密通訊模式，必須還要再增加外層 IP Header 以及 Onion Router 的資訊，會造成封包超過 MTU 的大小，使得封包會被切割成兩個 fragmentation，影響傳輸效率。

2. 調整 max_linkhdr 的大小，從 16-bytese 改為 134-bytes，原本的 max_linkhdr 是預留給 data-link layer 來配置此層的 Header，增加的空間是預留給 IP layer 來配置外層 IP Header 以及 Onion Router 的資訊，如圖 3.8。
3. 根據預留的空間，扣掉原來的 16 bytes 給 data-link layer 與外層 IP Header，最多可以配置 12 台 Onion Router。

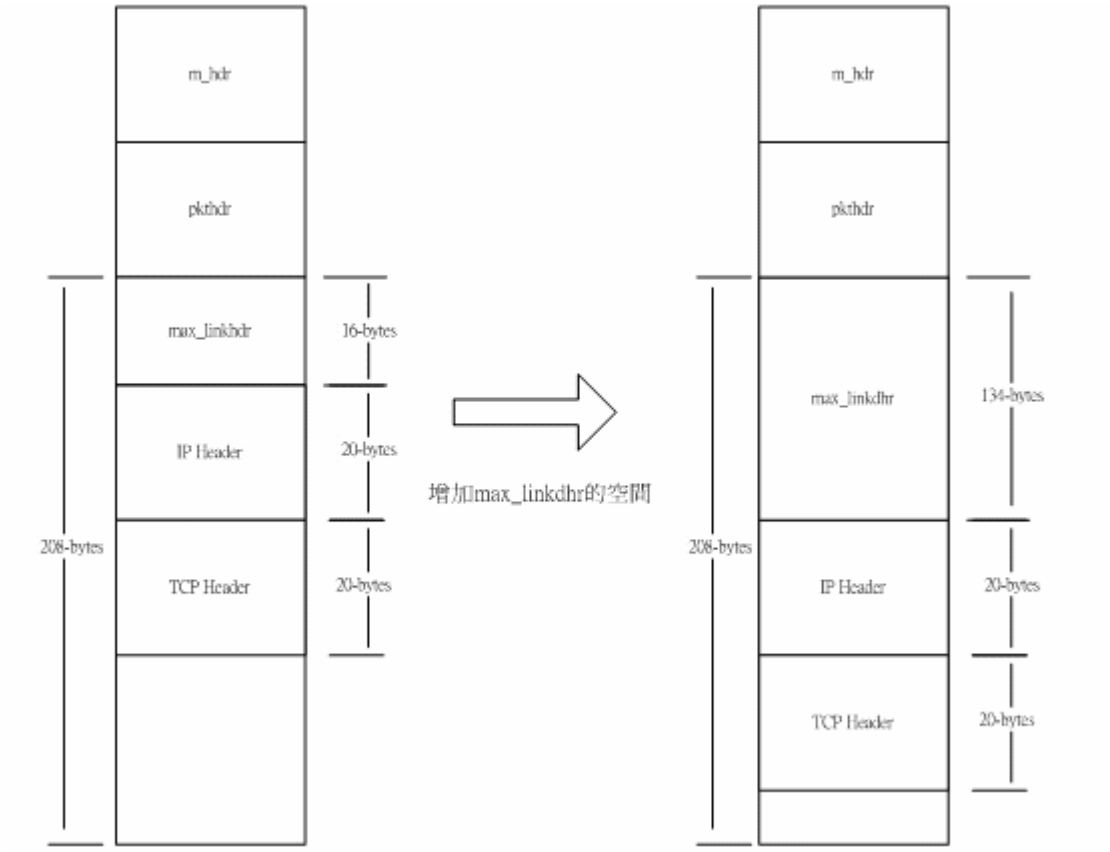

圖 3.8 調整 TCP 配置的記憶體空間

3.4.4 Makefile

原始的 Makefile 並沒有將加解密的相關檔案納入其中，如果想要 kernel 提供加解密的功能，必須要自行加入相關檔案，才可以編譯成功。在這裡使用的是 DES，

如下：

1.

```
des_setkey.o: $S/crypto/des/des_setkey.c
	${NORMAL_C}
des_enc.o: $S/crypto/des/des_enc.c
	${NORMAL_C}
des_ecb.o: $S/crypto/des/des_ecb.c
	${NORMAL_C}
```

2.

```
dOBJS=…\
	des_setkey.o des_enc.o des_ecb.o \
	…\
	…\
```

3.

```
CFILES=…\
	$S/crypto/des/des_setkey.c $S/crypto/des/des_enc.c $S/crypto/des/des_ecb.c\
	…\
	…\
```

將會在 ip_output.c 以及 ip_input.c 使用 DES 加密與解密的函式，必須要 include 以下檔案：

```
#include <crypto/des/des.h>
#include <crypto/des/des_locl.h>
#include <crypto/des/podd.h>
#include <crypto/des/sk.h>
```

3.5 系統運作

3.5.1 Onion Server

有三個主要部份，分別為與 Onion Router 通訊、與 Host 通訊以及檢查 Onion Router 是否 timeout。會使用多個 thread 來完成這三個部份，其中與 Onion Router 通訊的部份，對於每一個 Onion Router 都會產生相對應的 thread 來服務。

(1) 與 Onion Router 通訊：

所有的 Onion Router 都必須向 Onion Server 註冊，因此 Onion Server 知道所有 Onion Router 的資訊(包含 Onion Router 的 IP address、secret key 和對應的 key ID)，如圖 3.9。每一個 Onion Router 的 Key ID 都會有 2^{16} 個，分別對應 65536 把不同的 secret key，其目的是要讓每個 Host 對於同一個 Onion Router 都拿到不同的 secret key，因此每一個 Onion Router 可以服務 65536 個 Host。並且每一把 secret key 都有時間性，當 Host 的封包一段時間沒有經過某台 Onion Router 時，該把 secret key 便會改變。Onion Router 的資訊必須使用與 Onion Server 共有的 Secret key 加密。

當完成註冊的 Onion Router 每隔一段時間必須發送 hello message 給 Onion Server，如果 Onion Server 一段時間沒有收到 hello message，表示 Onion Router 已不存在，Onion Server 能夠立即更新。hello message 包含 secret key 的更新，如圖 3.9。

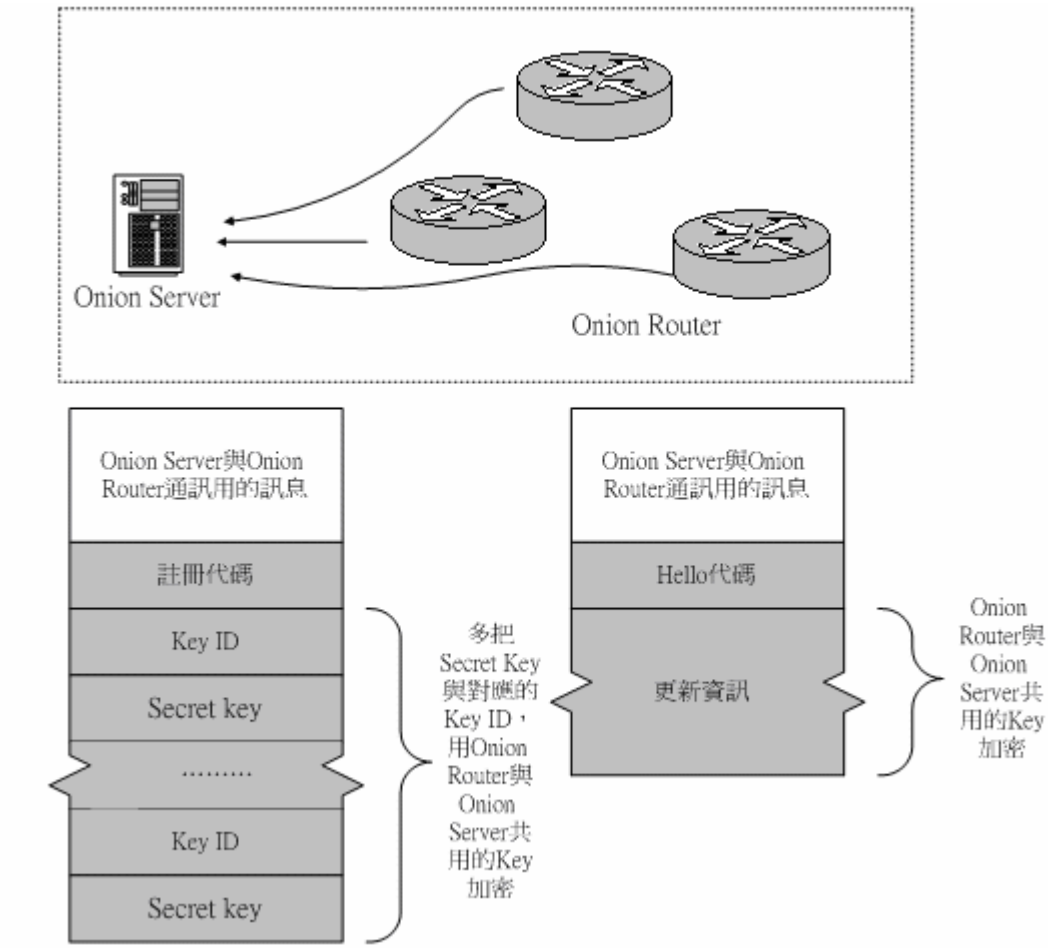

圖 3.9 Onion Router 與 Onion Server 的通訊

(2) 與 Host 通訊：

秘密通訊是由多群 Onion Server-Onion Routers 所組成，每一群 Onion Server-Onion Routers 會有一台 Onion Server 管理多個 Onion Routers。

當 Host 要作秘密通訊時，可以有三種選擇，第一種必須要向多個 Onion Servers 詢問有哪些 Onion Routers 可以使用，如圖 3.10，目的是要分散風險，避免某一台 Onion Server 被入侵而得知那台 Onion Server 所管轄的所有 Onion Router 的資訊，因此 Host 在選擇路徑時，不可以全部都選同一 Onion Server-Onion Routers 群內的 Onion Router。每台 Onion Server 回報現在已存在的一部分 Onion Routers 的資訊(包含 Onion Router 的 IP address、secret key 和對應的 key ID)給 Host。每一個 Host 所得到的 Onion Router 資訊可能都不

一樣，以避免每一個 Host 都使用相同的 Onion Router 與相同的 secret key。此 Onion Router 資訊必須使用 Onion Server 與 Host 共有的 secret key 來加密。當 Host 所擁有的 Onion Router 有變動時，Onion Server 會主動告知 Host 最新的 Onion Router 資訊，因此兩端 Host 在通訊時，並不會因為 Onion Router 的變動而造成斷線。

第二種必須要向一個 Onion Servers 提出詢問，且必須知道除了目的端之外的其他 Host，可以是一台或多台，如圖 3.11。當 Host A 要與 Host C 通訊時，Host A 選擇 Host B 當作中繼站。封包抵達中繼站 Host B 時，Host B 再選擇它所在群組的 Onion Router，封包經過 Host B 群組的 Onion Router 之後會抵達目的端 Host C，因此可以透過中繼站來跨越不同群組。

第三種只要向一個 Onion Servers 提出詢問，傳送的封包只會在單一群組的 Onion Router 之間行走，之後直接抵達目的端。

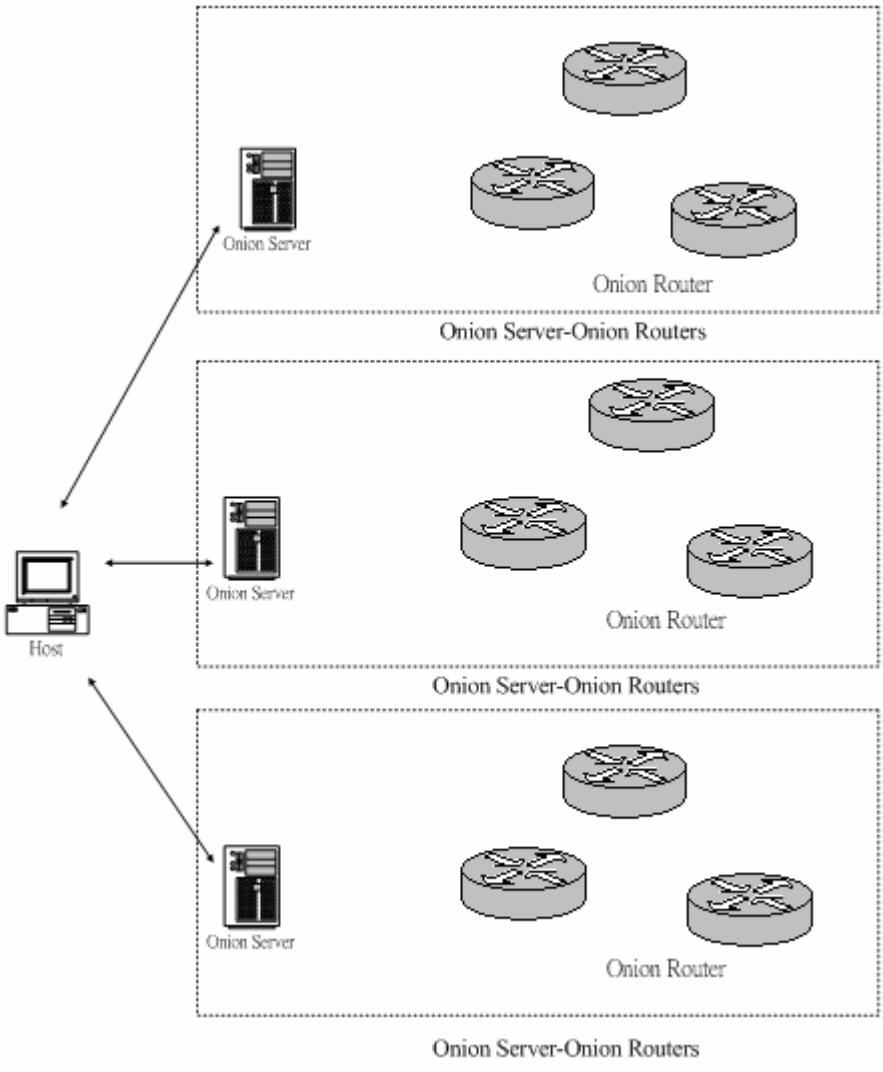

圖 3.10 向多台 Onion Server 取得 Onion Router 的資訊

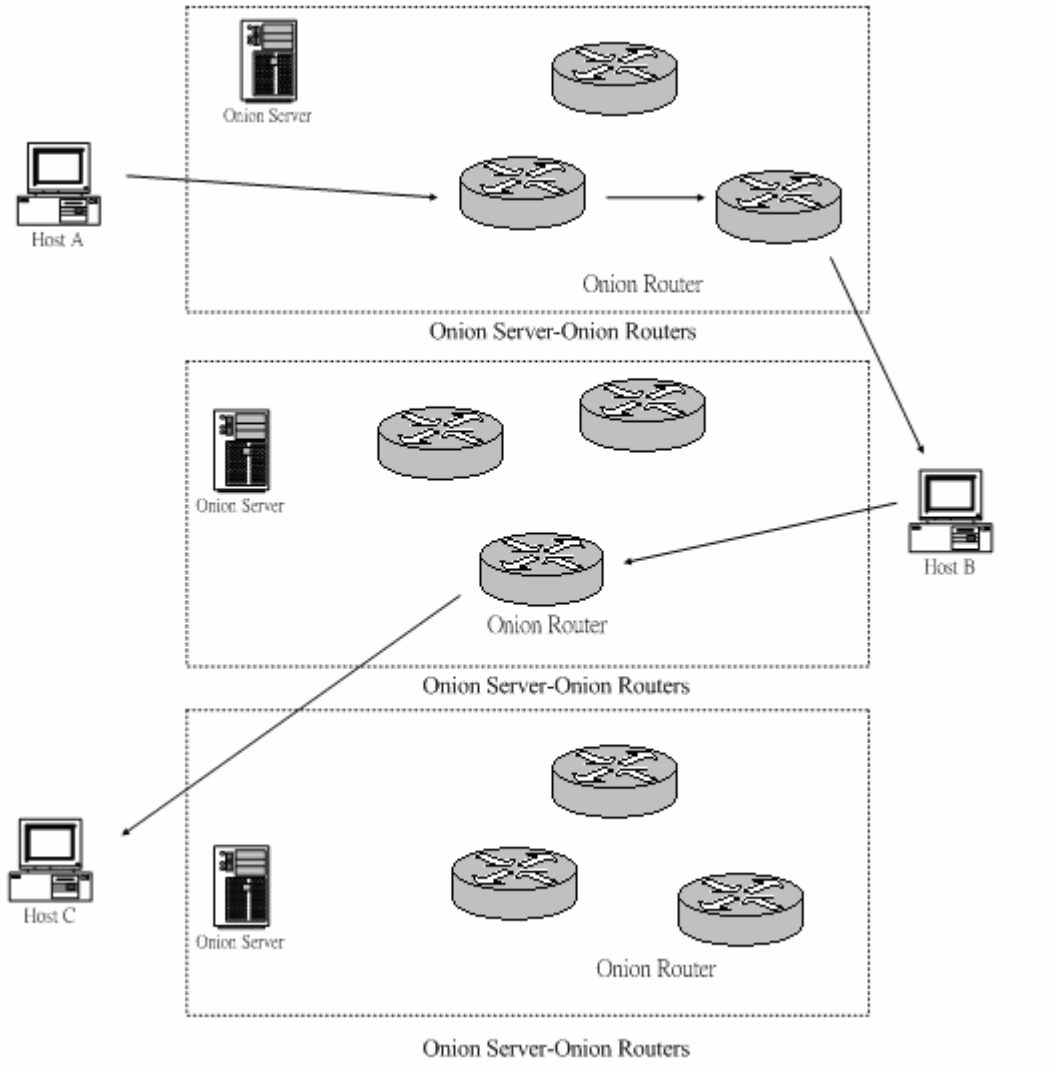

圖 3.11 Host 使用中繼站的秘密通訊

可藉由 Onion Server-Onion Routers 群數目的增加，讓秘密通訊的 Host 一直增加。因此可讓無限多個 Host 加入秘密通訊系統。

(3) 檢查 Onion Router 是否 timeout：

當 Onion Server 發現某一台 Onion Router 已經 timeout，便會通知擁有此 Onion Router 的 Host。可讓 Host 選擇路徑時，不會選到此台 Onion Router。

3.5.2 Host

◆ 秘密通訊模式的啟動

利用 Raw Socket 的特性，可將 message 傳送到 ip_output，來進行秘密通訊模式的開啟與關閉。

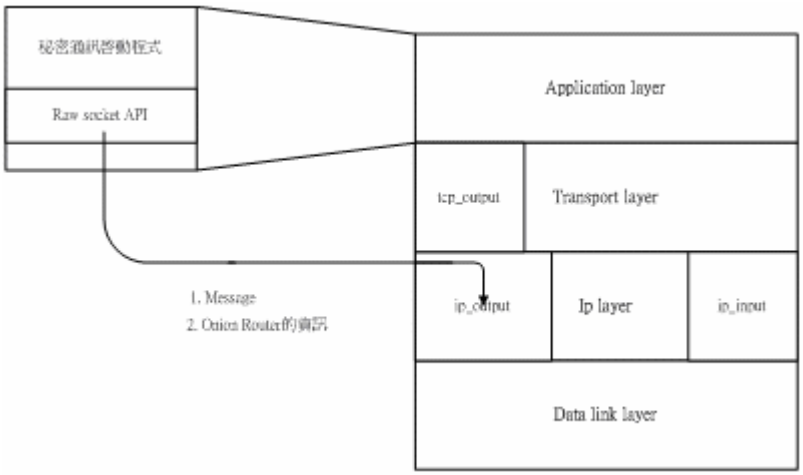

圖 3.12 啟動程式的系統內部運作圖

(1) 向 Onion server 取得 Onion Router 的資訊 (包括 Onion Router 的 IP address、secret key 和對應的 key ID)。

(2) 獲得 Onion Routers 資訊後，若是目前無存在任何 Onion Router，則無法開啟秘密通訊模式；若是有一台以上的 Onion Router 存在，則秘密通訊模式的啟動程式會傳送開啟秘密通訊模式的 message 以及 Onion Router 的資訊到 ip_output，如圖 3.12。

(3) 若有兩台以上 Onion Router，則可讓使用者設定封包傳送的路徑長度。

◆ 秘密通訊模式的運作

秘密通訊模式的運作是透過修改 tcp_output、ip_output、ip_input 的方式達成。

(1) 由 transport layer 傳下來的封包先將決定好的路徑資訊填入，再加入外層 IP Header，最後進行加密的動作。首先使用雙方事先協調好的 key 來對內

部原始封包加密，接著依序使用每台 Onion Router 的 secret key 對之後的 payload 加密，從 Onion Router (n)、Onion Router (n-1)、 … 、一直到 Onion Router (2)，如圖 3.13。

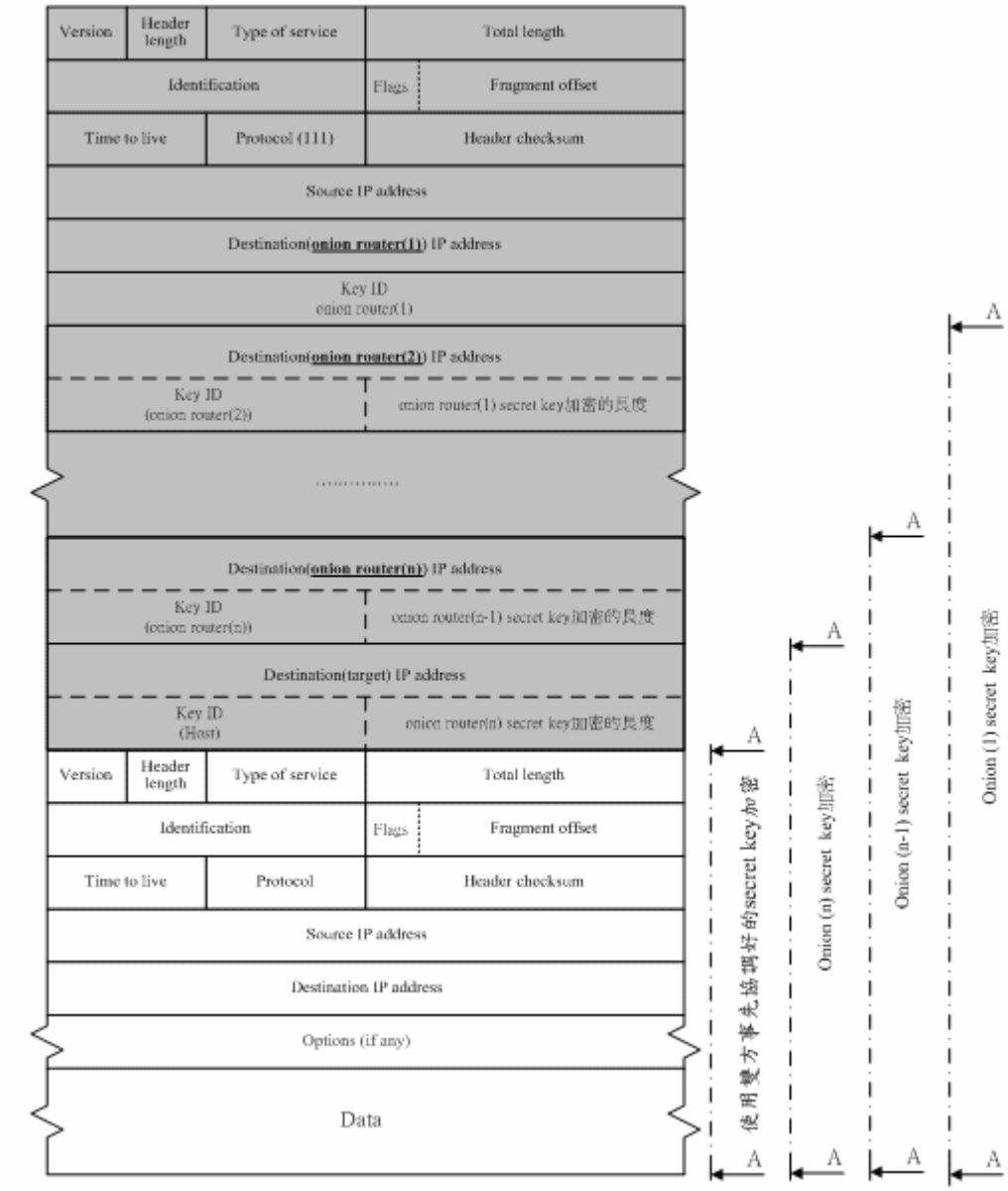

圖 3.13 Sender 送出的秘密通訊封包

(2) 一但 Onion Router 有所變動時，便會收到新的 Onion Router 資訊，因此會自動改變路徑。

(3) 當封包經過所有的路徑之後，最後會到達目的端 Host，收到的封包格式如圖 3.14：

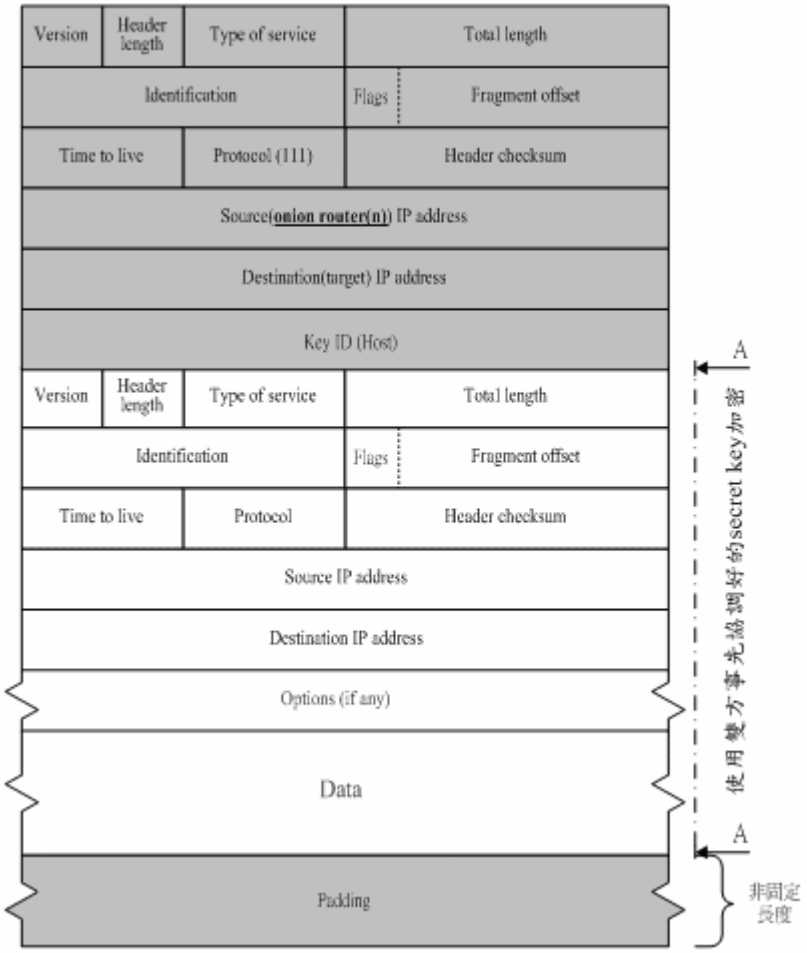

圖 3.14 Receiver 收到的秘密通訊封包

(4) 目的端 Host 在 ip_input 收到此格式的封包後，先去掉灰底的 IP header 與 padding，再解密原始 IP 封包內容，即可得到原始資訊。

3.5.3 Onion Router

有兩個主要部份，分別為 Register 及 Packet forward。使用兩個 thread 來完成這兩個部份。

(1) Register：

當 Onion Router 啟動時，首先產生 65536 把 Secret Key 與對應的 Key ID，Key ID 是用來識別每一把 Secret Key。接著必須向 Onion Server 註冊，註冊的資訊包含 Onion Router 的 IP address、secret key 和對應的 key ID，此資訊必

須用與 Onion Server 共有的 Secret key 加密。之後每隔一段時間發送 hello message 給 Onion Server 表示其 Onion Router 還存在。

(2) Packet forward：

利用 Raw Socket 可接收有特定 protocol 值的封包之特性，可將 IP layer 收到秘密通訊模式的封包傳送到 Application layer。因此便可以直接處理這類型的封包，之後再透過 Raw Socket，將封包直接傳送到 IP layer，便可完成 forward 的動作，如圖 3.15。

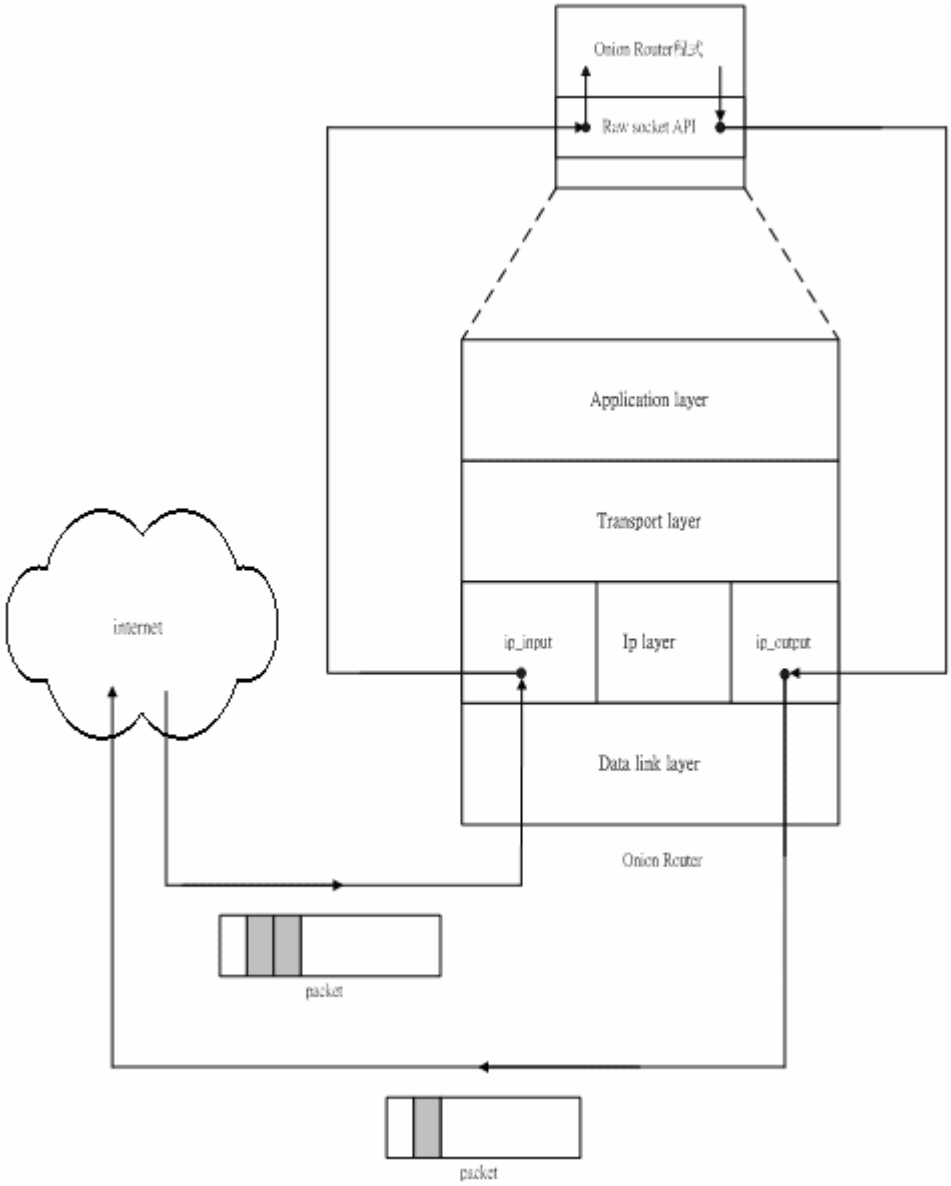

圖 3.15 Onion Router 處理秘密通訊封包的系統內部運作圖

當收到 packet 時，先從 Key ID 欄位得知使用加密的 secret key，將放在外層 IP Header 後面的資訊(已用此 Onion Router 的 Key ID 之 secret key 加密)解密。解密第一塊 block(8-bytes)後，可取得 1)下一站的位址；2)下一站的 Key ID；3)用此 secret key 加密的長度。由第 3)點可以將剩下的部分解密，再將外層的 IP Header 去除，重新產生一個新的外層 IP Header，Destination address 欄位填入 1)下一站的位址；Source address 欄位填入自己的 IP address。由第 3)點可以間接得知收到封包的 padding 長度，必須先去除此 padding，再產生隨機長度的 padding，其目的要讓收到的封包與送出的封包長度沒有相關性，具有混淆的效果。整個封包長度確定後，將 Total length 欄位填入，再將 IP Header 的其他欄位填入。最後 Key ID 欄位填入 2)下一站的 Key ID，便立刻送出。例如圖 3.16，當 Onion Router R1 收到 Packet 1 時，由外層 IP Header 之後的 Key ID 欄位為 23，表示必須使用 ID 為 23 的 secret key 解密。首先解開第一塊 block，獲得 1)下一站的位址，R1；2)下一站的 Key ID，50；3)用此 secret key 加密的長度，1478。由外層 IP Header 的 Total length 欄位得知 Packet 1 的長度為 1500 bytes 以及 3)加密的長度為 1478，及可判斷封包後面的 padding 長度；外層 IP Header 的長度為 20 bytes 加上加密的長度 1478 bytes 與 Key ID 的長度 2 bytes，共為 1500 bytes，表示 Packet 1 沒有 Padding。由 3)加密的長度的資訊，再將剩餘的部分解密後，分別將 1)和 2)的資訊填入新的外層 IP Header 相對應的欄位。最後決定 Padding 的長度為 7 bytes，Packet 2 的 Total length 為 1478+2-8+7+20=1499 bytes，Onion Router R1 就完成 Packet 2，將它傳送出去。

每個 Onion Router 只能夠解開下一站的 IP address，並無法得知其他的 IP address，也無法得知會經過多少個 Onion Router。

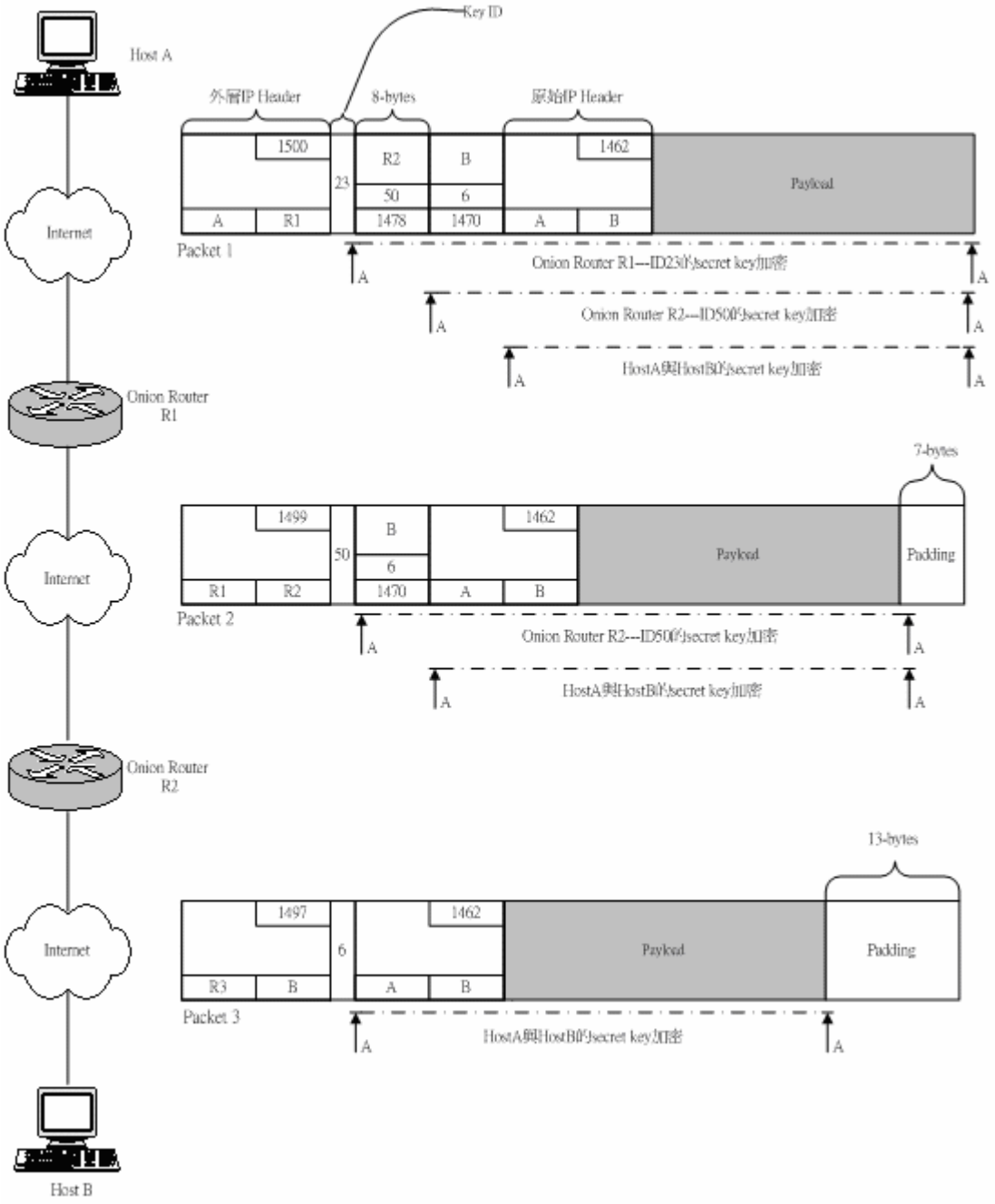

圖 3.16 秘密通訊的封包運作過程圖

第四章 實作結果

4.1 運作過程

使用設備如下：

Host A：AMD Athlon 1800+ ， IP Address：140.113.241.22

Host B：Intel Pentium 333 MHz ， IP Address：140.113.241.21

Onion Server：Intel Pentium 300 MHz ， IP Address：140.113.241.26

Onion Router(R1)：Intel Pentium3 700 MHz ， IP Address：140.113.241.28

Onion Router(R2)：AMD Duron 1.8 GHz ， IP Address：140.113.241.24

作業系統：FreeBSD 5.1

網路環境：100M/bits 的區域網路

Host A 與 Host B 進行秘密通訊，首先 Host A 與 Host B 分別向 Onion Server 取得 Onion Router 的資訊，為 R1 與 R2 兩台 Onion Router。Host A 與 Host B 決定路徑都是為 Sender → R1 → R2 → Receiver。

Host A 使用 FTP 向 Host B 做連線，分別上傳與下載 5139KB 大小的檔案。Host A、Onion Server、Onion Router(R1)、Onion Router(R2)的運作狀態如下，圖 4.1.1、圖 4.1.2、圖 4.1.3、圖 4.1.4、圖 4.1.5、圖 4.1.6：

首先必須先啟動秘通訊，如圖 4.1.1，向 Onion Server 取得 Onion Router 的資訊，而得知有兩台 Onion Server。接著使用者可以選擇路徑長度，選擇要經過兩台 Onion Router。啟動程序完成後，封包格式會變成祕密通訊的封包格式來傳送。

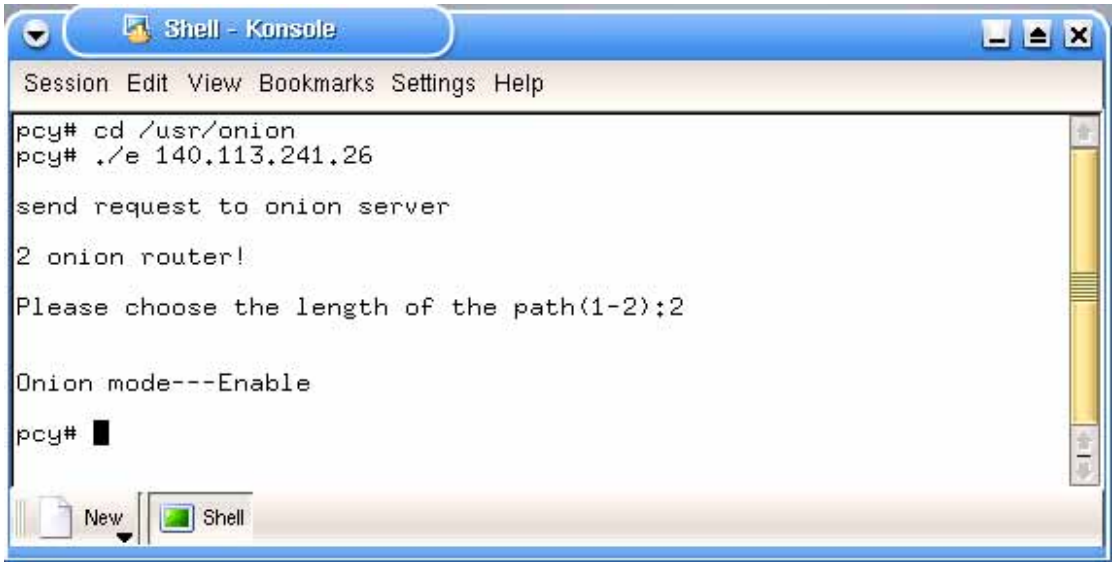

圖 4.1.1 啟動秘密通訊

使用秘密通訊來上傳與下載檔案，如圖 4.1.2，可發現傳輸的速度變的比正常通訊慢。

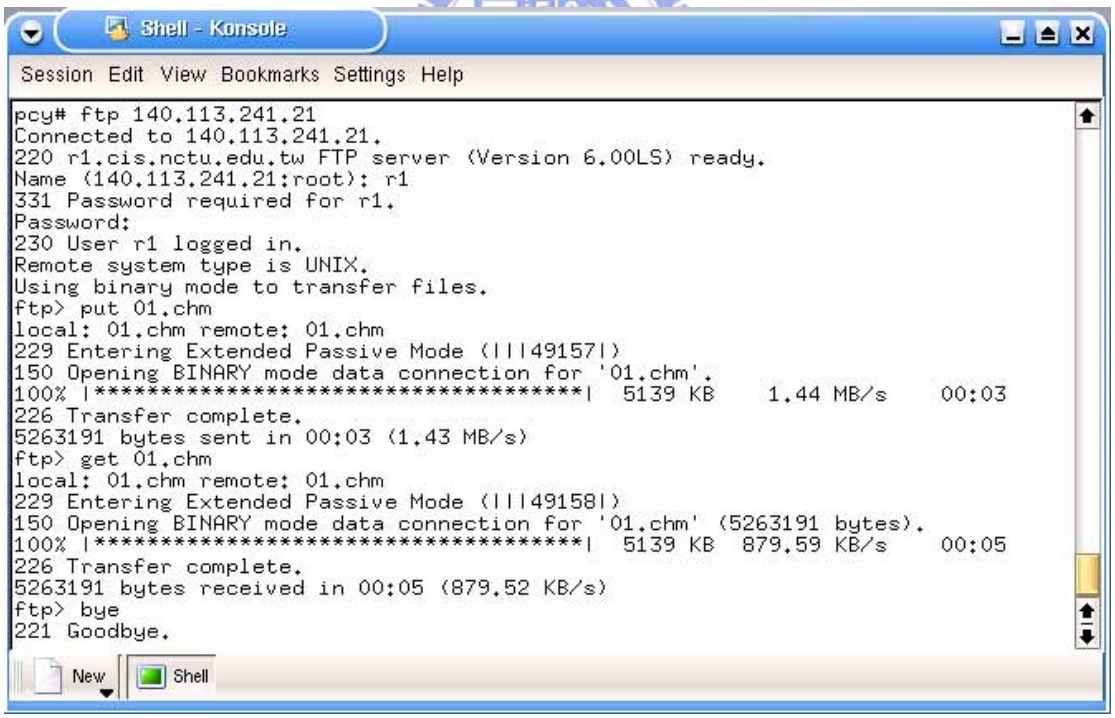

圖 4.1.2 Host A 的運作狀態

Onion Server 記錄目前存在的 Onion Router 資訊，當 Timer 一段時間沒更新，表示 Onion Router 已不存在，如圖 4.1.3。

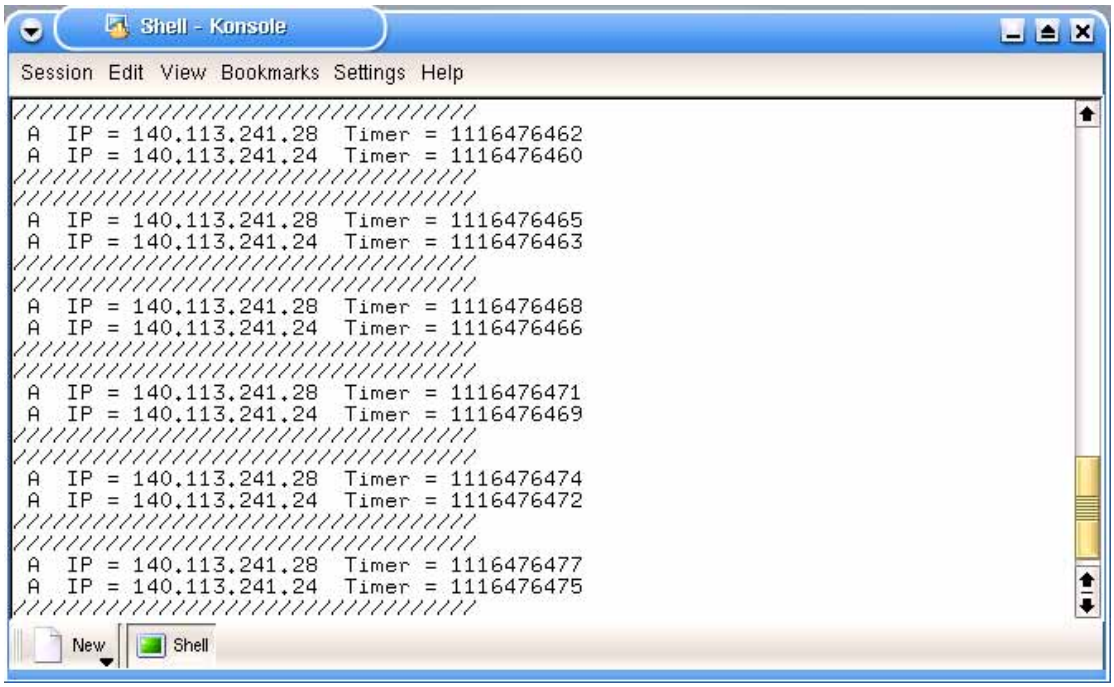

圖 4.1.3 Onion Server 的運作狀態

Onion Router R1 收到從 Host A 以及 Host B 傳來的封包，解密一層後往 Onion Router R2 傳送，如圖 4.1.4。

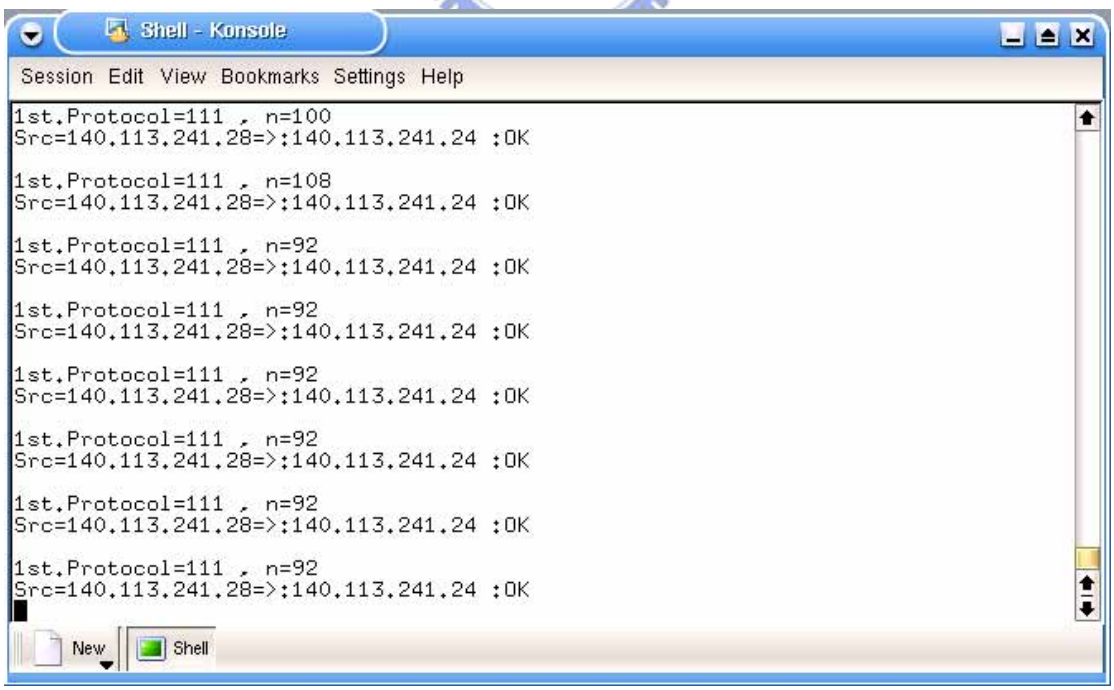

圖 4.1.4 Onion Router(R1) 的運作狀態

Onion Router R2 收到從 Onion Router R1 傳來的封包，由於 Onion Router R1 會對封包做 Padding，因此封包長度都不一樣。解密一層後，往 Host A 或 Host B 傳送，如圖 4.1.5。。

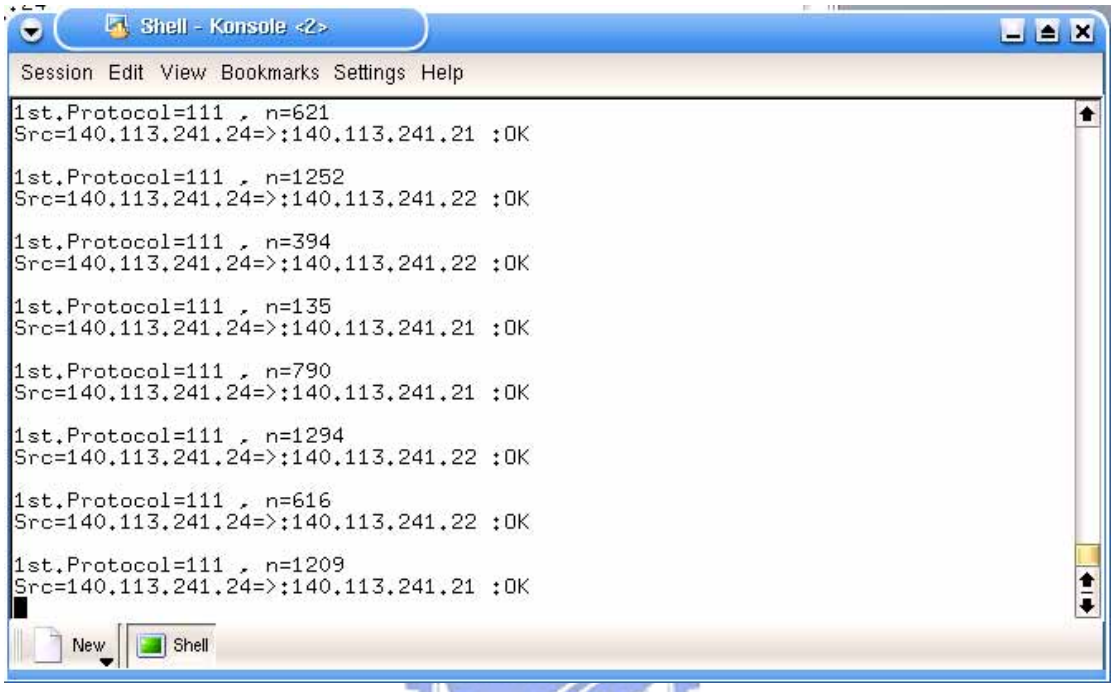

圖 4.1.5 Onion Router(R2) 的運作狀態

當想恢復原來的正常模式，可將秘密通訊模式關閉，如圖 4.1.6。

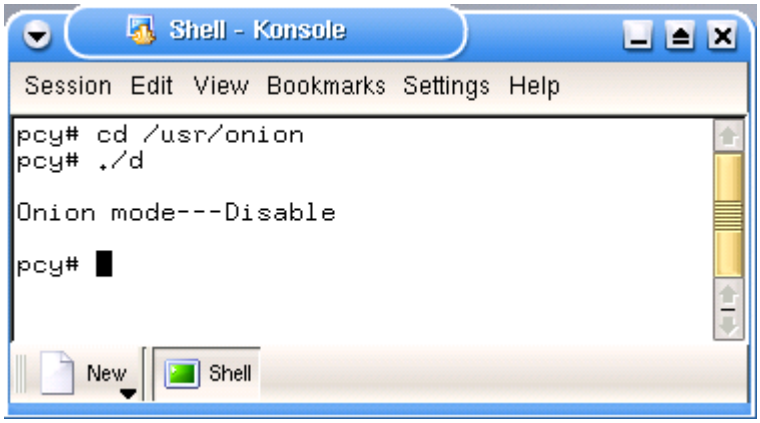

圖 4.1.6 關閉秘密通訊

4.2 實作數據與分析

4.2.1 實作數據

始用 FTP 來上傳檔案來分析效能，檔案大小為 5139KB。首先是一般正常連線，如圖 4.2.1。

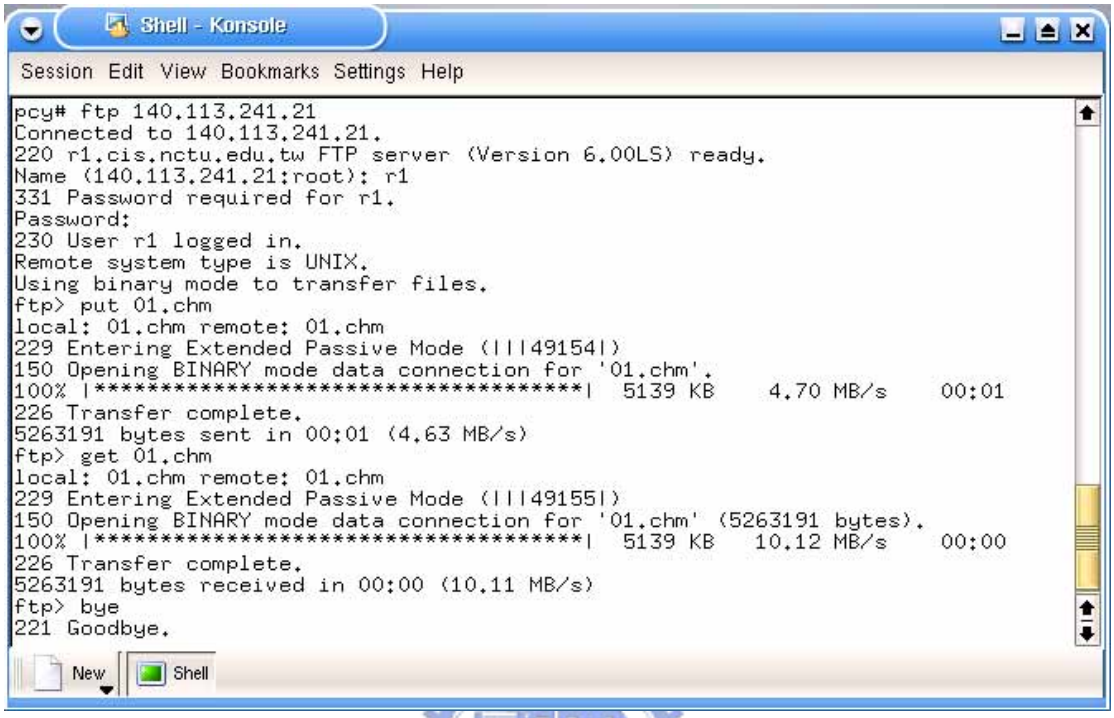

圖 4.2.1 一般正常連線

使用秘密通訊連線，但無加密，經過一台 Onion Router(R1)，如圖 4.2.2。

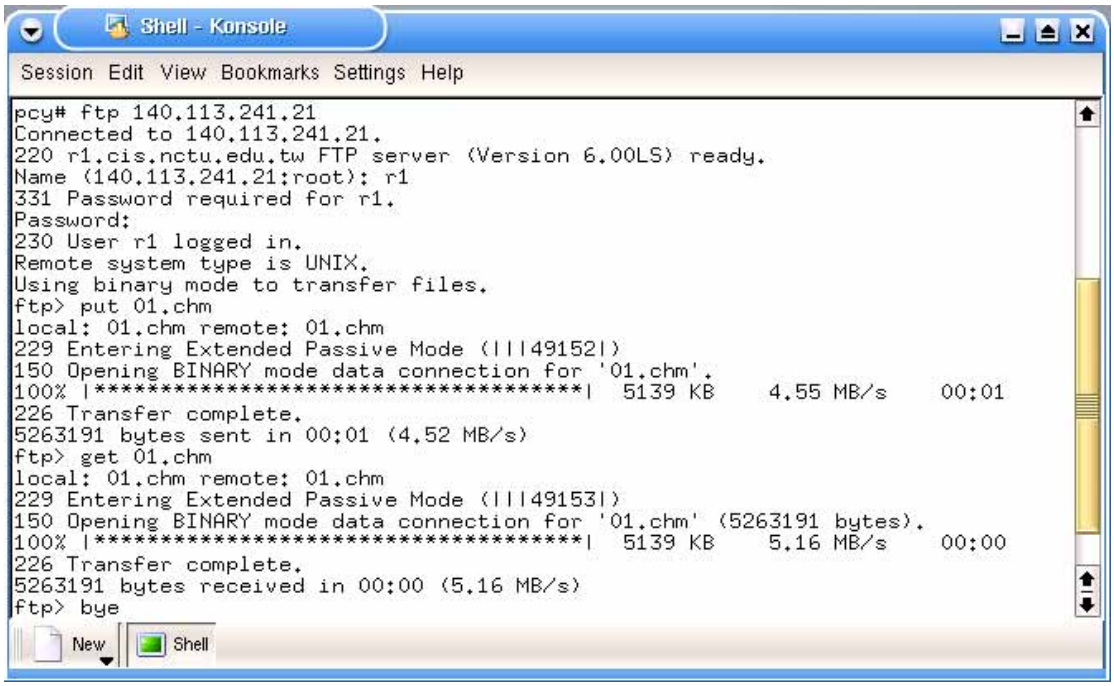

圖 4.2.2 秘密通訊連線，無加密

使用秘密通訊連線，有加密，經過一台 Onion Router(R1)，如圖 4.2.3。

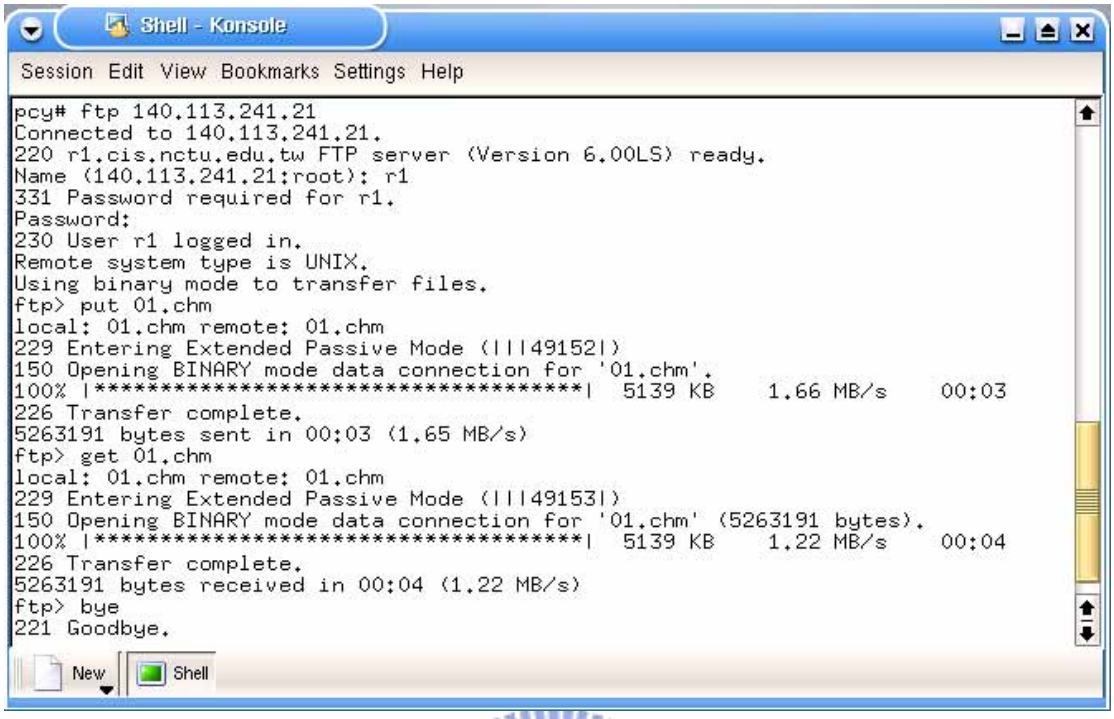

圖 4.2.3 經過一台 Onion Router(R1)

使用秘密通訊連線，有加密，經過兩台 Onion Router(R1、R2)，如圖 4.2.4。

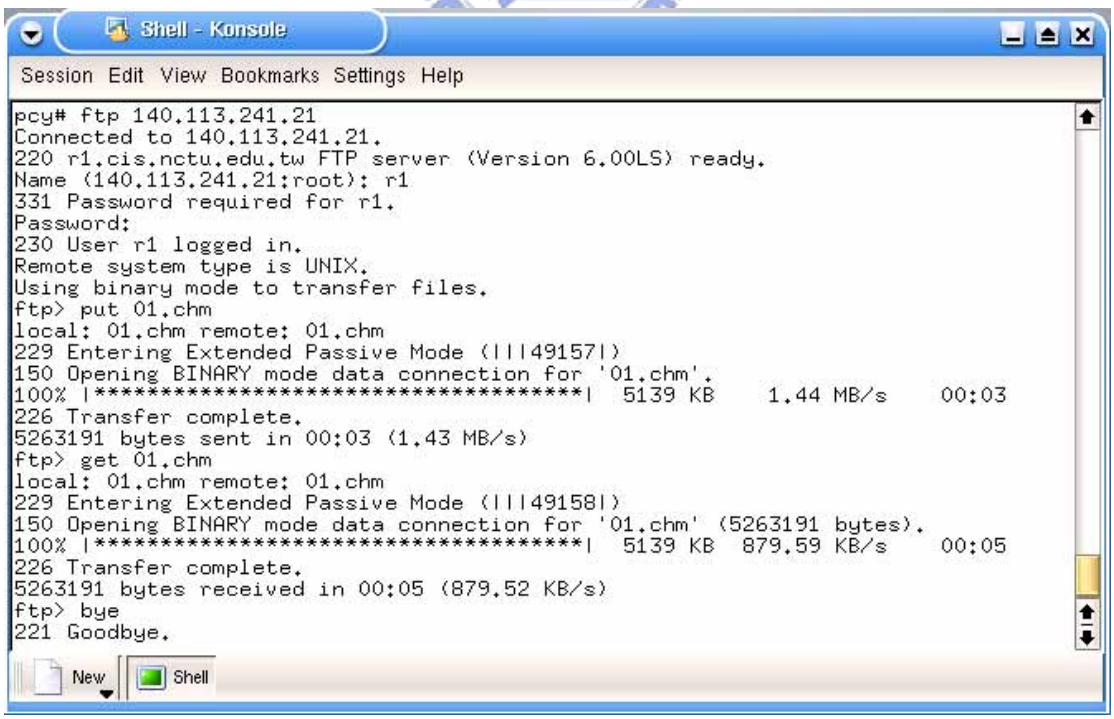

圖 4.2.4 經過兩台 Onion Router(R1、R2)

4.2.2 效能分析

關於實際效能的部份，使用三個不同大小的檔案，分別為 5139KB、10.69MB、173MB，透過上傳與下載這些檔案來分析不同情況的效能。我們將分成五種情況，第一種為正常連線，未啟動秘密通訊的一般正常連線；第二種為啟動秘密通訊，經過一台 Onion Router R1 但未加密；接下來的三種為啟動秘密通訊，有加密，分別經過的 Onion Router 為 R1 一台、R2 一台、R1 和 R2 兩台，見表 4.2.1。

上傳：Host A 將檔案傳往 Host B。

下載：Host B 將檔案傳往 Host A。

Size:5139KB	正常連線	無加密(R1)	加密(R1)	加密(R2)	加密(R1、R2)
上傳	4.72MB/s	3.73MB/s	1.65MB/s	1.81MB/s	1.43MB/s
下載	10.12MB/s	4.16MB/s	1.22MB/s	1.21MB/s	879.52KB/s

Size:10.69MB	正常連線	無加密(R1)	加密(R1)	加密(R2)	加密(R1、R2)
上傳	4.45MB/s	3.70MB/s	1.58MB/s	1.81MB/s	1.55MB/s
下載	10.13MB/s	4.11MB/s	1.20MB/s	1.21MB/s	853.47KB/s

Size:173MB	正常連線	無加密(R1)	加密(R1)	加密(R2)	加密(R1、R2)
上傳	4.47MB/s	3.69MB/s	1.59MB/s	1.77MB/s	1.48MB/s
下載	9.20MB/s	4.10MB/s	1.21MB/s	1.21MB/s	881.25KB/s

表 4.2.1 效能分析

由表 4.2.1 的正常連線欄位可得知，Host A 傳送檔案至 Host B 的極限為

4.5MB/s 左右，Host B 傳送檔案至 Host A 的極限為 10MB/s 左右。由無加密(R1)與加密(R1)兩個欄位，可得知加密對整體的效能影響最大，因此整個系統負擔最重為 Sender，由下載那一欄位便可看出，Host B 的 CPU 為 333MHz 導致效能會變的不理想。由加密(R1)與加密(R2)兩個欄位，可得知 Onion Router 的快慢的效能影響不大。檔案的大小則對效能毫無影響。

4.2.3 安全性分析

當一個封包從 Sender 端送出，如圖 4.2.5，經過一台 Onion Router(r1)，往第二台 Onion Router(r2)傳送，如圖 4.2.6，最後 Onion Router(r2)將封包傳送到 Receiver 端，如圖 4.2.7。一個封包在 Sender 端透過三層加密，再經由兩台 Onion Router 的一層解密與 Padding，最後抵達 Receiver 端。可讓單一封包從 Sender 端到 Receiver 端，被混淆成三個獨立的封包，這三個封包長度不一樣，內容也都不一樣，如圖 4.2.5、4.2.6、4.2.7。

原始的 IP Header 的 Source IP Address 與 Destination IP Address，記錄 Sender 與 Receiver 的身分，由於經過多層的加密以及不容易察覺原始的 IP Header 位於封包的相對位址，因此很難得知通訊兩方的身分。

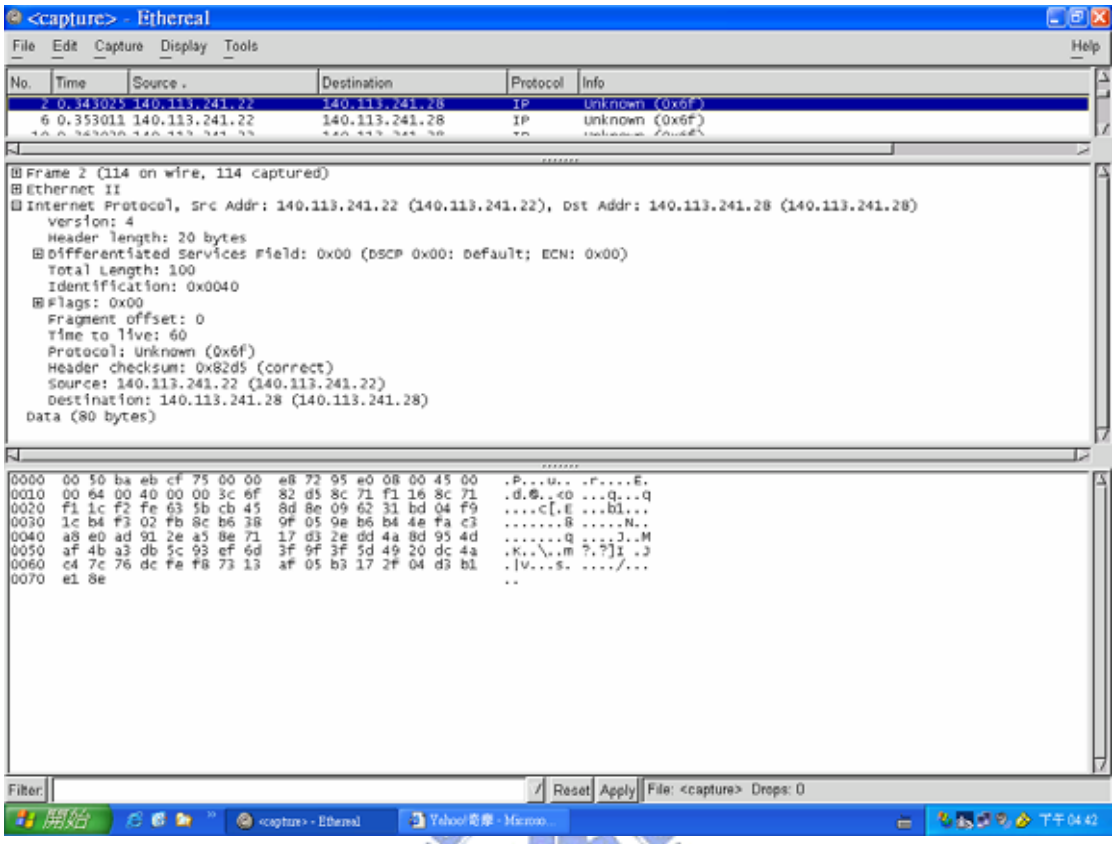

圖 4.2.5 封包分析(一)

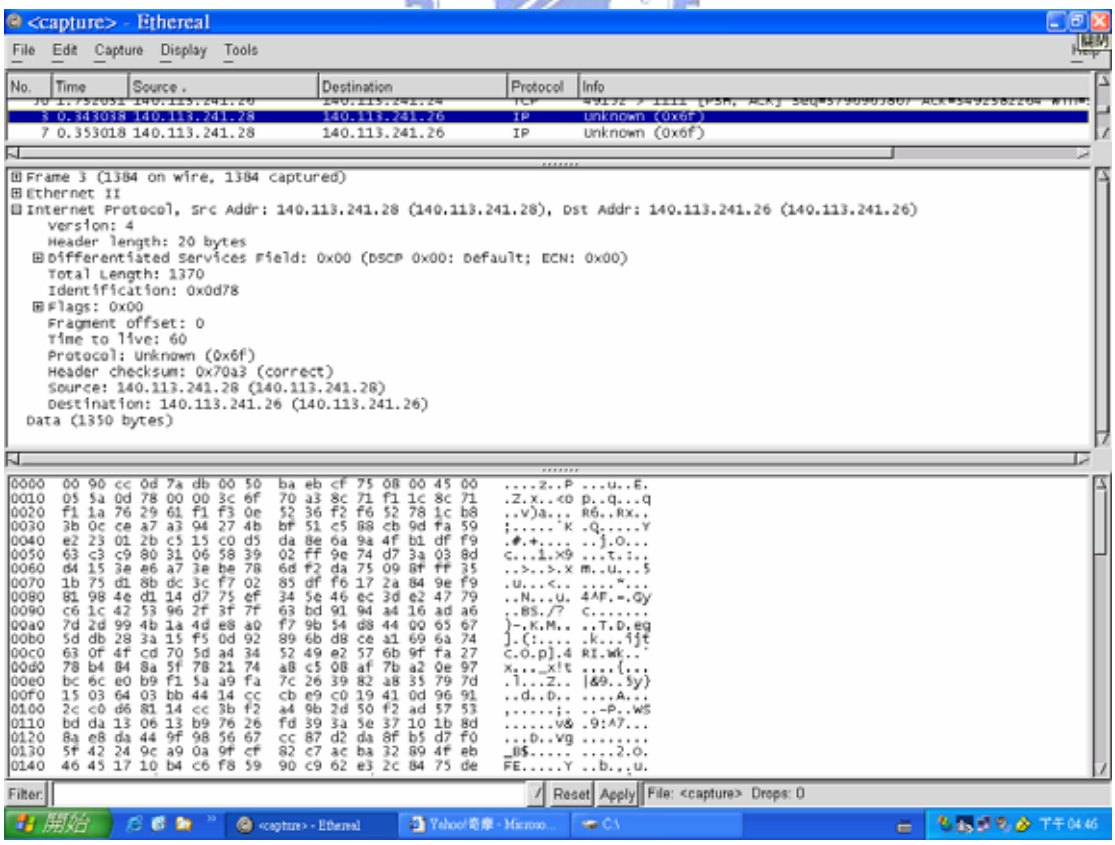

圖 4.2.6 封包分析(二)

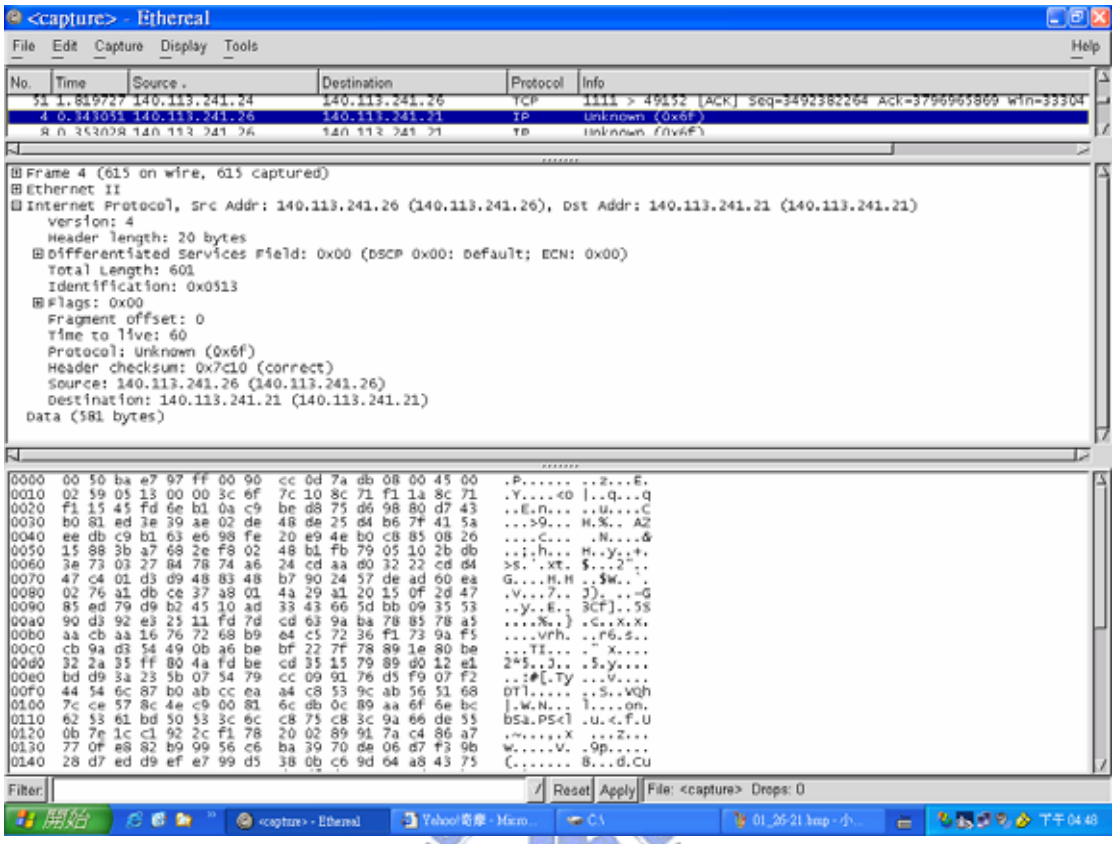
<capture> - Ethereal
File Edit Capture Display Tools
Help
No. Time Source Destination Protocol Info
51 1.849727 140.113.241.24 140.113.241.26 TCP 1111 > 49152 [ACK] Seq=3492382264 Ack=3796965869 Win=33304
4 0.343031 140.113.241.26 140.113.241.21 IP Unknown (0x6f)
Frame 4 (615 on wire, 615 captured)
Ethernet II
Internet Protocol, Src Addr: 140.113.241.26 (140.113.241.26), Dst Addr: 140.113.241.21 (140.113.241.21)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
Total Length: 601
Identification: 0x0513
Flags: 0x00
Fragment offset: 0
Time to live: 60
Protocol: Unknown (0x6f)
Header checksum: 0x7c10 (correct)
Source: 140.113.241.26 (140.113.241.26)
Destination: 140.113.241.21 (140.113.241.21)
Data (581 bytes)
0000 00 50 ba e7 97 ff 00 90 cc 0d 7a db 08 00 45 00 .P..... ..z...E.
0010 02 59 05 13 00 00 3c 6f 7c 10 8c 71 f1 1a 8c 71 .Y....<o |..q...q
0020 f1 15 45 fd 6e b1 0a c9 be d8 75 d6 98 80 d7 43 ..E.n... ..u....C
0030 b0 81 ed 3e 39 ae 02 de 48 de 25 d4 b6 7f 41 5a ...>9... H.%.. AZ
0040 ee db c9 b1 63 e6 98 fe 20 e9 4e b0 c8 85 08 26 ....c... .N....&
0050 15 88 3b a7 68 2e f8 02 48 b1 fb 79 05 10 2b db ..;.h... H..y..+.
0060 3e 73 03 27 84 78 74 a6 24 cd aa d0 32 22 c8 d4 >s.'.xt. $...2"..
0070 47 c4 01 d3 d9 48 83 48 b7 90 24 57 de ad 00 ea G....H.H ..$W....
0080 02 76 a1 db ce 37 a8 01 4a 29 a1 20 15 0f 2d 47 .v...7.. J). ..-G
0090 85 ed 79 d9 b2 45 10 ad 33 43 66 5d bb 09 35 53 ..y..E.. 3Cf]..5S
00a0 90 d3 92 e3 25 11 fd 7d cd 63 9a ba 78 85 78 a5 ....%..} .c..x.x.
00b0 aa cb aa 16 76 72 68 b9 e4 c5 72 36 f1 73 9a f5 ....vrh. ..r6.s..
00c0 cb 9a d3 54 49 0b a6 be bf 22 7f 78 89 1e 80 be ...TI... ." .x....
00d0 32 2a 35 ff 80 4a fd be cd 35 15 79 89 d0 12 e1 2*5..J.. .5.y....
00e0 bd d9 3a 23 5b 07 54 79 cc 09 91 76 d5 f9 07 f2 ..:#[.Ty ...v....
00f0 44 54 6c 87 b0 ab cc ea a4 c8 53 9c ab 56 51 68 DTl..... ..S..VQh
0100 7c ce 57 8c 4e c9 00 81 6c db 0c 89 aa 6f 6e bc |.W.N... l....on.
0110 62 53 61 bd 50 53 3c 6c c8 75 c8 3c 9a 66 de 55 bSa.PS<l .u.<.f.U
0120 0b 7e 1c c1 92 2c f1 78 20 02 89 91 7a c4 86 a7 .~...,.x ...z...
0130 77 0f e8 82 b9 99 56 c6 ba 39 70 de 06 d7 f3 9b w.....V. .9p.....
0140 28 d7 ed d9 ef e7 99 d5 38 0b c6 9d 64 a8 43 75 (....... 8...d.Cu
Filter:
Reset Apply File: <capture> Drops: 0
開始
<capture> - Ethereal
Yahoo!奇摩 - Micro...
C:\
01_26-21 hop - 小...
下午 04:48

圖 4.2.7 封包分析(三)

第五章 結論

5.1 結論

本篇論文中，我們設計一個可在網路上進行秘密通訊的系統來降低被第三者竊聽與 traffic analysis 的風險，在盡可能增加安全性的同時也能有好的效能表現。使用者可根據自己的需求決定路徑長度，路徑越長安全性越高，但是效能會越差，反之亦然。

在現今網路上沒有百分之百安全的系統能夠防止竊聽與 traffic analysis，只能增加被竊聽與 traffic analysis 的困難度，使安全性提高。我們利用密碼學與 Onion Router 的轉送封包來增加混淆程度，提高竊聽與 traffic analysis 的困難度。我們修改作業系統的核心，但還保有原始的功能，透過啟動程式可選擇要使用原始的連線方式或者是具有隱密性與安全性的秘密通訊連線，而不是一個只供特殊應用的單一系統。

5.2 未來發展方向

未來可發展的方向如下：

◆ Key 的交換

在兩端 Host 之間的 secret key、Onion Server 與 Onion Router 之間的 secret key、Onion Server 與 Host 之間的 secret key 是假設存在，未來可在這三個部份 key 的交換與使用加以實作或者使用其他的密碼學方法。

◆ 擴大規模

目前只針對單一群組作實驗，未來可擴大群組與 Host 來作實驗與測試。

◆ 自動啟動秘密通訊

目前 receiver(server)必須手動來啟動秘密通訊，未來可以判斷是否為秘密通

訊的特殊封包，如果是就自動啟動秘密通訊。

◆ 加解密

目前使用 DES 來加解密，而 DES 在現在已經不夠安全，未來可以使用更新更安全的加解密系統。

◆ 使用現有的 protocol 封包格式

秘密通訊模式的封包是使用特定的 protocol(111)格式，未來可以使用現有的 UDP 封包格式，利用 port 欄位來判斷是否為秘密通訊模式，而不是原來的 IP Header 的 protocol 欄位(111)，會使安全性更好。

參考文獻

[1] David Goldschlag, Michael Reed, Paul Syverson, “Anonymous connections and onion routing”, in Proc. IEEE Journal on Selected Areas in Communications, on Volume 16, Issue 4, May 1998

[2] David Goldschlag, Michael Reed, Paul Syverson, “Anonymous connections and onion routing”, in Proc. 1997 IEEE Symp. on Security and Privacy, Oakland, CA, May 1997

[3] David Goldschlag; Michael Reed, Paul Syverson, ”Onion routing for anonymous and private Internet connections”, Association for Computing Machinery. Communications of the ACM; Feb 1999

[4] David Goldschlag, Michael Reed, Paul Syverson, “Onion routing access configurations”, DARPA Information Survivability Conference and Exposition, 2000

[5] David Goldschlag, Michael Reed, Paul Syverson, “Proxies for anonymous routing”, in Proc.12th An. Computer Security Applications Conf., San Diego, CA, 1996

[6] Paul Syverson, “Onion routing for resistance to traffic analysis”, in proceedings of the DARPA Information Survivability Conference and Exposition on Volume 2, April 2003

[7] Andrew Oram, Steve Talbott , “Managing Projects with make, 2/e”, O'Reilly, 1991

[8] Behrouz A. Forouzan, Tyler Gregory Hicks, “TCP/IP Protocol Suite, 2/e”, McGraw-Hill, 2002

[9] Gary R. Wright, W. Richard Stevens, “TCP/IP Illustrated, Volume 2: The Implementation”, Addison Wesley, 1995

[10] Michael Lucas, Jordan Hubbard, “Absolute BSD: The Ultimate Guide to FreeBSD”, No Starch Press, 2002

[11] W. Richard Stevens, “UNIX Network Programming, Volume 1, 2/e”, Prentice

Hall, 1997

[12] W. Richard Stevens, “TCP/IP Illustrated, Volume 1: The Protocols”, Addison Wesley, 1995

[13] 林慶德, “UNIX 網路程式設計--網路應用程式設計介面 Sockets 與 XTI”, 培生, 2002

[14] 施勢帆, 林毓能, 吳國華, “FreeBSD 實務手冊”, 旗標, 2004

[15] 蔣大偉, “make 專案開發工具”, O'Reilly, 2000

[16] “Ethereal”, http://www.ethereal.com

[17] “OpenSSL”, http://www.openssl.org

[18] “Winpcap”, http://winpcap.polito.it/