Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA
Corso di Laurea in Ingegneria Informatica

TESI DI LAUREA TRIENNALE

Incapsulamento di una trasmissione Peer2Peer in un flusso HTTP (Peer2Peer Mask)

Candidato:
Manuel Olmino
Matricola 575561

Relatore:
Prof. Bertasi Paolo

Anno Accademico 2012/2013

Indice

Introduzione

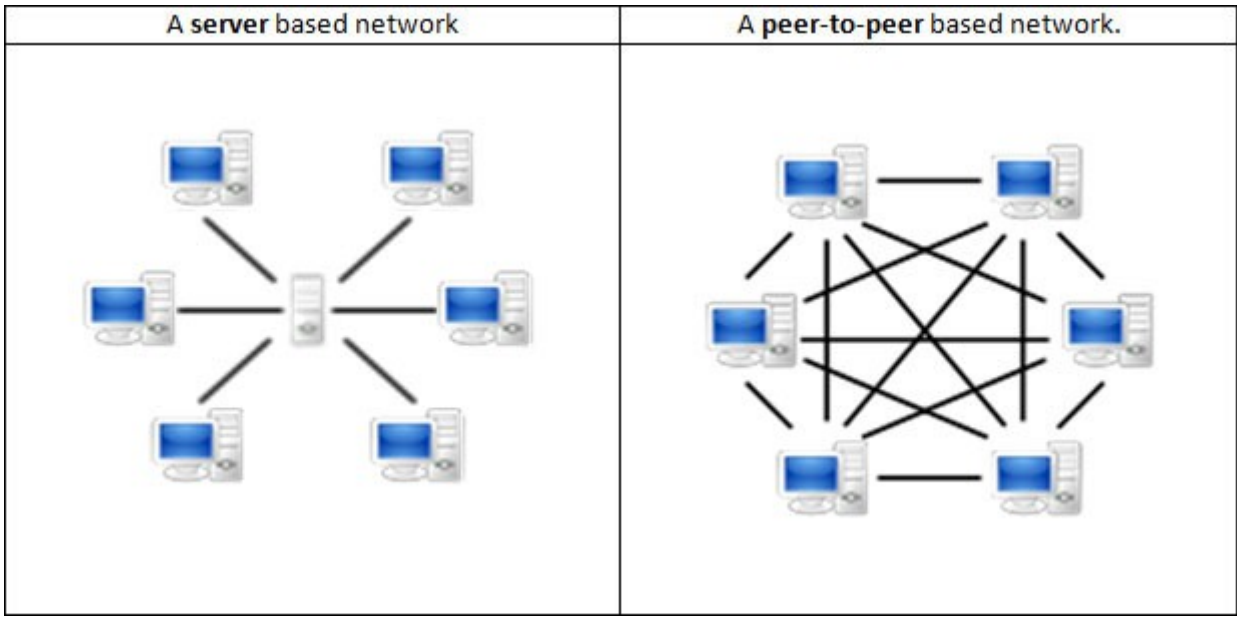
Il successo dei programmi peer-to-peer (P2P) è ormai innegabile. La suddivisione del carico di lavoro in più nodi "uguali", anziché in un solo o pochi server, permette a molti utenti il download di file di svariate dimensioni con estrema facilità. Inoltre la possibilità di condividere file da parte anche di computer casalinghi con modeste capacità di calcolo ha permesso una diffusione di contenuti mai vista prima. Molto spesso, tuttavia, dallo smodato utilizzo di questo mezzo ne consegue una veloce saturazione della banda di connessione e con conseguenti problema per i provider che fornisce il servizio. Tale problematica ha spinto un discreto numero di compagnie ad adottare un sistema di rilevazione e limitazione specifica per il traffico P2P. Spesso la limitazione imposta è così alta che annulla qualsiasi vantaggio nell'utilizzo del protocollo P2P.

Il seguente elaborato propone un sistema mascheramento mediante incapsulamento del traffico P2P volto ad aggirare alcuni di questi controlli e restrizioni. Ad una breve descrizione del protocollo P2P segue un approfondimento del software qui presentato, che verrà chiamato semplicemente "Peer2PeerMask", con particolare attenzione alla cattura e al mascheramento del traffico dati. Segue la spiegazione dettagliata per l'uso dell'applicazione da parte dell'utente. Nell'ultima sezione infine si cercherà di analizzare i pregi e difetti dell'applicazione sviluppata in modo da convincere il lettore che Peer2PeerMask assolve pienamente al compito prefissato.

1. Programmi Peer2Peer

I programmi P2P sono stati concepiti per realizzare una rete tra nodi equivalenti (PEER), i quali svolgono tutti sia la funzione di client che quella di server. Ogni nodo può quindi effettuare o accettare una o più connessioni con gli altri. Nella complessa rete che si viene a creare non vi sono nodi essenziali al suo funzionamento; nel caso anche un'intera parte della rete smettesse di funzionare la rete stessa non ne risentirebbe se non in efficienza. Ma il principale punto di forza sta nel fatto che tutti gli utenti che dispongono di un specifico file lo rendono disponibile al download a tutti gli altri. Anche se ognuno dedicasse solo una piccola parte della propria velocità di upload per condividerlo, l'alto numero garantirebbe un elevata velocità. Ne consegue che, a differenza degli altri protocolli, spesso la velocità di download è limitata solo dalle prestazioni della propria linea e non dalla fonte dalla quale si scarica.

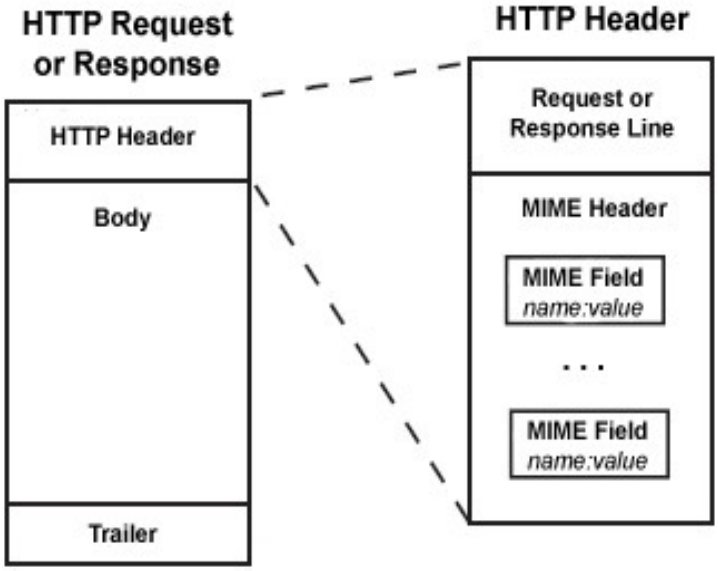
La rete P2P è un'evoluzione della più classica architettura client-server. Quest'ultima è solitamente definita da una o più macchine di considerevoli prestazioni dette server. La loro funzione consiste nel condividere nella rete una o più risorse (non necessariamente un file), le quali possono essere richieste da un numero più o meno elevato di altre macchine dette client. Si può notare come questa soluzione, sicuramente più semplice da implementare, risulti limitata dalle performance del server; a seconda del numero di client connessi può fornire un servizio più o meno di qualità. Si rende quindi necessario dedicare a questo scopo macchine di una certa potenza, e quindi di un certo costo. Inoltre in caso di malfunzionamento di un server, l'intero sistema smetterebbe di funzionare in assenza di un altro server equivalente/di supporto.



Rispetto quindi all'architettura client-server, le reti P2P evidenziano il duplice vantaggio di robustezza e velocità con il risultato di una sempre maggiore diffusione. Ciò ha portato nel tempo a un incremento sostanziale della banda occupata dai vari utenti connessi alla rete e conseguenti problemi legati alla qualità del servizio, non solo a coloro che usano tali programmi, ma a tutti gli utenti. Per tentare di risolvere questo problema, alcune compagnie italiane che forniscono il servizi di connessione (provider), hanno limitato la velocità del traffico P2P per diminuire la banda occupata.

1.1. Problema limitazione traffico Peer2Peer

Alcuni provider, come accennato nel capitolo precedente, hanno la capacità di individuare e limitare il traffico P2P. Per fare ciò è necessario un sistema che permetta di distinguere una trasmissione P2P da un'altra. Quasi tutte le applicazioni, comprese quelle P2P , comunicano attraverso una rete attraverso lo scambio di pacchetti. Si ricorda che i dati da inviare non vengono direttamente inviati nella rete, ma sono prima inseriti in pacchetti corredati da header e trailer.



L'header è la parte iniziale di ogni pacchetto e al suo interno sono presenti informazioni legate alla trasmissione e al contenuto dati. Leggendo quindi i primi byte di un pacchetto e sapendoli interpretare, è possibile ottenere tutta una serie di informazioni fra cui il tipo di traffico. Questa tecnica chiamata ispezione del pacchetto è uno dei sistemi più usati, non solo dai provider, per analizzare il traffico e può essere eseguito attraverso apposite applicazioni chiamate sniffer. Questa tipologia di software cattura tutto o buona parte del traffico passante su di una scheda di rete, per poi analizzarlo.

Questi, tuttavia, mostrano un'evidente debolezza: il tipo di traffico viene dedotto dall'header del pacchetto e non dal suo contenuto. Il programma Peer2Peer Mask

sfrutta proprio questo potenziale difetto incapsulando la trasmissione P2P in pacchetti di tipo HTTP, in modo da evitare il fenomeno sopra esposto. La scelta di usare pacchetti HTTP è motivata dal fatto che questo tipo di trasmissioni non vengono mai limitate e quindi sono adatti allo scopo.

2. Proxy

Il programma Peer2Peer Mask può mascherare le trasmissioni P2P; ma prima di entrare nel dettaglio del progetto occorre spiegare lo strumento usato per gestire le connessioni tra i PEER, il proxy. La funzione principale del proxy è quella di mettere in comunicazione un client con un server. In presenza di un proxy, il client non contatta direttamente il server ma invia una richiesta, secondo specifici protocolli, al proxy; qualora la richiesta venga accettata sarà il proxy stesso a contattare il server per poi metterlo in contatto col client chiamante.

Le varie richieste di connessione che vengono inviate sono composte da una serie di pacchetti, adeguatamente formattati, inviati a uno specifico Socket in ascolto. Grazie a questa particolarità il proxy non deve essere in esecuzione sulla medesima macchina in cui è in esecuzione il client, ma su di una qualsiasi della rete, da una piccola rete LAN alla rete internet. Solitamente viene installato nelle macchine utilizzate per mettere in contatto le reti private con Internet o in server pubblici. E' da notare che il server potrebbe non avere conoscenza del client con cui sta sta realmente comunicando, dando così la possibilità a quest'ultimo di mascherare o coprire del tutto la sua identità. Inoltre nel caso di reti private, tutte le connessioni in uscita devono obbligatoriamente passarvici attraverso, con conseguente possibilità di eseguire ulteriori funzioni di controllo e monitoraggio del traffico.

È importante sottolineare una particolarità del proxy che ne ha influenzato la scelta in questa sede: analogamente ai server, anche i client che hanno lanciato la richiesta non necessariamente sono a conoscenza dell'identità del server contattato. Ne consegue che il proxy può mettere in contatto il client con un qualsiasi server, anche se non è quello richiesto, essendo la connessione gestita al suo interno. Il motivo di questa scelta sarà chiarito più avanti.

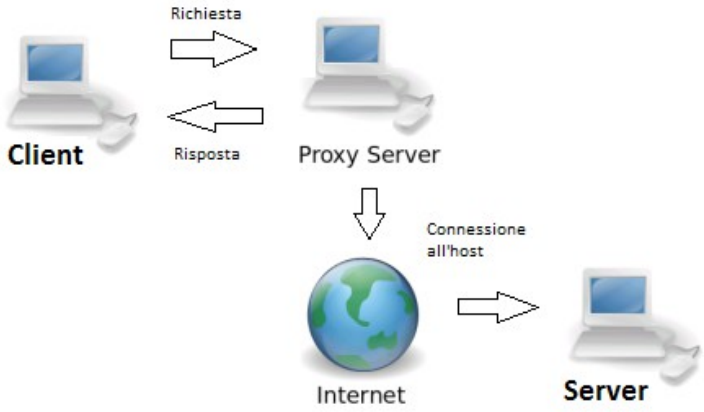
La differenza principale tra il proxy e altre soluzioni al medesimo problema sta nel fatto che il proxy è un programma indipendente da client e server. Indubbiamente un notevole vantaggio è data dalla facilità di installazione e uso: spesso è sufficiente solo avviarlo e configurare l'applicazione con cui lo si vuole usare per ottenere gli effetti desiderati. Quasi tutte le applicazioni di rete permettono l'ausilio di un proxy e per quelle poche che non lo consentono, sono possibili altre soluzioni spesso fornite dallo stesso sistema operativo della macchina. Tuttavia un difetto significativo è dato dal limite di compatibilità fornita dal protocollo usato dal proxy. Esistono diverse categorie di protocolli proxy e molti di essi sono stati ideati per essere usati solo in determinate circostanze, non risultando quindi compatibili con tutti i tipi di traffico. Per arginare a questo problema, Peer2Peer Mask fa uso del protocollo SOCKS, uno fra i più conosciuti e diffusi.

2.1. Proxy SOCKS

I SOCKS di fatto sono particolari tipi di proxy che utilizzano i protocolli SOCKS4 e SOCKS5, evoluzione di SOCKS4, per le richieste; rispetto ad altri protocolli proxy, essi hanno il vantaggio di essere indipendenti dal traffico rediretto. Ciò è legato al fatto che quando la richiesta viene accettata, il client e il server vengono messi in comunicazione in modo "trasparente", rendendo completamente ininfluente il tipo di dati scambiati fra di loro. Il SOCKS4 permette solo un limitato sistema di autenticazione, con l'uso di un userId, e consente solo di effettuare connessioni TCP. Il SOCKS5 invece permette di utilizzare anche connessioni UDP e sistemi di autenticazione più complessi.

Il proxy ha un Socket in ascolto su una porta specifica, che deve essere nota a priori insieme al suo IP, e su questo riceve varie richieste. Nel caso riceva la richiesta di una connessione TCP in uscita (comando *CONNECT*) il proxy si comporta come segue:

- Il proxy riceve la richiesta di connessione. Se accettata, esso apre un Socket verso l'host richiesto; successivamente un messaggio di risposta viene inviato indicando se la richiesta è stata accettata, rifiutata o segnalando l'eventuale impossibilità a stabilire la connessione.
- Se la richiesta è stata accettata e non ci sono stati errori, il proxy mette in comunicazione il Socket dal quale ha ricevuto la richiesta con quello sul quale ha effettuato la connessione. I byte ricevuti da uno vengono inoltrati direttamente all'altro ottenendo cosi una connessione trasparente che può essere di qualsiasi tipo senza problemi.



La connessione cosi instaurata rimane valida finché uno dei due Socket, quello su cui è stata ricevuta la richiesta o quello dell'host contattato, non viene chiuso. In questo caso il proxy chiude anche l'altro, terminando la connessione.

Anche se finora si è accennato solo alla possibilità da parte del proxy di gestire connessioni in uscita, è in realtà previsto anche un comando per le connessioni TCP in ingresso (comando *BIND*). Questo, oltre a non essere utilizzato dai

software P2P, non ha alcuna utilità nel progetto implementato. Per questo l'argomento non verrà affrontato nell'elaborato.

Le connessioni UDP del protocollo SOCKS5 (comando *UDP ASSOCIATE*) sono invece gestite in questo modo:

- Il proxy riceve la richiesta. Se accetta, apre un DatagramSocket in ascolto su una porta che verrà indicata nella risposta, la quale sarà inviata, come in precedenza, anche in caso di rifiuto o errore.
- Il richiedente può ora inviare pacchetti UDP adeguatamente strutturati secondo il DatagramSocket creato dal proxy. Quest'ultimo successivamente potrà decodificarli e inoltrarli verso l'host desiderato.

La connessione, quindi il DatagramSocket del proxy, rimane valida finché il Socket, su cui è stata ricevuta la richiesta, rimane aperta; quando quest'ultimo viene chiuso il proxy provvederà a chiudere il DatagramSocket.

3. Gestione dei collegamenti tra due PEER

Il software P2P, per gestire le connessioni così da farle passare attraverso l'applicazione Peer2Peer Mask, viene configurato per utilizzare un proxy. Tuttavia nel caso di utilizzo senza alcuna modifica esso non sarebbe adatto al nostro scopo. Normalmente il proxy viene impiegato solo per le connessioni in uscita e nel suo funzionamento standard, alla relativa richiesta si connetterebbe direttamente col server da contattare, nel caso specifico un'istanza del programma P2P, situata nell'altro host. Ogni tentativo di mascheramento non potrebbe essere così invertito e il programma P2P riceverebbe dei dati che non saprebbe come interpretare.

Risulta quindi necessario apportare alcune modifiche al funzionamento sopra esposto. Viene in aiuto una delle caratteristiche spiegate nella sezione precedente. I client che effettuano la richiesta di connessione non sono al corrente del server con cui vengono messi effettivamente in contatto. Tale situazione ci permette di aprire una connessione non direttamente col programma P2P, bensì con un altro server appositamente creato, il quale avrà lo scopo di smascherare la trasmissione per poi inoltrarla al programma P2P, a cui era in origine destinata.

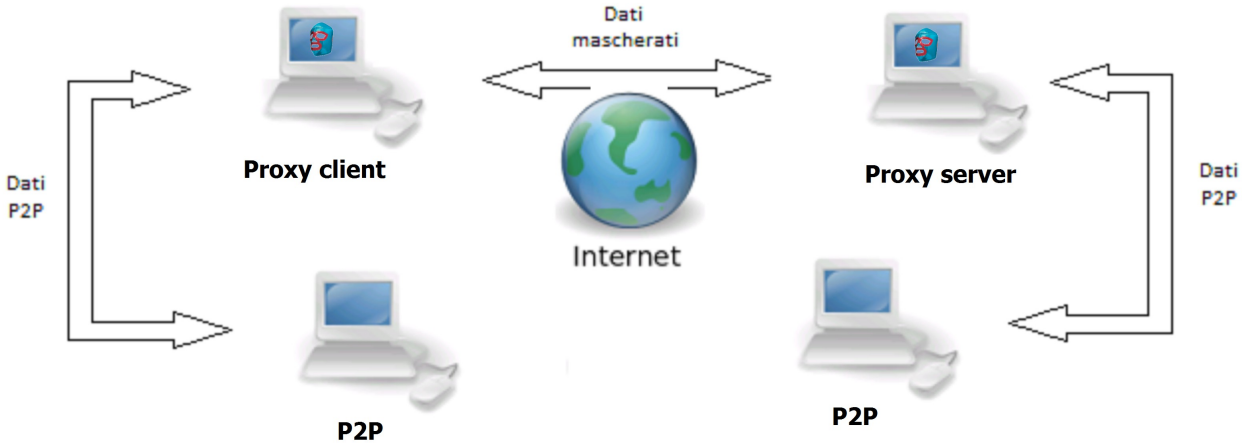
3.1. Deviazione della connessione in uscita tramite Proxy

Nella richiesta di connessione sono indicati IP e porta del server a cui il client vuole connettersi. Nel nostro caso essi sono relativi al programma P2P. Come spiegato, questo non può essere direttamente contattato con una trasmissione mascherata. Le situazioni che si possono presentare a questo punto sono due:

- presso l'IP indicato dal programma P2P (rete o macchina a seconda della situazione), l'host non è munito di un'altra istanza di Peer2Peer Mask; in questo caso non essendoci niente che possa smascherare la trasmissione modificata, Peer2Peer Mask (presente nell'host che effettua la connessione) non può che lasciare inalterato il traffico (modalità trasparente) da inviare.
- Un'istanza di Peer2Peer Mask è presente anche nel server contattato. Essa in questo contesto funge da server fittizio atto allo smascheramento e inoltro

della trasmissione al programma P2P dell'host destinatario. Ciò rende possibile la comunicazione fra client e server in modalità “mascherata”.

Per stabilire in quali delle due situazioni ci si trova è sufficiente tentare di contattare prima il server del programma Pee2Peer Mask, il quale, se presente, è in ascolto su di una porta fissa e quindi nota. Se questo risponde, dimostrandone la presenza, è possibile avviare una trasmissione mascherata, altrimenti è necessario connettersi in modalità trasparente all'IP e alla porta indicata nel pacchetto originale. Le modifiche effettuate al proxy si limitano quindi solo ad un iniziale tentativo di connessione su di una porta diversa da quella indicata nella richiesta, oltre ovviamente a tutto il sistema di mascheramento della trasmissione. Con questo intuitivo stratagemma è possibile controllare le connessioni tra i PEER permettendo di gestire il traffico dati da entrambi i lati. Inoltre il sistema, se adeguatamente configurato, mantiene il vantaggio, tipico del proxy, di funzionare anche se in esecuzione su di una macchina diversa da quella su cui è in funzione il/i software P2P; la motivazione verrà spiegata più avanti.



Sono previste 3 modalità di funzionamento di Peer2Peer Mask, selezionabili al suo avvio o successivamente tramite console.

- Transparent mode

In questa modalità il programma funziona esattamente come un normale proxy: elabora le richieste ricevute, tenta la connessione all’indirizzo e porta indicati, quindi mette in comunicazione chi ha fatto la richiesta con l’host in modo trasparente senza alcun tipo di mascheramento.

Inoltre il server software di Peer2Peer Mask non accetta connessioni, essendo queste solo eventuali trasmissioni mascherate provenienti da altre istanze del medesimo proxy.

- Mask mode

In questa modalità il programma tenta di deviare le connessioni. Alla richiesta di connessione a uno specifico indirizzo e porta, prima tenta di connettersi a quell'indirizzo sulla porta prestabilita per il server Peer2Peer Mask. In caso di risposta affermativa, un'altra istanza di Peer2Peer Mask è in esecuzione nell'host destinatario. E' perciò possibile instaurare, e mascherare, la connessione. Qualora il tentativo di connessione non andasse a buon fine (in altri termini: nell'host destinatario non è attiva un'istanza del nostro programma), si passerebbe alla modalità di funzionamento proxy normale, nella quale viene tentata una connessione all'indirizzo e porta indicati nella richiesta e, in caso di successo, viene avviata una trasmissione trasparente.

Le richieste dove l'IP è quello locale, sia privato che pubblico, in questa modalità vengono rifiutate. La scelta, apparentemente insensata, è dovuta a certi comportamenti anomali di alcuni programmi P2P consistenti in ripetuti tentativi di connessione a loro stessi. Questo può portare a tutta una serie di problematiche, anche gravi, tra cui il possibile ban dell'IP del proxy. Ciò porterebbe al rifiuto di tutte le connessioni in ingresso inoltrate dal server, con la conseguente impossibilità di ricevere trasmissioni mascherate in ingresso. Si fa notare che in questa modalità il programma Peer2Peer Mask può ancora essere utilizzato con un normale proxy. Si ha solo un ritardo di qualche secondo nell'instaurazione di una connessione trasparente dovuto al tempo necessario per accertarsi del fallimento di quella mascherata.

- only Mask mode

Come per la modalità precedente, dopo aver ricevuto la richiesta, si tenta una connessione prima alla porta prestabilita per il server di P2P Mask e in caso di successo viene avviata una trasmissione mascherata. La sostanziale differenza, rispetto a prima, è rappresentata dal fatto che, in caso di fallimento, il software si limita ad inviare direttamente un messaggio di errore al richiedente. Di conseguenza non viene eseguita alcuna connessione trasparente permettendo il passaggio solo a quelle mascherate. Anche in questa modalità le richieste all'IP locale vengono rifiutate.

3.2. Server per smascheramento delle connessioni in ingresso

In modalità Mask mode (o "only mask mode"), è necessario che la trasmissione passi attraverso il server costituito un'altra istanza di Peer2Peer Mask per lo smascheramento, il quale dovrà poi inoltrarla al software P2P di destinazione. Per l'inoltro si rende necessario conoscere IP e porta del destinatario. Queste informazioni, tuttavia, non sono note a priori al server e non sono

facilmente ricavabili dai dati ricevuti. Per ovviare al problemi si sono utilizzati i seguenti accorgimenti.

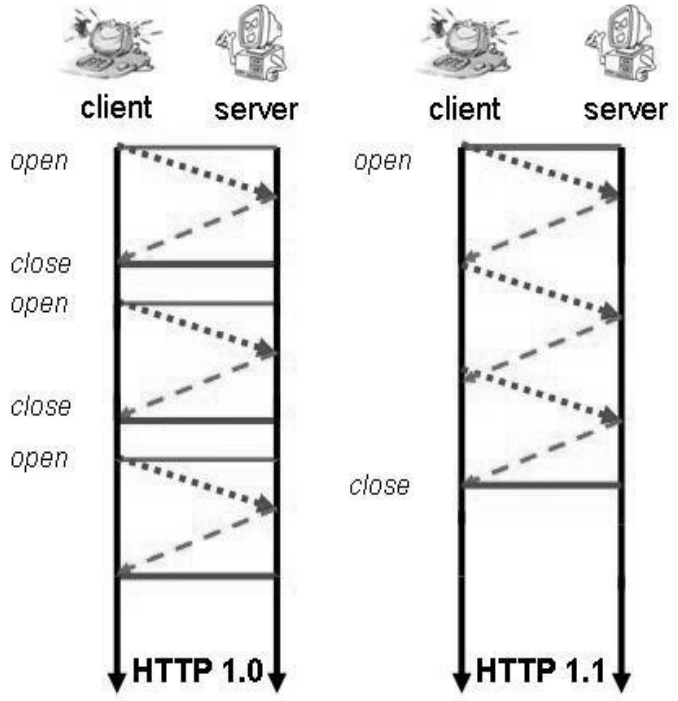
Innanzitutto bisogna conoscere la porta, poiché questa informazione, come verrà spiegato di seguito, è utilizzata per ottenere indirettamente l'IP; per individuarla viene in aiuto il proxy modificato che ha contattato il server. Questo infatti è a conoscenza della porta del programma P2P, essendo la stessa indicata nella richiesta di connessione. Risulta dunque sufficiente che il proxy, all'avvio di una comunicazione mascherata con il server, invii 2 byte con indicata quella porta. A tal punto, il server, per conoscere la porta, dovrà soltanto leggere questi primi 2 byte ogni volta che riceve una connessione.

Si fa necessaria più attenzione per il rilevamento dell'IP. Se l'istanza di Peer2Peer Mask, fungente da server, è eseguita nella stessa macchina del software P2P di destinazione, allora basta utilizzare l'IP locale. Tuttavia, come affermato in precedenza, si è voluto mantenere il vantaggio del proxy di funzionare in una qualsiasi macchina delle rete. Per ottenere questo risultato, si sono presi in esame due casi: uno nell'ipotesi in cui ci siano molteplici applicazioni P2P che utilizzano la medesima istanza di Peer2Peer Mask e l'altro nell'ipotesi in cui ce ne sia solo uno. Con il primo viene impiegato un sistema analogo al *port forwarding:* l'utente inserisce una serie di coppie porta\IP. Quando il server riceve la porta dal proxy controlla se essa è abbinata ad un IP e, nel caso lo sia, utilizza quest'ultimo per l'inoltro della trasmissione smascherata. La soluzione descritta permette di avere un numero arbitrario di programmi P2P in esecuzione, ma richiede che ognuno di essi sia in ascolto su di una porta diversa.

Se la porta non è presente nell'elenco, l'IP utilizzato sarà quello che ha effettuato il maggior numero di richieste al proxy. E' importante far notare che, in tal caso, il sistema funziona solo se ad usare il proxy è un solo programma P2P, indifferentemente che sia in esecuzione sulla stessa macchina o su una diversa. Nonostante questo, l'accorgimento ha il vantaggio di essere completamente automatico visto che non richiede alcun intervento da parte dell'utente finale e non presenta problemi nel caso in cui la configurazioni del sistema cambi.

4. Incapsulamento HTTP

Di seguito viene affrontato il mascheramento nel dettaglio. Come detto in precedenza, si vuole mascherare le trasmissioni P2P come HTTP. Il formato HTTP (*HyperText Transfert Protocol*) è il sistema principalmente utilizzato per la navigazione su web. L'utilizzo maggiore è per l'invio di file come pagine web, ma può essere impiegato anche per altri scopi, come ad esempio lo streaming audio/video. Il protocollo è di tipo request/response: il client crea una connessione TCP/IP con il server e invia un comando, solitamente la richiesta di un file, mediante un pacchetto. Il server a questo punto, invia anch'esso un pacchetto nel quale è contenuta la risposta alla richiesta ed eventualmente il file richiesto. A questo punto nella vecchia versione HTTP\1.0 il server si sconnette mentre nella HTTP\1.1 può rimanere in attesa di un'altra richiesta.



Sia gli header dei pacchetti di richiesta che quelli di risposta sono scritti in caratteri ASCII. Questo non risulta particolarmente efficiente, ma è un'eredità dovuta all'età del protocollo che ne semplificava la lettura da parte dell'utente.

4.1. Utilizzo della porta 80

Le trasmissioni di tipo HTTP fanno spesso uso della porta n.80, la quale è ormai uno standard per le stesse. Si è scelto quindi di usarla anche per le connessioni mascherate in modo da ottenere un traffico dati il più possibile simile ad una classica trasmissione HTTP. Questa scelta però ha anche alcuni svantaggi legati

proprio al suo ampio utilizzo. I principali problemi da tenere in considerazione sono:

- Per fare partire un server in ascolto sulla porta 80 è necessario avviare il programma come root/amministratore. In caso contrario il programma funzionerebbe ugualmente, poiché in grado di connettersi agli altri PEER, ma non potrebbe ricevere connessioni mascherate in ingresso.
- La porta 80 può essere in uso anche da altri programmi per le connessioni in ingresso, quali ad esempio Skype o server web. Se uno di questi programma è già in ascolto sulla porta, all'avvio del server questo non potrà utilizzarla avendo quindi gli stessi problemi spiegati nel punto precedente. Qualora si verificasse questa situazione, essa viene segnalata da un messaggio di errore all'avvio del programma Peer2Peer Mask. L'utente, in caso di questa segnalazione, sarà costretto a chiudere tutti i software in ascolto sulla porta 80 prima di poter avviare di nuovo P2P Mask.
- Se l'host da contattare ha un programma in ascolto sulla porta 80, il programma Peer2Peer Mask si comporterà in maniera anomala, cioè come se stesse comunicando con il server per lo smascheramento. Difatti non essendo previsto alcun protocollo tra proxy modificato e server, l'unico parametro usato per decidere se avviare o meno una comunicazione mascherata è la presenza di un programma in ascolto sulla porta 80. Ne consegue che, in presenza di questi host, non verrà mai tentata una connessione trasparente e quindi non sarà possibile avviare una comunicazione con il programma P2P.

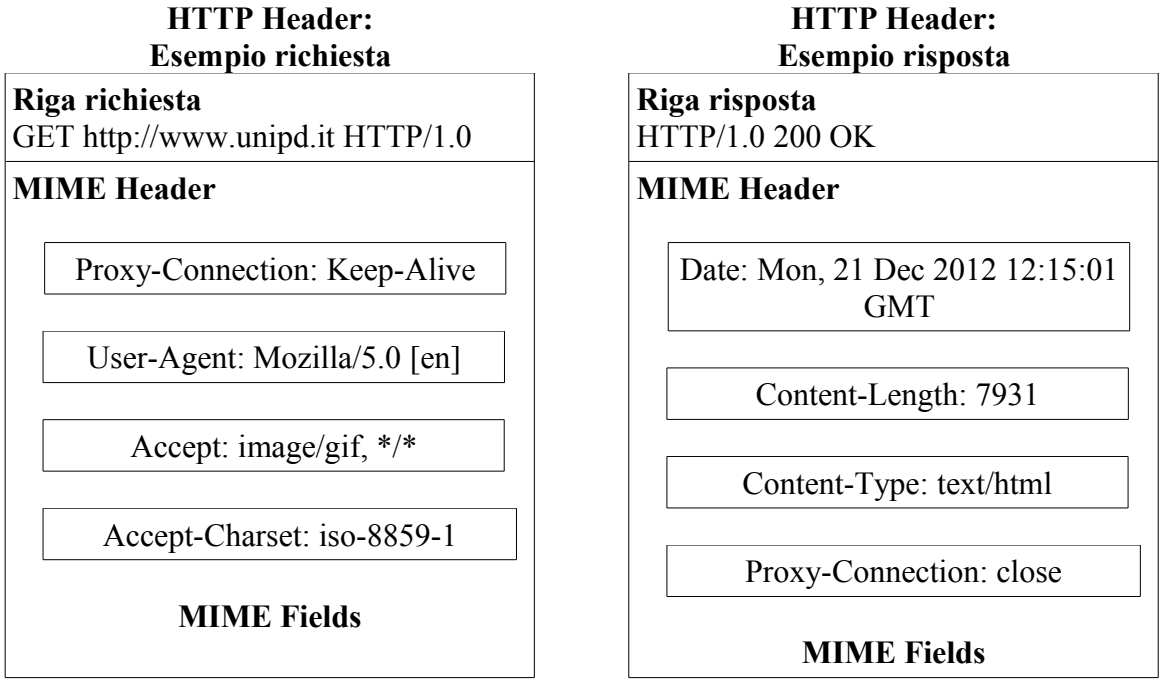
4.2. Aggiunta dell'header HTTP e GIF

E' importante notare che il protocollo P2P è profondamente diverso rispetto all'HTTP. Il primo prevede, dopo una fase di handshake, trasmissioni di tipo quasi unidirezionale. La maggior parte della trasmissione è inoltre composta da pacchetti contenenti i byte del file inviati da uno dei due PEER all'altro. L'HTTP invece rientra nella categoria dei request/response; a ogni pacchetto inviato dal client ce n'è uno di risposta dal server. Questa differenza non può essere colmata con il sistema di incapsulamento usato dal programma Peer2Peer Mask e, senza l'implementazione di un protocollo sottostante, non è possibile simulare in modo permetto una trasmissione HTTP. Ciò non viene fatto per mantenere il mascheramento in più leggero possibile, infatti l'incapsulamento, per quanto meno efficace, risulta essere più rapido ed efficiente con conseguente basso overhead di CPU e banda. Per ottenere un mascheramento il quando più possibile efficace, tutti i byte della trasmissione P2P vengono incapsulati in pacchetti di risposta contenenti immagini GIF. Ciò permette di simulare l'invio di immagini da parte di un server web.

Per il mascheramento si leggono buffer di grandezza massima 0x2000 (8192 byte), dai quali viene creato il pacchetto per l'invio. Ad ognuno di essi viene anteposto un header HTTP e poi uno GIF. La dimensione di 8 kB è stata scelta in quanto simile a quella di una tipica immagine GIF. Gli header HTTP sono composti da parametri scritti in codice ASCII separati dal carattere di new line e da un doppio new line per indicare la fine e l'inizio degli eventuali dati. Nei pacchetti di risposta del server il primo parametro è sempre il codice di stato della richiesta; il quale contiene la versione del protocollo HTTP usato (da anni la 1.1) seguito dal codice della risposta il quale è composto da un numero di 3 cifre, più la stringa corrispondente a quel codice. La prima delle tre cifre indica il tipo di risposta:

- 1 indica che la risposta contiene informazioni sullo stato della richiesta
- 2 indica che la richiesta è stata accettata
- 3 indica che la richiesta non presenta errori ma non è formulata in modo corretto. Nei dati del pacchetto è indicata la correzione
- 4 indica che la richiesta è errata
- 5 indica problemi al server

Il codice usato dal programma è sempre lo stesso, cioè "***HTTP/1.1 200 OK***", che corrisponde all'accettazione della richiesta fatta con invio del file richiesto, mediante protocollo 1.1. Il resto dei parametri non è obbligatorio e fornisce altre informazioni come il tipo di server web, la data dell'invio del pacchetto o il tipo di dati presente nel pacchetto. Questi sono formati da una stringa, che indica il tipo di parametro, seguita dal valore del parametro stesso, con "due punti" come carattere di separazione. L'unico di questi che ha un effettiva importanza per il mascheramento è quello indicante il tipo di dati inviati, cioè "*Content-Type*". Il buffer di byte che noi inviamo sono 'casuali', non sono cioè strutturati come un file e non possono certamente essere visti come file HTML, audio/video o immagini. Una possibile scelta, dunque, potrebbe essere quella di usare come tipo "*application/octet-stream*" il quale sta ad indicare un flusso dati relativo ad un programma incapsulato in un pacchetto HTTP. Tuttavia questo tipo di flusso potrebbe essere comunque controllato e/o direttamente limitato dai provider. Viene quindi utilizzato come tipo "*image/gif*" che indica, come è facile intuire, un file di tipo immagine GIF. Il parametro viene allora impostato nel seguente modo: "***Content-Type: image/gif***". L'header GIF viene inserito tra quello HTTP e i dati in modo che il pacchetto in questione non venga rilevato come corrotto in un eventuale controllo del suo contenuto. Relativamente al problema della frammentazione dei pacchetti, che verrà trattato successivamente, vengono aggiunti anche i parametri "***Accept-Ranges: bytes***" e "*Content-Length*". Il primo indica che la dimensione dei dati del pacchetto è in byte, mentre il secondo ne indica la dimensione. Vengono inseriti anche altri parametri che non hanno un'utilità specifica per il mascheramento ma che nella pratica vengono normalmente usati: "***Server: Apache/2.4.1 (Unix)***", "***Keep-Alive: timeout=5, max=100***", "***Connection: Keep-Alive***" e "*Date*".



Questo procedimento, come si può notare, è molto veloce, semplice da eseguire e comporta un bassissimo utilizzo della CPU. Tuttavia con l'inserimento di questi header, si ha un'aggiunta di circa 260 byte per ogni buffer di 8192 inviato; si ottiene quindi un overhead circa del 3% di byte inviati con conseguente diminuzione di banda disponibile per i dati P2P. Il valore non è ottimo visto l'uso di un semplice incapsulamento e potrebbe essere migliorato diminuendo il numero di parametri non indispensabili al mascheramento o aumentando la dimensione del buffer; tuttavia si è deciso di mantenere questa scelte per avere una maggiore sicurezza nell'ispezione.

4.3. Mascheramento byte trasmissione Peer2Peer

Con il mascheramento fin qui spiegato, durante il quale i buffer vengono incapsulati con l'aggiunta degli header HTTP e GIF, ad un'ispezione standard del flusso non ci sarebbero problemi. Per ispezione standard intendiamo l'ispezione degli header dei pacchetti HTTP e la verifica che il contenuto sia corrispondente a quello indicato. Purtroppo, nel caso di una ispezione più approfondita, potrebbero essere individuati gli header della trasmissione P2P.

Per risolvere il problema, il programma Peer2Peer Mask applica un'ulteriore mascheratura al buffer di byte che viene incapsulato eseguendone il not dei bit. Questa semplice operazione è sufficiente a rendere impossibile il rilevamento degli header P2P. L'ultima modifica esposta, inoltre, è semplice, non aumenta il carico

sulla CPU, non rallenta la velocità di trasmissione e, infine, risulta facilmente invertibile eseguendo un'ulteriore not dei bit.

4.4. Problema frammentazione e smascheramento

Quando un pacchetto viene inviato attraverso la rete internet, questo passa attraverso una serie si sotto reti, in ognuna delle quali la dimensione massima dei pacchetti che vi può transitare può variare notevolmente; nel caso in cui la dimensione del pacchetto vi sia superiore, quest'ultimo viene frammentato in pacchetti più piccoli. Questo comporta che un pacchetto HTML inviato dal proxy modificato, può arrivare diviso al server. Per questo motivo nei parametri dell'header HTTP è stato inserita la dimensione del pacchetto stesso.

Nel momento in cui il server riceve un pacchetto, ne legge l'header controllando solo il parametro della dimensione. In questo modo si può capire se il pacchetto è stato ricevuto per intero e nel caso contrario i byte arrivati vengono bufferizzati fino al suo completamento.

La fase successiva consiste nello smascheramento del pacchetto ricevuto; la prima parte di questa operazione è semplice perché sarà sufficiente togliere gli header HTTP e GIF, ma per fare ciò è necessario sapere da quanti byte sono composti. La dimensione di quello HTTP è variabile ma sapendo che termina con un doppio newline, è sufficiente cercare questi due caratteri per sapere dove finisce; diversamente quella GIF è costante.

Dopo aver cancellato i byte degli header, rimangono quelli della trasmissione P2P negati e sarà quindi sufficiente rieseguire il not dei bit per ottenere quelli originali.

Con il completamento di questa operazione i byte ora smascherati possono essere inoltrati al programma P2P.

5. Implementazione

Il codice scelto per implementare il programma Peer2Peer Mask è Java. Questa scelta è dovuta a un serie di motivazioni tecniche di cui la più importante è sicuramente l'alta compatibilità. Infatti, a differenza di quasi tutti gli altri linguaggi di programmazione, Java viene compilato di default per una specifica macchina virtuale chiamata *Java Virtual Machine* (JVM). Ne consegue che il codice compilato, chiamato *bytecode*, può essere eseguito su qualsiasi macchina purché abbia la JVM installata. Essendo Java, nella variante Javascript, ampiamente utilizzato sopratutto nel web e non solo, quasi tutte le macchine dispongono di una JVM rendendo quindi il programma Peer2Peer Mask utilizzabile senza dover installare o configurare niente.

La versione di Java utilizzata durante l'implementazione è la jre6 e il programma è stato testato su sistemi operativi Windows, Linux e Mac OS.

Il programma è composto da 5 file .java [Proxy.java, Socks4.java, Socks5.java, SupportThread.java, FrameProxy.java]. Viene di seguito illustrato, attraverso uno schema, il funzionamento del codice con i vari metodi principali chiamati nel corso di connessioni in uscita e ingresso.

OUT TCP

```
run (classe Proxy)
├ getSocks()
│ └ Socks4() -o- Socks5()
├ response()
│ └ connect()
├ getSocket2()
├ getSocket()
└ flowTCP()
  ├ ByteFlowTCP() [IN]
  │ └ packetBuffer()
  └ ByteFlowTCP() [OUT]
    └ mask()
```

OUT UDP

```
run (classe Proxy)
├ getSocks()
│ └ Socks4() -o- Socks5()
└ response()
  └ respnseUDP()
    └ flowUDP()
      └ ByteFlowUDP()
        └ decodePacket()
```

IN TCP

```
run (classe ServerTCP)
└ flowTCP()
  ├ ByteFlowTCP() [IN]
  │ └ packetBuffer()
  └ ByteFlowTCP() [OUT]
    └ mask()
```

5.1. Inizializzazione

Anche se sono presenti all'interno del codice diversi ***main()*** usati in fase di test e debug, il metodo invocato all'avvio del programma Peer2Peer Mask è quello della classe ***Proxy*** presente nell'omonimo file. Questo metodo ha lo scopo di eseguire i

controlli iniziali e inizializzare tutto quello che è necessario al programma per funzionare.

Per prima cosa legge e interpreta i parametri passati all'avvio, questi permettono di variare alcune impostazioni e saranno descritti nel dettaglio nel capitolo 6.2.

Successivamente, se non richiesto diversamente, inizializza l'interfaccia grafica mediante il metodo ***init()*** della classe ***FrameProxy*** *e il* ***ServerSocket*** sul quale vengono ricevute le richieste di connessione alla porta di default 1080.

Segue la stampa a console di alcune scritte contenenti informazioni riguardanti il programma come versione, modalità di funzionamento e porta in ascolto del proxy.

Infine avvia i 2 *Thread* per la gestione delle connessioni in uscita e ingresso (***Proxy*** e ***ServerTCP***) e quello per il controllo delle velocità di download e upload.

5.2. Connessioni in uscita

Le connessioni in uscita, gestite dalla classe ***Proxy***, vengono elaborate da diversi *Thread* in parallelo in modo da ottimizzare il processo. Ne vengono avviati 3 per ogni connessione, il primo (***Proxy***) è la scopo di elaborare la richiesta di connessione ricevuta dal proxy, inizializzare la connessione e avviare gli altri 2 *Thread* (***ByteFlowTCP***) incaricati uno di gestire i byte inviati e l'altro quelli ricevuti nel caso di connessione TCP, mentre 1 solo *Thread* (***ByteFlowUDP***) in caso di connessione UDP. Se così non fosse il programma risulterebbe altamente inefficiente in quanto in un dato istante sarebbe gestito solo l'invio o la ricezione byte di una sola connessione.

Quando il programma Peer2Peer Mask viene avviato viene inizializzato un *Thread* ***Proxy*** che rimane in attesa di una richiesta di connessione, quando questa viene ricevuta viene immediatamente inizializzato un altro *Thread* ***Proxy*** in modo che ce ne sia sempre uno in attesa.

Viene così inizializzato un *Socket* con il quale comunicare con il client. Da questo viene letto il primo pacchetto tramite il metodo ***getSocks(),*** il quale ha lo scopo di capire se si sta utilizzando il protocollo socks4 o socks5; successivamente invia il pacchetto al costruttore della classe corretta e restituisce un oggetto di tipo ***Socks***. Ciò può essere fatto in modo molto semplice leggendo il primo byte del pacchetto, se questo è 4 sarà creato e restituito un nuovo oggetto ***Socks4***, mentre se è 5 un oggetto ***Socks5***; diversamente, essendo un protocollo sconosciuto, viene lanciato un errore che fa terminare il *Thread* ***Proxy*** e di conseguenza la connessione con il client dal quale si è ricevuta la richiesta.

L'interfaccia ***Socks***, implementata dalle classi ***Socks4*** e ***Socks5*** presenti negli omonimi file .java, permette alla classe ***Proxy*** di gestire la richiesta nello stesso modo indipendentemente dal protocollo usato. Infatti, richiamando il metodo ***response()*** passandogli il Socket sul quale è stata ricevuta la richiesta, viene gestita la decodifica del pacchetto di richiesta, l'inizializzazione della connessione con l'host remoto e l'invio di un pacchetto contenente la risposta. Nello specifico,

le due classi si comportano in modo diverso nella gestione dei pacchetti di richiesta e risposta per via della diversità dei protocolli, ma la parte legata all'inizializzazione delle connessione TCP, metodo ***connect()***, è identica. Per prima cosa il metodo verifica quale comando è stato inviato nella richiesta; con *CONNECT* viene richiesto di contenersi a un host remoto, mentre con *BIND* è richiesto di mettersi in ascolto in attesa di una trasmissione in ingresso, entrambe tramite connessione TCP. Se è richiesta la connessione a un host viene prima verificata in quale modalità di funzionamento (vedi capitolo 3.1) è impostato il programma, se non si è in "*trasparente mode*" viene tentata la connessione, attraverso un *Socket*, alla porta della connessioni mascherate e in caso di successo viene inviata 2 byte contenenti la porta presente nella richiesta (vedi capitolo 3.2). In caso di insuccesso, se non si è in modalità "*only mask mode*", o in "t*rasparente mode*" viene tentata una connessione alla porta della richiesta. Nel caso l'IP a coi connettersi è uno di quelli locali non viene tentata alcuna connessione. Per il comando BIND il metodo rimane in attesa di una connessione in ingresso e termina quando questa arriva. Per entrambi i comandi il metodo ritorna un *int* per indicare lo stato del procedimento, viene restituito 0 se non è stato inizializzato nessun Socket, 1 se è stata avviata una connessione trasparente e 2 se una mascherata.

Attraverso il metodo ***getSocket()*** in caso di comando *CONNECT* e ***getSocket2()*** in caso di comando *BIND*, la classe ***Proxy*** riceve il Socket con il quale è stata avviata la connessione con l'host remoto e mediante il metodo ***flowTCP()*** mette in comunicazione quest'ultimo con quello dal quale è stata ricevuta la richiesta. Questo metodo inizializza due *Thread* ***ByteFlowTCP***, uno per i byte in ingresso e uno per quelli in uscita, fornendogli un *boolean* per indicare se la connessione è mascherata o meno. Sarà poi compito di questi due mascherare i byte in uscita e smascherare quelli in ingresso. Quando uno dei due *Thread* termina il metodo chiude anche l'altro terminando la connessione.

Per quanto riguarda le connessioni UDP del protocollo socks5 (comando *UDP_ASSOCIATE*), avendo un funzionamento completamente diverso dalla precedenti (vedi capitolo 2.1) il flusso di byte non viene gestito dalla classe ***Proxy*** ma direttamente all'interno di ***Socks5***. Il metodo ***response()*** non procede, come nel caso precedente, chiamando il metodo ***connect()*** dopo aver ricevuto il pacchetto della richiesta, ma chiama il metodo ***responseUDP()***. Questo inizializza un *DatagramSocket* su quale il client invierà i pacchetti UDP da inoltrare. La porta su cui è in ascolto viene incrementato di 1 ogni volta che questo metodo viene chiamato in modo da non avere conflitto. Infine viene chiamato il metodo ***flowUDP()*** presente nella classe ***Proxy***. Questo inizializza in *Thread* ***ByteFlowUDP*** il quale ha il compito di ricevere i pacchetti UPD, decodificarli e inoltrarli. Quando il *Socket* su cui è stata ricevuta la richiesta viene chiuso viene terminato anche il *Thread* e di conseguenza anche l'inoltro dei pacchetti.

5.3. Connessioni in ingresso

Anche per le connessioni in ingresso vi sono 3 *Thread* per la gestione di ognuna. Durante l'inizializzazione del programma Peer2Peer Mask viene avviato un *Thread* ***ServerTCP*** in attesa di una connessione in ingresso e, come per la connessioni in uscita, quando arriva subito avvia un altro *Thread* ***ServerTCP*** in modo da averne sempre uno in attesa. Queste connessioni, arrivando sulla porta (di default 80) dedicata alle trasmissioni mascherate in arrivo dal proxy modificato, vengono trattate come tali.

Dopo che la connessione è stata accettata, vengono letti i primi 2 byte, questi sono la porta del programma P2P (vedi capitolo 3.2) a cui inoltrare la trasmissione dopo che è stata smascherata. Viene quindi tentata una connessione all'IP e porta su cui dovrebbe essere in ascolto il programma P2P, se questa va a buon fine vengono messi in comunicazione i due Socket tramite il metodo ***flowTCP()***, spiegato nel capitolo precedente, indicando di mettere in comunicazione i due Socket mediante trasmissione mascherata.

Anche se, come spiegato nel capitolo 4.1, la porta 80 può essere utilizzata da altri applicazioni con conseguente possibilità di una trasmissione in ingresso non da un altro programma Peer2Peer Mask, non vi sono specifici controlli per queste situazioni. Questo però non causa problemi perché la classe ***ByteFlowTCP*** si aspetta sempre pacchetti HTTP formattati in un certo modo, per cui altre trasmissioni causerebbero subito il lancio di errori che farebbero terminare la connessione.

5.4. Classi ByteFlowTCP e ByteFlowUDP

Le classi ***ByteFlowTCP*** e ***ByteFlowUDP*** citate nei capitoli precedenti sono utilizzata per la gestione del flusso di byte in ingresso e uscita. Essendo estensioni della classe *Thread* possono essere eseguite in parallelo per ottimizzare la gestione delle connessioni.

La classe ***ByteFlowTCP*** serve a gestire il flusso di byte in una direzione di una connessione TCP, è quindi necessario avviarne 2 per avere un flusso bidirezionale. Al momento dell'inizializzazione è necessario passargli i due *Socket* da mettere in contatto, facendo attenzione di inserire nel corretto ordine quello dal quale leggere i byte e quello a cui inviarli, se il flusso è in ingresso o in uscita e se la connessione è mascherata. Tra flusso in ingresso o in uscita non c'è differenza quando la connessione è trasparente, ma se mascherata quello in ingresso deve essere smascherato mediante il metodo ***packetBuffer()*** mentre quello in uscita mascherato tramite ***mask()***. I buffer di byte vengono letti da un *Socket* per essere mascherati/smascherati e inviati all'altro.

Il metodo ***mask()*** è quello dedicato all'incapsulamento, gli vengono passati gli array di byte letti dal *Socket* connesso al programma P2P per essere negati bit a bit

e uniti agli header HTTP e GIF (vedi capitolo 4.2 e 4.3). Per l'header HTTP viene calcolata la dimensione dell'array più l'heder GIF per essere inserita come dimensione dei dati del pacchetto. Infine il risultato è restituito come array di byte per essere inoltrato attraverso il *Socket* all'host remoto.

Il metodo ***packetBuffer()*** ha invece il compito opposto di eliminare gli header per smascherare il flusso dati. Ricevuto l'array di byte letti dal *Socket* connesso all'host remoto, questo viene trattato in due modi differenti a seconda se il precedente pacchetto HTTP è stato completamente ricevuto o no. Nel primo caso il metodo verifica la presenza dell'header HTTP e legge il valore della dimensione dei dati; tale informazione è utilizzata per capire se il pacchetto è stato completamente ricevuto e in tal caso restituisce, dopo aver effettuato la negazione bit a bit, l'array privo degli header. In caso contrario l'array, sempre negato bit a bit e privo di header, viene bufferizzato in attesa della completa ricezione e restituito un array di dimensione 0. Se i byte che vengono ricevuti non coincidono con quello che il metodo si aspetta di trovare non viene tentata al con tentativo di ripristino o correzione, ma viene invece lanciato un errore che fa terminare il Thread e di conseguenza la connessione.

La classe ***ByteFlowUDP*** è usata per la gestione delle connessioni UDP del protocollo socks5. Quando viene inizializzato riceve un *DatagramSocket* sul quale il programma P2P invierà i pacchetti UDP da inoltrare. Ogni volta che viene ricevuto un pacchetto viene decodificato con il metodo ***decodePacket()*** e inviato. Infatti i pacchetti ricevuti non contengono solo le informazioni da inviare ma anche un semplice header contenente a chi inoltrarle.

5.5. Classi di supporto e interfaccia grafica

Oltre alle classi spiegati nei capitoli precedenti ne sono presenti altre con lo scopo di fornire funzioni di supporto per un miglior funzionamento del programma e controllo da parte dell'utente.

- ***Conn***: questa classe è dedicata alla memorizzazione delle connessioni in corso e recentemente terminate. A ogni nuova richiesta ricevuta o connessione in ingresso viene inizializzato e legato un nuovo oggetto ***Conn***. Appositi metodi all'interno di tutto il codice del programma Peer2Peer Mask mantengono aggiornate le informazioni sullo stato di quella connessione all'oggetto ***Conn*** legato. Tali informazioni racchiudono lo stato in cui si trova la richiesta o in generale la connessione, il tipo di trasmissione (TCP o UDP, mascherata o trasparente) e i byte ricevuti/inviati. Quando una trasmissione viene terminata vengono mantenute le informazioni solo delle ultime 10 più recenti, tutte le altre vengono eliminate. Nonostante non abbia una funziona essenziale nel funzionamento del programma, le informazioni che fornisce all'utente sono molto utili per permettere un controllo rapido

sullo stato attuale delle connessioni senza dover controllare tutti i messaggi di log.

- ***IpList***: mediante questa classe il programma Peer2Peer Mask gestisce quasi tutti gli aspetti legati agli IP. Mediante il metodo ***getClienteIp()*** permette di ricavare gli IP a cui inoltrare le connessioni in ingresso. Con ***addRequestIp()*** si può memorizzare quelli da cui è stata ricevuta una richiesta di connessione. Utilizzando ***addPortIp()*** è possibile creare una copia porta/IP utilizzata per inoltrare le connessioni in ingresso (vedi capitolo 3.2). Infine con ***isLocalIp()*** è possibile conoscere gli IP locali e viene utilizzato per rifiutare le richieste a questi IP. Si può notare che questo metodo può potenzialmente essere utilizzato anche per filtrare IP esterni a cui non si vuole che il programma P2P si connetta.
- **Output**: classe dedicata alla stampa dei messaggi del programma. I 2 metodi per la stampa (***print()*** e ***println()***) prevedono, oltre alla stringa da stampare, un intero per indicare il tipo di messaggio. A seconda di questo valore e dello stato della variabili interne viene deciso se stampare o meno il messaggio (vedi capitolo 6.4).
- **Shell**: in questa classe sono gestiti i comandi inseriti in console dall'utente. Prevede 2 procedure di funzionamento, la prima, utilizzata durante la modalità testuale, consiste nell'avvio della classe stessa come *Thread*. In questa modo legge i comandi dalla *stdin* per passarli al metodo ***command()*** il quale è dedicato alla decodifica ed esecuzione degli stessi. Nella seconda il metodo ***command()*** viene richiamato all'interno della classe dell'interfaccia grafica. Nel caso il comando non sia valido o non riconosciuto viene stampato a console l'help dei comandi generato dal comando ***help()***.
- **Speed**: questa è la classe che ha lo scopo di visualizzare le velocità totali di download e upload. Le velocità calcolate non coincidono con quelle visualizzate dal programma P2P poiché in queste sono considerati gli overhead dovuti al mascheramento.
- **Tools**: classe contenente tutti quei metodi utilizzati in più punti del codice. È un contenitore dei quei metodi che non avendo una funzione legata a solo una delle altre classi o al loro scopo.
- **FrameProxy**: questa classe è quella in cui è implementata l'interfaccia grafica. A differenza delle precedenti, è molto più articolata ed è inserita in un file (FrameProxy.java) dedicato. Estensione della classe *Applet*, viene inizializzata mediante il metodo ***init()***, il quale crea tutti componenti della form. La classe prevede la possibilità di utilizzare un trayicon in combinazione con la form se il sistema operativo la supporta. È quindi possibile nascondere l'interfaccia grafica mediate gli appositi tasti. Sono previsti anche metodi per permettere alle altre classi di scrivere

all'interno delle apposite aree e il metodo ***closeProgramm()*** per chiudere il programma in maniera corretta terminando prima tutte le componenti grafiche.

6. Uso del programma Peer2Peer Mask

Il programma P2P Mask, come spiegato in precedenza, utilizza un proxy SOCKS con conseguente vantaggio legato all'elevata compatibilità con quasi tutti i programmi P2P. Infatti se lo stesso è in esecuzione, indipendentemente che lo sia sulla macchina stessa o nella rete, è sufficiente che il programma P2P sia correttamente configurato.

Sorge però una limitazione dovuta all'uso della porta 80 impiegata per le connessioni mascherate. Essendo infatti la stessa porta utilizzata anche dai tracker, questi hanno dei problemi nel funzionamento attraverso il proxy modificato. Ciò comporta una limitazione nell'uso di quei soli programmi P2P che permettono di scegliere per quali connessioni utilizzare il proxy. Nello specifico deve essere impiegato esclusivamente per le connessioni tra i PEER.

Il programma utilizzato per i vari test è Vuze. Viene di seguito illustrata la configurazione nel caso il programma P2P Mask sia in esecuzione sulla stessa macchina.

6.1. Utilizzo combinato con Vuze

Vuze è tra i programmi torrent più conosciuti e usati. Evoluzione di Azureus, è un programma scritto in Java il che lo rende facilmente eseguibile su molte piattaforme. Permette una gestione dettagliata e avanzata di molti aspetti del download e l'aggiunta di plugin in maniera semplice. Inoltre ha integrato un sistema di ricerca dei file torrent.

Per impostare correttamente il programma occorre andare su sulle opzioni e cercare le impostazioni riguardanti il proxy (Strumenti => Opzioni (modalità esperto) => Connessione => Proxy). Va impostato l'uso del proxy solo per le connessioni tra i Peer e non per i tracker. La porta da utilizzare dipende dalla configurazioni del programma Peer2Peer Mask, di default la 1080, mentre IP dipende su quale macchina è in esecuzione, nel caso sia la stessa del programma P2P è 127.0.0.1.

6.2. Comandi all'avvio

All'avvio del programma è possibile passare una serie di parametri. Questi permettono di variare alcuni impostazioni; di cui una parte possono essere modificati solo nella fase di avvio.

Per avviare il programma, tramite consolle, va usato il comando

java -jar "Peer2Peer Mask 0.71.jar"

dopo il quale possono essere aggiunti dei comandi. La parte di comando tra virgolette è il nome del file jar, e può variare in funzione di questo.

Qui di seguito in elenco i comandi che si possono passare all'avvio:

-h: in presenza di questo parametro tutti gli altri vengono ignorati, il programma non viene avviato, e viene invece stampato a consolle l'help dei comandi di avvio

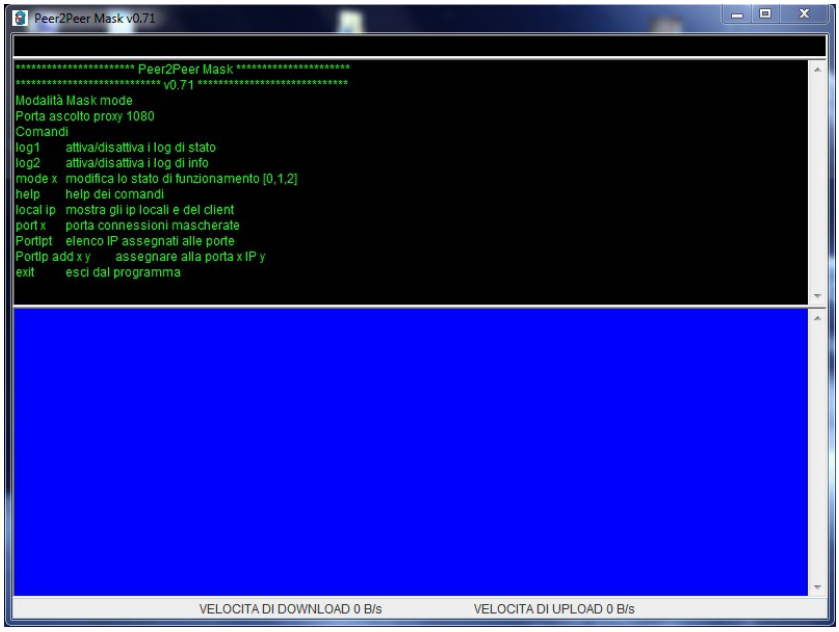
-t: questo parametro permette di scegliere se avviare il programma in modalità testuale, inserendo di seguito il numero 1, o con l'interfaccia grafica facendolo seguire dal numero 0. Nel caso questo parametro non sia presente o inserito in maniera errata, viene avviato di default di interfaccia grafica. Non può essere modificato dopo l'avvio. In modalità testuale vengono stampati solo i messaggi di log e velocità.

-p: con questo parametro si può modificare la porta su cui è in ascolto il proxy per la ricezione di richieste. Il proxy tenterà di mettersi in ascolta su quella porta, se questa non è valida o non può essere utilizzata, viene usata la porta di default 1080. Come per il parametro precedente, la porta scelta non può essere modificata dopo l'avvio.

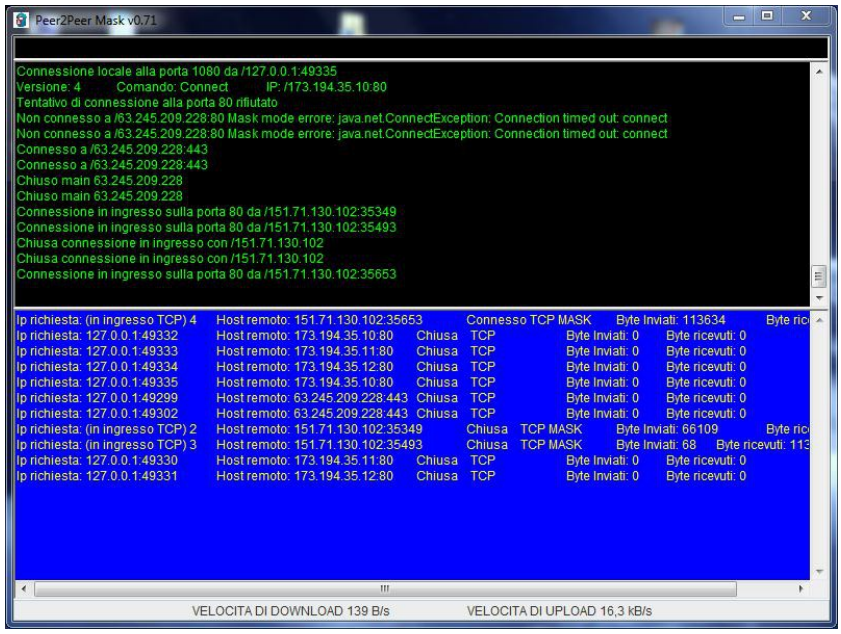
-m: tramite questo parametro è possibile scegliere in quale modalità di funzionamento avviare il programma, tra quelle descritte nel capitolo 3.1 Con il numero 0 si avvia la modalità trasparente, con il numero 1 la mask mode e con il numero 2 la only musk mode; di default, se questo parametro non viene inserito o viene inserito in maniera errata, viene avviata la modalità mask mode.

6.3. Interfaccia del programma

Quando il programma viene avviato con l'interfaccia grafica, si presenta come in figura.



L'interfaccia è divisa in tre parti: una nera, una blu e una grigia. La parte nera corrisponde alla console del programma nella quale vengono inseriti i comandi e stampati i vari messaggi di log. I comandi vengono inseriti nella prima riga in alto. All'avvio, come mostrato nello screenshot sopra, viene stampato il nome del programma e la versione, seguita dalla modalità di funzionamento, dalla porta di ascolto del proxy e l'help dei comandi. Quando il programma è già in esecuzione da un po' di tempo e ci sono già stati alcune connessioni, l'interfaccia si presenta in questo modo.



La parte blu è dedicata all'elenco delle connessioni in corso e alle 10 chiuse più recenti. Il primo valore assume un significato diverso a seconda del tipo di connessione; nel caso sia in uscita indica l'IP di chi ha effettuato la richiesta, se invece è una connessione in ingresso sul server del programma è indicato un numero crescente che aumenta a ogni connessione ricevuta. Nel caso di connessioni TCP, il valore della colonna successiva indica l'IP e la porta dell'host remoto con cui si è connessi, mentre se è UDP indica la porta su cui è in ascolto il proxy. Nella colonna successiva sono indicati lo stato e il tipo della connessione (UDP o TCP) e se mascherata oppure no. Infine nelle ultime due colonne vengono mostrati i byte totali inviati e ricevuti.

Nella riga grigia in fondo vengono visualizzate le velocità attuali totali di download e upload delle connessioni.

Se invece il programma è avviato in modalità testuale si presenta in questo modo.

6.4. Messaggi di log

Tutti i messaggi di log vengono stampati mediante la stessa classe all'interno del programma Peer2Peer Mask. Questi sono divisi in 4 categoria a seconda del loro significato e importanza.

- **IMPORTANT**: questo tipo di messaggio sono considerati importanti e vengono quindi sempre stampati a console. Rientrano in questa categoria quelli legati allo stato del programma come la visualizzazione della modalità di funzionamento, le porte utilizzate e le risposta ai comandi inseriti dall'utente.
- **STATUS**: questi sono messaggi legati allo stato delle connessioni. Sono considerati importanti per l'informazione dell'utente e vengono quindi stampati di default, ma possono essere disattivati mediante l'apposito comando "*log1*". Tra questi messaggi rientra la stampa di notifica quando viene ricevuta una richiesta di connessione visualizzando anche l'IP da cui è stata effettuata, la stampa dei dati presenti nella richiesta dopo averla elaborata, notifiche dei tentativi di connessione ed eventuale successo o fallimento con relativi dati, chiusura di una connessione e notifica delle connessioni in ingresso e inoltre delle stesse con relativi dati.
- **INFO**: questi messaggi sono considerati di minor importanza e sono utilizzati principalmente in fase di test per avere informazioni più dettagliate sullo stato delle connessioni. Di default non vengono stampati ma possono essere attivati mediante il comando "*log2*". Questi messaggi sono quelli relativi allo stato dei *Thread* di ogni connessione, quando vengono avviati e quando terminati, e lo stato e porta di quelli in attesa di connessione.

- **ERROR**: anche questi messaggi, come i primi, sono considerati importanti e sempre stampati, ma sono dedicati a errori nel funzionamento del programma. Ogni qual volta avvengono degli errori non previsti nel normale funzionamento che causa la chiusura di un *Thread* e conseguente terminazione di una connessione viene stampato uno di questi messaggi. La comparsa occasionale di questi messaggi non è causa di problemi nel funzionamento del programma, ma una continua visualizzazione di questi potrebbe essere sintomo di problemi di configurazione.

6.5. Comandi da console

Il programma una volta avviato permette l'uso di alcuni comandi per modificarne il funzionamento. Questi vanno inseriti nella riga nera in alto se è avviata l'interfaccia grafica, o direttamente in console nella modalità testuale.

L'elenco dei comandi, come riportato nel riquadro sottostante, viene stampato in automatico all'avvio del programma e ogni volta che viene usato il comando **help** o nel caso il comando inserito non sia riconosciuto.

```
Comandi
log1            attiva/disattiva i log di stato
log2            attiva/disattiva i log di info
mode x modifica lo stato di funzionamento [0,1,2]
help            help dei comandi
local ip mostra gli ip locali e del client
port x   porta connessioni mascherate
PortIpt elenco IP assegnati alle porte
PortIp add x y        assegnare alla porta x IP y
exit            esci dal programma
```

- **log1**: Con questo comando è possibile attivare o disattivare la stampa dei messaggi di log relativi allo stato delle connessioni. Di default sono attivati perché riportano informazioni essenziali.
- **log2**: Anche questo comando permette di attivare o disattivare la stampa di messaggi di log contenenti informazioni marginali e per questo motivo di default sono disattivati.
- **mode**: Usando questo comando seguito da un numero tra 0 e 2 è possibile modificare la modalità di funzionamento del programma tra quelle descritte nel capitolo 3.1. Per la modalità trasparente si utilizza lo 0, per la mask mode l'1 e per la only mask mode la cifra 2.
- **local ip**: Con questo comando viene stampato a console l'elenco degli IP locali compreso quello pubblico, se rintracciabile, e quello da cui sono state inviate più richieste al proxy. Come spiegato nel capitolo 3.1 le richieste di

connessione a uno degli IP locali vengono rifiutate, da qui l'importanza di conoscerli. Inoltre l'IP dal quale sono arrivate più richieste sarà quello usato dal server per inoltrare le connessioni mascherate in ingresso qualora la porta non sia assegnata come indicato nel capitolo 3.2.

- **port**: Questo comando deve essere usato con molta attenzione perché cambia la porta delle connessioni mascherate impostandolo al valore che viene indicato di seguito. Questa porta, come spiegato in precedenza, deve essere fissa e nota, altrimenti non è possibile per gli altri conoscerla. Il comando è stato inserito per essere usato in fase di debug e se usato durante il normale utilizzo compromette il funzionamento del programma.
- **portIp**: Utilizzato da solo stampa a console l'elenco delle coppie porta/IP usate dal server per inoltrare le connessioni in ingresso; se invece è seguito da **add** permette di creare una coppia tra la porta e IP con i due valori scritti di seguito.
- **exit**: Si chiude il programma.

7. Conclusioni

Grazie all'utilizzo del programma Peer2Peer Mask è quindi possibile mascherare una trasmissione P2P. Attente analisi del traffico mediante sniffer hanno verificato che il flusso dati tra due host, utilizzanti Peer2Peer Mask, viene visto come traffico HTTP.

Grazie a questo risultato, il problema della limitazione del traffico viene risolto attraverso una soluzione facile da utilizzare e abbastanza flessibile. Infatti si ricorda che il programma Peer2Peer Mask è compatibile con un'elevata quantità di applicazioni P2P e permette di essere utilizzato non solo in combinazione con una singola macchina, ma anche con un numero superiore purché tutte all'interno di una stessa rete. L'unica limitazione è legata al dover avere il programma Peer2Peer Mask in esecuzione su entrambi gli host che vogliono avviare una connessione mascherata.

Si vuole infine fare notare un'ultima caratteristica molto importante e finora non ancora evidenziata: anche se la cattura e il mascheramento delle trasmissioni è stato pensato per programmi P2P, l'implementazione è stata fatta considerando il passaggio di dati generici. Ne consegue che il programma Peer2Peer Mask è teoricamente compatibile con qualsiasi applicazione e protocollo. Potrebbe quindi essere utilizzato come strumento per mascherare qualsiasi tipo di trasmissione, non solo quelle P2P.

Si può quindi concludere che il programma Peer2Peer Mask assolve pienamente il compito per cui è stato pensato con vantaggi aggiuntivi legati a scelte implementative.