Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea in Scienze e Tecnologie Informatiche

VALUTAZIONE SPERIMENTALE DI TECNOLOGIE PER LA CREAZIONE DI VPN IN AMBITO LAN/WAN

Tesi di Laurea in Reti di Calcolatori

Relatore:
GABRIELE D'ANGELO

Presentata da:
MATTEO CORFIATI

♯ Sessione II
Anno Accademico 2011-2012

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

L'informatica, che ormai svolge un ruolo predominante nel mondo odierno, rivolge sempre più la sua attenzione alla comunicazione. Se il computer nasce per risolvere calcoli complessi, negli ultimi anni "perde" questo suo ruolo per diventare principalmente un *individuo* connesso a una rete. Ogni ambiente di lavoro ne possiede una e il PC risulta essere il mezzo per ottenere informazioni condivise all'interno di questa rete privata, una rete che ha costi bassi se creata all'interno di uno stabile o, comunque, di un intorno geografico limitato. I problemi nascono quando, ad esempio, una azienda è composta da diverse filiali dislocate nel territorio che devono poter comunicare in un canale sicuro. Diverse sono le possibilità attuabili, anche se solitamente una viene prediletta rispetto alle altre, ovvero la creazione di reti private virtuali. Le VPN (*Virtual Private Network*) sono strumenti che permettono la creazione di tunnel criptati tra due *endpoint*[1], utilizzando come mezzo di trasmissione una rete condivisa non sicura come Internet. Questo processo è nascosto agli occhi dell'utilizzatore, poiché i sistemi coinvolti, che possono essere da singoli calcolatori a gruppi di *intranet*, vengono mostrati come facenti parte della stessa rete privata. La trasparenza, caratteristica importante nell'informatica, soprattutto nel mondo del distribuito, è la proprietà che nasconde all'utente ciò che per lui non è rilevante; una volta configurata, una VPN mo-

[1]Sistemi terminali di una rete.

stra le due sottoreti collegate fra loro, nascondendo completamente la parte di *tunneling*, compito delegato all'interfaccia virtuale.

Viene utilizzato il termine virtuale perché, da un lato, non vengono utilizzate NIC (*Network Interface Card*) fisiche, dall'altro, non è la classica rete privata, che sia LAN (*Local Area Network*) o WAN (*Wide Area Network*). Seppur virtuali, le VPN rimangono reti private a tutti gli effetti, e questo le "obbliga" a utilizzare gli spazi di indirizzamento riservati dallo IANA (*Internet Assigned Numbers Authority*) e specificati nell'opportuna RFC (*Address Allocation for Private Internets*, RFC 1918).

Vengono sempre più utilizzate da enti, aziende e privati per connettere sistemi geograficamente distanti tra loro a prezzi notevolmente ridotti rispetto a collegamenti dedicati.

Essendo creata su un mezzo pubblico e non sicuro come Internet, ogni VPN gestisce un proprio livello di sicurezza, che solitamente comprende un meccanismo di autenticazione e di cifratura dei dati trasmessi.

Un'altra caratteristica fondamentale di una VPN è la flessibilità, poiché offre all'utente la possibilità di collegarsi a reti e risorse remote da ogni posto utilizzando semplicemente una connessione a Internet [1].

Esistono molte soluzioni VPN, sia di natura *open source* che proprietarie, le quali utilizzano proprie implementazioni di protocolli più o meno standard, che incidono sulla sicurezza e sulle *performance* del sistema. Il *tunneling* prevede l'incapsulamento dei dati all'interno di pacchetti logicamente separati dai dati stessi; solitamente si tende a preferire determinati protocolli per il trasporto delle informazioni (come UDP, il quale introduce un basso *overhead*) rispetto ad altri. Il fine della seguente analisi è quello di, scelte

alcune soluzioni VPN note[2], valutarne l'efficienza tramite alcuni test mirati. Lasciando a margine il fattore inerente alla sicurezza, si confrontano i comportamenti dei protocolli di livello superiore utilizzati e le caratteristiche fondamentali che determinano una prestazione migliore in uno scambio di dati in rete: i parametri controllati sono la *bandwidth* tra gli *endpoint*, gli *overhead*[3] dei pacchetti che formano il tunnel criptato, l'RTT (*Round Trip Time*) e altre caratteristiche analoghe. Non esiste un protocollo specifico per le VPN poiché lo scenario gioca un ruolo fondamentale dettato dalla propria struttura. Il *wireless*, che utilizza come implementazione lo standard IEEE 802.11, introduce una percentuale notevolmente maggiore di *packet loss* rispetto a una situazione *wired*. Internet tra le due terminazioni incrementa la possibilità di avere pacchetti duplicati o corrotti a causa di ritardi e collisioni. I collegamenti cablati hanno una latenza minore rispetto a reti mobili come UMTS e 3G. Per questi motivi i test verranno riproposti nei due ambienti analizzati, mostrando come una soluzione ottenga prestazioni migliori in un contesto piuttosto che in un altro. Verranno inoltre indicati i problemi riscontrati durante le rilevazioni, tentando di fornire spiegazioni plausibili in base al contesto.

Quattro tipologie di VPN sono state create per l'analisi (tutte utilizzano un'architettura *client-server*), una che utilizza OpenVPN su UDP e una su TCP, una che utilizza il protocollo di *tunneling* PPTP e una proprietaria, NeoRouter; queste vengono confrontate in due scenari differenti, ovvero una LAN *wireless* e una LAN cablata.

[2]La scelta delle soluzioni analizzate è dettata dalle seguenti motivazioni: diversità di protocolli utilizzati per la creazione del tunnel, gratuità del prodotto (preferibile un prodotto *open source*), compatibilità con i sistemi operativi maggiormente diffusi.

[3]Somma degli *header* che non costituiscono la parte dei dati di un pacchetto, ma le informazioni aggiuntive dettate dal protocollo utilizzato.

Capitolo 1

Soluzioni di *Virtual Private Networking*

Come per qualsiasi altro prodotto commerciale, esistono VPN con diverse caratteristiche; tra loro possono differenziarsi per prezzo, protocolli utilizzati, livello di sicurezza e, più in generale, per la qualità del servizio offerto.
Il primo capitolo mostra le caratteristiche strutturali e funzionali delle soluzioni VPN utilizzate durante l'analisi[1]. Ognuna viene descritta riferendosi alle documentazioni ufficiali, ponendo l'attenzione sui protocolli, sulle configurazioni richieste, sulle modalità di autenticazione e di scambio di messaggi. Nonostante alcune di esse supportino anche il P2P (*Peer-to-Peer*), la struttura degli ambienti monitorati è sempre di tipo *client-server*, per il semplice motivo di rendere il più possibile confrontabili i diversi scenari.

[1]Le soluzioni sono tutte *open source* o *freeware*; le scelte sono state dettate dalla volontà di avere diversi protocolli e VPN strutturate in modo differente.

1.1 OpenVPN

OpenVPN è una delle soluzioni VPN più conosciute, almeno nel campo dell'*open source*. Basata sul protocollo SSL/TLS[2], garantisce i più alti livelli di sicurezza [4]. Le caratteristiche fondamentali di OpenVPN, che gli hanno garantito il successo nel mercato delle soluzioni VPN, sono l'estrema semplicità di installazione e le tante possibilità di configurazione (come mostrato nell'appendice A). In base a come impostata la VPN, il tunnel[3] viene creato su UDP (scelto nella configurazione di default) o TCP. I *payload* dei segmenti che creano il tunnel sono criptati, e la connessione viene instaurata previa autenticazione tramite certificati. OpenVPN permette la scelta di diverse possibili soluzioni: vengono supportate configurazioni *Peer-to-Peer* oppure ogni server può gestire diversi client contemporaneamente e possono essere create strutture *host-to-host*, *site-to-site* o *host-network* [1].

L'autenticazione è gestita tramite un complessa infrastruttura di sicurezza, formata da certificati e chiavi private propri del server e di ogni client e da un certificato CA (*Certificate Authority*) utilizzato per $firmare$ i certificati privati; entrambe le parti, prima di analizzare le informazioni dei certificati, verificano che questi siano stati sottoscritti con il certificato CA. Tutte le chiavi e i certificati possono essere creati con appositi *script* contenuti nel pacchetto del prodotto.

I file da utilizzare possono essere logicamente organizzati in due directory (posizionate nei due sistemi terminali), *server* e *client*[4]:

[2]OpenVPN utilizza la libreria di crittografia OpenSSL.

[3]Diverse configurazioni permettono la creazione di *device* TUN o TAP: entrambe sono interfacce virtuali di rete, ma TUN lavora con pacchetti IP mentre TAP con *frame* Ethernet.

[4]I nomi dei file elencati sono quelli utilizzati nella documentazione ufficiale di OpenVPN.

- Server

  server.crt Certificato del server

  server.key Chiave privata del server

  ca.crt Certificato CA (*Certificate Authority*) condiviso

  key.txt Chiave pubblica statica (o TLS)

  dh1024.pem Parametri per il protocollo Diffie-Hellman[5]

  server.conf File di configurazione del server

- Client

  client1.crt Certificato del client

  client1.key Chiave privata del client

  ca.crt Certificato CA (*Certificate Authority*) condiviso

  key.txt Chiave pubblica statica (o TLS)

  client.conf File di configurazione del client

Quelli appena descritti sono tutti file che contengono informazioni sull'autenticazione degli host tranne i due di configurazione server.conf e client.conf. Nella prima appendice viene riportato un esempio[6] di configurazione di un server creato con OpenVPN.

[5]Il protocollo Diffie-Hellman permette a due utenti di scambiarsi chiavi segrete tramite un mezzo non sicuro.

[6]L'esempio mostrato riporta le impostazioni utilizzate nella creazione di una VPN per i test che seguiranno. Dal file originale sono stati rimossi alcuni commenti superflui e parametri non interessanti nella configurazione utilizzata, riguardanti l'*ethernet bridging*, il massimo numero di client e un insieme di altri fattori non utili all'analisi.

1.2 PPTP

PPTP (*Point-to-Point Tunneling Protocol*) è un protocollo di rete sviluppato da Microsoft che permette la creazione di VPN su una rete pubblica. É un protocollo orientato alla connessione ed è formato da due componenti principali: una connessione di controllo[7] creata su TCP e un tunnel formato incapsulando in datagrammi IP pacchetti PPP (*Point-to-Point Protocol*), già incapsulati in GRE (*Generic Routing Encapsulation*).
La *Control Connection* può essere stabilita da entrambe le parti e viene mantenuta attiva tramite messaggi *echo*. È semplicemente una sessione TCP nella quale vengono inviate informazioni di controllo e gestione del collegamento PPTP.
Logicamente associate alle connessioni di controllo vi sono i tunnel, formati incapsulando successivamente diversi protocolli. I datagrammi del livello di rete contengono come dati pacchetti di una versione estesa di GRE[8], che provvede al controllo di flusso e di congestione tramite l'utilizzo di numeri di riconoscimento. Oltre al numero di sequenza, che identifica univocamente un pacchetto all'interno di una sessione, diversi protocolli di rete prevedono anche degli *acknowledgment number*, che permettono a un *host* di avere informazioni sulla avvenuta ricezione di pacchetti da lui precedentemente inviati. La differenza con gli ACK di TCP è che non vengono utilizzati per rinvii dei dati ma per determinare il tasso di trasmissione degli stessi: in linea teorica questo meccanismo dovrebbe provvedere ad utilizzare in maniera efficiente l'ampiezza di banda disponibile.

[7] Nella RFC 2637 chiamata letteralmente *Control Connection*

[8] *Generic Routing Encapsulation* è un protocollo descritto nella RFC 1701 che, indipendentemente dal *payload* e dai protocolli di livello inferiore, fornisce una modalità di trasmissione il più possibile generalizzata in modo da avere un ampio spettro di utilizzo

L'*header* GRE viene aggiunto al rispettivo *payload*, un pacchetto PPP[9]. I pacchetti GRE non gestiscono in alcun modo la sicurezza, non sono protetti in modo crittografico né prevedono alcun *handshake* in fase di autenticazione; l'intera gestione di meccanismi di protezione dei dati e sicurezza della connessione è lasciato a PPP: se i *payload* di questo protocollo non sono protetti, il tunnel risulta sprovvisto di una parte fondamentale solitamente offerta dalle VPN.

I client PPTP sono ormai previsti di *default* nella maggior parte dei sistemi operativi utilizzati, come Linux, Windows, OSx e addirittura in campo *mobile* con Android e iOS. La soluzione VPN lato server che utilizza questo protocollo sul sistema operativo Linux prende il nome di PPTPD (*Point-to-Point Tunneling Protocol Daemon*).

L'installazione del demone PPTP è molto semplice, in particolare nei sistemi Linux. Dopo l'aggiunta degli utenti, vi è la possibilità di modificare due file di configurazione che gestiscono le caratteristiche del processo (pptpd.conf e pptpd-options). Nel caso in cui gli indirizzi assegnati dalla VPN (di default 192.168.1.1-100) entrino in contrasto con gli IP assegnati dal DHCP automatico dei router, in pptpd.conf è possibile modificare l'IP locale del server e quelli dei client remoti.

[9] *Point-to-Point Protocol* è un protocollo descritto nella RFC 1661 che fornisce un modello per il trasferimento di datagrammi su un collegamente punto-punto.

1.3 NeoRouter

NeoRouter è una soluzione VPN multi-piattaforma molto semplice ed intuitiva; è un prodotto proprietario definito come privo di configurazione[10]. È la soluzione *freeware* tra un insieme di altre più complesse e a pagamento, che offrono migliorie in termini di sicurezza e prestazioni. Essendo un prodotto commerciale la sua struttura e il suo funzionamento non vengono mostrati interamente; la caratteristica principale è che vengono utilizzati dei *NeoRouter domains* per collegarsi alle proprie reti: il server in un primo momento crea un dominio e successivamente lo associa al proprio IP; i client possono connettersi alla VPN tramite un *Network Manager* (o semplicemente da terminale) inserendo nome di dominio e credenziali. Una volta stabilita la connessione ogni calcolatore all'interno della LAN virtuale può vedere, sempre tramite il *Network Manager*, la lista dei dispositivi connessi. L'utilizzo di questo metodo di autenticazione e gestione della VPN, che rende completamente trasparente la connessione alla rete, aiuta notevolmente un utente inesperto, aumentando tuttavia la latenza soprattutto in fase di instaurazione del collegamento, quando il client deve effettuare una richiesta a una sorta di server DNS proprietario. La struttura di una VPN NeoRouter non è limitata a una semplice architettura client-server, ma ne viene gestita una ibrida con il P2P: i client si connettono al server (responsabile dell'autenticazione degli utenti), stabilendo successivamente connessioni P2P su richiesta (creabili su TCP o UDP) con gli altri *peer* (Figura 1.1). In tal modo, togliendo al server il carico di lavoro più oneroso, si incrementa la velocità di trasferimento dei

[10]Nella sezione *How it works* della documentazione ufficiale di NeoRouter si parla di *cross-platform zero-configuration VPN solution*. In realtà possono essere modificate le varie caratteristiche sia dai file di configurazione che dalle GUI di amministrazione fornite col prodotto, come spiegato nello *User's manual*.

dati.

Adotta i più alti livelli di sicurezza informatica, utilizzando il protocollo SSLv3[11] per il canali tra client e server e un insieme di protocolli come RSA 2048bit, DH e AES-256 per la protezione delle connessioni P2P tra i client.

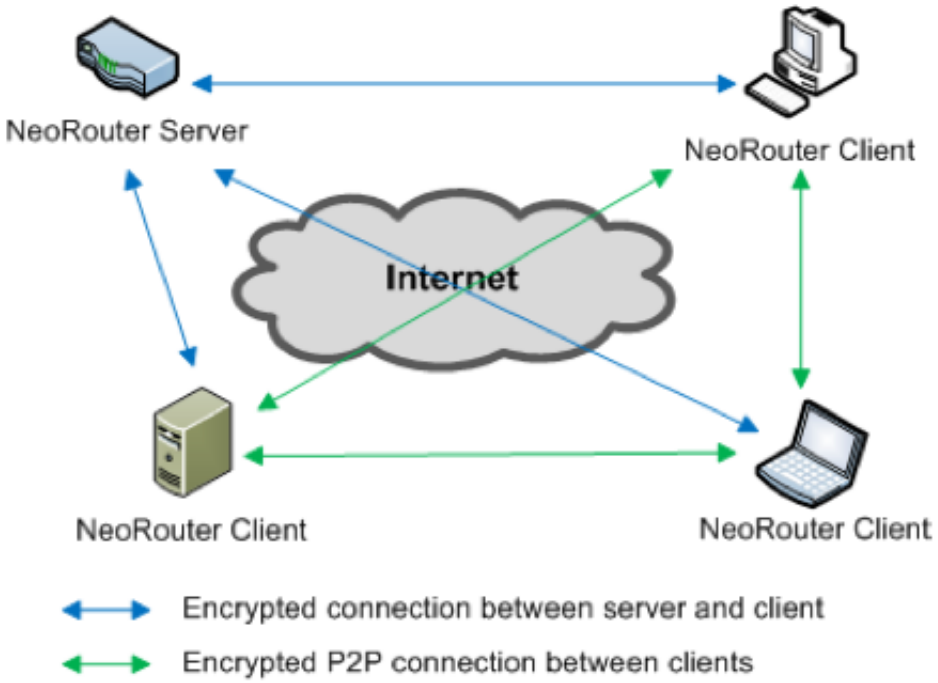

Figura 1.1: Architettura *hybrid P2P* di NeoRouter. Figura tratta da *NeoRouter: How it works*, pag. 4

[11]Utilizzando il protocollo SSL, NeoRouter può essere considerato una soluzione di SSL VPN, pur non utilizzando il browser come *client program*.

Capitolo 2

Strumenti per l'analisi delle prestazioni

Sono diversi i fattori che possono influenzare le prestazioni di una VPN e diversi sono i parametri che ne danno una misura delle *performance*. La scelta del protocollo di tunneling è sicuramente una delle caratteristiche più incisive (banalmente TCP darà risultati diversi rispetto a UDP), ma è seguita da una lunga lista di altri fattori: l'utilizzo di algoritmi di compressione, l'aumento del *delay* causato dall'uso di diversi meccanismi di sicurezza, differenti tipologie di rete per il collegamento al server (es. *Wired*, *Wireless*, LAN, WAN, ...). Per valutare l'efficienza di una VPN si utilizzano i metri di misura soliti del mondo delle reti:

Il throughput medio misura la quantità di dati trasmessi nell'unità di tempo (es. un secondo); è un parametro molto importante nell'analisi di una rete poiché indica la velocità di trasmissione dei dati (senza contare l'*header* aggiunto dal tunnel).

L'overhead, che nel campo del *networking* indica l'insieme degli *header*

aggiunti al *payload* dei dati, incide sulla quantità di dati inviati rispetto al reale *peso* degli stessi.

L'RTT, per esteso *Round Trip Time*, fornisce informazioni riguardo al tempo di latenza e al *jitter*, ovvero la variazione della latenza stessa. È il tempo che impiega un pacchetto (di piccole dimensioni) per viaggiare da un host a un altro della stessa rete e ritornare indietro.
Altre considerazioni vanno fatte anche sulla possibile presenza di errori, quindi pacchetti duplicati o corrotti.

Per l'analisi occorrono quindi degli strumenti specifici, in grado di esaminare nel dettaglio le caratteristiche delle VPN sopra elencate.

2.1 Wireshark

Wireshark, che ha le sue radici nel predecessore Ethereal, è un analizzatore di pacchetti di rete, ovvero un programma che cattura i dati e le informazioni che attraversano una *network interface*.
Questi *packet sniffer* vengono spesso utilizzati per controllare la correttezza delle configurazioni, per valutare la complessità e la 'resistenza' dei meccanismi di sicurezza o comunque, più in generale, per effettuare *debugging* su una rete informatica.
Wireshark gestisce la cattura di oltre 1300 protocolli di rete, e la sua natura open-source permette agli utenti di crearsi *plugin* per aggiungerne altri, in modo da allargare la quantità di informazione decodificabile dal programma stesso; questo lo rende uno dei tool di *network analysis* più utilizzati.
Oltre alla fase di cattura vera e propria dei pacchetti, Wireshark offre un complesso sistema di statistiche create sulla base dell'analisi dei dati ottenuti. All'interno di queste statistiche ritroviamo diversi livelli di dettaglio che vanno dal mostrare un riepilogo sommario della cattura effettuata a una struttura ad albero formata dai protocolli utilizzati, dalla lista dei sistemi terminali che han preso parte allo scambio alla creazione di grafici sui vari aspetti della comunicazione. Più nello specifico troviamo il dettaglio degli scambi di informazione, ampiamente utilizzato nei test successivi: per ogni *conversation*[1] vengono mostrati i sistemi terminali che ne han preso parte, le porte scelte, il numero di pacchetti e i bytes scambiati, sia totali che divisi per direzione, la durata in secondi e il throughput medio, anch'esso diviso tra bps da A a B e bps da B ad A.

[1] "A network conversation is the traffic between two specific Endpoints"[13]. L'esempio nella documentazione di Wireshark mostra che una conversazione IP contiene tutti i pacchetti IP inviati e ricevuti tra due indirizzi IP.

2.2 Ping

Ping (Packet InterNet Groper) è un programma diagnostico nato all'inizio degli anni '80. Fa uso di pacchetti ICMP (Internet Control Message Protocol) per misurare l'RTT tra due host. Nello specifico il client ping invia un messaggio di ECHO_REQUEST e ottiene un messaggio di ECHO_REPLY; il tempo viene calcolato, in millisecondi, effettuando una differenza tra l'istante in cui il client ottiene la risposta e quello in cui aveva inviato la richiesta. I server ping sono ormai inseriti all'interno dei maggiori sistemi operativi, che vengono solitamente predisposti per rispondere con un messaggio ECHO_REPLY a seguito di una richiesta dello stesso tipo.
L'output fornito da ping non è uguale in tutti i sistemi operativi anche se, settando opportuni parametri opzionali, si possono ottenere le stesse informazioni.
L'uso che se ne è fatto nei test seguenti riguarda il calcolo del *roundtrip time*: lo standard output di Ping su Linux riporta il numero di pacchetti trasmessi, quanti ricevuti correttamente e quanti danneggiati, la percentuale di pacchetti persi e l'RTT minimo, massimo, medio e la deviazione standard, tutti tempi misurati in millisecondi. Di seguito è mostrato l'esempio[2] di invio di due pacchetti all'host google.it:

```
$ ping -c 2 google.it
PING google.it (173.194.35.191) 56(84) bytes of data.
64 bytes from muc03s02-in-f31.1e100.net (173.194.35.191): icmp_req=1 ttl=53
time=48.4ms
64 bytes from muc03s02-in-f31.1e100.net (173.194.35.191): icmp_req=2 ttl=53
time=39.1ms


--- google.it ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 39.165/43.817/48.469/4.652 ms
```

[2]L'output mostrato è quello che si ottiene richiamando il comando in un sistema Linux con distribuzione Ubuntu 11.10.

2.3 Netcat

Netcat è uno strumento *opensource* utilizzato ampiamente nel campo dell'amministrazione di rete. La capacità principale del *tool* è quella di leggere e scrivere dati attraverso connessioni TCP o UDP: questo permette di utilizzarlo in diversi modi, ad esempio come client di posta elettronica, per far richieste a server HTTP o per eseguire il *port-scanning* su un *host*.
Un altro insieme di utilizzi prevede una comunicazione tra due computer che usano entrambi Netcat, uno come server e uno come client: in tal modo è possibile creare una chat, scambiarsi file o testare la sicurezza del proprio sistema.
Netcat nasce sui sitemi UNIX nel 1995 e viene poi implementato anche per Windows NT nel 98 da Weld Pond: in entrambi i casi il software viene rilasciato insieme ai codici sorgente con licenza GNU (*General Public License*).
Ciò che ha reso così importante Netcat nel corso degli anni è, da una parte, la sua semplicità, e dall'altra, la facilità con cui si può interfacciare ad altri software e script.
L'esempio base prevede il server (o *listener*) in ascolto su una determinata porta

```
nc -l -p <port>
```

e un client che si connette alla porta specificata

```
nc <IP address> <port>
```

2.4 Iperf/Jperf

Iperf è uno strumento utilizzato per la misurazione delle *performance* di una rete, in particolare per analisi specifiche sulla *bandwidth*. È un software *open source*, scritto in C++ e sviluppato dal NLANR/DAST (*National Laboratory for Applied Network Research/Distributed Applications Support Team*).

Iperf ha una architettura client/server e mette a disposizione dell'utente una ampia scelta di opzioni, in modo da permettere una personalizzazione elevata dei test. Una prima scelta riguarda la creazione del flusso dati, che può essere formato da segmenti TCP o UDP. Nel primo caso è possibile definire anche l'MSS (*Maximum Segment Size*), la lunghezza dei buffer e della finestra[3], mentre nel secondo caso occorre impostare una *bandwidth* massima ed è possibile modificare la capacità dei pacchetti UDP.

Altre opzioni configurabili gestiscono la fase di trasmissione, come la possibilità di inviare per un certo lasso di tempo o di spedire un preciso quantitativo di dati deciso dall'utente, e il formato dell'output. Sul server si può impostare il numero massimo di connessioni contemporanee accettate, sul client è possibile generare più flussi paralleli. A partire dalla versione 1.2, il contenuto di un file può essere utilizzato come campo dati dei segmenti inviati.

Nel corso degli anni diversi progetti sono stati creati a partire dalla base offerta da Iperf, uno dei quali è Jperf: questo estende Iperf, nato con la sola interfaccia a linea di comando, con una *graphical user interface* fatta in Java. La principale utilità di questa estensione sta nella visualizzazione dell'output, che riporta oltre alla rappresentazione canonica dei risultati, un grafico

[3]L'MSS indica la massima quantità di dati che può essere inserita in un segmento. La finestra di ricezione è una variabile collegata ai buffer TCP, che serve per implementare un servizio di controllo di flusso, ovvero un modo per evitare l'*overflow* dei buffer stessi.

automaticamente generato a partire dalle rilevazioni.

L'esempio[4] seguente mostra un possibile output di *iperf* ottenuto lato server:

```
$ iperf -s
------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------
[  4] local 10.242.184.114 port 5001 connected with 10.242.184.113 port 50567
[ ID] Interval       Transfer     Bandwidth
[  4] 0.0-10.4 sec   4.25 MBytes  3.43 Mbits/sec
```

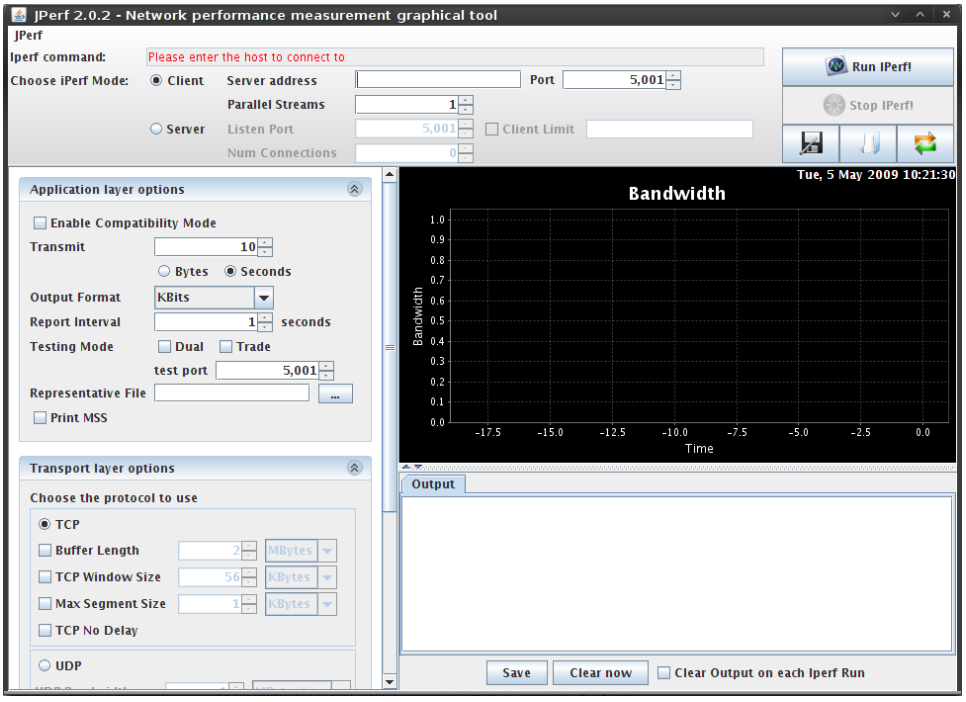

Figura 2.1: Jperf 2.0.2 UI. Figura tratta da [18]

[4]Lato client il comando è 'iperf -c 10.242.184.114 -P 1 -i 1 -p 5001 -f k -t 10': eseguito in client-mode (-c), si connette al server 10.242.184.114 sulla porta 5001 (-p 5001) con un unico thread (-P 1), avendo un secondo di pausa tra due report successivi (-i 1), mostrati in Kbit (-f k), e invia pacchetti per un totale di 10 secondi (-t 10).

2.5 ifconfig

Ifconfig (*Interface Configurator*) è un comando Unix-like per la configurazione e l'amministrazione di interfacce di rete, molto simile allo strumento $ipconfig$ utilizzato nei sistemi Microsoft.
All'avvio dei sistemi, molti di questi utilizzano $ifconfig$ per inizializzare le interfacce per mezzo di script. Analogamente a questo uso, l'amministratore può configurare una *network interface* utilizzando la CLI (*Command Line Interface*) del comando e le opzioni rese disponibili dallo stesso.
Il classico utilizzo di $ifconfig$ si limita a mostrare i parametri importanti riferiti a una o più interfacce di rete presenti nel sistema.
Le due sintassi possibili sono:

- ifconfig interfaccia [aftype] opzioni | indirizzo
- ifconfig [interfaccia]

Il primo caso mostra la sintassi da utilizzare nel caso si vogliano configurare i parametri di una interfaccia di rete. $Aftype$ indica la *address family* utilizzata per la decodifica e la visualizzazione degli indirizzi (ne sono supportate diverse, tra cui *inet*, che corrisponde agli indirizzi delle reti TCP/IP, $ax25$, *ddp*, *ipx* e *netrom*); tra le opzioni vi sono l'impostazione dell'MTU (*Maximum Transfer Unit*) o dell'hardware su cui si appoggia l'interfaccia.
Il secondo caso è meno complesso ma più importante per i successivi test. Se il comando non viene seguito da altro, l'output mostra le configurazioni attuali di tutte le interfacce di rete presenti nell'host; se viene specificato il nome di una interfaccia, le impostazioni mostrate sono riferite alla sola interfaccia indicata.
L'esempio[5] che segue propone un ipotetico output ottenuto richiamando il

[5] L'output mostrato è quello che si ottiene richiamando il comando in un sistema Linux con distribuzione Ubuntu 11.10. Gli altri sistemi Unix-like hanno output leggermente differenti.

comando 'ifconfig eth0':

```
$ ifconfig eth0
eth0    Link encap:Ethernet  HWaddr 00:1d:ba:f4:1d:e5
        inet  addr:10.242.184.114  Bcast:10.242.191.255  Mask:255.255.248.0
        inet6  addr: fe80::222:fbff:fe82:53a8/64  Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:471 errors:0 dropped:0 overruns:0 frame:0
        TX packets:695 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:160637 (156.8 KB)   TX bytes:86193 (84.1 KB)
        Interrupt:185
```

Le prime righe mostrano informazioni riguardo alla *link encapsulation*, all'indirizzo MAC[6] (*Media Access Control*) dell'hardware associato all'interfaccia e al riepilogo degli indirizzi associati (in questo caso indirizzi IP essendo in una rete TCP/IP). Seguono indicazioni sullo stato corrente, sull'MTU e sulla metrica (valore che 'esprime' il livello di priorità di una interfaccia rispetto alle altre). Le due righe seguenti mostrano il numero di pacchetti rispettivamente ricevuti e trasmessi dalla *network interface*; nella penultima riga i dati ricevuti e trasmessi sono espressi sotto forma di byte.
Collisions, se maggiore di 0, indica che si sono verificate collisioni, quindi sintomo di una rete congestionata. *Txqueuelen* denota la lunghezza della coda di trasmissione.
Le interfacce gestibili con $ifconfig$ non sono limitate a quelle collegate a device fisici, ma con questo strumento sono indicate anche le eventuali NIC (*Network Interface Card*) virtuali.

[6]Il *MAC address* identifica univocamente i *device* fisici utilizzati per la connessione a una rete (schede di rete, ecc.).

Capitolo 3

Valutazione delle prestazioni

La fase empirica dell'analisi si suddivide in due blocchi differenti, che corrispondono ai due scenari previsti. Per ogni scenario vengono effettuati i vari test su ognuna delle quattro VPN create (OpenVPN UDP, OpenVPN TCP, PPTP, NeoRouter) e, per dare un metro di paragone, anche nella modalità senza VPN.

Le prove effettuate sono tre: l'analisi del *throughput* durante l'invio di un file di grandi dimensioni[1] utilizzando Netcat e durante l'invio di pacchetti creati *ad hoc* tramite Iperf, la valutazione dell'RTT e dei pacchetti duplicati o corrotti a seguito dell'invio di 1000 ping ICMP (*Internet Control Message Protocol*).

In questo capitolo verranno riportati, dopo aver mostrato in dettaglio come sono stati creati i vari scenari, i risultati ottenuti nelle diverse prove esaminate, analizzando i valori di test tramite opportune statistiche.

Per una corretta lettura dei dati occorre sapere che questi vengono mostrati

[1]File creato tramite il comando dei sistemi operativi Unix e Unix-like *dd*. Il comando seguente, simile a quello utilizzato, crea un file da 1MB, diviso in 1000 blocchi di 1KB ciascuno: *dd if=input-file of=file-to-create bs=1k count=1000*

secondo uno schema ben definito: una prima divisione riguarda gli scenari, di ognuno dei quali vengono riportate caratteristiche e problemi riscontrabili; la seconda divisione riguarda le varie VPN utilizzate (incluso il caso senza VPN); per ultimi si riportano i risultati, schematizzati in tabelle, di ogni test. Alla fine di ogni scenario viene mostrato un confronto delle diverse tecnologie, utile per valutare le principali differenze riscontrate.
Le unità di misura dei tempi sono sempre riportate in ms, mentre il *throughput* è mostrato in KB/sec o in Kb/sec, in base al test considerato.
Di seguito vengono mostrate nello specifico le caratteristiche proprie di ogni test.

Ping test

Il primo test riguarda l'invio di 1000 pacchetti ICMP da 64 bytes tramite il programma Ping. Come già detto più volte, questo test ha il compito di analizzare il tempo di latenza. Nonostante il ping test sia molto importante per il confronto tra le diverse tecnologie, progettarlo risulta essere molto semplice. Il comando utilizzato è `ping -c 1000 x.y.z.t`. L'unica opzione settata è -c 1000 per forzare l'host a inviare esattamente 1000 pacchetti ICMP, mentre il resto rimane immutato, lasciando le opzioni di default di ping.
Tra le numerose opzioni offerte dal programma, l'unica interessante per un eventuale test sarebbe stata -f, che genera un flusso continuo di pacchetti; tuttavia in un test sulla latenza, un *ping flood* sarebbe stato controproducente nel caso in cui si fosse raggiunto un'*overflow* dei buffer del destinatario. Nel modo utilizzato l'intervallo di tempo tra due *request* successive è di un secondo.

Netcat test

Il test più complesso è quello che ha come azione principale l'invio di un file di grandi dimensioni tramite Netcat.
Questa prova è, a sua volta, divisa in tre test minori, che si differenziano per il tipo di file "utilizzato". Il primo esame è completamente irrealistico, poiché propone come file di download uno di 500 MB riempito con tutti zeri. Come già detto nei capitoli precedenti, una caratteristica importante delle VPN è l'uso di meccanismi di compressione che riducono il carico inviato all'interno del *tunnel*. Di conseguenza, le soluzioni che, per configurazione, comprimono i dati prima di inviarli al destinatario, avranno risultati vistosamente migliori con questo primo file, anche rispetto al caso senza VPN. Il secondo file ha le stesse dimensioni del primo, ma è stato riempito con un input pseudo-casuale; in questo modo la compressione di algoritmi come LZO non incide in maniera così irrealistica sul test. L'ultimo caso propone il download di un file di 311 MB creato dal *grab*[2] di centinaia di pagine web, includendo prevalentemente html e immagini. Questa dovrebbe essere la prova più significativa delle tre, sempre in considerazione del problema della compressione[3].
I primi due file sono stati creati con il comando Unix *dd*[24], modificando solo la sorgente di input:

- Primo file: dd if=/dev/zero of=file1 bs=1M count=500
- Secondo file: dd if=/dev/urandom of=file2 bs=1M count=500

Il terzo file è stato invece creato con *tar* [26], software di generazione file per l'archiviazione e il backup.

[2]Il *mirroring* delle pagine web è stato fatto tramite wget, programma di gestione di download incluso in Unix e Linux[23].

[3]Il file è diviso in due parti che occupano circa lo stesso spazio: la parte di testo è comprimibile, mentre le immagini non completamente.

Tramite Netcat si crea un *listener*, che riceverà il file ascoltando su una determinata porta (`nc -l -p 50000 >nomeFile`), e un *sender*, che invierà il flusso di dati senza l'utilizzo di alcun protocollo di livello applicazione (`nc 10.8.0.1 50000 <nomeFile`).

Quando vengono riportati i risultati di questo test, si troveranno tre risultati di *throughput*, ognuno relativo al file utilizzato (A corrisponde al file formato da tutti zeri, B a quello avente come input numeri pseudocasuali e C corrisponde all'ultimo, generato unendo file di diverso tipo e contenuto).

Per ottenere il *throughput* vi sono molti modi diversi. Wireshark mette a disposizione un insieme di statistiche che comprendono anche l'analisi della velocità di trasferimento: questa potrebbe essere una possibilità adottabile, ma non è stata scelta per evitare che il programma aggiungesse *overhead* a livello di ricezione. Il *throughput* voluto è dato dalla quantità di byte trasferiti diviso il tempo impiegato dalle informazioni ad arrivare dal mittente al destinatario. Wireshark non è quindi adatto per una analisi specifica visto che al tempo di trasferimento potrebbe aggiungere quello richiesto per storicizzare il contenuto dei pacchetti e per creare le statistiche corrette. Perciò si è scelta la strada che forniva il *throughput* migliore minimizzando l'eventuale *overhead*, ricadendo nell'utilizzo del comando Unix `time`. `time` prende come argomento un comando: quando quest'ultimo termina, viene scritto nello *standard error* il tempo impiegato dall'esecuzione dell'argomento ottenuto come differenza tra i due tempi di *clock*[4]. Il *throughput* è quindi dato da una semplice divisione tra i byte del file e il tempo stimato dal comando.

Riassumendo, se il server Netcat è generato con il comando

[4]Questo viene indicato come *real time* ed è il tempo realmente interessante per i test. L'output di `time` fornisce anche altri due tempi, lo *user time* e il *system time* che rappresentano rispettivamente il tempo di utilizzo della CPU per eseguire qualche azione del comando e il tempo di utilizzo della CPU per richiamare delle *system call*.

`nc -l -p 50000 >nomeFile`,
sul client si richiederà il file tramite la chiamata
`time nc 10.8.0.1 50000 <nomeFile`.

Iperf test

Il terzo test ha anch'esso come obiettivo quello di analizzare la *bandwidth* utilizzata dalle varie tecnologie, tuttavia utilizzando un programma mirato e specifico per lo scopo. Quello che nella pratica viene fatto è "battezzare" un host come server[5] e uno come client. L'host che esegue la parte server di *iperf* rimane in ascolto di connessioni entranti sulla porta TCP 5001. Il client, che per semplicità utilizza *jperf* e la sua GUI, invia un flusso TCP per 300 secondi. Per evitare che il contenuto dei pacchetti influisca sull'analisi, un file contenente numeri pseudo-casuali fa da input per i dati dei segmenti. Il comando *iperf* associato al client è:

```
iperf -c x.y.z.t -P 1 -i 1 -p 5001 -f K -t 300 -F file_path/file
```

I report non vengono successivamente mostrati tutti, perché per ogni tecnologia e per ogni scenario ne avremmo 300 (si ricorda che '-i 1' richiede che ogni secondo venga generato il report dal programma). Per questo motivo si è deciso di riportare i dati tramite grafici[6].

Overhead test

Pur non essendo un test a parte, l'analisi dell'*overhead* aggiunto dalle VPN rappresenta un valore di confronto importante. Per ottenere questa in-

[5] Con il comando `iperf -s`.

[6] Nonostante *jperf* dia la possibilità di esportare anche il solo grafico ottenuto, questo è stato creato tramite il foglio elettronico di OpenOffice, per avere una visualizzazione migliore.

formazione si è utilizzato *ifconfig* durante uno scambio FTP. Essendo i pesi degli *header* tutti valori fissi, sia all'interno dello stesso scenario che in scenari diversi, è stato sufficiente ottenerli una volta e non ripetere l'operazione per ogni download.

ifconfig mostra il numero di byte trasmessi su una determinata interfaccia; terminato il download, una semplice sottrazione tra il valore mostrato dal comando e l'effettiva grandezza del file fornisce un *overhead* relativo (ovviamente non rappresenta la somma degli *header* di un singolo pacchetto, ma dell'intera trasmissione del file), facilmente confrontabile con gli altri ugualmente ottenuti.

I valori seguenti sono molto indicativi e danno una informazione approssimativa degli *header* aggiunti; nonostante questo test sia stato fatto nello scenario *wired*, eventuali ritrasmissioni non vengono escluse dall'analisi e questo ci potrebbe mostrare dati errati. Si sottolinea quindi, ancora una volta, che i seguenti dati non sono precisi, ma utili per avere una visione di massima riguardo i byte aggiunti dalle diverse soluzioni.

Soluzione VPN	**Overhead relativo**
OpenVPN UDP	20.1 MB
OpenVPN TCP	20.2 MB
PPTP	19.5 MB
NeoRouter	26.4 MB

Tabella 3.1: Overhead relativi per ogni soluzione VPN. A questi valori, che corrispondono solo agli overhead effettivi, vanno aggiunti i 500 MB di dati veri e propri.

3.1 Primo scenario: LAN cablata

Il primo e più semplice dei due scenari è quello che ha le VPN che girano all'interno di una rete locale collegata tramite cavi Ethernet. Creare una VPN all'interno di una LAN può avere un valore puramente sperimentale se installata in una casa, visto la quasi totale assenza di utilità pratica dell'architettura, ma può essere utile se creata in realtà più ampie, come aziende di grandi dimensioni. Ovviamente i benefici li possiamo ottenere sulla gestione dei diversi tipi di accessi a particolari dati e sulla sicurezza degli stessi se il canale è condiviso.
In questo scenario i pacchetti sfruttano tutta la *bandwidth* disponibile nella LAN[7] e sono rari fenomeni di *packet loss* o duplicazione di pacchetti. I dati raccolti non mostrano particolari differenze tra le varie opzioni, probabilmente perché la distanza tra gli *host* è talmente breve da non amplificare i ritardi dovuti a incapsulamenti, compressioni e caratteristiche simili.

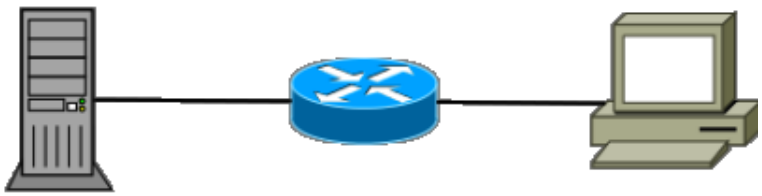

Figura 3.1: Primo scenario: LAN cablata. Nella figura è rappresentato a sinistra l'host che ha il ruolo di server VPN e a destra il client. I due sono interconnessi a un router attraverso cavi Ethernet.

[7]Le rilevazioni sono importanti se confrontate tra loro, non tanto come valori assoluti. Per questo non ci si sofferma sulle caratteristiche della specifica architettura utilizzata.

3.1.1 LAN cablata senza VPN

Ping test

Trasmissioni Corrette	MinRTT	AvgRTT	MaxRTT	Deviazione Standard
1000/1000	0.412	0.511	4.781	0.143

Tabella 3.2: Ping LAN cablata senza VPN. Tutti i tempi sono misurati in secondi.

Netcat download test

Throughput A	Throughput B	Throughput C
10012.7 ± 87.4	10094.0 ± 64.7	10158.5 ± 239.1

Tabella 3.3: Netcat in LAN cablata senza VPN. Il *throughput* A fa riferimento al file comprimibile, il *throughput* B a quello non comprimibile e il *throughput* C a quello parzialmente comprimibile. I due valori, rispettivamente in alto e in basso, sono la media e la deviazione standard di 5 rilevazioni. L'unità di misura utilizzata è KB/secondo.

Iperf test

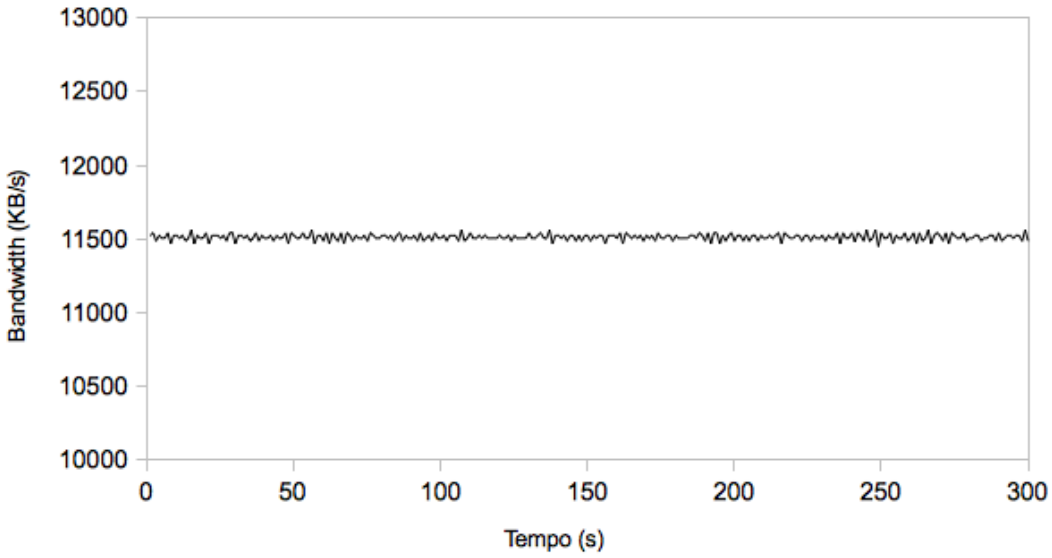

Figura 3.2: Iperf in LAN Wired senza VPN, con una media di 11511.77 KB/s su 7 rilevazioni

3.1.2 OpenVPN UDP in LAN cablata

Ping test

Trasmissioni Corrette	MinRTT	AvgRTT	MaxRTT	Deviazione Standard
1000/1000	0.807	0.904	1.107	0.046

Tabella 3.4: Ping LAN cablata su OpenVPN UDP. Tutti i tempi sono misurati in secondi.

Netcat download test

Throughput A	Throughput B	Throughput C
16908.5 ± 594.5	9436.9 ± 75.3	9545.3 ± 116.0

Tabella 3.5: Netcat in LAN cablata su OpenVPN UDP. Il *throughput* A fa riferimento al file comprimibile, il *throughput* B a quello non comprimibile e il *throughput* C a quello parzialmente comprimibile. I due valori, rispettivamente in alto e in basso, sono la media e la deviazione standard di 5 rilevazioni. L'unità di misura utilizzata è KB/secondo.

Iperf test

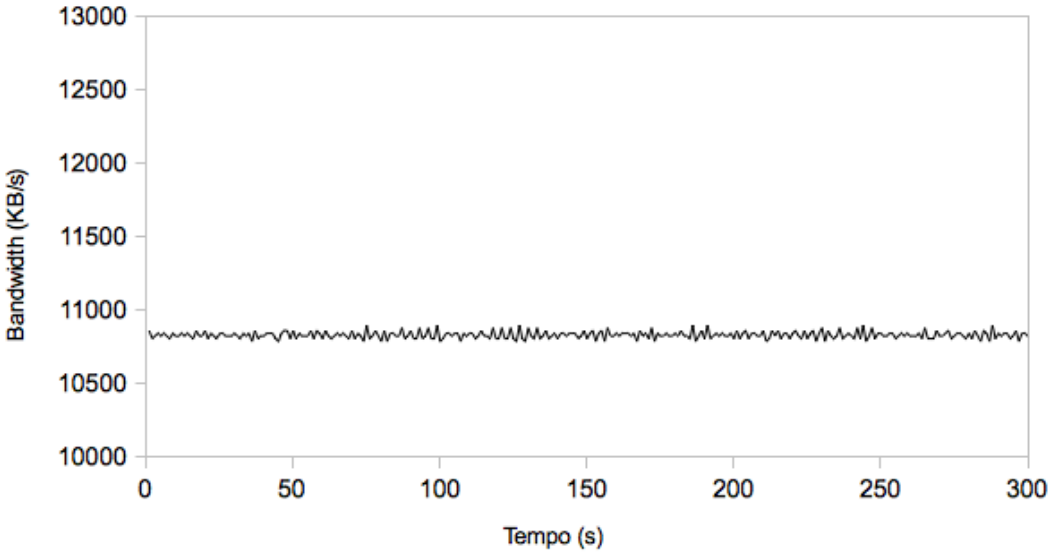

Figura 3.3: Iperf in LAN Wired con OpenVPN UDP, con una media di 10832.5 KB/s su 7 rilevazioni

3.1.3 OpenVPN TCP in LAN cablata

Ping test

Trasmissioni Corrette	MinRTT	AvgRTT	MaxRTT	Deviazione Standard
1000/1000	0.586	1.068	1.749	0.070

Tabella 3.6: Ping LAN cablata su OpenVPN TCP. Tutti i tempi sono misurati in secondi.

Netcat download test

Throughput A	Throughput B	Throughput C
18019.8 ± 367.7	9206.6 ± 134.4	9042.1 ± 233.2

Tabella 3.7: Netcat in LAN cablata su OpenVPN TCP. Il *throughput* A fa riferimento al file comprimibile, il *throughput* B a quello non comprimibile e il *throughput* C a quello parzialmente comprimibile. I due valori, rispettivamente in alto e in basso, sono la media e la deviazione standard di 5 rilevazioni. L'unità di misura utilizzata è KB/secondo.

Iperf test

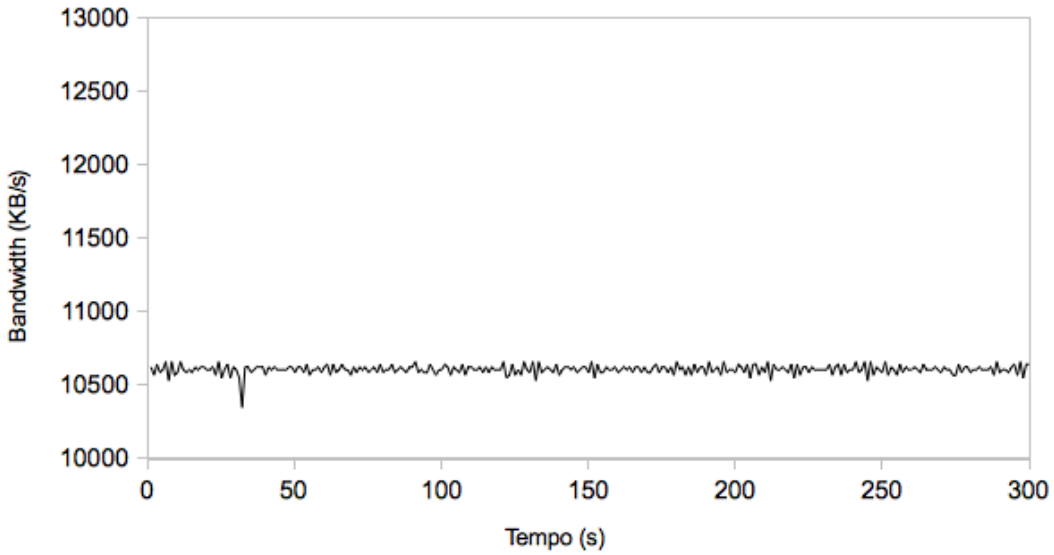

Figura 3.4: Iperf in LAN Wired con OpenVPN TCP, con una media di 10607.2 KB/s su 7 rilevazioni

3.1.4 PPTP in LAN cablata

Ping test

Trasmissioni Corrette	MinRTT	AvgRTT	MaxRTT	Deviazione Standard
1000/1000	0.722	0.886	1.043	0.042

Tabella 3.8: Ping LAN cablata su PPTP. Tutti i tempi sono misurati in secondi.

Netcat download test

Throughput A	Throughput B	Throughput C
9619.4 ± 104.6	9712.1 ± 24.2	9642.6 ± 167.4

Tabella 3.9: Netcat in LAN cablata su PPTP. Il *throughput* A fa riferimento al file comprimibile, il *throughput* B a quello non comprimibile e il *throughput* C a quello parzialmente comprimibile. I due valori, rispettivamente in alto e in basso, sono la media e la deviazione standard di 5 rilevazioni. L'unità di misura utilizzata è KB/secondo.

Iperf test

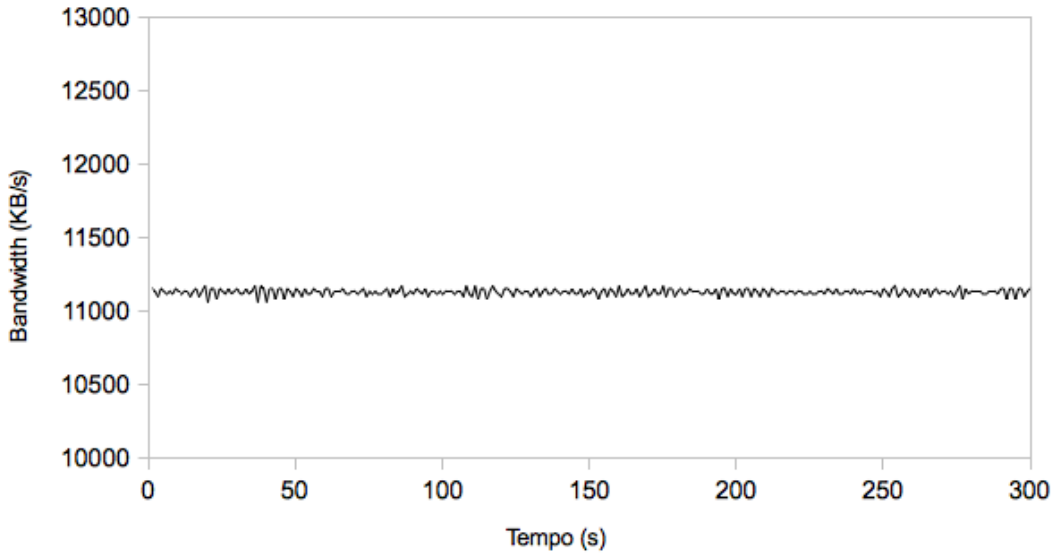

Figura 3.5: Iperf in LAN Wired con PPTP, con una media di 11132.0 KB/s su 7 rilevazioni

Di seguito viene mostrato un piccolo esempio di cattura di una sessione PPTP tramite Wireshark (Figura 3.6). Come specifica l'RFC, i pacchetti inviati attraverso l'interfaccia virtuale (nella caso della figura seguente pacchetti TCP e FTP-DATA) vengono incapsulati nei protocolli GRE e PPP. Nell'esempio notiamo che il protocollo riconosciuto da Wireshark è "PPP Comp": come specificato da una apposita RFC [27], esiste un modulo di PPP chiamato CCP (*Compression Control Protocol*) che si occupa della gestione della compressione. Ne vengono supportati diversi tipi tra cui quella dei dati e quella degli *header*. Nonostante la configurazione utilizzata per PPTP non abiliti alcun algoritmo di compressione, parte del compito è demandato a questa parte di PPP. Dalla cattura sembrerebbe che la compressione viene utilizzata; stando ai *throughput* ottenuti, l'unica possibilità è che CCP com-

prima unicamente le intestazioni, infatti i dati rimangono constanti in tutte e tre le ipotesi analizzate. Tuttavia, il guadagno ottenuto dalla compressione non supera il calo di prestazioni generato dallo stesso meccanismo e da quello di cifratura. Rimane comunque un ottimo compromesso se si vuole avere un dato costante indipendente dalla tipologia di file scambiato.

10.0.0.1	10.0.0.2	FTP-DATA
10.8.0.3	10.8.0.2	GRE
10.0.0.1	10.0.0.2	FTP-DATA
10.8.0.2	10.8.0.3	PPP Comp
10.0.0.1	10.0.0.2	FTP-DATA
10.8.0.3	10.8.0.2	GRE
10.0.0.1	10.0.0.2	FTP-DATA
10.8.0.2	10.8.0.3	PPP Comp
10.0.0.2	10.0.0.1	TCP
10.8.0.3	10.8.0.2	PPP Comp
10.0.0.2	10.0.0.1	TCP
10.8.0.3	10.8.0.2	PPP Comp
10.0.0.2	10.0.0.1	TCP
10.8.0.3	10.8.0.2	PPP Comp
10.0.0.1	10.0.0.2	FTP-DATA
10.8.0.3	10.8.0.2	PPP Comp

Figura 3.6: Cattura con Wireshark di PPTP. Gli IP 10.0.0.1 e 10.0.0.2 sono quelli virtuali, attraverso i quali vengono inviati pacchetti FTP e TCP. Attraverso gli IP 10.8.0.2 e 10.8.0.3 passano invece pacchetti GRE e PPP Comp, con all'interno i dati criptati.

3.1.5 NeoRouter in LAN cablata

Ping test

Trasmissioni Corrette	MinRTT	AvgRTT	MaxRTT	Deviazione Standard
1000/1000	0.906	1.165	1.424	0.069

Tabella 3.10: Ping LAN cablata su NeoRouter. Tutti i tempi sono misurati in secondi.

Netcat download test

Throughput A	Throughput B	Throughput C
8707.6 ± 181.8	8813.0 ± 314.5	8740.7 ± 340.3

Tabella 3.11: Netcat in LAN cablata su NeoRouter. Il *throughput* A fa riferimento al file comprimibile, il *throughput* B a quello non comprimibile e il *throughput* C a quello parzialmente comprimibile. I due valori, rispettivamente in alto e in basso, sono la media e la deviazione standard di 5 rilevazioni. L'unità di misura utilizzata è KB/secondo.

Iperf test

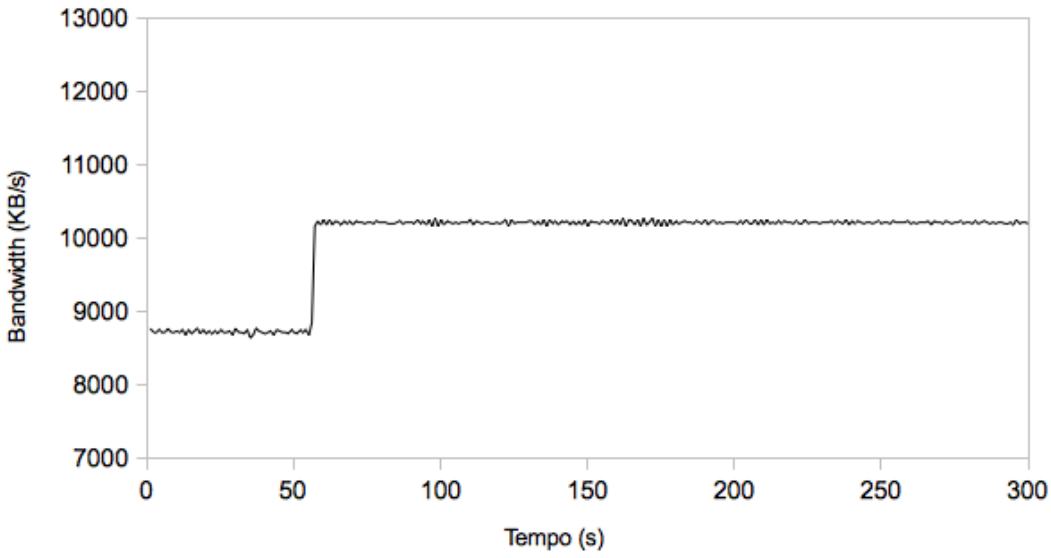

Figura 3.7: Iperf in LAN Wired con NeoRouter, con una media di 9940.2 KB/s su 7 rilevazioni

Dalla documentazione di NeoRouter non è chiaro come effettivamente avvenga una comunicazione tra client e server. Dalle catture tramite Wireshark si può tuttavia dedurre che lo scambio diretto tra i due host non inizi nell'istante in cui si connettono, ma dopo un certo tempo. Durante la prima parte, ad esempio, di un download tramite FTP, NeoRouter sembra triangolare la comunicazione tra il client e il server verso l'esterno, quindi in Internet. Durante il login, il client può connettersi in due modi: specificando il dominio, soluzione ovviamente impraticabile se la LAN non è connessa a Internet perché, come già detto nel capitolo 2, il client deve prima connettersi a un server di NeoRouter per risolvere il dominio 'proprietario', oppure direttamente l'indirizzo IP del server, unico modo per poter utilizzare la soluzione senza Internet. La triangolazione iniziale, che avviene nel caso si sia specifi-

cato un nome di dominio, ritarda lo scambio di dati con ovvie conseguenze sulle prestazioni, come mostra il test Iperf (Figura 3.7 e Figura 3.10). Nella prossima figura (Figura 3.8) viene mostrato come, finché non inizia l'effettivo download, la connessione passa attraverso un indirizzo IP esterno.

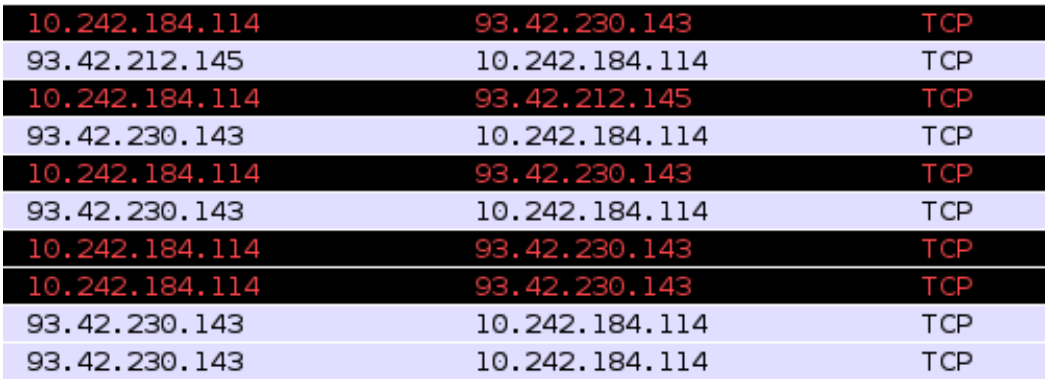

Figura 3.8: L'IP privato 10.242.184.114 comunica con un IP esterno per tutta la prima parte della connessione, se la LAN è connessa a Internet. L'indirizzo 93.42.230.143 appartiene al *provider*, ma questa comunicazione avviene solo all'inizio di una comunicazione di questo tipo, facendo pensare che l'host a questo indirizzo si comporti da *proxy*, effettuando richieste a un server di NeoRouter.

10.242.184.113	10.242.184.114	UDP
10.0.0.3	10.0.0.2	FTP-DATA
10.242.184.113	10.242.184.114	UDP
10.0.0.2	10.0.0.3	TCP
10.242.184.113	10.242.184.114	UDP
10.0.0.3	10.0.0.2	FTP-DATA
10.242.184.113	10.242.184.114	UDP
10.0.0.3	10.0.0.2	FTP-DATA
10.242.184.113	10.242.184.114	UDP
10.0.0.2	10.0.0.3	TCP

Figura 3.9: Il vero scambio di dati avviene normalmente, passando da un host a un altro, entrambi connessi alla VPN, tramite UDP.

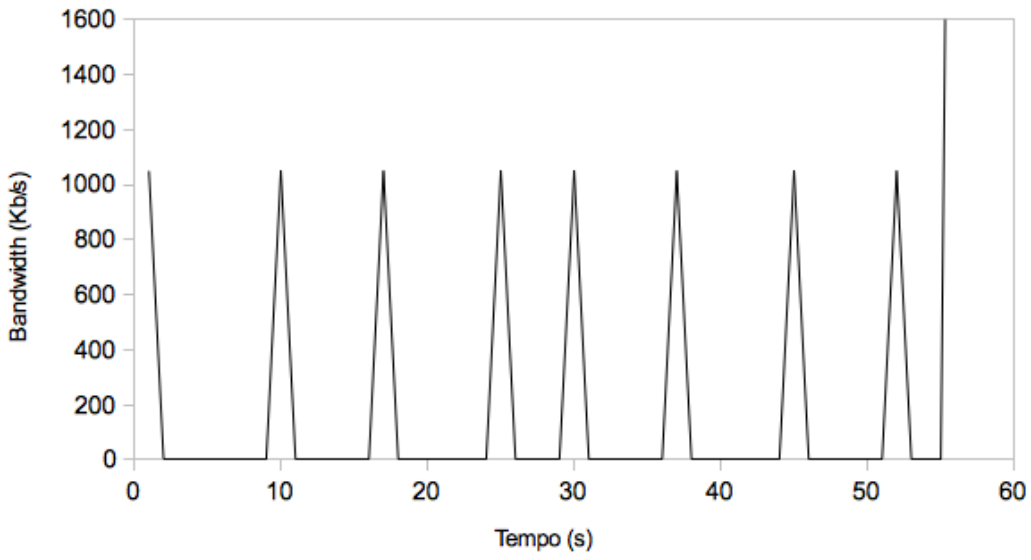

Figura 3.10: Dettaglio della prima parte del test Iperf con NeoRouter in LAN Wired. Nei primi cinquanta secondi di trasmissione della prima rilevazione, non si supera una bandwidth massima di 1094 Kb/s. Successivamente, il *throughput* aumenta mantenendosi costante fino al termine del test.

3.1.6 Confronto in LAN cablata

Ping test

Strumento	Trasmissioni Corrette	MinRTT	AvgRTT	MaxRTT	Deviazione Standard
Senza VPN	1000/1000	0.412	0.511	4.781	0.144
OpenVPN UDP	1000/1000	0.807	0.904	1.107	0.046
OpenVPN TCP	1000/1000	0.586	1.068	1.749	0.070
PPTP	1000/1000	0.722	0.886	1.043	0.042
NeoRouter	1000/1000	0.906	1.165	1.424	0.069

Tabella 3.12: Confronto di Ping in LAN cablata. I tempi sono misurati in secondi.

Netcat download test

	Senza VPN	OpenVPN UDP	OpenVPN TCP	PPTP	NeoRouter
A[8]	10012.7 ± 87.4	16908.5 ± 594.5	18019.8 ± 367.7	9619.4 ± 104.6	8707.6 ± 181.8
B[8]	10094.0 ± 64.7	9436.9 ± 75.3	9206.6 ± 134.4	9712.1 ± 24.2	8813.0 ± 314.5
C[8]	10158.5 ± 239.1	9545.3 ± 116.0	9042.1 ± 233.2	9642.6 ± 167.4	8740.7 ± 340.3

Tabella 3.13: Confonto di download con Netcat in LAN cablata. In ogni cella viene riportata la media e la varianza, misurate in KB/secondo.

Iperf test

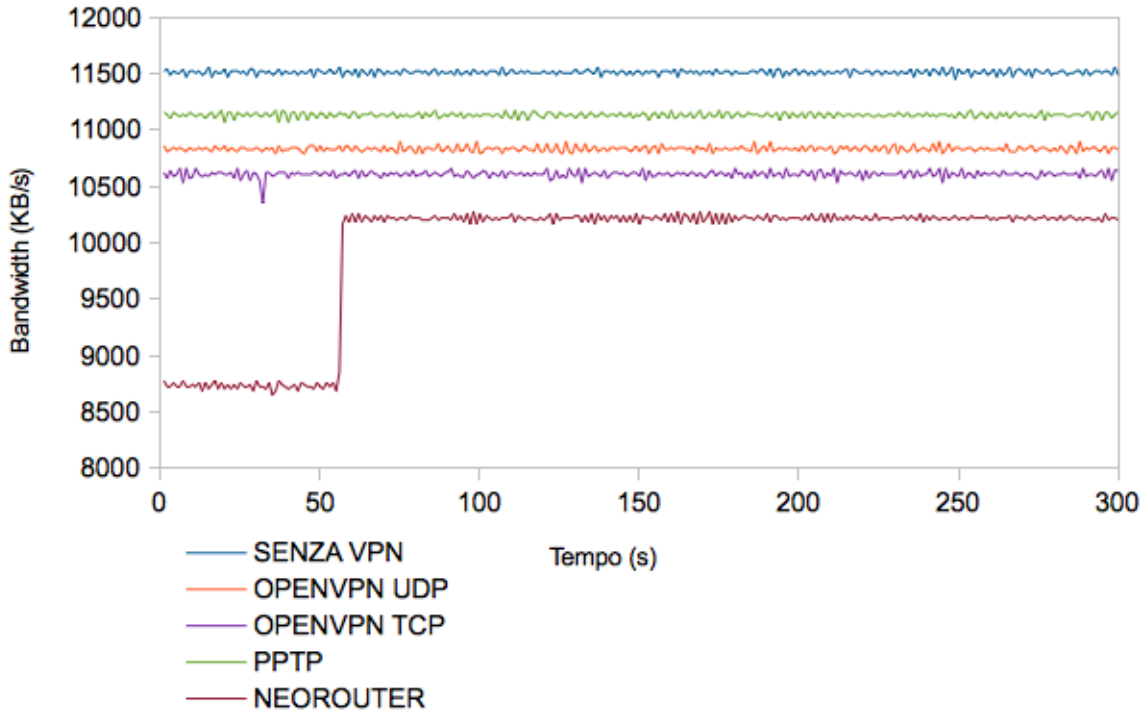

Figura 3.11: Confronto Iperf in LAN Wired. Il grafico mostra le medie, in ogni secondo, di 7 rilevazioni per ogni VPN.

[8] A, B, C indicano il *throughput* ottenuto dal download dei tre file, rispettivamente quello completamente comprimibile, quello creato con valori casuali e quello utilizzando come contenuto file di diverso genere.

Raggruppando i dati ottenuti dalle rilevazioni è possibile analizzare le informazioni e fornire delle considerazioni sui confronti fatti. Le prime riguardano la latenza: come ci si poteva aspettare, in tutti e cinque i casi i pacchetti inviati sono arrivati a destinazione in modo corretto; altro aspetto facilmente prevedibile è che l'RTT migliore lo otteniamo nel caso in cui non è stata creata nessuna VPN tra i due host, perché non viene aggiunto nessun *overhead* ai pacchetti ICMP di Ping. Analogamente anche le altre soluzioni VPN hanno una latenza media che cresce all'aumentare dell'overhead, ovvero maggiore è il carico di informazioni aggiunte al pacchetto, maggiore è l'RTT. Un dato che risulta invece inatteso riguarda la deviazione standard legata alle diverse rilevazioni: si può notare come senza VPN lo scarto quadratico medio è maggiore rispetto agli altri, come se la creazione del tunnel dia una stabilità più alta sulla latenza.
L'analisi del *throughput* durante il download di file tramite l'utilizzo del comando Netcat mostra diversi aspetti caratteristici delle soluzioni utilizzate. Per prima cosa si nota che la velocità di trasferimento maggiore la otteniamo (escludendo il primo file, considerato irrealistico) senza l'utilizzo di VPN. Anche in questo caso, come nell'analisi di Ping, il dato era facilmente prevedibile poichè è proprio l'aggiunta delle migliorie che le VPN portano a generare ritardi nel trasferimento.
Nei tre casi (corrispondenti ai tre file), ogni soluzione mantiene grossomodo gli stessi valori, con le uniche eccezioni di OpenVPN UDP e OpenVPN TCP. In entrambi i casi il fattore di compressione si manifesta aumentando il *throughput* medio calcolato in fase di ricezione durante l'invio del primo file, quello totalemente comprimibile perchè formato da tutti zeri. Nonostante appositamente non si sia specificata la natura della LAN, in questo caso è interessante sottolineare come, con dati estremamente compressi, la velocità

di trasferimento percepita supera il *throughput* massimo consentito dalle caratteristiche fisiche della LAN Ethernet (OpenVPN trasferisce il primo file a una media di 130-140 Mb/s, superando la massima velocità consentita di 100 Mb/s). PPTP, che pur avendo da configurazione disabilitato l'uso di questi algoritmi, utilizza il servizio di compressione offerto da PPP, ottenendo dei risultati indipendenti dal tipo di file scambiato.

Il caso che più si avvicina a un possibile scambio reale coincide con il terzo file. Nonostante sia diviso in modo equo tra testo e immagini, probabilmente il guadagno ottenuto dall'invio di dati compressi viene perso proprio durante le fasi di compressione e decompressione, essendo testato in uno scenario a bassa latenza.

Un'altra analisi interessante riguarda le deviazioni standard, sempre maggiori durante il download del primo file: anche in questo caso il problema potrebbe essere determinato dal tentativo di comprimere i dati.

Il confronto rispecchia esattamente quello fatto per il Ping test: anche in questo caso a *overhead* maggiore corrisponde una minore prestazione.

In ultimo viene analizzato il test specifico sulla *bandwidth*: riportando in un unico grafico le cinque curve si riconferma quello che già è stato mostrato dagli altri due test, ovvero che le prestazioni delle soluzioni hanno un andamento proporzionale al carico aggiunto dal tunnel. Inoltre, come ci si può aspettare, questo scenario è sostanzialmente esente da problemi che possono verificarsi in una rete più complessa, e questo è confermato dalla stabilità delle rilevazioni ottenute (ad esclusione del caso di NeoRouter, problema già mostrato precedentemente).

3.2 Secondo scenario: LAN wireless

Il secondo scenario prevedere un'architettura simile alla precedente, con la differenza che il client e il server VPN sono connessi al *router* tramite una rete *wireless* che utilizza lo standard IEEE 802.11. Questo ambiente risulta essere più interessante, sempre in campo di test, rispetto alla LAN Ethernet, perché permette di sottolineare i problemi di prestazioni che i protocolli hanno in caso di perdita o duplicazione dei pacchetti, problemi più frequenti nel mondo WiFi che in quello *wired*.
Probabilmente è uno scenario strano da ritrovare nella vita reale poiché, solitamente, le grandi aziende che richiederebbero l'uso di VPN interne alla rete locale utilizzano collegamenti *wired* dedicati; tuttavia, in linea teorica, sarebbe opportuno utilizzare una VPN in un ambiente *wireless* per proteggere i dati che transitano sul canale condiviso.
Nel test con Netcat si potrebbe riscontrare un effetto noto come *meltdown*: è un problema specifico delle situazioni di $TCP-over-TCP$ (questa possibilità verrà spiegata nel prossimo capitolo nell'ambito dei problemi riscontrati e riscontrabili). Nonostante GRE, protocollo in cui incapsula PPTP, abbia caratteristiche molto simili a TCP, il diverso uso che fa dei numeri di sequenza preserva la VPN dalla possibilità di incorrere in questo problema.

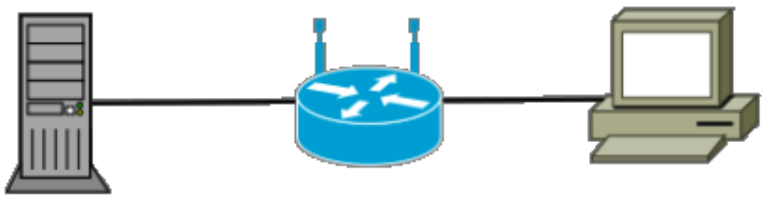

Figura 3.12: Secondo scenario: LAN wireless

3.2.1 LAN wireless senza VPN

Ping test

Trasmissioni Corrette	MinRTT	AvgRTT	MaxRTT	Deviazione Standard
986(7*d*)/1000	2.029	244.697	461.375	83.810

Tabella 3.14: Ping LAN *wireless* senza VPN. Tutti i tempi sono misurati in secondi. Tra i 986 pacchetti ricevuti correttamente, 7 di questi sono duplicati, ovvero hanno un identificativo (*icmp_sequence*) uguale ad altri.

Netcat download test

Throughput A	Throughput B	Throughput C
361.2	355.9	316.5
±	±	±
14.7	40.5	31.0

Tabella 3.15: Netcat in LAN *wireless* senza VPN. Il *throughput* A fa riferimento al file comprimibile, il *throughput* B a quello non comprimibile e il *throughput* C a quello parzialmente comprimibile. I due valori, rispettivamente in alto e in basso, sono la media e la deviazione standard di 5 rilevazioni. L'unità di misura utilizzata è KB/secondo.

Iperf test

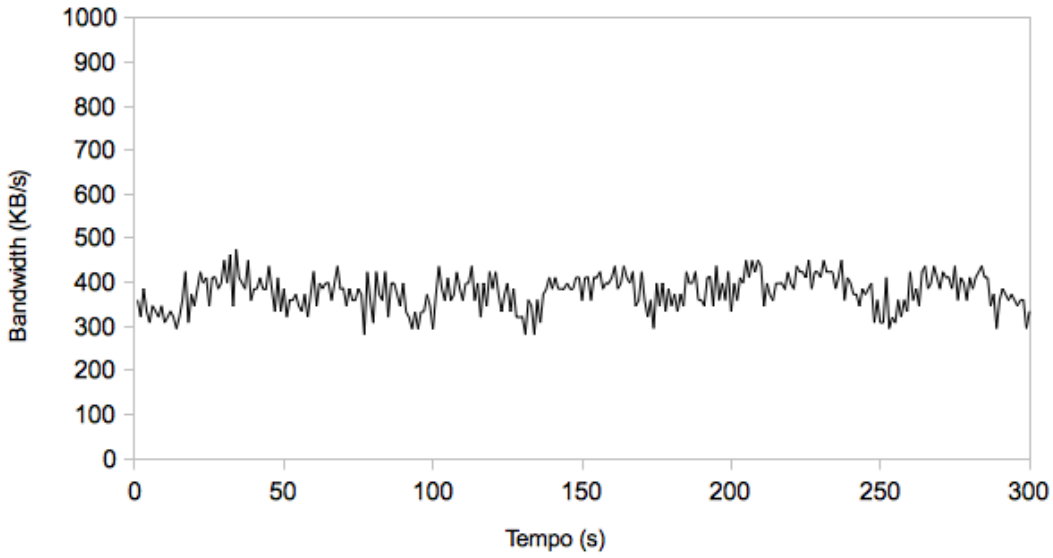

Figura 3.13: Iperf in LAN Wireless senza VPN, con una media di 378.5 KB/s su 10 rilevazioni

3.2.2 OpenVPN UDP in LAN wireless

Ping test

Trasmissioni Corrette	MinRTT	AvgRTT	MaxRTT	Deviazione Standard
978/1000	3.184	242.101	456.233	88.543

Tabella 3.16: Ping LAN *wireless* su OpenVPN UDP. Tutti i tempi sono misurati in secondi.

Netcat download test

Throughput A	Throughput B	Throughput C
563.3 ± 14.8	240.0 ± 26.5	286.3 ± 79.4

Tabella 3.17: Netcat in LAN *wireless* su OpenVPN UDP. Il *throughput* A fa riferimento al file comprimibile, il *throughput* B a quello non comprimibile e il *throughput* C a quello parzialmente comprimibile. I due valori, rispettivamente in alto e in basso, sono la media e la deviazione standard di 5 rilevazioni. L'unità di misura utilizzata è KB/secondo.

Iperf test

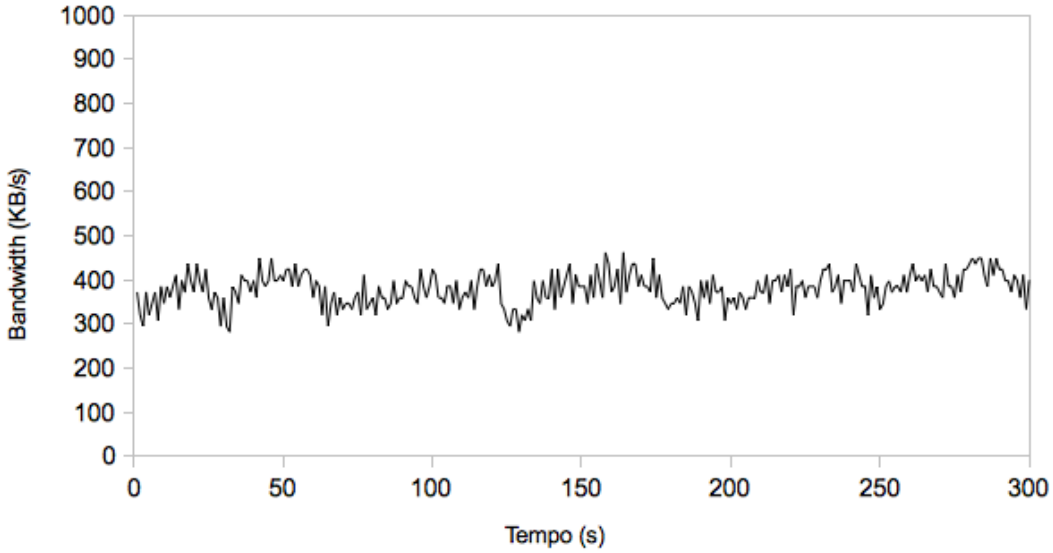

Figura 3.14: Iperf in LAN Wireless con OpenVPN UDP,
con una media di 379.5 KB/s su 10 rilevazioni

3.2.3 OpenVPN TCP in LAN wireless

Ping test

Trasmissioni Corrette	MinRTT	AvgRTT	MaxRTT	Deviazione Standard
1000/1000	3.284	266.358	1176.032	107.520

Tabella 3.18: Ping LAN *wireless* su OpenVPN TCP. Tutti i tempi sono misurati in secondi.

Netcat download test

Throughput A	Throughput B	Throughput C
1310.3 ± 120.3	151.4 ± 44.3	191.1 ± 22.3

Tabella 3.19: Netcat in LAN *wireless* su OpenVPN TCP. Il *throughput* A fa riferimento al file comprimibile, il *throughput* B a quello non comprimibile e il *throughput* C a quello parzialmente comprimibile. I due valori, rispettivamente in alto e in basso, sono la media e la deviazione standard di 5 rilevazioni. L'unità di misura utilizzata è KB/secondo.

Iperf test

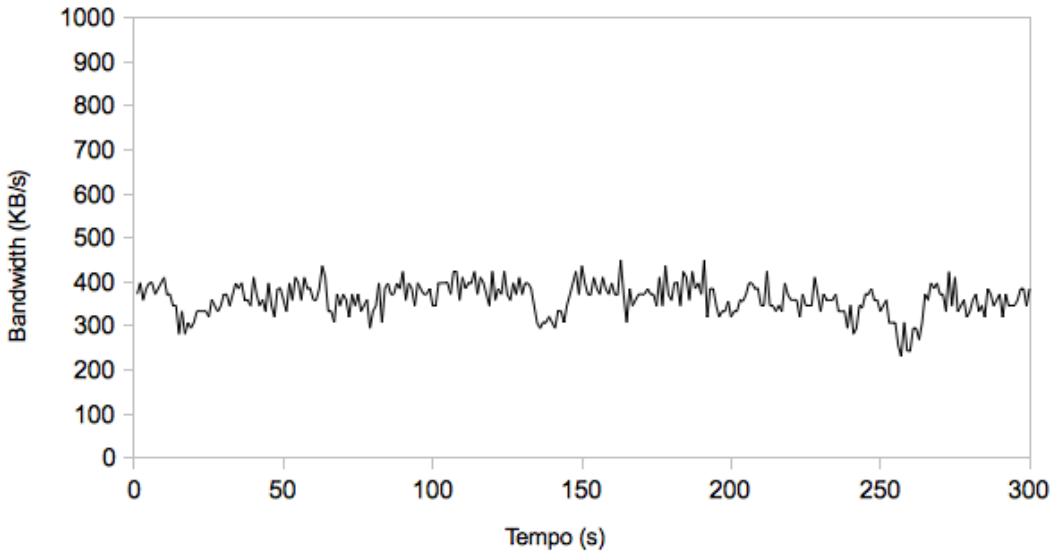

Figura 3.15: Iperf in LAN Wireless con OpenVPN TCP, con una media di 362.2 KB/s su 10 rilevazioni

3.2.4 PPTP in LAN wireless

Ping test

Trasmissioni Corrette	MinRTT	AvgRTT	MaxRTT	Deviazione Standard
986/1000	2.631	249.232	1408.226	154.252

Tabella 3.20: Ping LAN *wireless* su PPTP. Tutti i tempi sono misurati in secondi.

Netcat download test

Throughput A	Throughput B	Throughput C
321.7 ± 36.8	288.7 ± 49.1	260.3 ± 10.7

Tabella 3.21: Netcat in LAN *wireless* su PPTP. Il *throughput* A fa riferimento al file comprimibile, il *throughput* B a quello non comprimibile e il *throughput* C a quello parzialmente comprimibile. I due valori, rispettivamente in alto e in basso, sono la media e la deviazione standard di 5 rilevazioni. L'unità di misura utilizzata è KB/secondo.

[8] I valori di *throughput* sono espressi in KB/s. Tra parentesi la quantità in MB di file effettivamente scaricato.

Iperf test

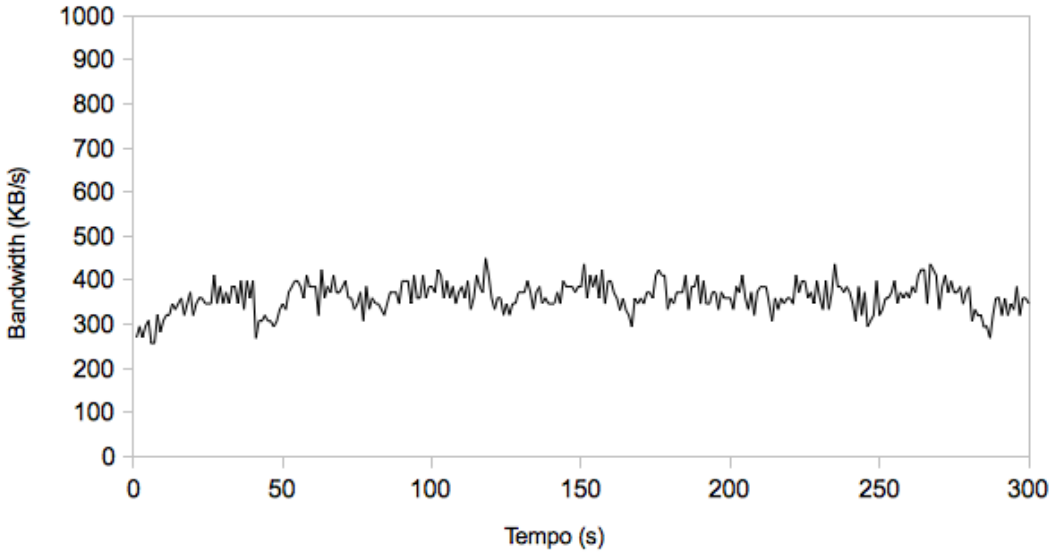

Figura 3.16: Iperf in LAN Wireless con PPTP, con una media di 361.4 KB/s su 10 rilevazioni

3.2.5 NeoRouter in LAN wireless

Ping test

Trasmissioni Corrette	MinRTT	AvgRTT	MaxRTT	Deviazione Standard
994(21*d*)/1000	2.999	252.210	426.131	78.346

Tabella 3.22: Ping LAN *wireless* su NeoRouter. Tutti i tempi sono misurati in secondi. Tra i 994 pacchetti ricevuti correttamente, 21 di questi sono duplicati, ovvero hanno un identificativo (*icmp_sequence*) uguale ad altri.

Netcat download test

Throughput A	Throughput B	Throughput C
207.9	328.2	210.3
±	±	±
15.0	86.9	29.1

Tabella 3.23: Netcat in LAN *wireless* su NeoRouter. Il *throughput* A fa riferimento al file comprimibile, il *throughput* B a quello non comprimibile e il *throughput* C a quello parzialmente comprimibile. I due valori, rispettivamente in alto e in basso, sono la media e la deviazione standard di 5 rilevazioni. L'unità di misura utilizzata è KB/secondo.

Iperf test

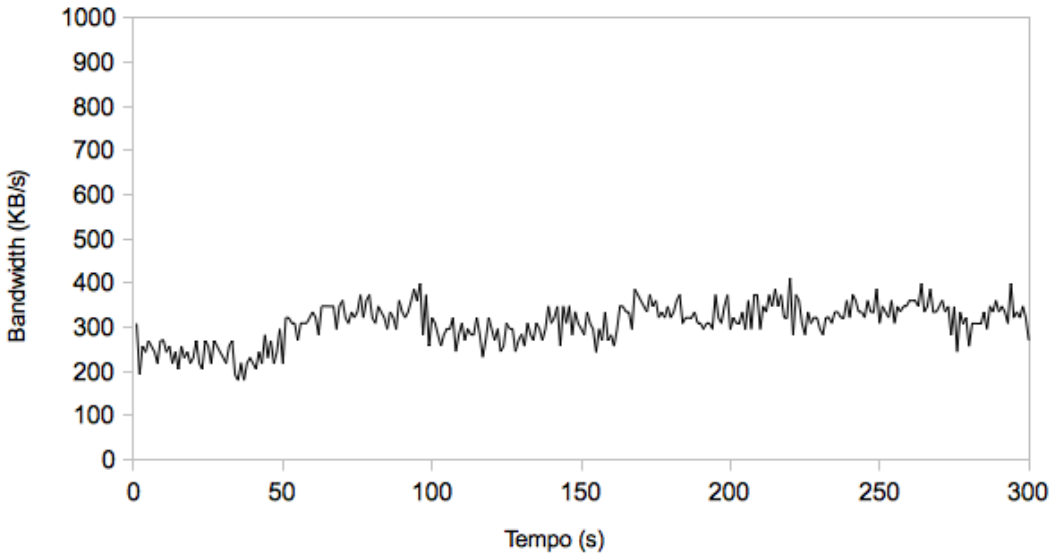

Figura 3.17: Iperf in LAN Wireless con NeoRouter, con una media di 307.9 KB/s su 10 rilevazioni

3.2.6 Confronto in LAN wireless

Ping test

Strumento	Trasmissioni Corrette	MinRTT	AvgRTT	MaxRTT	Deviazione Standard
Senza VPN	986(7*d*)/1000	2.029	244.697	461.375	83.810
OpenVPN UDP	978/1000	3.184	242.101	456.233	88.543
OpenVPN TCP	1000/1000	3.284	266.358	1176.032	107.520
PPTP	986/1000	2.631	249.232	1408.226	154.252
NeoRouter	994(21*d*)/1000	2.999	252.210	426.131	78.346

Tabella 3.24: Confronti di Ping in LAN *wireless*. I tempi sono misurati in secondi.

Netcat download test

	Senza VPN	OpenVPN UDP	OpenVPN TCP	PPTP	NeoRouter
A[9]	361.2 KB/s ± 14.7 KB/s	563.3 KB/s ± 14.8 KB/s	1310.3 KB/s ± 120.3 KB/s	321.7 KB/s ± 36.8 KB/s	207.9 KB/s ± 15.0 KB/s
B[9]	355.9 KB/s ± 40.5 KB/s	240.0 KB/s ± 26.5 KB/s	151.4 KB/s ± 44.3 KB/s	288.7 KB/s ± 49.1 KB/s	328.2 KB/s ± 86.9 KB/s
C[9]	316.5 KB/s ± 31.0 KB/s	286.3 KB/s ± 79.4 KB/s	191.1 KB/s ± 22.3 KB/s	260.3 KB/s ± 10.7 KB/s	210.3 KB/s ± 29.1 KB/s

Tabella 3.25: Confonti di download con Netcat in LAN *wireless*. In ogni cella vengono riportate media e varianza, misurate in KB/secondo.

Iperf test

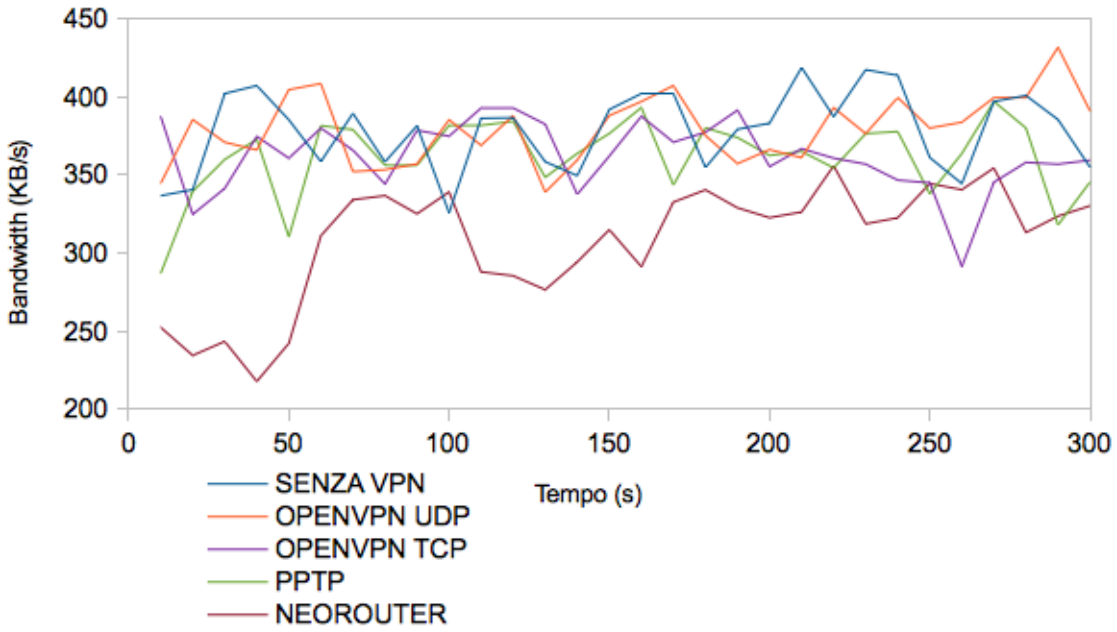

Figura 3.18: Confronto Iperf in LAN Wireless. Il grafico riporta il confronto sul test con Iperf. Vengono riportati i dati aggregati (ogni valore corrisponde a una media di 10 secondi) partendo da quelli ottenuti da 10 rilevazioni.

[9] A, B, C indicano il *throughput* ottenuto dal download dei tre file, rispettivamente quello completamente comprimibile, quello creato con valori casuali e quello utilizzando come contenuto file di diverso genere.

Lo scenario della rete locale con collegamenti *wireless* mette alla prova l'effettiva efficienza delle soluzioni e configurazioni utilizzate. Per prima cosa si può notare che OpenVPN TCP è l'unico tunnel che ha permesso a tutti i 1000 pacchetti ping di arrivare a destinazione; maggiore è l'RTT medio, più pacchetti arrivano a destinazione. Se si effettua contemporaneamente una cattura tramite Wireshark su entrambe le interfacce, reale e virtuale, si nota la corrispondenza uno-a-uno tra pacchetto ICMP e lo stesso 'incapsulato nel tunnel' (Figura 3.19). Anche NeoRouter si comporta in questo modo, anche se alterna l'invio dei normali pacchetti di REQUEST e di REPLY (incapsulati in UDP), alla solita triangolazione verso l'esterno, e forse è questo il motivo che amplifica la quantità di pacchetti duplicati.
Contrariamente allo scenario precedente, in ambiente wireless si notano particolarmente gli effetti della compressione, soprattutto in OpenVPN TCP: con una media di 1300 KB/s, supera di quasi quattro volte il *throughput* ottenuto senza l'installazione di VPN. Pur essendo un risultato totalmente irrealistico, è utile per capire la capacità di un meccanismo di compressione di questo tipo, se portato all'estremo. Anche se non così evidenziata, l'utilità della compressione (se il file lo "permette") si riscontra anche in OpenVPN UDP e PPTP: questi sono le uniche tre soluzioni che comprimono i dati prima di trasmetterli o con algoritmi specifici o, nel caso dell'ultima, utilizzando il meccanismo offerto da PPP.
In linea generale non è qui possibile determinare un chiaro andamento dei diversi protocolli, ovvero non hanno un comportamento stabile. Questa instabilità si nota sia dai dati numerici che dal grafico di confronto; anche tralasciando i risultati ottenuti dal primo download, considerato un caso non ottenibile nella realtà, non è possibile delineare un andamento univoco seguito dalle diverse soluzioni, e questo è dovuto alle caratteristiche proprie

dell'ambiente. Tuttavia si può notare come in entrambe le configurazioni di OpenVPN il terzo *throughput* si collochi tra i due precedenti, come era prevedibile, poiché, almeno in parte, si sentono i benefici della compressione.
Paragonando i tre dati ottenuti con PPTP si nota come il peggior *throughput* sia quello ottenuto dal download del terzo file. Probabilmente in uno scambio così ravvicinato non si riesce a notare il guadagno ottenuto dalla compressione, poiché è un guadagno che viene compensato dal tentativo di compressione stesso.
NeoRouter, come nello scenario *wired*, riporta anche in questo ambiente il problema che lo lega a *throughput* bassi durante la prima parte del test; il motivo rimane lo stesso, la triangolazione iniziale con un host su internet, che rallenta lo scambio di pacchetti (Figura 3.18) e genera risultati completamente imprevedibili.

10.8.0.1	10.8.0.6	ICMP
10.242.184.114	10.242.184.113	UDP
10.8.0.6	10.8.0.1	ICMP
10.242.184.113	10.242.184.114	UDP
10.8.0.1	10.8.0.6	ICMP
10.8.0.6	10.8.0.1	ICMP
10.242.184.114	10.242.184.113	UDP
10.242.184.113	10.242.184.114	UDP
10.8.0.1	10.8.0.6	ICMP
10.8.0.6	10.8.0.1	ICMP
10.242.184.114	10.242.184.113	UDP
10.242.184.113	10.242.184.114	UDP

Figura 3.19: Cattura con Wireshark di OpenVPN UDP. 10.8.0.1 e 10.8.0.6 sono gli IP virtuali, che si scambiano pacchetti ICMP; 10.242.184.113 e 114 sono gli IP dell'interfaccia fisica, che incapsulano all'interno di segmenti UDP i pacchetti ping.

Capitolo 4

Valutazione basata su FTP

Inizialmente i test che poi sono stati fatti utilizzando Netcat, non erano stati previsti. Al loro posto la valutazione del *throughput* durante l'invio di file era fatto analizzando una sessione FTP. La differenza tra i due test sta proprio nei protocolli utilizzati per lo scambio: se da un lato le informazioni vengono spedite direttamente all'interno di un flusso TCP, dall'altro il protocollo di trasferimento file FTP gestisce l'invio. Per il resto i due test sono stati fatti in maniera identica: in entrambi sono state fatte lo stesso numero di prove, con gli stessi tre file. Ciò che ha spinto a cambiare la tipologia di test è stato l'ottenere diversi errori nei download tramite FTP che avrebbero compromesso un confronto corretto tra le diverse soluzioni.
Per semplicità non vengono riportati tutti i dati, ma solo le medie ottenute.

	Senza VPN	OpenVPN UDP	OpenVPN TCP	PPTP	NeoRouter
A[1]	12001.7 ± 423.2	12329.1 ± 2317.5	10877.4 ± 1275.5	11334.7 ± 589.3	10679.3 ± 271.2
B[1]	12193.6 ± 19.4	11562.5 ± 38.3	11339.1 ± 36.5	11784.9 ± 24.2	10852.3 ± 150.4
C[1]	12169.7 ± 257.9	11581.1 ± 40.3	11315.5 ± 26.3	11779.7 ± 32.4	10989.4 ± 10.0

Tabella 4.1: Confonto di download FTP in LAN cablata

	Senza VPN	OpenVPN UDP	OpenVPN TCP	PPTP	NeoRouter
A[1]	230.7 KB/s ± 45.9 KB/s	488.9 KB/s ± 88.9 KB/s	1979,8 KB/s ± 763,7 KB/s	270.5 KB/s ± 37.3 KB/s	72.7 KB/s ± 35.1 KB/s
B[1]	339.4 KB/s ± 68.2 KB/s	379.7 KB/s ± 32.8 KB/s	216.2 KB/s ± 124.4 KB/s	185.6 KB/s ± 25.1 KB/s	154.5 KB/s ± 91.7 KB/s
C[1]	234,4 KB/s ± 3,7 KB/s	479.2 KB/s ± 18.5 KB/s	150.0 KB/s ± 45.93 KB/s	163.9 KB/s ± 30.6 KB/s	314.3 KB/s ± 28.1 KB/s

Tabella 4.2: Confonti di download FTP in LAN wireless

[1]A, B, C indicano il throughput ottenuto dal download dei tre file, rispettivamente quello completamente comprimibile, quello creato con valori casuali e quello utilizzando come contenuto file di diverso genere.

Per analizzare più nello specifico i dati che non ci hanno permesso di fare dei confronti opportuni, vengono mostrati i risultati ottenuti in LAN *wireless* con OpenVPN TCP e PPTP.

Numero test	$ThroughputA$[2]	$ThroughputB$[2]	$ThroughputC$[2]
1	1593.512794 (105.3)	445.7378988	105.7388938
2	1383.000206 (191.4)	234.4221275	100.8357725
3	1676.898047 (189.9)	122.5993613	150.8547063
4	1759.238016 (203.5)	182.3293763	165.76638
5	3486.509705	96.01772625	226.9602675

Tabella 4.3: FTP LAN wireless OpenVPN TCP

Numero test	$ThroughputA$[2]	$ThroughputB$[2]	$ThroughputC$[2]
1	208.3298913 (173.3)	171.1292788 (78.6)	213.6692288 (60.3)
2	260.7988325 (32.9)	188.9997875 (33.1)	186.0493638 (29.7)
3	314.6334413 (29.0)	158.7068038 (18.4)	138.92947 (40.7)
4	265.9026825 (21.1)	177.2501075 (91.8)	139.41949 (21.3)
5	302.7300725 (34.5)	231.8966613 (21.6)	141.6472888 (40.1)

Tabella 4.4: FTP LAN wireless PPTP

Come si può notare dalle tabelle mostrate, spesso il download non viene completato e la sessione FTP viene chiusa prima di essere effettivamente ultimata. Questo avviene solo con le due soluzioni sopra riportate e, nel caso

[2]I valori di throughput sono espressi in KB/s. Tra parentesi la quantità in MB di file effettivamente scaricato.

di OpenVPN TCP, solo con il file interamente comprimibile.
Diverse ipotesi possono essere fatte a riguardo, anche se si è stati obbligati a cambiare il tipo di test per avere i risultati più omogenei possibile.
Un problema improbabile ma possibile in una connessione che utilizza il *tunneling* è il *TCP-over-TCP meltdown*, ovvero un errore legato all'incapsulamento di segmenti TCP in altri dello stesso protocollo (Figura 4.1). Quando un pacchetto viene eliminato o perso, TCP utilizza dei meccanismi di ritrasmissione per garantire un servizio di trasferimento dati affidabile. Nel caso in cui TCP incapsuli un pacchetto della stessa natura, entrambi i flussi, quello esterno e quello interno, cercheranno di risolvere il problema, entrando in conflitto tra loro e generando un degrado elevato delle prestazioni. Un discorso analogo potrebbe essere fatto per PPTP, visto che GRE ha caratteristiche molto simili a TCP, anche se la diversità di utilizzo dei numeri di sequenza escluderebbe questa eventualità.

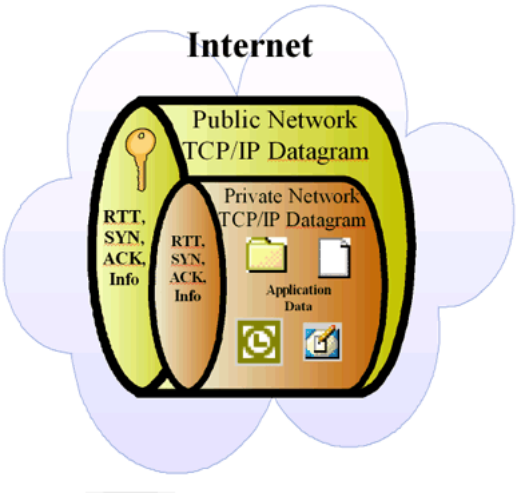

Figura 4.1: Incapsulamento di segmenti TCP in segmenti TCP. Figura tratta da *http://www.neoaccel.com/gi-index.php*

L'ipotesi del *meltdown*, tuttavia, va scartata: pur essendo possibile in una realtà in cui i pacchetti per arrivare dall'origine alla destinazione eseguono diversi *hop*, è abbastanza irreale la possibilità di avere il problema in una LAN. Inoltre, dall'analisi dei pacchetti con Wireshark, se si fosse ottenuto il *meltdown* si noterebbero molte ritrasmissioni, e questo non avviene. Infine, nel caso di OpenVPN TCP, il problema lo otteniamo solo con il primo file; questo potrebbe portarci a pensare che la compressione incide sui risultati.
Il server FTP utilizzato è Proftpd; analizzando il log, il quale viene inserito direttamente in quello di sistema, si nota che è il server a chiudere la sessione con un messaggio del tipo "*FTP: Broken pipe*". Probabilmente l'ipotesi più plausibile è una eventuale saturazione dei buffer di ricezione che, andando in *overflow*, generano un time out della sessione. In questo modo il server FTP è costretto ad inviare una serie di pacchetti *TCP-reset*[3] che terminano lo scambio.

[3]Il flag RST (*reset*) appartiene all'*header* TCP e viene settato nel caso in cui un *host* voglia terminare la connessione in modo anomalo.

Conclusioni

Le VPN portano un vantaggio fondamentale che è quello di creare un collegamento "dedicato" in un canale che non nasce come tale. Il ruolo principale è quindi quello di connettere due sistemi in modo sicuro. L'implementazione, che avviene tramite il meccanismo del *tunneling*, aggiunge tutto un insieme di caratteristiche aggiuntive come la trasparenza, l'uso di compressione e cifratura dei dati. Nonostante siano presenti alternative valide alle VPN, lo stesso problema viene superato con prezzi decisamente minori.

Tuttavia, con questa soluzione si presentano anche svariati problemi, primo fra tutti l'aumento della latenza. Ai ritardi classici delle reti come Internet (ritardo di elaborazione, di accodamento, di trasmissione e di propagazione), si aggiungono, nei sistemi terminali, quello di criptazione (e decriptazione), incapsulamento (e decapsulamento) e compressione (con la relativa decompressione). Nonostante la compressione, se abilitata, possa portare a una diminuzione della latenza, in generale questo risultato non è predicibile poiché dipende fortemente dalla natura dei dati scambiati e da quanto questi siano predisposti a essere compressi. Se l'algoritmo utilizzato non riesce a comprimere abbastanza i dati, il ritardo ottenuto da questo tentativo non viene compensato da un adeguato guadagno, perciò si ottiene un calo delle prestazioni.

Un ulteriore ritardo è quello dovuto all'aggiunta di *overhead*. Oltre al nor-

male tempo di incapsulamento, gli *header* aggiunti aumentano la grandezza in byte del pacchetto e, nel caso questa superi l'MTU (*Maximum Transmission Unit*), occorre dividerlo in parti più piccole in modo da poter superare il collegamento. Conseguenza di ciò è l'aumento del tempo di trasmissione generato da una maggiore frammentazione dei dati rispetto all'invio degli stessi senza l'utilizzo di una VPN.

La scelta del protocollo

L'obiettivo iniziale di mostrare con prove empiriche quale fosse il protocollo di *tunneling* migliore per la creazione di VPN non è stato raggiunto completamente. Per prima cosa è da notare che non solo il protocollo determina la bontà di una soluzione, ma anche l'implementazione della stessa, che specifica come il protocollo scelto viene utilizzato. Un esempio visto di ciò è la diversità di realizzazione di OpenVPN configurato per usare UDP e NeoRouter: entrambi utilizzano lo stesso protocollo ma i risultati sono molto dissimili proprio per come è strutturata la soluzione. Raramente è preferibile utilizzare una soluzione che crei il tunnel su TCP: l'unica motivazione valida si avrebbe nel caso in cui i protocolli incapsulati offrano un servizio *best-effort*, mentre la comunicazione voluta sia affidabile[4].
Per avere una valutazione complessiva migliore occorrerebbe estendere il lavoro su più fronti.
Da un lato l'aggiunta di protocolli e implementazioni diverse delle soluzioni

[4]Se incapsuliamo TCP all'interno di un protocollo di *tunneling* non affidabile come UDP, sarà il protocollo interno a garantire l'affidabilità. Se avvenisse il contrario, analogamente sarebbe sempre TCP a offrire le opportune garanzie. L'ultimo caso è il TCP-over-TCP che, come mostrato nei capitoli precedenti, può non portare benefici ma, addirittura, cali di prestazioni.

VPN amplierebbe il raggio di scelte possibili, poiché quelle presentate fan parte di un piccolo sottoinsieme di soluzioni esistenti. Ne esistono alcune che utilizzano protocolli sia a livello di trasporto che a livello di rete: esempi che sarebbe opportuno considerare sono IPSec (*IP Security*) ed L2TP (*Layer 2 Tuneling Protocol*). IPSec estende IP con una suite di protocolli per la crittazione e l'autenticazione; le due modalità di utilizzo sono la *tunnel mode*, nella quale il pacchetto IP viene incapsulato completamente in un nuovo datagramma tramite IPSec, e *transport mode*, nel quale solo al payload di IP vengono aggiunti gli *header* che gestiscono la sicurezza (Figura 4.2).

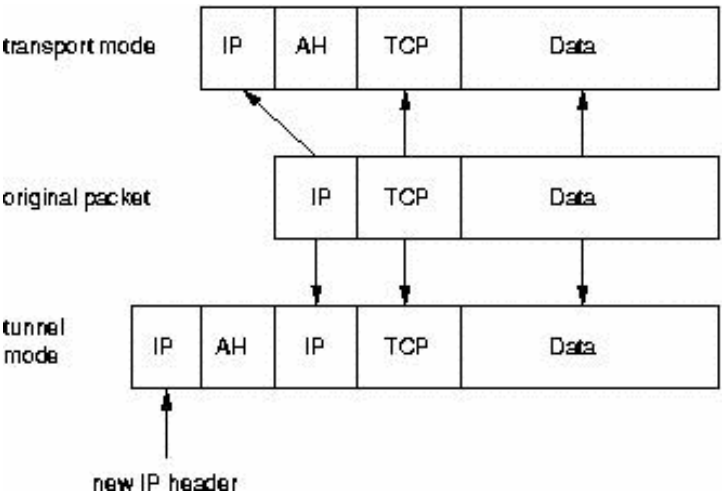

Figura 4.2: Le due modalità di utilizzo di IPSec. Figura tratta da *http://www.ipsec-howto.org/italian/x151.html*

L2TF è un protocollo di *tunneling* a livello di collegamento che fonda le sue origini nei due protocolli L2F (*Layer 2 Forwarding Protocol*) di Cisco e PPTP. Pubblicata inizialmente nel 1999 (RFC 2661), nel 2005 viene proposta come standard una nuova versione del protocollo, L2TPv3 (RFC 3931). L2TP non gestisce alcun meccanismo di sicurezza, per questo è necessario

abbinarlo a protocolli di autenticazione e criptazione. Una combinazione spesso utilizzata prevede l'uso congiunto di L2TP per la parte *encapsulation* e IPSec come gestore della sicurezza.

Lo stesso protocollo, come già detto, può avere prestazioni diverse in base all'implementazione della soluzione: per questo motivo sarebbe interessante aggiungere altri strumenti VPN, come *OpenS/WAN* e *Tinc*.

Questa tesi si è concentrata sulla valutazione delle *performance* dei sistemi, tralasciando una delle caratteristiche fondanti delle VPN, la sicurezza. La sicurezza è un parametro non trascurabile nell'analisi di questi tipi di reti, perciò l'incentrarsi maggiormente su questa caratteristica, sia a livello di autenticazione che a livello di cifratura dei dati, darebbe informazioni aggiuntive nella valutazione complessiva.

L'ultimo punto importante con il quale poter estendere l'analisi riguarda l'introduzione di un terzo scenario di valutazione: nonostante le VPN possano essere create in LAN, nascono e vengono più spesso utilizzate per connettere due *host* (o due sottoreti) geograficamente molto distanti tra loro, che richiederebbero l'installazione di collegamenti dedicati per poter comunicare liberi da possibili intrusioni. Perciò lo scenario migliore sarebbe quello che vede Internet come base di collegamento tra i due *endpoint* (Figura 4.3). Visto che Internet introduce una maggiore complessità, aggiungendo eventualità difficilmente ottenibili in una rete locale, sarebbe opportuno seguire i metodi di *testing* consigliati dall'RFC 2544 (*Benchmarking Methodology for Network Interconnect Devices*).

Figura 4.3: A sinistra il server VPN, a destra la sotto-rete che contiene il client. La VPN utilizza Internet per veicolare i pacchetti.

Infine è interessante sottolineare una caratteristica delle VPN che in questo lavoro è passata in secondo piano: la flessibilità, ovvero l'essere indipendente dalla posizione del client. La mobilità ha una grande importanza, tanto più il poter accedere ai propri dati da un qualsiasi luogo. I più importanti sistemi operativi di smartphone e tablet forniscono già dei client VPN integrati, e molti altri sono disponibili come applicazioni esterne. Interessante sarebbe l'estensione dei test anche in questo ambito, con connessioni create attraverso sistemi quali 3G, UMTS e LTE.

Appendice A

Configurazione server OpenVPN

```
#################################################
# Sample OpenVPN 2.0 config file for            #
# multi-client server.                          #
#                                               #
# This file is for the server side              #
# of a many-clients <-> one-server              #
# OpenVPN configuration.                        #
#################################################

# Which TCP/UDP port should OpenVPN listen on?
port 1194

# TCP or UDP server?
;proto tcp
proto udp

# "dev tun" will create a routed IP tunnel,
```

```
# "dev tap" will create an ethernet tunnel.
;dev tap
dev tun

# SSL/TLS root certificate (ca), certificate
# (cert), and private key (key).  Each client
# and the server must have their own cert and
# key file.  The server and all clients will
# use the same ca file.
ca ca.crt
cert server.crt
key server.key  # This file should be kept secret

# Diffie hellman parameters.
dh dh1024.pem

# Configure server mode and supply a VPN subnet
# for OpenVPN to draw client addresses from.
# The server will take 10.8.0.1 for itself,
# the rest will be made available to clients.
server 10.8.0.0 255.255.255.0

# Maintain a record of client <-> virtual IP address
# associations in this file.  If OpenVPN goes down or
# is restarted, reconnecting clients can be assigned
# the same virtual IP address from the pool that was
# previously assigned.
```

```
ifconfig-pool-persist ipp.txt

# Uncomment this directive to allow different
# clients to be able to "see" each other.
# By default, clients will only see the server.
;client-to-client

# The keepalive directive causes ping-like
# messages to be sent back and forth over
# the link so that each side knows when
# the other side has gone down.
keepalive 10 120

# For extra security beyond that provided
# by SSL/TLS, create an "HMAC firewall"
# to help block DoS attacks and UDP port flooding.
# The server and each client must have
# a copy of this key.
# The second parameter should be '0'
# on the server and '1' on the clients.
tls-auth key.txt 0 # This file is secret

# Select a cryptographic cipher.
cipher BF-CBC        # Blowfish (default)
;cipher AES-128-CBC   # AES
;cipher DES-EDE3-CBC  # Triple-DES
```

```
# Enable compression on the VPN link.
comp-lzo

# It's a good idea to reduce the OpenVPN
# daemon's privileges after initialization.
user nobody
group nogroup

# The persist options will try to avoid
# accessing certain resources on restart
# that may no longer be accessible because
# of the privilege downgrade.
persist-key
persist-tun

# Output a short status file showing
# current connections, truncated
# and rewritten every minute.
status openvpn-status.log

# By default, log messages will go to the syslog (or
# on Windows, if running as a service, they will go to
# the "\Program Files\OpenVPN\log" directory).
log         openvpn.log
;log-append  openvpn.log

# Set the appropriate level of log
```

```
# file verbosity.
#
# 0 is silent, except for fatal errors
# 4 is reasonable for general usage
# 5 and 6 can help to debug connection problems
# 9 is extremely verbose
verb 3
```

Appendice B

Configurazione demone PPTP

pptpd.conf

```
############################################################
# $Id$                                                     #
#                                                          #
# Sample Poptop configuration file /etc/pptpd.conf         #
#                                                          #
# Changes are effective when pptpd is restarted.           #
############################################################

# TAG: ppp
# Path to the pppd program, default '/usr/sbin/pppd' on Linux
#
#ppp /usr/sbin/pppd

# TAG: option
# Specifies the location of the PPP options file.
# By default PPP looks in '/etc/ppp/options'
```

```
#
option /etc/ppp/pptpd-options

# TAG: debug
# Turns on (more) debugging to syslog
#
debug

# TAG: stimeout
# Specifies timeout (in seconds) on starting ctrl connection
#
# stimeout 10

# TAG: noipparam
#       Suppress the passing of the client's IP address to PPP, which is
#       done by default otherwise.
#
#noipparam

# TAG: logwtmp
# Use wtmp(5) to record client connections and disconnections.
#
logwtmp

# TAG: bcrelay <if>
# Turns on broadcast relay to clients from interface <if>
#
```

```
#bcrelay eth1

# TAG: localip
# TAG: remoteip
# Specifies the local and remote IP address ranges.
#
#       Any addresses work as long as the local machine takes care of the
#       routing.  But if you want to use MS-Windows networking, you should
#       use IP addresses out of the LAN address space and use the proxyarp
#       option in the pppd options file, or run bcrelay.
#
# You can specify single IP addresses seperated by commas or you can
# specify ranges, or both. For example:
#
# 192.168.0.234,192.168.0.245-249,192.168.0.254
#
# IMPORTANT RESTRICTIONS:
#
# 1. No spaces are permitted between commas or within addresses.
#
# 2. If you give more IP addresses than MAX_CONNECTIONS, it will
#    start at the beginning of the list and go until it gets
#    MAX_CONNECTIONS IPs. Others will be ignored.
#
# 3. No shortcuts in ranges! ie. 234-8 does not mean 234 to 238,
#    you must type 234-238 if you mean this.
#
```

```
# 4. If you give a single localIP, that's ok - all local IPs will
#    be set to the given one. You MUST still give at least one remote
#    IP for each simultaneous client.
#
# (Recommended)
localip 10.0.0.1
remoteip 10.0.0.2
# or
#localip 192.168.0.234-238,192.168.0.245
#remoteip 192.168.1.234-238,192.168.1.245
```

pptpd-options

```
###############################################################################
# $Id$                                                                        #
#                                                                             #
# Sample Poptop PPP options file /etc/ppp/pptpd-options                       #
# Options used by PPP when a connection arrives from a client.                #
# This file is pointed to by /etc/pptpd.conf option keyword.                  #
# Changes are effective on the next connection.  See "man pppd".              #
#                                                                             #
# You are expected to change this file to suit your system.  As               #
# packaged, it requires PPP 2.4.2 and the kernel MPPE module.                 #
###############################################################################


# Authentication
```

```
# Name of the local system for authentication purposes
# (must match the second field in /etc/ppp/chap-secrets entries)
name VPNPPTP

# Optional: domain name to use for authentication
# domain mydomain.net

# Strip the domain prefix from the username before authentication.
# (applies if you use pppd with chapms-strip-domain patch)
#chapms-strip-domain



# Encryption
# Debian: on systems with a kernel built with the package
# kernel-patch-mppe >= 2.4.2 and using ppp >= 2.4.2, ...
# {{{
refuse-pap
refuse-chap
refuse-mschap
# Require the peer to authenticate itself using MS-CHAPv2 [Microsoft
# Challenge Handshake Authentication Protocol, Version 2] authentication.
require-mschap-v2
# Require MPPE 128-bit encryption
# (note that MPPE requires the use of MSCHAP-V2 during authentication)
require-mppe-128
# }}}
```

```
# Network and Routing

# If pppd is acting as a server for Microsoft Windows clients, this
# option allows pppd to supply one or two DNS (Domain Name Server)
# addresses to the clients.  The first instance of this option
# specifies the primary DNS address; the second instance (if given)
# specifies the secondary DNS address.
# Attention! This information may not be taken into account by a Windows
# client. See KB311218 in Microsoft's knowledge base for more information.
#ms-dns 10.0.0.1
#ms-dns 10.0.0.2

# If pppd is acting as a server for Microsoft Windows or "Samba"
# clients, this option allows pppd to supply one or two WINS (Windows
# Internet Name Services) server addresses to the clients.  The first
# instance of this option specifies the primary WINS address; the
# second instance (if given) specifies the secondary WINS address.
#ms-wins 10.0.0.3
#ms-wins 10.0.0.4

# Add an entry to this system's ARP [Address Resolution Protocol]
# table with the IP address of the peer and the Ethernet address of this
# system.  This will have the effect of making the peer appear to other
# systems to be on the local ethernet.
# (you do not need this if your PPTP server is responsible for routing
# packets to the clients -- James Cameron)
```

```
proxyarp

# Debian: do not replace the default route
nodefaultroute


# Logging

# Enable connection debugging facilities.
# (see your syslog configuration for where pppd sends to)
debug

# Print out all the option values which have been set.
# (often requested by mailing list to verify options)
#dump


# Miscellaneous

# Create a UUCP-style lock file for the pseudo-tty to ensure exclusive
# access.
lock

# Disable BSD-Compress compression
nobsdcomp
```

Bibliografia

[1] Michael Hall, 2008,
Performance Analysis of OpenVPN on a Consumer Grade Router.

[2] C. Scott, P. Wolfe, M. Erwin, 1999,
Virtual private networks, (second ed.), O'Reilly

[3] D. Napolitano, 2006,
Analisi sperimentale di soluzioni open-source per la realizzazione di VPN

[4] Charlie Hosner, 2004,
OpenVPN and the SSL VPN Revolution.

[5] Stijn Huyghe, 2004,
OpenVPN 101: introduction to OpenVPN.

[6] Home page OpenVPN, http://openvpn.net/

[7] NeoRouter Inc., 2010,
How it works, http://download.neorouter.com/Documents/nr_howitworks_en.pdf

[8] NeoRouter Inc., 2010,
User's manual, http://download.neorouter.com/Documents/nr_usermanual_10en.pdf

[9] *Request For Comments (RFC), 2637, http://www.ietf.org/rfc/rfc2637.txt*

[10] *Request For Comments (RFC), 1661, http://www.ietf.org/rfc/rfc1661.txt*

[11] *Request For Comments (RFC), 1701, http://www.ietf.org/rfc/rfc1701.txt*

[12] Wireshark User's guide,
http://www.wireshark.org/docs/wsug_html_chunked/index.html

[13] Wireshark Wiki,
http://wiki.wireshark.org/

[14] Linux Ping Manual,
http://linuxreviews.org/man/ping/index.html.it

[15] Wikipedia: Ping,
http://it.wikipedia.org/wiki/Ping

[16] Tom Armstrong, 2001,
Netcat - The TCP/IP Swiss Army Knife

[17] Iperf,
http://iperf.sourceforge.net/

[18] Jperf 2.0.2,
http://code.google.com/p/xjperf/

[19] Martin A. Brown, 2007,
Guide to IP Layer Network Administration with Linux, Appendix C.1

[20] Linux Network Administrators Guide, 5.8 All about ifconfig, *http://www.faqs.org/docs/linux_network/x-087-2-iface.ifconfig.html*

[21] Ubuntu manuals, ifconfig man page, *http://manpages.ubuntu.com/manpages/oneiric/en/man8/ifconfig.8.html*

[22] Software "Wget" Manual, http://www.gnu.org/software/wget/manual/wget.html

[23] Software "Wget" Wiki, http://en.wikipedia.org/wiki/Wget

[24] Software "dd" Manual, http://www.linuxmanpages.com/man1/dd.1.php

[25] Software "dd" Wiki, http://en.wikipedia.org/wiki/Dd_(Unix)

[26] Software "TAR" Wiki, http://it.wikipedia.org/wiki/TAR_(software)

[27] *Request For Comments (RFC), 1962, http://tools.ietf.org/html/rfc1962*

[28] O. Honda, H. Ohsaki, M. Imase, M. Ishizuka, J. Murayama, *Understanding TCP over TCP: Effects of TCP Tunneling on End-to-End Throughput and Latency*, pag. 1

[29] *Request For Comments (RFC), 2401, http://tools.ietf.org/html/rfc2401*